UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CENTRO DE EDUCACIÓN A DISTANCIA Y GESTIÓN DEL CONOCIMIENTO

APUNTE ELECTRÓNICO

Informática III.
Análisis y Diseño de Sistemas Estructurado

Licenciatura en Informática

UNAM FCA CEDGEC Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

COLABORADORES

DIRECTOR DE LA FCA
Mtro. Tomás Humberto Rubio Pérez

SECRETARIO GENERAL
Dr. Armando Tomé González

– – – –

COORDINACIÓN GENERAL
Mtra. Gabriela Montero Montiel
Jefe del Centro de Educación a Distancia y Gestión del Conocimiento

COORDINACIÓN ACADÉMICA
Mtro. Francisco Hernández Mendoza
FCA-UNAM

COORDINACIÓN DE MULTIMEDIOS
L.A. Heber Javier Mendez Grajeda
FCA-UNAM

– – – –

COAUTORES
Mtro: Virgilio Sámano Núñez
Lic. Uriel Marín Monterde

REVISIÓN PEDAGÓGICA
Mtro. Joel Guzmán Mosqueda

CORRECCIÓN DE ESTILO
Mtro. Carlos Rodolfo Rodríguez de Alba

DISEÑO DE PORTADAS
L.CG. Ricardo Alberto Báez Caballero

DISEÑO EDITORIAL

UNAM FCA
CED:GEC
Centro de Educación a Distancia y Gestión del Conocimiento

OBJETIVO GENERAL

Al finalizar el curso, el alumno aprenderá a desarrollar sistemas utilizando el análisis y diseño apropiados, según el enfoque estructurado.

TEMARIO OFICIAL

(64 horas)

	Horas
1. Introducción	10
2. Análisis de sistemas	26
3. Diseño de sistemas	28
TOTAL	64

INTRODUCCIÓN

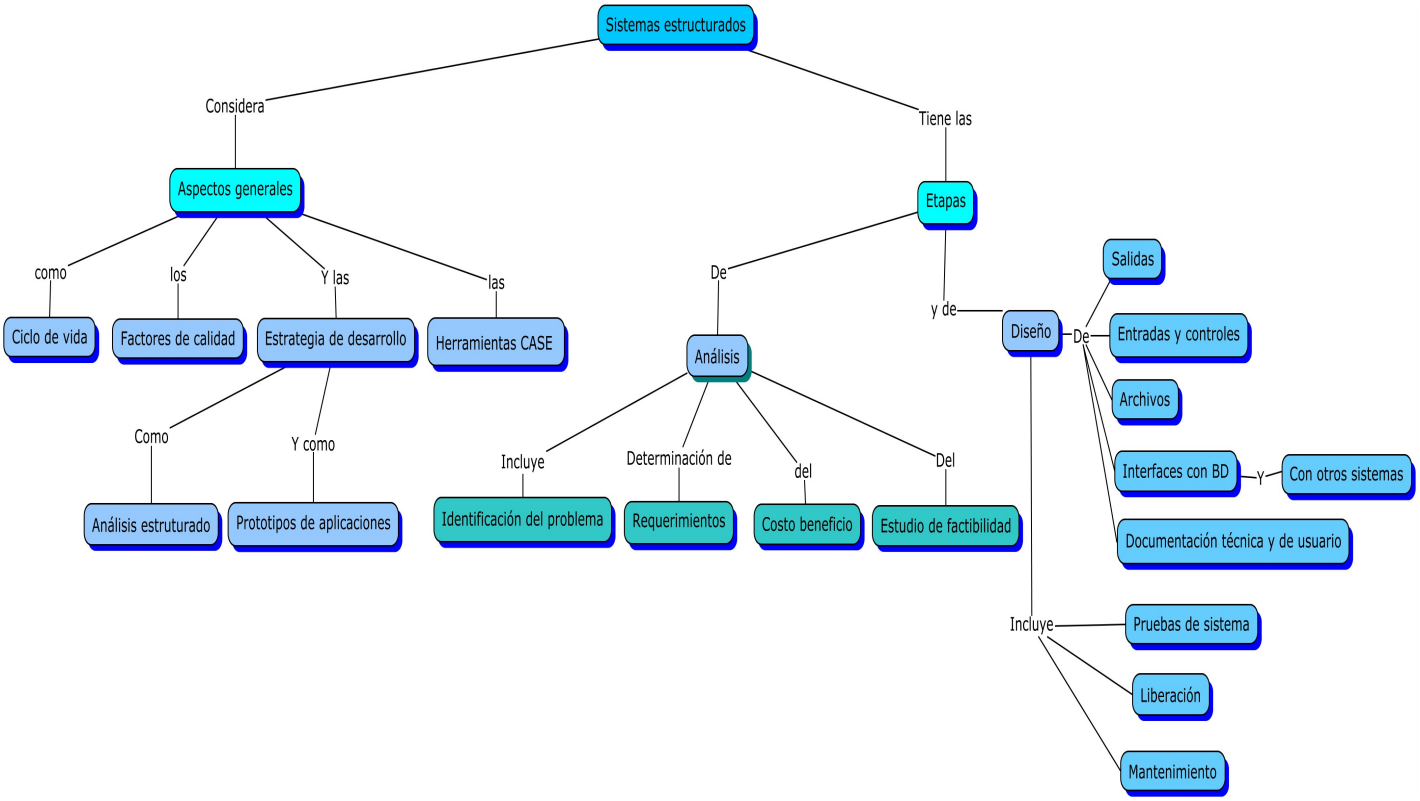
Esta asignatura tiene como propósito señalar los conceptos más relevantes que dan sustento al análisis y diseño de sistemas como etapas del ciclo de vida de los propios sistemas, para que puedas identificarlos. La estructura de la materia se conforma de tres unidades:

1. **Introducción**. Se explican los conceptos que conforman la Teoría General de Sistemas que hoy en día siguen estando vigentes y tienen gran relevancia en el desarrollo de sistemas. Asimismo, se abordan los modelos del ciclo de vida de sistemas que conforman las bases de los procesos de desarrollo de *software*.

2. **Análisis Estructurado**. Se explican los conceptos que intervienen en la administración de requerimientos y en el análisis de sistemas.

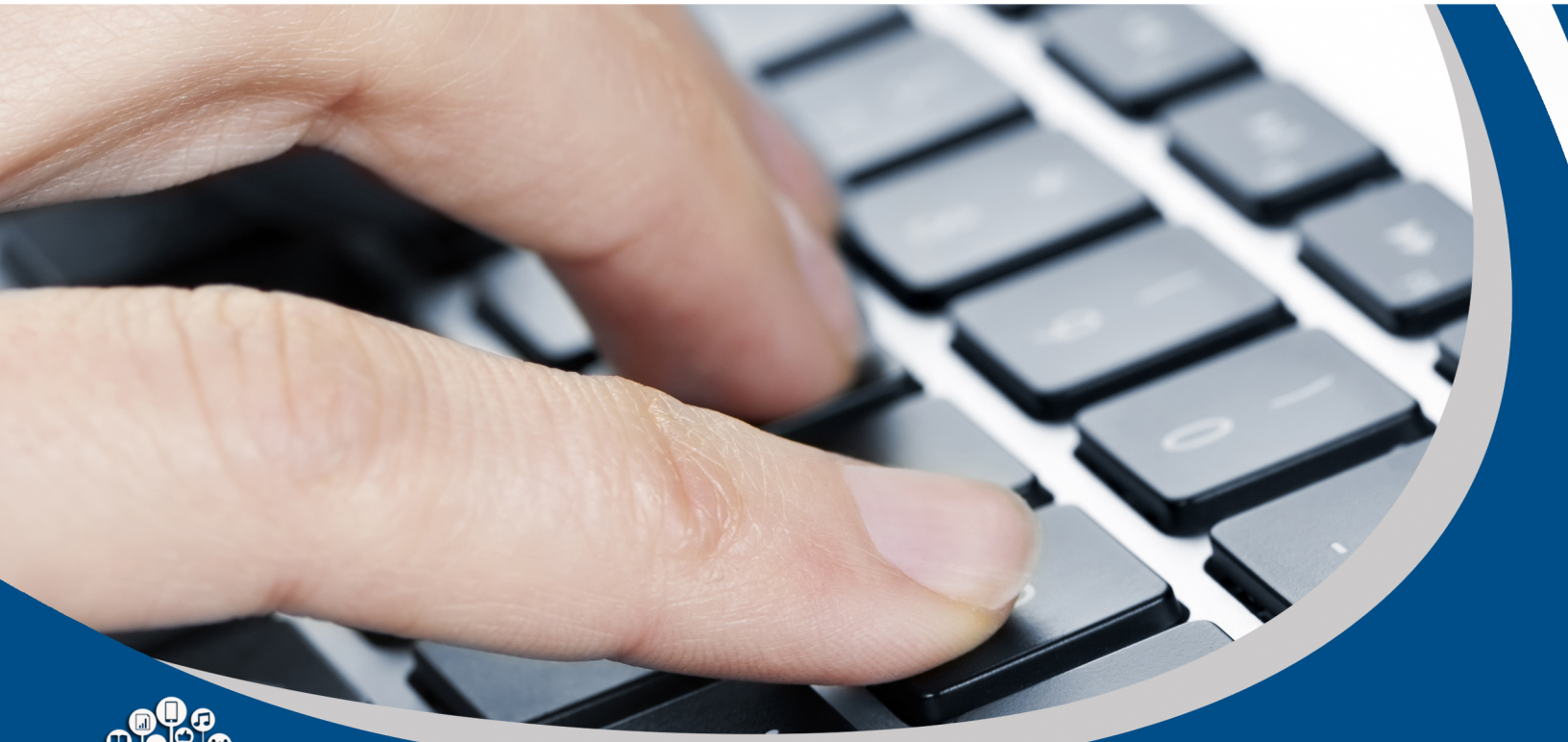
3. **Diseño Estructurado**. El contenido de esta asignatura es fundamental para tu formación como Licenciado en Informática; la asignatura de *Análisis y diseño estructurado de sistemas* conforma la base para asignaturas que cursarás en el mismo semestre, así como para futuros semestres, por tal razón, necesitas poner especial atención en los conceptos y procedimientos que aquí se explicarán.

ESTRUCTURA CONCEPTUAL



UNIDAD 1

INTRODUCCIÓN



UNAM FCA
CEDIGEC
Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

OBJETIVO PARTICULAR

Identificar las fases del ciclo de vida de sistemas y ubicar las responsabilidades del analista y del diseñador en dicho ciclo.

TEMARIO DETALLADO
(10 horas)

1. Introducción

1.1. Concepto de análisis y diseño
1.2. Ciclo de vida de los sistemas
1.3. Factores de calidad del *software*
1.4. Estrategia de desarrollo por análisis estructurado
1.5 Estrategia de desarrollo por prototipos de aplicaciones
1.6 Herramientas asistidas por computadora para el desarrollo de sistemas (CASE)

INTRODUCCIÓN

Al hablar del tema de sistemas, es recomendable comenzar por hacer algunos bosquejos, e ir visualizando qué se quiere obtener como resultado final.

Los profesionales en informática que se dedican al desarrollo de *software* o sistemas, aplican diversas herramientas y elaboran diagramas de flujo, pruebas de escritorio, pseudocódigos, etc., para comenzar a realizar la planeación del desarrollo de algún sistema.

Para desarrollar un sistema de información se hace uso de las etapas que conforman varios pasos, por ejemplo, en el ciclo de vida de los sistemas (análisis, diseño, implementación, pruebas, implantación, mantenimiento). De éstas, el análisis y diseño son procesos que examinan una situación de necesidades de manejo de información, se especifican los procedimientos y componentes necesarios para lograr el objetivo.

Al referirnos al desarrollo de *software*, no se puede hablar de una sola metodología; pero en la literatura sobre el tema, se llega a concluir que la metodología consiste en un conjunto de pasos y procedimientos que se deben seguir para el desarrollo óptimo del *software*.

De acuerdo con Cataldi, que a su vez cita a Maddison, se define la metodología como “un conjunto de fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información” (Cataldi, 2014:14).

El objetivo de las metodologías es ayudar a los desarrolladores de sistemas de información a controlar el desarrollo del *software*, facilitar su mantenimiento y estimación de recursos a utilizar, y hacer eficiente el tiempo requerido.

Una metodología se compone de elementos que especifican:

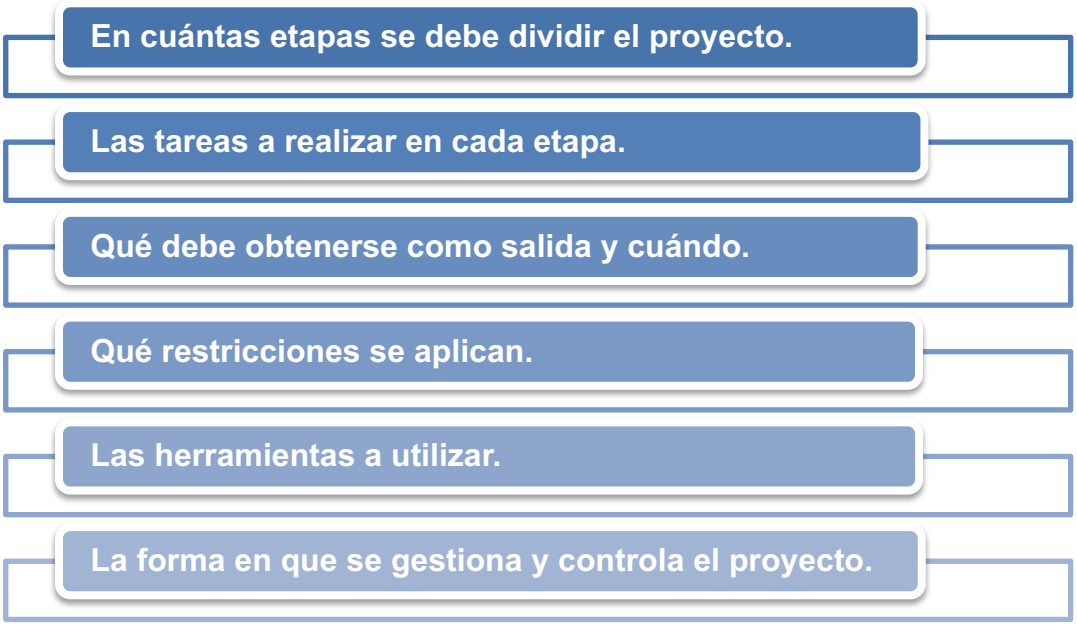


Figura 1.1 Elementos que integra la metodología.

El uso correcto de una metodología evita inconsistencias, desfases de tiempo, malos dimensionamientos y omisiones durante el análisis y diseño. Por consiguiente, en el desarrollo del sistema pueden presentarse algunos errores en la etapa de pruebas generando atrasos importantes en la entrega e implantación de la solución informática, lo que conlleva a tener un mayor gasto económico y físico.

Desde el año 1970 se han propuesto y evolucionado varios métodos para el modelado. Sin embargo, ahora dos tendencias dominan el panorama: El

estructurado o convencional que se verá en esta materia y las orientadas a objetos como RUP (*Rational Unified Process*), UML (lenguaje unificado de modelado), XP (*Extreme Programming*) y Métrica Versión 3, entre otros basados en componentes y objetos que modularizan la información y el procesamiento del problema a resolver.

Para el modelado estructurado existen 3 estrategias o métodos de desarrollo.

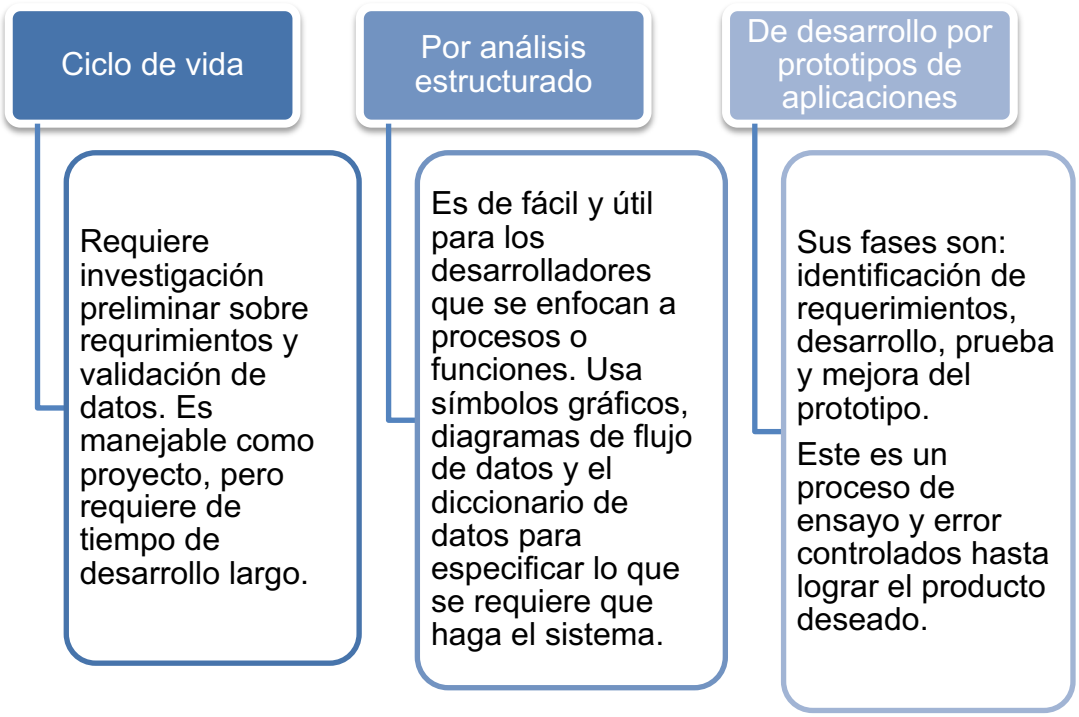


Figura 1.2. Estrategias o métodos de desarrollo. Elaboración propia con base en Senn, 1992: 33;40-41;45.

Aunque se elija la mejor metodología y se lleve a cabo con todo rigor, es casi imposible desarrollar sistemas libres de errores. Los analistas buscan la manera de evitarlo, por eso hay varias propuestas conocidas como modelos de calidad de *software*, cada uno con sus factores a considerar para garantizar la calidad del sistema. Algunos autores como Senn, agrupan estos factores en confiabilidad y mantenimiento. En cambio,

McCally Cavano los categorizan en productos de operación (operativos), productos de revisión (mantenimiento) y productos de transición (evolutivos), bajo la premisa de que los sistemas deben ser confiables, fáciles de mantener y de preferencia evitar mejoras. En el punto 1.3 de esta unidad se mencionarán los modelos y estándares más reconocidos, con el fin de que se tenga un panorama más amplio de esta rama en la que muchos de nuestros colegas se han especializado, ya sea como asesores o como auditores de calidad del *software*.

En la última parte de esta unidad, se aborda el tema de las herramientas asistidas por computadora para el desarrollo de sistemas conocidos como CASE, las cuales le permiten al analista disminuir el tiempo necesario para llevar a cabo las tareas, reduce la intensidad de trabajo, seguimiento de los procedimientos y mejora de la calidad del sistema. Constan de 5 componentes: herramientas de diagramación, depósito de información, generadores de interfaces, generadores de código y herramientas de administración.

1.1. Concepto de análisis y diseño

El análisis y diseño son los componentes más importantes para desarrollar un sistema: el análisis nos da una idea de qué se debe hacer y el diseño el cómo realizarlo.

Análisis

En su primera acepción, elimológicamente, este término proviene del griego ἀνάλυσις: *análysis.* Que significa, distinción y separación de las partes de algo para conocer su composición.[1]

Esta fase contesta la pregunta ¿qué? y es una descripción de las causas, necesidades y problemas existentes, por lo que se puede observar y modelar cómo funciona la organización o sistema actual. Otro nombre que recibe esta fase, es el de “requerimientos” y sus principales actividades son las siguientes:

- Diagnosticar la necesidad o problema a resolver.
- Identificar a los participantes, así como sus necesidades.
- Establecer los objetivos que persigue el sistema.
- Establecer los alcances y exclusiones.
- Crear un modelo de dominio o negocio con su respectiva especificación, describiendo la situación actual.

El analista de sistemas es una persona experta en la identificación y comprensión de los problemas y oportunidades; con madurez, visión y una amplia experiencia en la solución de problemas, el cual usa tecnologías de información para conjugar sus conocimientos y juicios al momento de

[1] DRAE. Recuperado el 11 de marzo de 2019 de: https://dle.rae.es/?id=2Vga9Gy

crear soluciones, y actúa como enlace entre la ingeniería de sistemas y los usuarios.

Dependiendo de las funciones de un analista de sistemas se puede clasificar en:

Analista diseñador de sistemas	**Programador de sistemas**
"Expertos en tecnología que traducen las necesidades y las restricciones manifestadas por los usuarios de la empresa en soluciones técnicas"[2] Algunas de las tareas que realiza son: Algunas actividades que realizan son: -Actualización o volver a diseñar un sistema viejo. -Planificar el nuevo sistema. -Capacitar a los usuarios del sistema. -Elaboración de manuales del usuario. -Revisan minuciosamente el programa y si tiene las funciones que promete para dar satisfacción al cliente.[3]	Son expertos en el uso de varios lenguajes de programación. Realizan tareas de -Investigación -Diseño -Desarrollo de programas de una computadora. Por lo regular siguen las especificaciones proporcionadas por un analista diseñador de sistemas informáticos, desglosando cada uno de los pasos que deberán tener los diagramas del programa. Su propósito es que los sistemas informáticos tengan un funcionamiento eficiente, por ejemplo, en el manejo de datos, salida de datos a impresoras y conexiones a sistemas de telecomunicaciones.[4]

[2] Definición tomada de: Fernández Alarcón Vicenc, (2006) *Desarrollo de sistemas de información. Una metodología basada en el modelado*. p. 212.
[3] Educaweb (s/f) Recuperado el 28 de marzo de 2019 de: https://www.educaweb.com/profesion/analista-sistemas-informaticos-362/
[4] Educaweb (s/f) Recuperado el 28 de marzo de 2019 de: https://www.educaweb.com/profesion/programador-sistemas-informaticos-363/

Tabla 1.1. Definición y funciones de los analistas diseñadores de sistemas y Programadores de sistemas. Elaboración propia.

Tanto los analistas diseñadores y programadores de sistemas trabajan de manera conjunta en el desarrollo de un *software*. Cada uno se diferencia por las actividades que definen sus denominaciones. “El analista de sistemas más valioso y mejor calificado es aquel que sabe programar y conoce los estándares de calidad” (Senn (1992: 12).

En su calidad de analista de sistemas, debe promover el cambio de tal manera que involucre el uso de los sistemas de información. También es parte de su tarea enseñar a los usuarios el proceso del cambio, ya que las modificaciones a un sistema de información no sólo afectan a éste, sino que provocan cambios en el resto de la organización (Senn, 1992: 12).

Responsabilidades del analista

- Asociar las necesidades con el problema principal por resolver o la oportunidad por conseguir.
- Planear las actividades de análisis de sistemas.
- Escoger (o diseñar) y utilizar los métodos, técnicas y herramientas más adecuadas para el análisis del sistema.
- Estudiar el sistema de planeación, organización, dirección y control de la empresa.
- Representar con algún tipo de especificación los procesos que se realizan en la organización y que están relacionados con el sistema por desarrollar.
- Modelar los aspectos dinámicos y estáticos del sistema actual.
- Proponer y aplicar las medidas de carácter organizativo que se requieran para perfeccionar los procesos.

- Revisar los resultados obtenidos y elaborados por los programas.
- Elaborar los datos de prueba para comprobar la calidad de los programas individualmente y en su conjunto.
- Capacitar a los usuarios en la operación del sistema.

Kendall (2011: 6) define el perfil de los analistas de sistema basándose en las fases impuestas en el paradigma Ciclo de Vida de Desarrollo de Sistemas (CVDS) como:

Consultor. Cuando se comienza el CVDS el analista cumple en papel de consultor, asesorando a la empresa sobre los mejores métodos y sistemas que se pueden emplear para la óptima gestión de información, recomendando sistemas ya sean de tipo manual o de tipo informático, se puede traducir en una ventaja porque los consultores externos tienen una perspectiva fresca de la cual carecen los demás miembros de una organización. También se puede traducir en una desventaja porque alguien externo nunca conocerá la verdadera cultura organizacional.

Experto de soporte. En este rol el analista recurre a su experiencia profesional en el uso del *hardware* y *software* que pueda darse en una organización o negocio, aunque no necesariamente debe ser requerido en todo el proyecto, por lo regular sugiere o realiza pequeñas modificaciones o es tomado en cuenta para algunas decisiones.

Agente de cambio. Kendal lo define como:

> El rol más extenso y responsable del analista de sistemas es el de agente de cambio, ya sea interno o externo, para la empresa. Podemos definir a un agente de cambio como una persona que actúa como catalizador para el cambio, desarrolla un plan de cambio y trabaja con otros para facilitarlo (Kendall, 2011: 7).

Diseño

Para los fines de este tema, se retoma la segunda acepción que presenta del término la RAE:

Del italiano: *disegno*: Proyecto, plan que configura algo.[5]

Aplicado el término al tema que nos ocupa, esta fase contesta a la pregunta ¿cómo?, se refiere a cómo se va a dar solución a las necesidades y/o problemas. Sus actividades principales son:

- Diseñar procesos y procedimientos.
- Diseñar algoritmos.
- Diseñar la base de datos.
- Diseñar la distribución de los componentes.

Según Senn:

> El diseño de sistemas es el proceso de planificar, reemplazar o complementar un sistema organizacional existente. Pero antes de llevar a cabo esta planeación es necesario comprender, en su totalidad, el viejo sistema y determinar la mejor forma en que se pueden, si es posible, utilizar las computadoras para hacer la operación más eficiente. El análisis de sistemas, por consiguiente, es el proceso de clasificación e interpretación de hechos, diagnóstico de problemas y empleo de la información para recomendar mejoras al sistema. (…) El análisis especifica qué es lo que el sistema debe hacer. El diseño establece cómo alcanzar el objetivo (1992:12-13).

[5] DRAE, Recuperado el 11 de marzo de 2019 de: https://dle.rae.es/?id=DuKP0H9

1.2. Ciclo de vida de los sistemas

La concepción de *sistemas* tiene su origen en las ciencias naturales por tratar de describir un ser vivo como un conjunto de partes, el cual tiene un ciclo de vida (*nace, crece, se reproduce y muere*).

El método para el desarrollo de sistemas, *conocido como ciclo de vida*, es un modelo teórico que tiene como propósito agrupar las actividades para construir un sistema en un conjunto de pasos o fases, dichos pasos son: *análisis, diseño, implementación, pruebas, implantación y mantenimiento*.

Este modelo también es llamado de cascada o lineal secuencial y dependiendo del autor que se consulte, las etapas o fases pueden ser más o pueden ser menos, integrarse en una sola o cambiar de nombre, pero en esencia tienen el propósito de organizar la información y los procedimientos para desarrollar e implementar un sistema de información.

Análisis

Su propósito es traducir los requerimientos a especificaciones que describan cómo implementar el sistema y "permite al ingeniero de sistemas especificar las características operacionales del *software* (función, datos y rendimientos), indica la interfaz del *software* con otros elementos del sistema y establece las restricciones que debe cumplir el *software*" (Pressman, 2002: 182).

Algunos autores incluyen en esta etapa la investigación preliminar que considera estudios de factibilidad técnica, operacional y económica.

Diseño

Si en el análisis se traducen los requerimientos, en esta etapa se deben transformar esos requerimientos a un diseño de sistema que indique los datos de entrada, cálculo y salida apoyándose en diagramas, tablas, símbolos especiales y herramientas automatizadas, así como seleccionar la mejor estrategia de implementación.

Una vez detallado el diseño, éste se entrega al área de desarrollo o implementación del *software* y es probable que surjan dudas que los diseñadores deberán aclarar a los desarrolladores.

Implementación

Esta fase se encarga de construir los componentes del *software* que le darán funcionalidad al sistema. Otros nombres que recibe esta fase son los de “desarrollo”, “programación” o “construcción” y sus actividades principales son:

- Codificar algoritmos a programas.
- Hacer o probar la infraestructura de cómputo necesaria para que el sistema por implantar funcione adecuadamente (redes, equipo, bases de datos, etc.).
- Construir todos los archivos de datos necesarios.
- Hacer programas de apoyo para mejorar o adaptar el sistema por implementar, esto es importante sobre todo si el sistema fue comprado.

Pruebas

Esta fase se encarga de asegurar que la calidad del *software* sea la correcta: por un lado, se verifica que el *software* se construya de manera correcta, es decir, que se apegue a los estándares establecidos, y, por el otro, se valida que el *software* que se construya sea el correcto; o sea, que satisfaga las necesidades por las cuales se concibió el sistema y que garantice no tener fallas. Otros nombres que recibe esta fase son los de "verificación", "validación" o "evaluación". Lo verás con más detalle en el tema de factores de calidad del *software* de esta unidad.

Implantación

Esta fase se encarga de poner en marcha el *software* construido y capacitar a las personas involucradas para su uso. Sus fases son:

- Verificación de funcionamiento.
- Instalación del nuevo sistema.
- Entrega del sistema para su puesta en marcha.
- Entrenar a los usuarios.

Dependiendo del tamaño de la organización y el riesgo de uso, esta etapa puede realizarse área por área de la organización, funcionando el viejo sistema y el nuevo, o desechando completamente el viejo.

Mantenimiento

"El soporte y mantenimiento del *software* vuelve a aplicar cada una de las fases precedentes a un programa ya existente y no a uno nuevo" (Pressman, 2002: 20).

Esta fase se enfoca en realizar las estrategias, procedimientos o procesos para corregir errores, hacer adaptaciones conforme el uso del sistema va evolucionando y mejoras requeridas por el cliente, el entorno o desarrollo de nuevas tecnologías que afecten de manera directa al funcionamiento del sistema, un ejemplo común de ello son las constantes actualizaciones de los sistemas operativos. Estas formas de mantenimiento reciben los nombres de:

a) Mantenimiento correctivo

La calidad de un sistema implica eliminar cualquier posibilidad de error o falla, pero no hay sistema perfecto o infalible. Este tipo de mantenimiento corrige errores no previstos y por lo regular se realiza (y debe realizarse) después de que el cliente ha usado el sistema por un tiempo.

b) Mantenimiento preventivo

Tiene como propósito evitar fallas que sucederían en un futuro, por ejemplo, la depuración de archivos temporales generados por algún programa o sistema para evitar la saturación del disco.

De acuerdo con Pressman (2002), los programas de computadora se deterioran debido a que cambian continuamente y por eso se recomienda el mantenimiento preventivo. Es preciso que cualquier *software* sea de utilidad y exclusivamente para las necesidades de los usuarios finales, el mantenimiento correctivo se encarga de hacer cambios en la computadora, de manera que se puedan corregir, adaptar y mejorar continuamente.

c) Mantenimiento adaptativo.

Sucede cuando hay cambios o mejoras en la tecnología usada en el sistema como pueden ser los dispositivos periféricos del servidor o las terminales, un sistema operativo nuevo o cambiar las interfaces del sistema a otras plataformas como Web o Android, que pueden mejorar el acceso y desempeño del sistema. También puede ser por nuevas leyes gubernamentales (ej. factura impresa a factura digital), cambios en los procedimientos administrativos o incluso cambio de dueño de la empresa. "El mantenimiento adaptativo produce modificación en el *software* para acomodarlo a los cambios de su entorno externo" (Pressman, 2002:15).

1.3. Factores de calidad del software

En relación al desarrollo de *software* anteriormente, la calidad de una aplicación sólo se centraba en que hiciera lo que el usuario requería y sólo se cuidaba a la hora de probar el sistema, con el paso del tiempo, se vio que esto implicaba pérdidas económicas y de tiempo cuando, una vez terminado un producto, acababa siendo desechado después de las pruebas.

Por ello la calidad de un sistema de información depende de su diseño, desarrollo, pruebas e implantación, por lo que debe estar presente en todas las fases de desarrollo del *software* desde la determinación de requerimientos, hasta el mantenimiento, independientemente del modelo de desarrollo que se elija, y bajo estándares específicos que definen criterios de desarrollo para la ingeniería de *software*.

Desde los primeros años en que se comenzaron a desarrollar sistemas, los errores empezaron y también las propuestas que consideran otros factores y procedimientos de verificación o auditoría que aseguren la calidad del *software* de acuerdo a estándares o certificaciones.

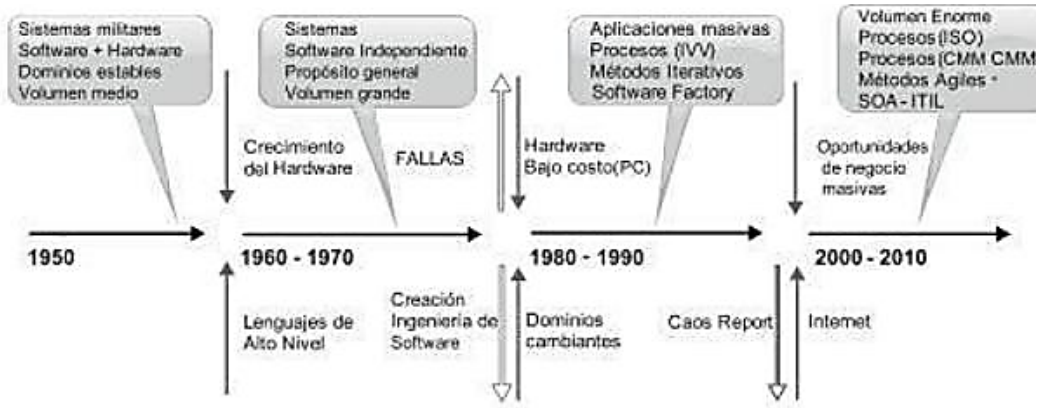


Figura 1.3. Fuente: Pantaleo, G. (2012). *Evolución histórica del desarrollo de software y del concepto de calidad asociado. Calidad en el desarrollo de software.* Marcombo, p. 20.

PRIMEROS FACTORES DE CALIDAD DEL SOFTWARE

En 1977 McCall y Cavano, hicieron los primeros intentos de proponer métricas de la calidad del *software* y en él se han basado muchos de los nuevos modelos e incluso estándares. Este modelo clasifica los factores de calidad del *software* en 3 categorías o ejes con sus respectivos criterios que el analista debe considerar para evaluar la calidad de su sistema.

1) Operación del producto (características operativas).
2) Revisión del producto (capacidad de modificarlo).
3) Transición del producto (capacidad de adaptarlo a nuevos entornos).

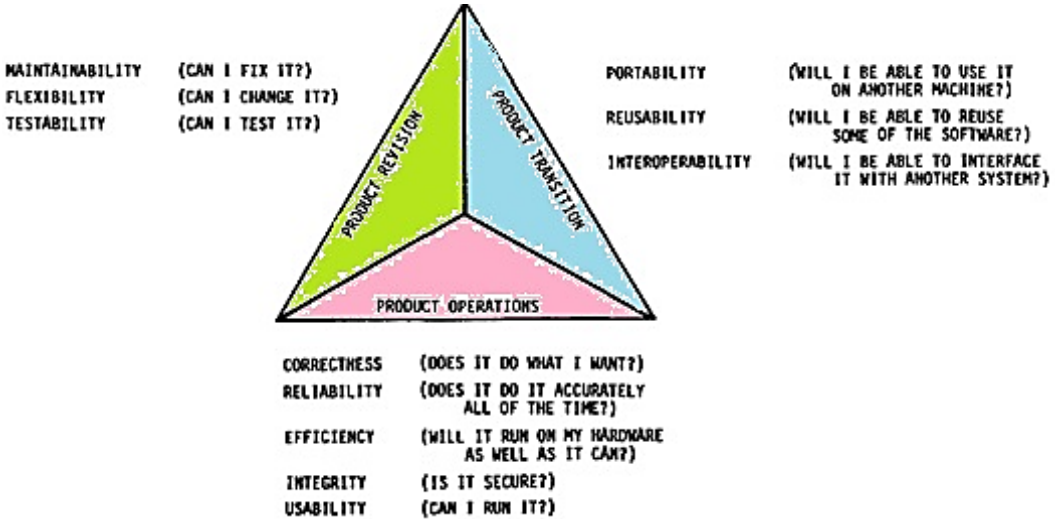


Figura. 1.4 Fuente: Cavano y McCall (1977). "Software Quality Factors. A framework for the measurement of software quality". *Acm sigmetrics performance evaluation review* 7 (3-4): 133-139.

El siguiente cuadro resume el trabajo de McCall y Cavano.

Ejes	Factor	Criterios
OPERACIÓN DEL PRODUCTO	Facilidad de uso El esfuerzo necesario para aprender a operar con el sistema, preparar los datos de entrada e interpretar las salidas (resultados) de un programa. ¿Puedo ejecutarlo?	- Facilidad de operación: El *software* tiene atributos que facilitan la operación del *software*. - Facilidad de comunicación: El *software* tiene atributos que proporcionan entradas y salidas fácilmente asimilables. - Facilidad de aprendizaje: El *software* tiene atributos que facilitan la familiarización inicial del usuario con el *software*. - Formación: El grado en que el *software* ayuda a los nuevos usuarios a manejar el sistema.
	Integridad Hasta dónde se puede controlar el acceso al *software* o a los datos por personas no autorizadas. ¿Es seguro?	- Control de accesos. Atributos del *software* que proporcionan control de acceso al *software* y los datos que maneja. - Facilidad de auditoría: Atributos que facilitan la auditoría de los accesos al *software*. - Seguridad: Mecanismos que controlen o protejan los programas o los datos de accesos no autorizados.
	Corrección Hasta dónde satisface un programa su especificación y logra los objetivos propuestos por el cliente. ¿Hace lo que quiero?	- Completitud: El *software* tiene implementadas todas las funciones requeridas. - Consistencia: Grado con que se ha logrado la implementación total de una función. - Trazabilidad o rastreabilidad: La capacidad de seguir una representación del diseño hasta los requisitos con respecto a un entorno operativo concreto.

	Fiabilidad Hasta dónde se puede esperar que un programa lleve a cabo su función con la exactitud requerida. ¿Lo hace con precisión todo el tiempo?	- Precisión: Grado requerido en los cálculos y los resultados. -Tolerancia a fallos: Capacidad del *software* para continuar funcionando en caso de errores o fallas. -Modularidad: Independencia funcional en la estructura del *software.* -Simplicidad: Grado de facilidad con que se puede entender un programa.
	Eficiencia La cantidad de recursos informáticos y de código necesarios para que un programa realice su función. ¿Funcionará en mi *hardware* como se espera?	-Eficiencia en ejecución: Rendimiento en el tiempo de procesamiento. -Eficiencia en almacenamiento: Uso mínimo de espacio de almacenamiento
REVISIÓN DEL PRODUCTO	Facilidad de Mantenimiento Esfuerzo necesario para localizar y arreglar un error en un programa. ¿Puedo arreglarlo?	-Concisión: Programación con la menor cantidad de código posible. -Auto descripción: Grado en que el código fuente proporciona explicación significativa de sus funciones.
	Facilidad de prueba Esfuerzo necesario para probar un programa y asegurarse de que realiza su función correctamente. ¿Puedo probarlo?	-Instrumentación: Atributos del *software* que posibilitan la observación del comportamiento del *software* durante su ejecución para facilitar las mediciones del uso o la identificación de errores.

	Flexibilidad Esfuerzo necesario para modificar un programa que ya está en funcionamiento. ¿Puedo cambiarlo?	-Capacidad de expansión: Grado de expansión del *software* en cuanto a diseño arquitectónico o procedimental, capacidades funcionales y datos. -Generalidad: Amplitud de las funciones implementadas y los componentes del programa.
TRANSICIÓN DEL PRODUCTO	Reusabilidad Capacidad de volver a utilizar un programa o partes de él de acuerdo al alcance de las funciones que realiza. ¿Puedo utilizar alguna parte? Portabilidad Esfuerzo requerido para transferir el programa desde un *hardware* y/o un entorno de sistema de *software* a otro. ¿Puedo usarlo en otro sistema?	-Independencia entre sistema y *software*: Grado de independencia respecto a las características del lenguaje de programación no estándar, características del sistema operativo y otras restricciones del entorno. - Independencia del *hardware*: Grado que determina la dependencia del *software* con el *hardware* donde opera. (Los criterios se consideran en ambos factores)
	Interoperabilidad Esfuerzo necesario para acoplar un sistema con otro. ¿Puedo conectarlo con otro sistema?	-Compatibilidad de comunicaciones: Atributos del *software* que posibilitan el uso de protocolos de comunicación e interfaces estándar. -Compatibilidad de datos: Atributos del *software* que posibilitan el uso representaciones de datos estándar. -Estandarización en los datos: El empleo de estructuras de datos y de tipos estándar a lo largo de todo el programa.

Tabla. 1.2. Elaboración basada en: Pressman, Roger. (2002). *Ingeniería del software. Un enfoque práctico*. [5ª ed.] Madrid: McGraw-Hill, pp. 324 y 325.

Para comprender la calidad del *software*, es preciso retomar el concepto de Pressman: “La concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo documentados y con las características implícitas que se esperan de todo *software* desarrollado profesionalmente” (2002:186).

Para ello deben realizarse las siguientes actividades:

1. Aplicación de metodología y técnicas de desarrollo.
2. Reutilización de procesos de revisión formales.
3. Prueba del *software.*
4. Ajustes a los estándares de desarrollo.
5. Control de cambios, mediciones y recopilación de información.
6. Gestión de informes sobre el control de calidad.

Según Senn, el aseguramiento de la calidad es:

“La revisión de los productos y documentación relacionada con el *software* para verificar su cobertura, corrección, confiabilidad y facilidad de mantenimiento” (Senn, 1992: 793).

Para ello se usan 4 niveles:

Prueba	Verificación	Validación	Certificación.
Proceso de ejecutar un programa de forma parcial o total con la intención explícita de hacer que el sistema falle.	Tiene la intención de hallar errores bajo un ambiente simulado.	Busca errores en un ambiente no simulado con grupos de control o instalaciones de prueba.	Es una garantía sometida a auditorías y estándares. Se realiza por lo regular para sistemas comerciales en lugar de crearlo.

Tabla 1.3. Niveles de aseguramiento de la calidad.

Este autor propone un modelo de calidad, donde los objetivos a lograr dentro de la calidad del *software* son: la confiabilidad y el mantenimiento de la misma aplicación. Para lograrlo los analistas de sistemas llevan a cabo una serie de pasos (metodología) que aseguren que el sistema es confiable cuando sea instalado y que la confiabilidad se mantenga después de la puesta en marcha.

Confiabilidad

Un sistema confiable es aquel que no produce peligros o fallas cuando se utiliza en forma razonable; es decir, de la manera que un usuario común espera que sea normal. La confiabilidad depende de dos niveles:

1. **Que el sistema cubra los requerimientos correctos**. Aquí el analista deberá realizar una completa y efectiva determinación de los requerimientos del sistema para evitar errores en el diseño y pérdidas de tiempo en el desarrollo.

2. **El trabajo efectivo proporcionado al usuario**. Aquí el desarrollo del *software* depende de que la aplicación final que se entregue al usuario sea la que él necesita.

Se debe estar consciente de que a pesar de los mejores esfuerzos que se hagan, es imposible alcanzar este propósito en su totalidad; sin embargo, en los programas se debe buscar este objetivo. Los errores no sólo son de sintaxis, existen muchos tipos de ellos, aquí se identifica a los de mayor importancia en las etapas de Análisis y Diseño. A continuación, se mencionan los errores que interesan en la etapa de análisis:

1) No obtener requerimientos en forma correcta;
2) No obtener los requerimientos adecuados; y
3) No interpretar los requerimientos en forma clara y entendible de manera que los programadores los puedan poner en marcha en forma apropiada.

El Mantenimiento de sistemas

Cuando existen equivocaciones y son detectadas después de que el sistema ya se haya puesto en marcha, se recurre al mantenimiento. Éste siempre será vigente, mientras el sistema se encuentre vigente. Aquí el trabajo del analista es tomar las precauciones para asegurar que la necesidad de mantenimiento sea controlada a través del diseño y prueba. También se incluyen las adaptaciones a las versiones iniciales del *software*.

En cuanto a las adaptaciones del *software,* se considera que la mayor cantidad de trabajo de mantenimiento es para mejoras a solicitud del usuario, de la documentación o del registro de componentes del sistema

para una mayor eficiencia. Incluso muchas de las tareas se podrían evitar si se llevara una metodología formal de análisis y diseño apropiada.

Actualmente, existen varios modelos que agrupan los controles de calidad de forma jerárquica, proponen fases e incluso aseguran la calidad del *software* por medio de auditorías y certificaciones[6]. En la actualidad, muchos informáticos e ingenieros son expertos en estos modelos y se dedican a la asesoría y auditoría en gestión, planificación y control de la calidad de sistemas.

Los modelos de calidad, se agrupan de acuerdo a las fases de desarrollo de software y para garantizar el correcto funcionamiento de las aplicaciones, sugieren que se debe mejorar el proceso de desarrollo para entregar al usuario final un producto satisfactorio. A continuación, se muestran los niveles mencionados.

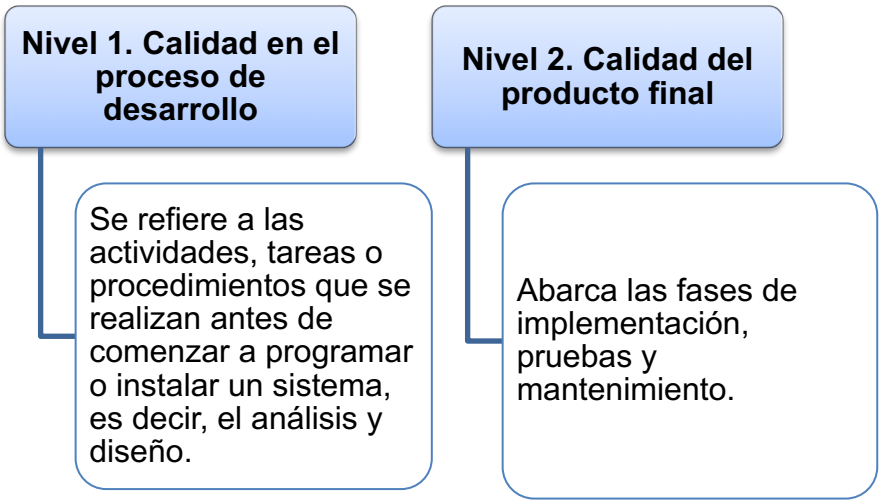


Figura 1.5. Niveles de calidad en el desarrollo de *software*. Elaboración propia.

[6] Nota: En cuanto a las certificaciones que se mencionan, cabe aclarar que lo que se certifica son los procedimientos del desarrollo del sistema.

MODELOS ACTUALES PARA GARANTIZAR LA CALIDAD DEL *SOFTWARE*

Modelo de calidad del software al proceso de desarrollo		
Nombre	**Organización**	**Factores o niveles**
CMMi (*Capability Matury Model Integration*)	SEI (*Software Enginering Institut*e) www.sei.cmu.edu	-Visitas por niveles y capacidades. -Área de proceso. -Grupos de áreas de procesos. -Ingeniería (7 áreas de procesos) -Proyectos (5 áreas de procesos). -Organizativas (6 áreas de procesos).
PSM (*Practical Software Measurement*) **Relacionado con CMMi y la norma ISO/IEC 15939**		-**Desempeño:** Lo que produce el proceso. **-Estabilidad**: Comportamiento del proceso. -**Conformidad**: Fallas dentro de un rango aceptable. -**Capacidad**: El proceso se ajusta a los requerimientos. -**Mejoramiento:** Mejora el desempeño del proceso.
PSP (Personal *Software* Process)		Niveles **-Inicial.** Seguimiento y control de proyectos, planeación de los proyectos. **-Repetible** Revisión entre colegas, manejo integrado del *software*, definición del proceso de software y foco del proceso de *software*. -**Definido** Control de calidad y administración cuantitativa del

		proyecto. **-Controlado** Administración de los cambios del proceso, administración del cambio tecnológico y prevención de defectos. **<u>Fases.</u>** **-PSPO** (proceso personal de arranque): Proceso base, registro de tiempos, registro de errores, estándar de tipo de errores, estándar de codificación, medición de tamaño y propuesta de mejoramiento del proceso. **PSP1 (Proceso personal de administración):** Estimación del tiempo, reporte de pruebas planeación de actividades y planeación de tiempos. **PSP2 (Proceso personal de calidad):** Revisión de codificación, revisión de diseño y formatos de diseño. **PSP2 (Proceso cíclico):** Desarrollo de ciclos.
ITIL (Information Technology Infraestructure Library)	United Kingdom's Office of Government Commerce (OGC) www.itil-officialsite.com/home/	**<u>Áreas de aplicación</u>** **-Administración de servicios:** *Service soport y service delivery.*

		-Guías operacionales: *ICT Infraestructure: management, security, management, business perspective, aplication management and software asset management***.** **-Guías de implementación:** *Planning to implement service management and ITIL small-scale implementation*. **<u>Fases</u>** ITIL service strategy ITIL service design ITIL service transition ITIL service operation ITIL continual service improvement
MOPROSOFT (Modelo de procesos para la industria del *software*)	Asociación mexicana para la calidad en ingeniería de *software*. www.comunidadmoprosoft.org.mx Concebido en México, basado en el modelo CMM y en las normas ISO/IEC 15504 para las PyME desarrolladoras de *software*.	**<u>Procesos estratificados según el organigrama típico:</u>** **-Alta dirección:** Gestión de negocio (visión y objetivos). **-Gerencia:** Gestión de procesos, gestión de proyectos y gestión de

		recursos. **-Operación** Administración de proyectos desarrollo y mantenimiento de *software*. **<u>-Directrices</u>** -Los procesos deben definirse manteniendo una relación integradora. -El proceso de administración de proyectos es atómico. -La ingeniería de productos lleva a cabo procesos de verificación, validación, documentación y control de la documentación para garantizar y controlar la calidad de los mismos.
TickIT **<u>Basado en la norma ISO 9001:2000</u>**	National Accreditation Council of Certification Bodies http://www.qcin.org/nabcb/	**-Efectividad** **-Eficiencia** **-Confidencialidad** **-Integridad** **-Disponibilidad** **-Conformidad** **-Confiabilidad**
Bootstrap	European Strategic Programme for Research in Information Technology (ESPRIT) http://cordis.europa.eu/esprit/home.html	**-Procesos dependientes del ciclo de vida.** **-Procesos independientes del ciclo de vida.** **-Gestión de procesos** **- Procesos relacionados**

		Definición de procesos Mejora de procesos Evaluación de procesos Medición
DMADV (Define - Measure - Analyze - Design - Verify)	Isixsigma http://www.isixsigma.com	**-Definir** **-Medir** **-Analizar** **-Diseñar** **-Verificar**
DMAIC (Define - Measure - Analyze - Improve - Control)		**-Definir** **-Medir** **-Analizar** **-Mejorar** **-Controlar**
EFQM (European Foundation for Quality Management)	Fundación Europea para la Gestión de la Calidad http://www.efqm.es/	Tiene varios criterios, pero el de procesos es donde se evalúa el desarrollo del *software*. **Criterios** **-Liderazgo** **-Política y estrategia** **-Gestión del personal** **-Gestión de los recursos** **-Procesos** Identifarlos Gestionarlos Revisarlos

	Establecer objetivos Mejorar los procesos mediante creatividad e innovación Modificar los procesos y evaluar las ventajas **Satisfacción del cliente** **Satisfacción del personal** **Impacto en la sociedad** **Resultados empresariales**
FMEA (Failure mode and effects analysis)[7]	**Identificar la falla** **Determinar el índice de severidad** **Estimar tasa de ocurrencia** **Definir el número de prioridad de riesgo** **Proceso de optimización** Reahacer la solución para reducir la severidad o su causa Mejorar la fiabilidad para minimizar la ocurrencia de la falla Mejora el proceso de detección para evitar que el usuario encuentre la falla

Tabla 1.4. Elaboración basada en: Pantaleo, Guillermo (2012). *Calidad en el desarrollo de software.* Barcelona: MARCOMBO.

7Villamil y García (2003), *Introducción al proyecto de ingeniería*. Disponible en: http://materias.fi.uba.ar/6612/archives/Libro_materia.pdf Recuperado: 07/09/2018.

Modelos de calidad del *software* a nivel producto.		
Nombre	**Organización o autor**	**Factores o niveles**
Gilb	Tom Gilb y Susannah Finzi	• Capacidad de trabajo • Disponibilidad • Adaptabilidad • Utilidad • Grado de corrección • Facilidad de mantenimiento • Integridad • Facilidad de uso
Mc Call	Joseph P. Cavano y James A. McCall	• Operación del producto • Revisión del producto • Transición del producto
GQM	Basili y Weiss (1984)	• Planificación • Definición • Recopilación de datos • Interpretación
FURPS (functionality, usability, reliability, performance, supportability)	Robert Grady y Heweltt Packard	• Funcionalidad • Facilidad de uso • Confiabilidad • Performance • Facilidad de soporte Categorías de requerimientos: • Funcionales • No funcionales
Boehm	Barry Boehm (1978)	• Portabilidad • Confiabilidad • Eficiencia • Ingeniería humana • Facilidad de prueba • Facilidad de comprensión • Facilidad de modificación

SATC (Software Assurance Technology Center)		• **Calidad de los Requerimientos**
		  ○ Ambigüedad
		  ○ Integridad
		  ○ Facilidad de entender
		  ○ Volatilidad del requerimiento
		  ○ Trazabilidad
		• **Calidad del Producto**

		○ Estructura ○ Arquitectura ○ Facilidad de mantenimiento ○ Reusabilidad ○ Documentación interna ○ Documentación externa • **Efectividad de la implementación** ○ Uso de los recursos ○ Porcentaje de terminación • **Efectividad de la prueba** ○ Corrección
Dromey	Geoff Dromey	• **Corrección** ○ Funcionalidad ○ Confiabilidad • **Aspectos internos** ○ Facilidad de mantenimiento ○ Eficiencia ○ Confiabilidad • **Aspectos del contexto** ○ Facilidad de mantenimiento ○ Confiabilidad ○ Portabilidad • **Aspectos descriptivos** ○ Facilidad de mantenimiento ○ Facilidad de uso ○ Eficiencia ○ Confiabilidad
C-QM	www.cqm-tech.com	• **Funcionalidad** ○ Adaptabilidad ○ Integridad • **Reusabilidad** ○ Modularidad ○ Comprensión ○ Construcción según especificaciones • **Facilidad de mantenimiento**

		○ Modularidad ○ Abstracción ○ Facilidad de cambio • **Conformidad** ○ Estándar ○ De acuerdo al modelo de referencia
Metodología SQAE (Software Quality Assessment Exercise)	MITRE Corporation	Dependiendo del área de calidad se considera un porcentaje de factor. • **Mantenimiento** ○ Modularidad (20%) ○ Descripción media (20%) ○ Simplicidad en el diseño (15%) ○ Consistencia (15%) ○ Control de anomalías (15%) ○ Documentación (15%) • **Evolución** ○ Modularidad (20%) ○ Simplicidad en el diseño (25%) ○ Control de anomalías (20%) ○ Documentación (10%) ○ Descripción media (25%) • **Portabilidad** ○ Independencia (40%) ○ Modularidad (20%) ○ Documentación (15%) ○ Descripción media (25%) • **Descripción** ○ Descripción media (50%) ○ Documentación (50%)
WebQEM (Web Quality Evaluation	www.webqem.com	• **Facilidad de uso** ○ Comprensibilidad global del sitio ○ Mecanismos de ayuda y

Method)	retroalimentación en línea ○ Aspectos de interfaces y estéticos ○ Soporte a lenguaje extranjero • **Funcionalidad** ○ Aspectos de búsqueda y recuperación ○ Aspectos de navegación y exploración ○ Aspectos del dominio orientados al estudiante • **Confiabilidad** ○ Errores de enlaces ○ Errores varios • **Eficiencia** ○ Performance ○ Accesibilidad

Tabla. 1.5. Elaboración con base en: Scalone, 2006: 129-148.

Un sistema debe tomar en cuenta otros factores, es un atributo a compararse con estándares conocidos, en el caso del *software* existe el estándar ISO/IEC 25000 que remplazó al ISO/IEC 9126. Conocido también como SquaRE (*Softwareproduct Quality Requirements and Evaluation*) basado en el modelo McCall que establece criterios para definir métricas y evaluar los requisitos de calidad del *software*.

Muchos de estos modelos están regidos por estándares que también se agrupan a nivel de proceso de desarrollo y nivel de producto.

Estándares de calidad del *software* a nivel proceso de desarrollo			
Organización: ISO			
ISO 90003	ISO/IEC 12207 Organización: ISO	ISO 15504 (SPICE)	ISO/IEC 9001:2000
-Sistema de Gestión de la Calidad **Documentación de requisitos**	**Procesos principales**	**Dimensión proceso -Cliente-Proveedor**	-Objeto y campo de aplicación -Referencias normativas -Términos y definiciones -Sistema de Gestión de la Calidad -Responsabilidad de la Dirección -Gestión de los Recursos, -Realización del Producto **Medición, Análisis y Mejora**
•Responsabilidad de la dirección **Compromiso de la dirección** **Enfoque al cliente** **Política de calidad** **Planificación** **Responsabilidad, autoridad y comunicación** **Revisión por la dirección** -Gestión de los Recursos **Provisión de recursos** **Recursos humanos** **Infraestructura** **Ambiente de**	Proceso de adquisición Proceso de suministro Proceso de desarrollo Proceso de operación Proceso de mantenimiento **-Procesos de soporte** Proceso de documentación Proceso de gestión de la configuración Proceso de Aseguramiento de la Calidad Proceso de	Proceso de adquisición Establecimiento de contratos Identificar las necesidades del cliente Realizar auditorías y revisiones conjuntas Empaquetar, entregar e instalar el software Mantenimiento del software Proporcionar servicio al cliente Valorar la satisfacción del cliente **-Ingeniería** Establecer los requisitos y diseño del sistema Análisis de requerimientos Desarrollar el diseño Implementar el diseño Integración y pruebas Mantenimiento **-Soporte**	

trabajo -Realización del Producto **Planificación de la realización del producto** **Procesos relacionados con el cliente** **Diseño y desarrollo** **Compras** **Producción y prestación de servicios** **Control de los dispositivos de seguimiento y de medición** -Medida, Análisis y Mejora **Seguimiento y medición** **Control del producto no conforme** **Análisis de los**	Verificación Proceso de Validación Proceso de Revisión Proceso de Auditoría Proceso de Resolución del problema **-Procesos de la organización** Proceso de Gestión Proceso de Infraestructura Proceso de Mejora **-Proceso de formación**	Documentación Gestión de la configuración Garantía de la calidad Verificación del producto Validación del producto Realizar revisiones conjuntas Auditoría Resolución de problemas **-Gestión** Del proceso Del proyecto Gestionar la calidad Gestionar los riesgos **Organización** Diseño del negocio Definir el proceso Mejora del proceso Entrenamiento Reutilización Proporcionar soporte informático Proporcionar facilidad de trabajo **Proyecto** Planificar el ciclo de vida del proyecto Establecer el plan del proyecto Armar los equipos de proyecto Gestionar los requisitos Gestionar la calidad	

datos **Mejora**	Gestión de riesgos Gestión de recursos y calendario Administrar los contratos	

Tabla 1.6. Estándares de calidad del *software* a nivel proceso de desarrollo. Elaboración propia.

Estándares de calidad del software a nivel producto		
Nombre	**Organización**	**Factores o niveles**
ISO 9126-1	ISO	**-Funcionalidad** Aptitud Precisión Interoperabilidad Conformidad Seguridad Trazabilidad **-Confiabilidad** Madurez Tolerancia a fallas Facilidad de recuperación Disponibilidad Degradabilidad **-Facilidad de uso** Comprensibilidad Facilidad de aprendizaje Operatividad Explicitud Adaptabilidad al usuario Atractividad Claridad Facilidad de ayuda Amigable al usuario **-Eficiencia** Respecto al tiempo Respecto a los usuarios **-Facilidad de mantenimiento** Facilidad de análisis Facilidad de cambio Estabilidad Facilidad de prueba **-Portabilidad** Adaptabilidad Facilidad de instalación Conformidad Facilidad de reemplazo
ISO 25000 (SQUARE)		**-Especificación de requerimientos** **-Planificación** **-Medición** **-Evaluación**
IEEE Std 1061-1998		**-Establecer requisitos de calidad de software**

		Identificar una lista de posibles requisitos de calidad Determinar la lista de los requisitos de calidad Cuantificar cada factor de calidad **Identificar las métricas de calidad de software** Aplicar las métricas de calidad de *software* Realizar un análisis de costo-beneficio Obtener el compromiso de las métricas establecidas **Implementar las métricas** Definir los procedimientos de recogida de datos Prototipo del proceso de medición Recopilar los datos y calcular los valores de la métrica **-Analizar los resultados de las métricas** Interpretar los resultados Identificar la calidad del *software* Hacer predicciones de calidad del *software* Garantizar el cumplimiento de los requisitos **Validar las métricas** Aplicar la metodología de validación Aplicar criterios de validez procedimiento de validación

Tabla. 1.7. Estándares de calidad del software a nivel producto. Elaboración propia.

1.4. Estrategia de desarrollo por análisis estructurado

En la actualidad, los sistemas informáticos permiten a los altos directivos conocer la información generada en las diferentes áreas, sucursales, oficinas, puntos de venta y, a la vez, obtener información centralizada; con el propósito de tener mayor control de las actividades que se desarrollan en la organización o bien, incrementar la efectividad en la operación de las mismas.

Mucho de ello sucede gracias a que hubo personas que invirtieron tiempo, experiencia y esfuerzo para analizar, desarrollar e implantar un sistema de información.

Sin embargo, es muy común que un proyecto de sistema de información sea sobre un área de la organización de la que el analista tiene poco conocimiento. Es por ello que el analista debe tomar en cuenta aspectos como las tecnologías de información con que se cuenta, el impacto directo o indirecto del nuevo sistema sobre el personal, las características que el nuevo sistema debe tener o tendrá después de cierta evolución y en qué se va a utilizar.

Según Senn, se espera que los analistas hagan lo siguiente para implementar o mejorar un sistema:

- Aprendan los detalles y procedimientos del sistema en uso.
- Obtengan una idea de las demandas futuras de la organización.
- Documentar detalles del sistema actual para su revisión y discusión.

- Evaluar la eficiencia y efectividad del sistema actual y sus procedimientos.
- Fomentar la participación de gerentes y empleados en todo el proceso.
- Documentar las características del nuevo sistema que permita comprender todos sus componentes (Senn, 1992: 174).

De acuerdo con el perfil de egreso del profesional en informática en la FCA de la UNAM, el contexto laboral se caracteriza por ser partícipe de la globalización que predomina en la actualidad en todos los ámbitos de la vida; por ejemplo, la economía, el mercado, las finanzas que afectan de manera directa a las organizaciones. Además, se caracteriza por ser formado para ser competitivo y estar a la altura de las exigencias actuales en cuanto a avances tecnológicos se refiere, con una formación sólida y una visión integradora, que posibilite su desempeño con "conocimientos, actitudes y destrezas que les permitan una participación comprometida y eficiente, económica y socialmente, dentro del ámbito de su campo profesional".[8]

En el contexto laboral en las organizaciones, este profesional trabaja en equipo con otros profesionales con formación afín, por lo que en ocasiones, no es quien programa el sistema; ya sea porque hay un especialista o grupo especializado en programación, bases de datos, telecomunicaciones o cualquier otra tecnología de información y comunicaciones. Sin embargo, la participación del profesional en informática como analista es una parte medular para el éxito del proyecto que lleve a una mejora con un mínimo de interrupciones para el área donde se usará dicho sistema.

[8] FCA, UNAM (2016) *Actualización del plan y programas de estudio de la licenciatura de informática*, Licenciatura en informática. 2.1. Demandas del contexto. p. 13. Recuperado el 01 de abril de 2019 de: **http://licenciaturas.fca.unam.mx/docs/informatica/plan_2012-2016_sua/proyecto_li_v2.pdf**

Para lograr lo anterior, se propusieron tres estrategias (tratados como métodos por otros autores), éstos son: Ciclo de vida (explicado en el punto 2 de esta unidad), el análisis estructurado (tema que abordamos) y por prototipo de sistemas (que veremos en las siguientes secciones de esta unidad).

¿QUÉ ES EL ANÁLISIS ESTRUCTURADO?

> El análisis estructurado es un método para el análisis de sistemas manuales o automatizados, que conduce al desarrollo de especificaciones para sistemas nuevos o para efectuar modificaciones a los ya existentes (…) permite al analista conocer un sistema o proceso (actividad) en una forma lógica y manejable al mismo tiempo que proporciona la base para asegurar que no se omite ningún detalle pertinente (Senn, 1992: 176).

Es decir que:

- Se busca darles una estructura a los procesos.
- Los procesos se organizan incluyendo los detalles relevantes que describen al sistema.
- Facilita la identificación de requerimientos, mejores soluciones y estrategias de implementación.
- Es un eficiente medio de comunicación para todos los que participan en el desarrollo o mejora del sistema.

A finales de los 70, Edward Yourdon propuso esta estrategia de desarrollo de *software* bajo la filosofía de programación estructurada, a raíz del escrito de Tom DeMarco de finales de los 70 titulado *Structured Analysis and Systemas Specification* y de la obra de Gane titulada *Structured System Analysis*: *Tools and Techniques*.

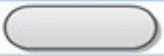
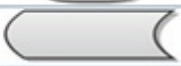
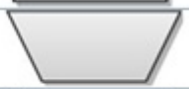
Esta estrategia permite comprender sistemas grandes y complejos por medio de la división del sistema en componentes y la construcción de un modelo que organice las tareas asociadas con la determinación de requerimientos y le permite al analista conocer:

- Detalles y procedimientos del sistema en uso para su revisión y discusión con los usuarios.
- Demandas futuras de la organización como resultado del crecimiento del sistema.
- Documentar las características del nuevo sistema de tal manera que otras personas lo puedan entender.

Esta estrategia, no establece cómo cumplir con los requerimientos o la forma de implantarlo, porque ello se realiza en la fase de diseño y desarrollo, respectivamente.

COMPONENTES DEL ANÁLISIS ESTRUCTURADO

- Símbolos gráficos. Por medio de determinadas figuras, se señalan las funciones del sistema sin definir detalles físicos, como procesos manuales o computarizados, tipo de *hardware*, comunicaciones, individuos o departamentos, de tal modo que cualquier persona pueda entender la forma en que interactúan los elementos del sistema, como se muestra a continuación:

Nombre	Símbolo	Función
Inicio/Final		Se utiliza para representar el inicio o fin de un proceso o programa
Entrada/Salida		Se utiliza para representar la introducción de datos por medio de periféricos.
Proceso		Se utiliza para representar cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transformaciones, etc.
Decisión		Se utiliza para indicar operaciones lógicas o de comparación entre datos.
Documento		Se utiliza para representar la salida de datos por impresora, pero en ocasiones es usado para mostrar datos o resultados.
Desplegar/Mostrar		Este es utilizado para representar la salida o para mostrar la información por medio del monitor o la pantalla.
Proceso predefinido		Se utiliza para representar procesos ya definidos tales como llamada a procedimientos o funciones y el inicio del mismo.
Base de datos		Se utiliza para representar la escritura o almacenado de datos en la base de datos.
Almacenamiento de datos		Se utiliza para representar la escritura o almacenado de datos en disco o en línea.
Unir		Se utiliza para acoplar segmentos del diagrama o para recibir la línea de flujo.
Multi-documento		Se utiliza para representar la salida, despliegue o impresión de varios documentos.
Entrada manual		Representa la intervención de usuario para dar una entrada a datos requeridos (No se confunda con el símbolo de Entrada / Salida).
Operación manual		Representa la intervención del usuario para realizar un proceso manual.
Almacenamiento interno		Se utiliza para representar el almacenamiento en memoria de algún proceso o valor.
Cinta magnética		Representa datos grabados en una cinta magnética.

Límite de ciclo		
Preparación		Expresa proceso de llamada a un proceso subalterno.
Tarjeta		Representa la entrada de datos o lectura de datos de una tarjeta perforada o recientemente de memorias de almacenamiento.
Retraso		Representa atraso para poder iniciar el siguiente proceso o tarea.
Conector (dentro de página)		Sirve para enlazar dos partes cualesquiera de un diagrama a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página del diagrama
Conector (fuera de página)		Sirve para enlazar dos partes cualesquiera de un diagrama a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en distinta página del diagrama
Línea de flujo		Indica el sentido de la ejecución de las operaciones

Figura. 1.6. "Simbología y significado". Fuente: Díaz Guzmán, Yazmín (2013). Información recuperada de: http://jazzfantastic.blogspot.mx/ 7/09/2018.

- Diagrama de flujo de datos. Es una herramienta gráfica empleada para describir y analizar el movimiento de datos, procesos a través de un sistema y lugares para almacenar los datos. Según el autor Senn, (1992: 190), Los diagramas de flujo de datos son de dos tipos, físicos y lógicos:

  I.- Los diagramas físicos de flujo de datos. Muestran qué tareas y cómo se llevan a cabo, dependiendo de la implantación. Algunas características físicas son:

  - Nombres de personas
  - Nombres o números de formatos y documentos.
  - Nombres de departamentos.
  - Archivos maestros y de transacciones
  - Equipo y dispositivos utilizados.
  - Ubicaciones.

- Nombres de procedimientos.

Algunas razones para usarlos son:

1. Los analistas de sistemas comienzan por identificar el movimiento de personas, documentos e información entre departamentos. Y es mucho más fácil describir la interacción entre los componentes físicos que comprender las políticas empleadas para administrar la aplicación.
2. Los diagramas físicos de flujo de datos son de utilidad para comunicarse con los usuarios y pueden señalar con rapidez cuando un paso es correcto o equivocado.
3. Los diagramas físicos de flujo de datos proporcionan un camino para validar o verificar el punto de vista del usuario.
4. No se usan donde la tarea la realiza una sola persona y no existe un flujo de datos. En cambio, se usa donde los procesos tienen varias tareas y el flujo de datos abarca diferentes personas o localidades.

II.- Diagramas lógicos de flujo de datos.

Se centran en el flujo de información entre los procesos independientemente de la implantación, sin considerar los dispositivos específicos, la localización de almacenes de datos, personas o áreas. Aquí no se indican las características físicas. Para su elaboración se usan básicamente cuatro elementos:

- Flujo de datos: Trayecto de los datos en el sistema en forma de documentos, formatos, transferencias o algún otro medio de comunicación.

- Procesos: Tareas o procedimientos que transforman, usan o producen datos. Un proceso puede desglosarse usando diagramas de flujo cada vez más detallados, de tal forma que el analista tenga suficientes detalles para comprender en su totalidad el sistema o la parte que se esté analizando.

- Fuentes de datos: Se señalan los datos necesarios para un proceso generados por personas, documentos, organizaciones u otras entidades que interactúan con el sistema que pueden estar dentro o fuera de su frontera.

- Almacenamiento de datos: Lugar donde se guardan los datos a los que hacen referencia los procesos en el sistema. El almacenamiento puede ser manual (Ej. Documentos o formatos) o de forma automática en almacenamiento de cómputo primario (ej. memoria caché o RAM) o secundario (ej. disco duro).

Reglas generales para el dibujo de diagramas lógicos de flujo de datos.[9]

1. Cualquier flujo de datos que abandone un proceso debe de estar basado en los datos que entran al proceso.
2. Todos los flujos de datos que reciben un nombre, el nombre refleja los datos que fluyen entre los procesos, almacenes de datos fuentes o destinos.
3. Sólo deben entrar al proceso los datos necesarios para llevarlo a cabo.

[9] Senn, James (1992). *Análisis y diseño de sistemas de información* [2ª ed.]. México: McGraw-Hill, pp. 201-202.

4. Un proceso no debe saber nada de ningún otro en el sistema, debe ser independiente; la única *dependencia* que debe existir es aquella que debe estar basada en sus propios datos de entrada y de salida.
5. Los procesos siempre están en continua ejecución, no se inician ni tampoco se detienen. Los analistas siempre deben suponer que un proceso siempre está listo para funcionar o realizar el trabajo.
6. La salida de los procesos puede tomar una de las siguientes formas:
    a. Flujo de datos con información añadida por el proceso.
    b. Una respuesta o cambio en la forma de los datos.
    c. Un cambio de condición.
    d. Un cambio de contenido.
    e. Cambios en la organización.

Por último, se presentan dos ejemplos de un mismo proceso para observar la diferencia entre un diagrama físico y uno lógico.

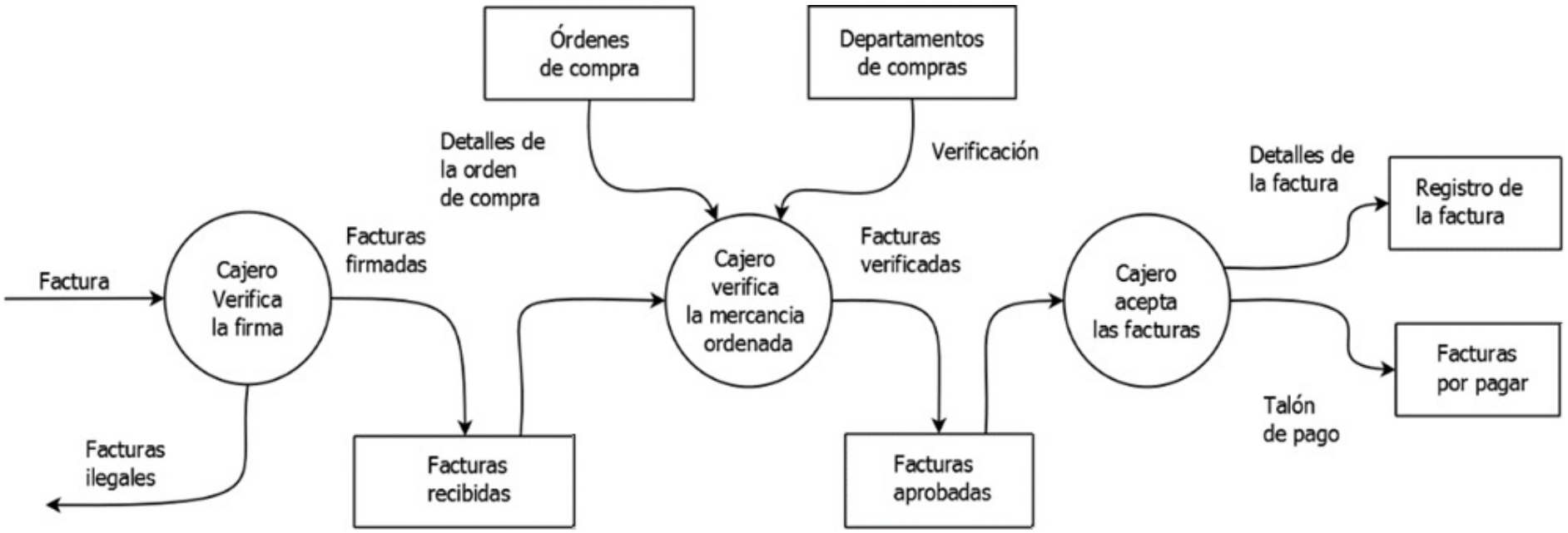


Figura. 1.7. Ejemplo de diagrama físico para el proceso de autorización de facturas, elaboración propia basada en Senn, 1992: 198.

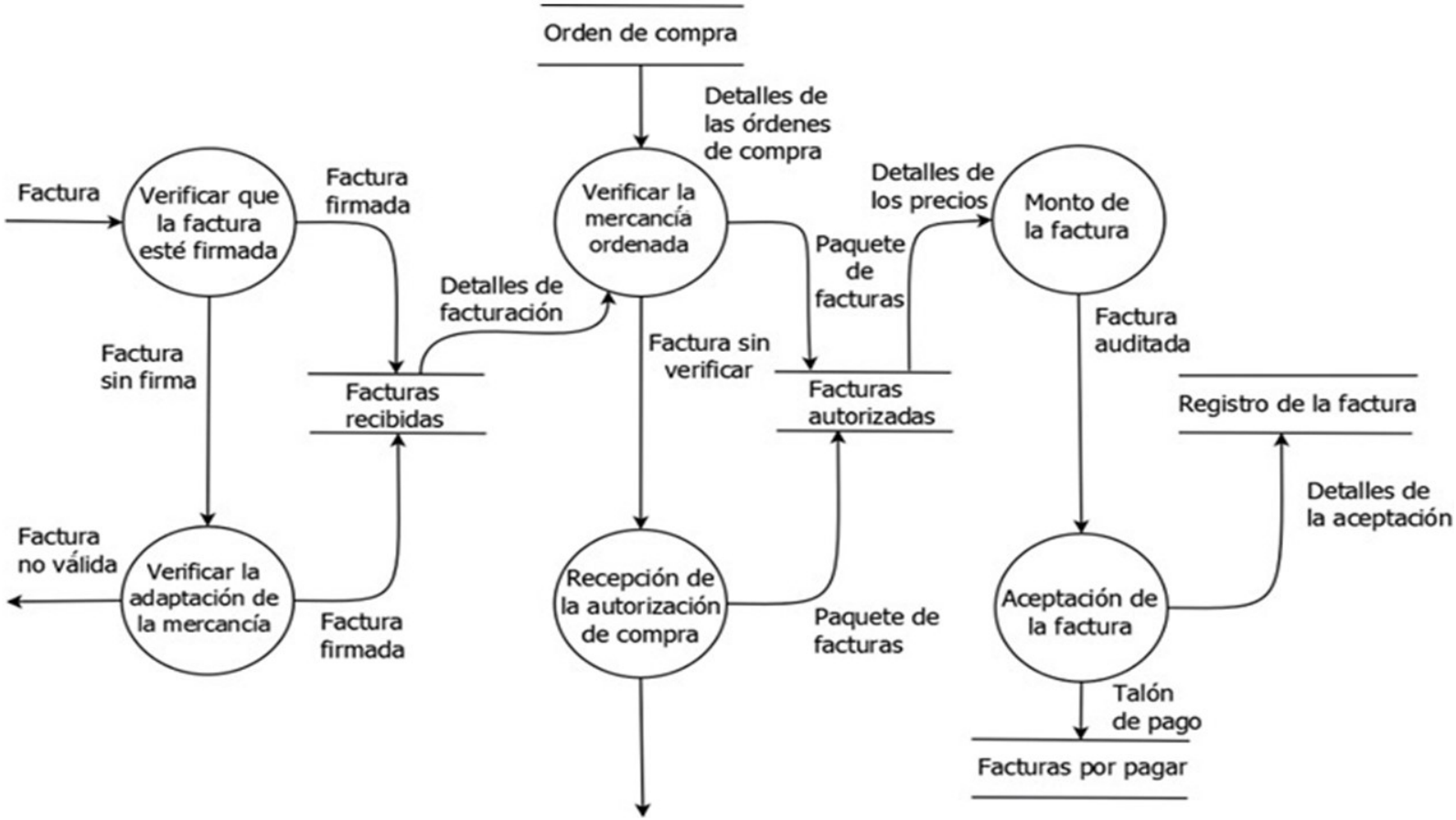


Figura. 1.8. Ejemplo de diagrama lógico para el proceso de autorización de facturas, elaboración propia basada en: Senn, 1992: 205.

Ventajas de los diagramas de flujo:

- Cualquier persona que forma parte del sistema puede comprender el funcionamiento de algún proceso.
- Mejora la comunicación entre usuarios y analistas, lo primeros haciendo sugerencias para describir una actividad con más exactitud y los segundos entendiendo mejor los procesos manuales.
- Rapidez para detectar problemas y hacer correcciones antes de realizar el diseño.

- Diccionario de datos. Contiene las descripciones de los datos y procesos usados en el sistema, como puede ser el tamaño del campo, dónde se utiliza o qué otro nombre recibe. Esto permite:
  - A cualquier miembro encargado del proyecto tener la misma información.
  - Mejor manejo de detalles en sistemas grandes.
  - Unificar significados para todos los elementos del sistema.
  - Facilita cambios en el sistema.
  - Localizar errores y omisiones.

Según Senn, la descripción de todos los elementos del diagrama de flujo incluye a cada almacén de datos y elemento dato:

Almacén de datos	Facturas autorizadas
Descripción	Solicitudes por parte del vendedor para procesamiento. Detalla la mercancía recibida, costo de cada artículo y contiene la firma del empleado que recibe la mercancía.

Flujo de datos recibidos	1.1 factura con firma 1.2 factura con firma –cuando la firma es necesaria	
Flujo de datos proporcionados	Detalles de los artículos asentados en el lote de facturas.	
Descripción de datos	Detalles del vendedor. Número de factura. Fecha de expedición de la factura. Referencia de la orden de compra.	Detalles de los artículos. Cantidad adeudada.
Volumen	200 al día, crecimiento anual 10%: la mayor carga de trabajo se presenta al inicio de cada mes.	
Acceso	Retrasado hasta completar el lote; una vez completado, se tiene acceso a cualquiera de ellas; el procedimiento dentro del lote es secuencial.	
Elemento dato	Número de orden de compra.	
Descripción	Identificación y autorización de cada orden otorgada a un proveedor externo.	
Tipo	AN N.	
Longitud	7.	
Alias	OC, requisición.	
Rango de valores	Valor representativo	Aumentando desde 10,000

Lista de valores específicos	Prefijos válidos	Significado
	AC	Contabilidad
	AD	Publicidad
	EX	Oficina ejecutiva
	PE	Personal
	PU	Compras
	RD	Investigación y desarrollo
	SA	Ventas
Otros detalles de edición	Número de la orden de compra que incluye un número de cinco dígitos y el prefijo del departamento.	

Tabla. 1.8. Descripción de los elementos del diagrama de flujo. Fuente: Senn, 1992: 228.

A continuación, se presentan otros ejemplos de diccionarios de datos, el primero es para un sistema que usa bases de datos y el segundo indica el significado en un sistema de los símbolos de diagrama de flujo.

Target					Source				Transformation
Table Name	Column Name	Data Type	Table Type	SCD Type	Database Name	Table Name	Column Name	Data Type	
EMPLOYEE_DIM	EMPLOYEE_KEY	NUMBER	Dimension	1				NUMBER	Surrogate key.
EMPLOYEE_DIM	EMPLOYEE_ID	NUMBER	Dimension	1	HR_SYS	EMPLOYEES	EMPLOYEE_ID	NUMBER	Natural Key for employee in HR system
EMPLOYEE_DIM	BIRTH_COUNTRY_NAME	VARCHAR2(75)	Dimension	1	HR_SYS	COUNTRIES	NAME	VARCHAR2(75)	select c.name from employees e, states s, countries c where e.state_id = s.state_id and s.country_id = c.country
EMPLOYEE_DIM	BIRTH_STATE	VARCHAR2(75)	Dimension	1	HR_SYS	STATES	DESCRIPTION	VARCHAR2(255)	select s.description from employees e, states s where e.state_id = s.state_id
EMPLOYEE_DIM	DISPLAY_NAME	VARCHAR2(75)	Dimension	1	HR_SYS	EMPLOYEES	FIRST_NAME	VARCHAR2(75)	select initcap(salutation) || ' '|initcap(first_name) || ' '| initcap(last_name) from employee
EMPLOYEE_DIM	BIRTH_DATE	DATE	Dimension	1	HR_SYS	EMPLOYEES	DOB	DATE	trunc(DOB)
EMPLOYEE_DIM	SALUTATION	VARCHAR2(12)	Dimension	1	HR_SYS	EMPLOYEES	SALUTATION	VARCHAR2(12)	initcap(salutation)
EMPLOYEE_DIM	FIRST_NAME	VARCHAR2(30)	Dimension	1	HR_SYS	EMPLOYEES	FIRST_NAME	VARCHAR2(30)	initcap(first_name)
EMPLOYEE_DIM	LAST_NAME	VARCHAR2(30)	Dimension	1	HR_SYS	EMPLOYEES	LAST_NAME	VARCHAR2(30)	initcap(last_name)
EMPLOYEE_DIM	MARITAL_STATUS	VARCHAR2(12)	Dimension	2	HR_SYS	MARITAL_STATUS	DESCRIPTION	VARCHAR2(12)	select nvl(m.name,'Unknown') from employee e marital_status m where e.marital_status_id = m.marital_status_id
EMPLOYEE_DIM	DIVERSITY_CATEGORY	VARCHAR2(30)	Dimension	1	HR_SYS	EMPLOYEES	EEO_CLASS	VARCHAR2(30)	decode(eeo_class,null, 'Not Stated', decode(eeo_class,'N', 'Not Stated',eeo_class))
EMPLOYEE_DIM	GENDER	VARCHAR2(12)	Dimension	1	HR_SYS	EMPLOYEES	SEX	VARCHAR2(12)	nvl(sex, 'Unknown')
EMPLOYEE_DIM	EMPLOYEE_STATUS	VARCHAR2(24)	Dimension	1	HR_SYS	EMPLOYEES	STATUS	VARCHAR2(24)	select es.name from employee e employee_status es where e.employee_status_id = m.employee_status_id
EMPLOYEE_DIM	POSITION_CODE	VARCHAR2(12)	Dimension	2	HR_SYS	POSITIONS	POSITION_CODE	VARCHAR2(12)	select p.code from employees e, positions p where p.position_id = e.position_id
EMPLOYEE_DIM	POSITION_CATEGORY	VARCHAR2(30)	Dimension	2	HR_SYS	POSITIONS	POSITION_CATEGORY	VARCHAR2(30)	select p.category from employees e , positions p where p.position_id = e.position_id
EMPLOYEE_DIM	HIRE_DATE	DATE	Dimension	1	HR_SYS	EMPLOYEES	DATE_HIRED	DATE	trunc(date_hired)
EMPLOYEE_DIM	DEPARTMENT_CODE	VARCHAR2(12)	Dimension	2	HR_SYS	DEPARTMENTS	CODE	VARCHAR2(12)	select d.code from employee p, employee_department pd, departments d where p.employee_id= pd.employee_id and pd.department_id = d.department_id
EMPLOYEE_DIM	DEPARTMENT_NAME	VARCHAR2(75)	Dimension	2	HR_SYS	DEPARTMENTS	DESCRIPTION	VARCHAR2(75)	select d.description from employee p, employee_department pd, departments d where p.employee_id= pd.employee_id and pd.department_id = d.department_id

Tabla 1.9. Ejemplo de sistema que usa bases de datos.
Fuente: Espinosa Roberto (2010) "*Identificación orígenes de datos. Utilizando Data Profiling*" Recuperado el 12 de marzo de 2019 de: http://churriwifi.wordpress.com/2010/05/03/16-1-identificacion-origenes-de-datos-utilizando-data-profiling/

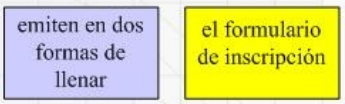
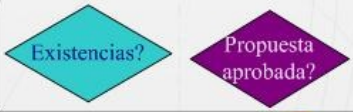
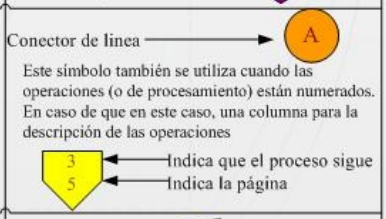
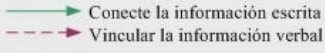
Componente	Descripción	Símbolo
Terminal	Terminal se utiliza para representar al comienzo o al final del proceso, sus zonas de frontera, o para referirse a otro proceso que no es el objeto de estudio	
Operación	representa ninguna medida para crear, procesar, analizar o dar una transacción (o transformación). En el símbolo, que describe el objetivo de la demanda. Este símbolo se utiliza también como una descripción de la operación (o processanmento) se hace dentro del símbolo, con, en este caso, la columna de descripción de las transacciones.	
Ejecutor	representa la zona (o de la persona / oficina) que realiza la acción	
Documento	representa cualquier documento creado o transformado en el flujo del proceso. En la representación por debajo de, por ejemplo, muestra que la nota fiscal deberá publicarse en dos maneras.	
Información verbal	representa los contactos intercambios verbales entre los participantes de la proceso.	
Archivo	representa el cierre de la documentación	
Decisión	Indica un punto en el proceso que se presenta acciones limitaciones (si), donde hay caminos alternativos, si se producen ciertos acontecimientos (sí o no)	
Conector	indica que la secuencia sigue la corriente. En la representación más adelante, indica que la continuación del proceso se produce en otra página.	
Material	representa el material que circula en el proceso	
Dirección de circulación	flechas se utilizan para interconectar los distintos símbolos, lo que indica el flujo del proceso	
Transporte	representa un elemento de referencia a otro	

Tabla. 1.10. Fuente: s/a, s/f, *Ejemplo diagrama de flujo proceso PDF,* Recuperado el 13 de marzo de 2019 de: https://diagram-ideas.com/ejemplo-diagrama-de-flujo-proceso-pdf/

1.5. Estrategia de desarrollo por prototipos de aplicaciones

La estrategia de desarrollo por prototipos de aplicaciones, prueba ideas y supuestos tanto de analista como de usuarios de un modo interactivo en casi todas las actividades del proceso de *software*. Es decir, haciendo las correcciones pertinentes hasta satisfacer los requerimientos del usuario contempladas en la identificación de requerimientos por medio de una aplicación terminada de manera simple, rápida y económica llamada prototipo. Otro autor como, por ejemplo, Roger Pressman, llama a esta estrategia *modelo de construcción de prototipos* y aconseja: "Cuando el cliente tiene una necesidad legítima, pero está desorientado sobre los detalles, el primer paso es desarrollar un prototipo" (Pressman, 2002: 21) y Seen, por su parte, señala:

> Este método hace que el usuario participe de manera más directa en la experiencia de análisis y diseño (…), la construcción de prototipos es muy eficaz bajo las circunstancias correctas (…), útil si se emplea en el momento adecuado y la forma apropiada (1992: 43).

Sommerville lo incluye como etapa de construcción de prototipos dentro de un modelo en espiral.[10]

Senn define al prototipo como un modelo de prueba, dice que el diseño va evolucionando con el uso y con la participación y opiniones de las personas que usarán el sistema al final, esta fase de prueba es lo que va puliendo su correcto funcionamiento (Senn, 1992: 44).

[10] Si deseas profundizar sobre este modelo en espiral, revisa el documento "Modelo Espiral de un proyecto de desarrollo de software", recuperado el 26 de marzo de 2019 de: http://ojovisual.net/galofarino/modeloespiral.pdf

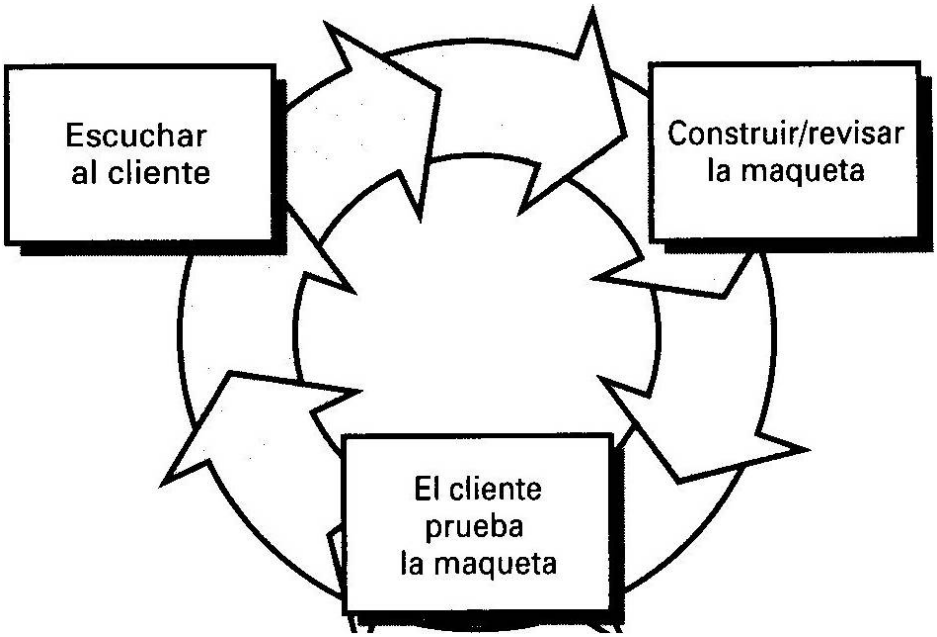


Figura. 1.9. Modelo de prueba. Fuente: (Pressman: 2002: 21)

Otras razones, para usar esta estrategia de desarrollo, son:

- Su eficacia para definir más claramente los requerimientos de los usuarios y verificar la factibilidad del proyecto.
- Minimiza el riesgo de perder tiempo y dinero por errores de diseño o desarrollo incorrecto usando otro tipo de estrategias de desarrollo.

Pressman (2002: 192), menciona que puede haber prototipo desechable o de enfoque cerrado que como su nombre lo indica, son elaborados únicamente como prueba y después deben desecharse, con la intención de comprender mejor los requerimientos del cliente para después considerar otra solución mejorada. Y prototipos evolutivos o de enfoque abiertos que van modificándose poco a poco, para ello se deben considerar factores como:

- Conocimiento sobre las características del sistema propuesto.
- Definición y evaluación de requerimientos.
- Costo.

- Área de aplicación.
- Nivel de riesgo.
- Complejidad.
- Tecnología disponible.
- Experiencia y disponibilidad de los usuarios.

Stephen J. Andriole, citado por Pressman, propone seis preguntas para determinar el tipo de prototipo:

Pregunta	Prototipo desechable	Prototipo evolutivo	Trabajo preliminar adicional requerido
¿Se entiende el dominio de la aplicación?	Sí	Sí	No
¿Se puede modelar el problema?	Sí	Sí	No
¿Está el cliente suficientemente seguro de los requisitos básicos del sistema?	Sí/No	Sí/No	No
¿Están establecidos los requisitos y son estables?	No	Sí	Sí
¿Hay requisitos ambiguos?	Sí	No	Sí
¿Hay contradicciones en los requisitos?	Sí	No	Sí

Tabla. 1.11. Tipo de prototipo. Fuente: Pressman, 2002: 193.

ETAPAS DE LA ESTRATEGIA DE DESARROLLO POR PROTOTIPO DE APLICACIONES

Autores como Pressman, Senn y Pfleeger dan nombres diferentes a la estrategia o método[11], pero coinciden en la mayoría de los pasos para su desarrollo. A continuación, se presentan dichos pasos y se indican los que no tienen coincidencia entre los autores mencionados:

No.	Autor	Pasos para su desarrollo
1	Pressman, 2002: 191	Recolección de requisitos
2	Pressman, 2002: 191	Definición de objetivos globales
3	Senn, 1992: 45	Identificación de requisitos que el usuario conoce
4	Senn, 1992: 45	Desarrollo de un prototipo que funcione o diseño rápido (Pressman, 2002: 21)
5	Senn, 1992: 45	Revisar el prototipo con base en la experiencia que tuvo el usuario
6	Senn, 1992: 45	Repetir los pasos anteriores hasta obtener un sistema satisfactorio
7		Decidir implantar o abandonar el prototipo.

Tabla. 1.12. Etapas de la estrategia de desarrollo, de acuerdo con algunos autores.

Shari Pfleeger, autora del libro "*Software engineering*", propone que hay 5 tipos de prototipos para el modelado de procesos, dependiendo de la fase de desarrollo de *software* en que se aplique (Pfleeger, 2010: 66):

[11] De acuerdo con los planteamientos de Pressman, 2002: 19, la estrategia de desarrollo se conforma por medio de un equipo de ingenieros que definen los procesos, los métodos y capas que implican la creación de un software, por lo que se considera más amplia que el método, en esto radica la diferencia.

- Prototipo de requisitos. Es como el prototipo desechable propuesto por Pressman, el cual permite conocer mejor los requerimientos del cliente.
- Prototipo de análisis. Similar al diseño rápido que genera una solución con las características básicas de acuerdo a los requerimientos.
- Prototipo de diseño. Es una solución de prueba para detectar posibles omisiones o inconsistencias de diseño.
- Prototipo vertical. Muestra una solución parcial para ser evaluada por el usuario.
- Prototipo de factibilidad. Es una propuesta que ayuda a determinar si el proyecto es viable en función del tiempo, costo y factores técnicos.

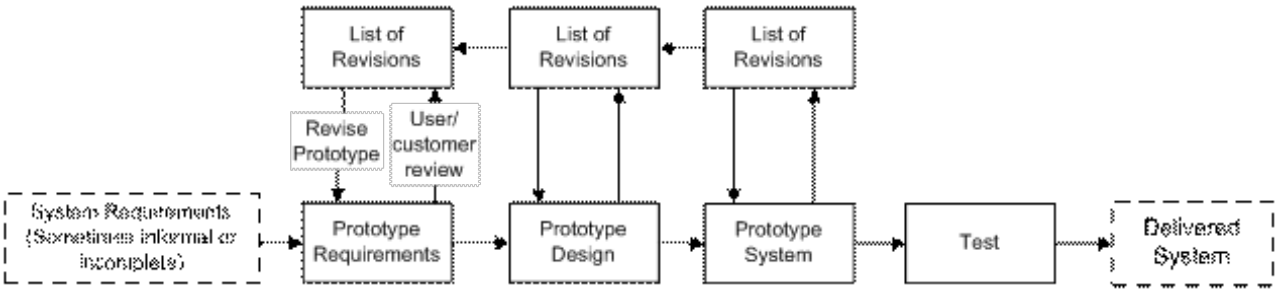


Figura 1.10. Fuente: Pfleeger (2010). *Software Engineering, Theory and Practice* [4a. ed.]. New Jersey: Pearson p. 68.

HERRAMIENTAS PARA EL DESARROLLO DE PROTOTIPOS

Para la creación de prototipos (o *prototyping* en inglés), hay varias herramientas agrupadas en las siguientes clasificaciones:

Clasificación	Descripción	Ejemplos
Lenguajes de cuarta generación	Son lenguajes de consulta e informes de bases de datos,	FOCUS, SQL, INTELLECT

	creados de aplicaciones de escritorio, web o para dispositivos móviles.	
Lenguajes no orientados a procedimientos	También llamados lenguajes no procedurales, es decir, que un solo comando puede ejecutar varias tareas, por ejemplo, SELECT en el lenguaje SQL.	Oracle, PostgreSQL, MySQL
Generadores de reportes	Recuperan y presentan información en determinados formatos usando pocas instrucciones.	Oracle reports, Cristal reports, Latex
Generadores de aplicaciones	Facilitan la creación de *software* de alto nivel que realicen cálculos, rutinas lógicas o manejo de datos.	Java, Python, PHP, PL/SQL
Generadores de pantalla	Actualmente, se les conoce como *frameworks* o aplicaciones para crear interfaces para la entrada, etiquetado y presentación de datos. Están enfocadas más en el diseño que en la funcionalidad.	PowerBuilder, Fireworks, NinjaMock, Photoshop
Componentes de *software* reutilizable.	Estos programas agilizan la creación de aplicaciones porque se pueden ensamblar por componente de *software* reutilizable.	Java, Python

Sistemas de diccionario de datos	Almacenan descripciones de los datos y procesos utilizados en el proyecto.	JustProto, ProtoShare
Especificaciones formales y entornos para prototipos.	Herramientas automáticas que traducen del lenguaje natural a código ejecutable y con ello el desarrollador refina los requisitos hasta el producto final.	Prototype Composer Arena simulation

Tabla. 1.13. Clasificación de herramientas para el desarrollo de prototipos. Elaboración propia.

1.6. Herramientas asistidas por computadora para el desarrollo de sistemas (CASE)

Los sistemas CASE, son herramientas de ingeniería de *software* que ayudan al analista de sistemas a organizar la información y hacer eficiente el diseño y desarrollo de sistemas.

Por medio de las herramientas de ingeniería, es posible obtener *software* de costo bajo y mayor calidad a manera de sistema automatizado para el desarrollo de sistemas, cuya utilidad es organizar la información y tornar eficientes los procesos o procedimientos que realiza una persona, institución, empresa, negocio, etc.

CASE viene de *Computer Aided Assisted Software* por sus siglas en inglés, es decir Ingeniería del *software* asistida por computadora (Pressman, 2002: 559). Para Pressman son:

> Herramientas que ayudan a los gestores y practicantes de la ingeniería del software en todas las actividades asociadas a los procesos del software. Automatizan las actividades de gestión de proyectos, gestionan todos los proyectos de los trabajos elaborados a través del proceso, y ayudan a los ingenieros en el trabajo de análisis, diseño y codificación. (2002: 559).

Con las herramientas CASE se pueden realizar tareas como gestión y almacenamiento de información del proyecto a través de repositorios,

diccionarios y reportes; diseñar proyectos, crear diagramas de diversos tipos, generar código e interfaces gráficas y detección de errores.

CASE también se puede utilizar en cualquier etapa del proceso de ingeniería de *software,* independientemente de si la metodología es por ciclo de vida, análisis estructurado o desarrollo de prototipos, pero por lo regular los instrumentos para diagramas se usan en las primeras etapas y las de generación de código e interfaces en las finales.

Piattini menciona que: “La tecnología CASE supone información de la informática, es decir, la automatización del desarrollo del software, contribuyendo así a elevar la productividad y la calidad en el desarrollo de sistemas de información” (Piattini, 2004: 656).

Los objetivos de las CASE son:

- Automatizar e integrar las tareas de las distintas etapas del ciclo de vida, junto con la gestión de proyectos de *software*.
- Mejorar la calidad mediante la automatización de la comprobación de errores.
- Automatizar la generación de documentación.
- Facilitar que se pueda compartir la información entre varios proyectos y la reutilización del *software*.
- Aportar un entorno de desarrollo interactivo.
- Acercar el desarrollo al usuario facilitando la creación de prototipos.
- Simplificar la labor de mantenimiento.[12]

Para fines didácticos, se muestra a continuación la siguiente línea del tiempo, que ilustra las diversas etapas que han tenido las herramientas CASE:

[12] Antonio Joxe (s/f) Recuperado el 20 de marzo de 2019 de: https://es.scribd.com/document/315842800/Uso-de-Herramientas-Case-Final

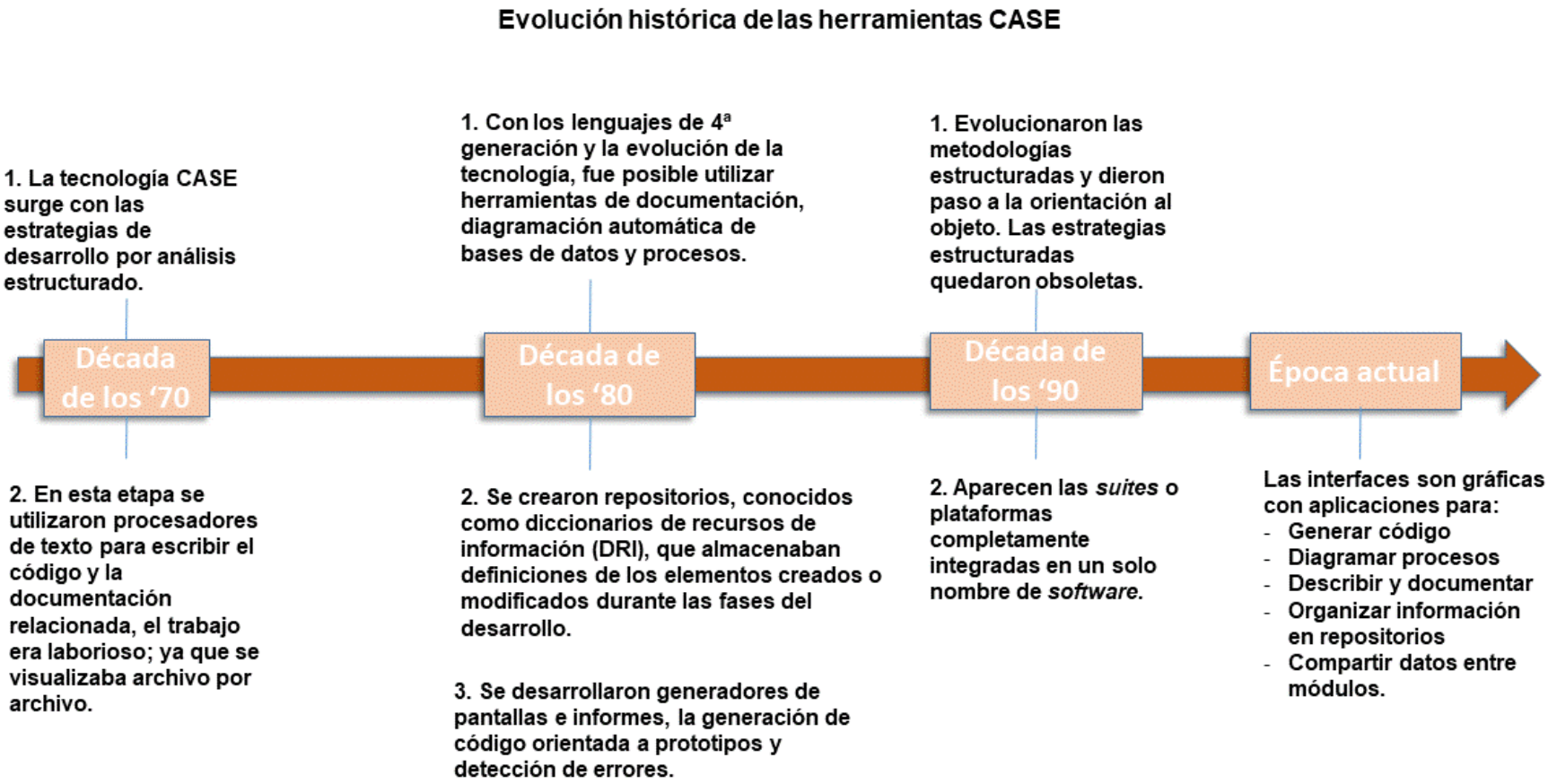


Figura. 1.11. Evolución histórica de las herramientas CASE. Elaboración propia.

Beneficios de las herramientas CASE

Si bien un sistema de información puede crearse con una aplicación para generar código, y otra para diagramar, este conjunto de aplicaciones no se considera un CASE. En su más reciente edición del libro Ingeniería de *software*, Presman menciona que: “Cuando se integran las herramientas de modo que la información creada por una pueda ser utilizada por otra, queda establecido un sistema llamado ingeniería de *software* asistido por computadora que apoya en desarrollo de *software*”. (Pressman, 2010: 12).

Con esta base los desarrolladores de *software* pueden crear una producción automatizada e incluso reutilizada en sistemas futuros.

Tanto Pressman como Senn coinciden en los beneficios de las herramientas CASE:

- Mejoran la productividad
  Al reducir el esfuerzo para producir un producto con herramientas de desarrollo y diseño, programación, gestión de bases de datos y reingeniería.

  Seen menciona que, al utilizar las herramientas correctas, ayudan en cierta forma a obtener un nivel de productividad que hace posible la realización de una tarea que no sería posible realizarla de otra manera. Seen hace una analogía del martillo y serrucho indispensables para que un carpintero haga su trabajo con el menor tiempo y esfuerzo (Senn, 1992: 285).

  Un ejemplo informático es el uso de *software* para crear diagramas de flujo, esto reduce el tiempo del a veces tedioso trabajo de

diagramar un proceso o sistema. Otro ejemplo son los generadores de código que reducen el tiempo necesario para preparar un prototipo por medio de técnicas de reutilización y reingeniería.

- Mejoran la eficiencia

  Al usar herramientas de análisis, simulación y construcción de prototipos.

  > (…) no se utiliza un martillo para atornillar tornillos (…). Identificar los requerimientos del usuario, trasladarlos en una forma comprensible y comunicarlos a todas las partes interesadas, puede ser un proceso de desarrollo más eficiente a la hora de hacer cambios o corregir errores (Senn, 1992: 286).

- Mejoran la calidad

  A través de herramientas de integración, pruebas y control de calidad que realizan auditorías y métricas del código fuente con el fin de ver si es posible que se ajuste o no a estándares del lenguaje requerido. (Pressman, 2002: 563-565).

  Retomando el ejemplo que presenta Seen, es necesario que, para poder realizar paredes con sus ángulos correctos, las personas que realicen el trabajo lo hagan con las herramientas correctas para obtener dichos resultados (Senn, 1992: 286).

Las herramientas CASE son importantes para generar sistemas de calidad porque utiliza métodos manuales dificulta el desarrollo *software* altamente complejo sin mencionar el tiempo y costo. Con el diseño automático se puede analizar mejor la complejidad de un sistema hasta el mínimo detalle, encontrar errores y hacer modificaciones de manera ágil; ya sea por corrección o mantenimiento, lo cual ahorra tiempo y dinero. Pero no

sólo basta con tener gráficas para crear o reutilizar código, también se requiere de un conjunto completo de información de una o varias partes del sistema de tal forma que tanto sus interrelaciones o cualquier actualización sean visibles para todos los involucrados o afectados en determinada modificación, evitando datos erróneos o redundantes.

Según González (1995:196) las capacidades de las herramientas CASE son:

- Capacidades gráficas para representar los modelos y diagramas.
- Comprobación de errores integrada dentro de las herramientas.
- Depósito de información o enciclopedia para almacenar y gestionar toda la información del sistema de forma global.
- Conjunto altamente integrado de herramientas para asistir a todas y cada una de las fases del ciclo de vida, dentro de un interfaz común de cara al usuario.
- Soporte de una metodología.
- Generación automática de código desde las especificaciones del diseño y soporte en la obtención de prototipos.
- Soporte de herramientas para el mantenimiento de sistemas antiguos: reestructuración (*restructuring*), ingeniería inversa (*reverse engineering*), reingeniería (*reengineering*).

Componentes de las herramientas CASE

Para lograr las capacidades antes descritas, un sistema elaborado con las herramientas CASE cuenta con un conjunto de instrumentos. Basado en SENN (1992: 293) estos instrumentos son:

- De diagramación

Con ellas se pueden elaborar diagramas de flujo de datos, modelos entidad-relación para bases de datos, estructuras de datos y técnicas matriciales que facilitan el análisis, diseño y desarrollo de sistemas complejos con un alto nivel de detalle. También se pueden ver varios

diagramas al mismo tiempo, añadir información adicional para aclarar algún punto concreto del diseño y, a la vez, disponible en cualquier fase del desarrollo o cuando sea necesario realizar cambios.

- Depósito de información.

El depósito de información también es conocido como repositorio o diccionario de recursos de información (DRI) o diccionario de datos (el nombre varía), es uno de los componentes CASE más importantes porque contiene información detallada sobre los componentes del sistema, tales como datos, gráficos, informes, modelos, flujo de datos, informes y procesos con sus respectivos niveles de autorización, variación de procesos, procedimientos para verificar inconsistencias en las descripciones y la gestión de cambios o versiones visibles para todos los involucrados. Esta última debe ser capaz de facilitar la localización de elementos, analizar las consecuencias de un cambio y los módulos que afecta.

Además de su importancia como fuente de información para todos los involucrados en todos los niveles y procesos; es una pieza imprescindible para la reutilización de código.

- Generadores de interfaces (instrumentos de prototipado)

Actualmente, una aplicación no se puede concebir sin su interfaz siendo ésta el instrumento de comunicación entre el usuario y el sistema a través de menús, pantallas de presentación y formatos de informes. Estas herramientas permiten visualizar cómo se verá el *software* en pantalla para el usuario final.

Además, las herramientas CASE se usan para crear prototipos con los cuales se puede tener una idea factible del proyecto con la opción de saber si cumple con los objetivos, determinar los requisitos técnicos para el buen funcionamiento del sistema y hacer los cambios necesarios antes de la versión final. Por ello son conocidas también como herramientas de prototipado, aunque, hay autores como Pressman que tratan por separado las herramientas de diseño y las de prototipos, ambas son mutuamente incluyentes, los cambios realizados en un prototipo obligan a cambiar la interfaz y por eso deberían tratarse como uno solo.

A estos prototipos se les conocen como productos beta o en fase beta y las compañías desarrolladoras las ofrecen al público para recibir una retroalimentación sobre errores o mejoras al *software*.

- Generadores de código.

Estos convierten las especificaciones del sistema en código ejecutable con las siguientes características:

  - Crearse casi de manera automática en su totalidad, aunque siempre es necesario hacer algunos ajustes de manera manual.
  - Actualmente se busca que el código tenga portabilidad, es decir, que pueda ser ejecutado sin importar el sistema operativo o *hardware*.
  - Generado con lenguaje estándar, privado (creado por una empresa o institución no conocida por el público en general) o especializado (para aplicaciones militares, médicas, científicas o multimedia).
  - Al integrarse con los otros componentes, los generadores de código junto con los repositorios se vuelven básicos en la

reutilización de código, una práctica común en los desarrolladores de sistema.

- Herramientas de gestión.

Con ellas se puede llevar un seguimiento de los tiempos de desarrollo de un proyecto contrastándolo con la planeación establecida en los repositorios, también ayuda con la asignación de recursos y tareas específicas al personal.

Por su parte, Pressman (2002: 561) hace un desglose más amplio de los componentes CASE:

- Herramientas de ingeniería de procesos de negocio

Estas herramientas modelan -por medio de objetos de datos la información del negocio, sus relaciones y la forma en que fluyen estos objetos de datos en las áreas del negocio.

- Modelado de procesos y herramientas de gestión

Se utilizan para representar los elementos clave del proceso, de manera que sea posible entenderlo mejor y proporcionar vínculos con descripciones de procesos, que ayuden al personal implicado a comprender las tareas que se requieren para llevar a cabo ese proceso.

- Herramientas de planificación de proyectos

Crea una red de tareas centradas en la estimación de costos, duración, y número de personas involucradas en el proyecto.

Tipos de herramientas	Descripción
De análisis de riesgos	Con las herramientas de análisis de riesgos, se busca identificar posibles riesgos y un plan para mitigar, monitorizar y gestionar esos riesgos.
De gestión de proyectos	Estas herramientas, recogen métricas que indican la calidad del producto del *software*.
De seguimiento de requisitos	El seguimiento de proyectos, genera y monitoriza información por medio de bases de datos y documentación para verificar que el proyecto se apega a los requisitos que debe cumplir el sistema.
De métrica y de gestión	Éstas herramientas recogen métricas centradas en características de procesos y productos.
De documentación	Las herramientas de documentación generan y gestionan el proceso de documentación.
De software de sistema	Estas herramientas se refieren a la tecnología de estaciones de trabajo y de comunicación.
De control de calidad	Son herramientas de métricas que hacen una auditoría del código fuente para determinar si se ajusta o no a ciertos estándares del lenguaje.
De gestión de bases de datos	Son también conocidos como sistemas gestores de bases de datos relacionales como puede ser Oracle, PostgreSQL o MySQL.
De gestión de configuración de *software*	Estas herramientas se encuentran en el núcleo de los entornos CASE y ofrecen asistencia para identificación, control de versiones, control de cambios, auditoría y contabilidad.
De análisis y diseño	Permiten a los desarrolladores eliminar errores y crear modelos que representan los datos, función, comportamiento, arquitectura e interfaz.
PRO/SIM	Se refieren a herramientas para la creación de prototipos y simulación que proporcionan al ingeniero de *software* la capacidad de predecir el comportamiento de un sistema en tiempo real antes de construirlo. Esto también puede generar nuevas ideas acerca de su funcionamiento, comportamiento y respuesta.
De desarrollo y diseño de interfaz	Son elementos que integran una interfaz como son los menús, botones, iconos, ventanas, controladores de dispositivos, mecanismos de desplazamiento, por citar los más básicos.

De programación	Estas herramientas abarcan compiladores, editores, depuradores, entornos de programación orientada a objetos y generadores de aplicaciones. Dependiendo del lenguaje de programación en algunos casos se usan traductores, lenguajes de consulta de bases de datos y máquinas virtuales.
De desarrollo de Webs	Existen una gran variedad de herramientas para crear aplicaciones basadas en web; todas ellas ayudan en la generación de texto, gráficos, multimedia y conexión a bases de datos.
De integración y pruebas	Se enfocan a prestar asistencia en la planificación, desarrollo y control de pruebas. Adquieren datos para realizar medidas estáticas (sin ejecutar casos de prueba) y medidas dinámicas (utilizan el código fuente durante la ejecución).
De análisis dinámico. (Estáticas de prueba)	En estas se utilizan tres tipos de herramientas estáticas de prueba: 1 Las que están basadas en código. Analizan el código fuente para generar casos de prueba. 2 Las que se basan en requisitos. Aíslan los requisitos del usuario y sugieren casos de prueba. 3 Lenguajes de prueba específicos. Permiten elaborar especificaciones de prueba detalladas para describir todos los casos de prueba y la logística de su ejecución.
De gestión de pruebas	Generalmente se utilizan para controlar y coordinar todos los pasos principales de las pruebas. Dar el formato adecuado a los datos de entrada, comparar los resultados de salida para cada prueba.
De pruebas cliente/servidor	Básicamente hacen mediciones de comunicación entre la interfaz del cliente y la respuesta del servidor.
De reingeniería	Esta categoría se puede dividir en 2 subdivisiones: 1 Herramientas de reestructuración y análisis de código, donde se analiza la sintaxis del programa, se genera una gráfica de control de flujo y un programa

	estructurado. 2 Herramientas de reingeniería para sistemas on-line se utilizan para modificar sistemas de bases de datos *on-line*.

Tabla. 1.14 Clasificación de tipos de herramientas.

Clasificaciones

Piattini (2004: 658), menciona que la tecnología CASE tiene una terminología confusa que origina numerosas clasificaciones de las herramientas CASE. Algunos autores los clasifican con base en el apoyo a los usuarios, otros por las fases del ciclo de vida que cubren, su funcionalidad o arquitectura. En esta unidad sólo se tomarán las aportaciones de Senn, Kendall y Piattini. Cabe mencionar que los planteamientos de kendall y Piattini son más actuales, por lo que dan por hecho que las herramientas CASE están integradas.

Según Senn, estas herramientas se agrupan en:

- Front-end

  > Automatizan las primeras actividades del proceso de desarrollo de sistemas (…) para ayudar al analista a preparar especificaciones formales que carezcan de ambigüedades, a validar las descripciones del sistema con el objeto de determinar su consistencia y completud, y a seguir la evolución de los requerimientos de la aplicación en características que formen parte del sistema que finalmente será implantado.

- Back-end

  Conocidas como herramientas para programación asistida por computadora "tienen como finalidad ayudar al analista a formular la lógica del programa, los algoritmos de procesamiento y la descripción física de datos, también ayudan a la interacción con los dispositivos (para entrada y salida), etc." (1992: 289).

- Integrales

  Estas herramientas se caracterizan por proponer un ambiente que automatiza las actividades durante todo el ciclo de vida del sistema. Fundamentalmente facilitan el manejo de aspectos de análisis y desarrollo, además del diseño, administración y mantenimiento del código. Así como ser eficiente en la administración general de los sistemas (creación, almacenamiento, manipulación y documentación).

Los primeros sistemas CASE eran un conjunto de herramientas individuales usadas para una parte determinada del proceso de *software*, sin interfaces estandarizadas y diferentes para cada tipo de *hardware*, sistema operativo y lenguaje de programación. Se necesitaba entonces que los responsables del desarrollo del sistema pasaban manualmente las especificaciones, datos y tareas generadas en herramientas de *front-end* a *back-end* y viceversa. Cuando las herramientas CASE tuvieron la posibilidad de integrarse se les llamó iCASE.
Senn dice: “Las herramientas integrales proporcionan un ambiente que automatiza tareas clave a lo largo de todo el proceso de desarrollo (…), brindan un ambiente para crear, almacenar, manipular, administrar y documentar sistemas.” (1992: 291)

Para Seen, la integración de herramientas ocurre en tres formas:

• Creación de una interface para desarrollo uniforme o adaptable. En un sistema CASE integrado debe haber un menú desde donde se acceda a todas las herramientas, sin necesidad de salir de una aplicación para activar otra y la información debe estar en un formato uniforme.

• Proporcionar la facilidad para transferir datos entre las herramientas

La integración de las herramientas, permite que la información obtenida con una de ellas sea utilizada por otra dentro del mismo proyecto.

En esta tarea, el repositorio es la fuente de datos que recibe y de la cual se alimentan todas las herramientas donde se almacenan datos sobre el problema a resolver, modelos o metodología de solución e implementación, bitácora de proyectos, recursos, presupuestos, organigramas y demás datos necesarios.

• Unir las actividades de desarrollo

Teniendo una fuente central de datos uniformes a través de todo el proceso de desarrollo, se crean enlaces, sea en forma manual o automatizada. Un ejemplo de unir las actividades de análisis a desarrollo es realizar un diagrama de flujo para después convertirlo a código fuente, esto se lograba con herramientas workbench.

A manera de resumen sobre la idea de sistemas integrados en la época de los 90, cito textualmente a Senn:

> La herramienta ideal (aunque todavía no existe) debe tener la capacidad para volver a conformar la salida de una actividad en una entrada para la siguiente actividad. Por ejemplo, los diagramas de flujo de datos, las descripciones de procesos y los almacenes y flujos de datos definidos en la fase de análisis, deberían transformarse de manera automatizada en diagramas estructurados, funciones y módulos para el proceso de diseño. (1992: 300).

La integración de datos (repositorio), de control (registro de eventos) y de presentación (interfaz homogénea) representa una de las evoluciones más importantes de las herramientas CASE y a raíz de ello surgieron estándares como ANSI X3H6C (*Tool Integrarion Models* Comité), OMG (CORBA, *Common Object Request Broker Architecture*) y el IEEE (*The Institute of Electrical and Electronics Engineers*. Inc).

Clasificación según Piattini (2004: 658):

- Herramientas de gestión. Encargadas de la estimación, planificación y seguimiento del proyecto para visualizar mejor el flujo de datos entre los elementos que participan en el desarrollo del sistema desde la determinación de requisitos hasta el desarrollo del producto final.
- Herramientas técnicas. Se dividen en:

  - CASE frontales (*front-end*) o superiores (*Upper CASE*), éstas abarcan las primeras fases de análisis y desarrollo.
  - CASE dorsales (*back-end*) o inferiores (*Lower CASE*), cuyo objetivo es el diseño detallado y generación de código.

- Herramientas de soporte. Son sistemas de integración como los repositorios, gestión de redes, gestión de bases de datos, herramientas de control de calidad y supervisión, información de seguridad y demás documentación necesaria.

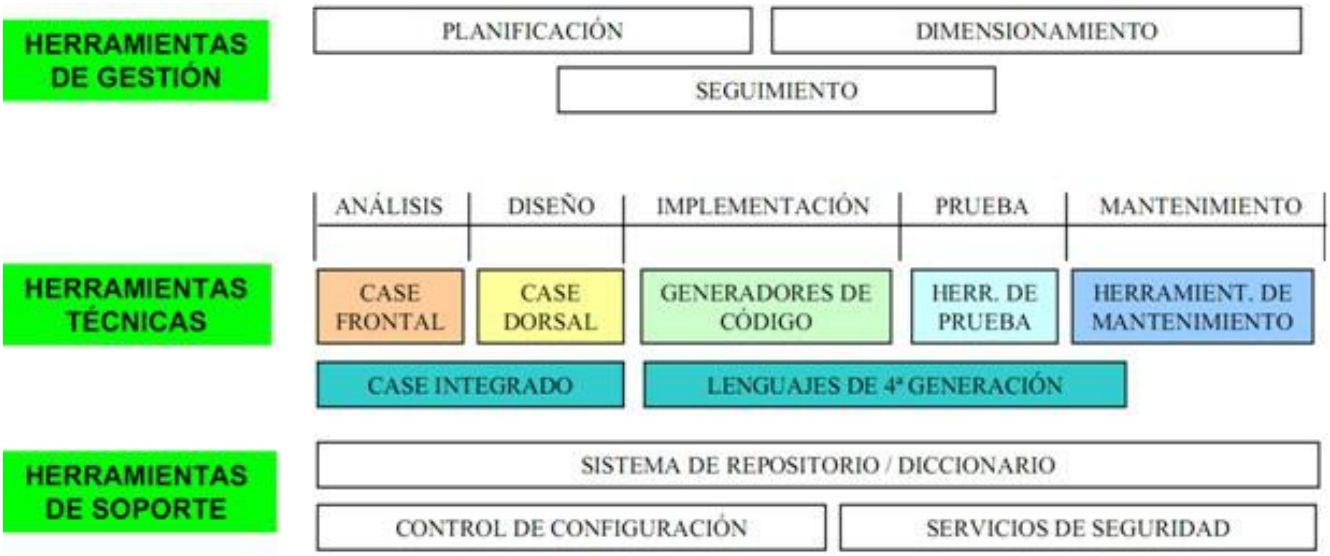


Figura. 1.12. Fuente: Piattini (2004), *Análisis y diseño de aplicaciones informáticas de gestión: una perspectiva de ingeniería de software*. México: Alfaomega, pág. 659.

Kendall y Kendall (2005:16) mencionan que estas herramientas se clasifican en:

- CASE de alto nivel (U-CASE) son aquéllas que ayudan en el análisis y diseño de sistemas e incluyen los generadores de interfaces, diagramación, prototipos y repositorios.
- CASE de bajo nivel (L-CASE) mayormente usadas por programadores y quienes implementan y dan soporte a los sistemas.

Actualmente, las herramientas de desarrollo son y deben ser integrales, es decir, tienen todos sus elementos integrados, de tal modo que proporcionan un ambiente comunicado que automatiza tareas clave a lo largo de todo el ciclo de vida de la aplicación.

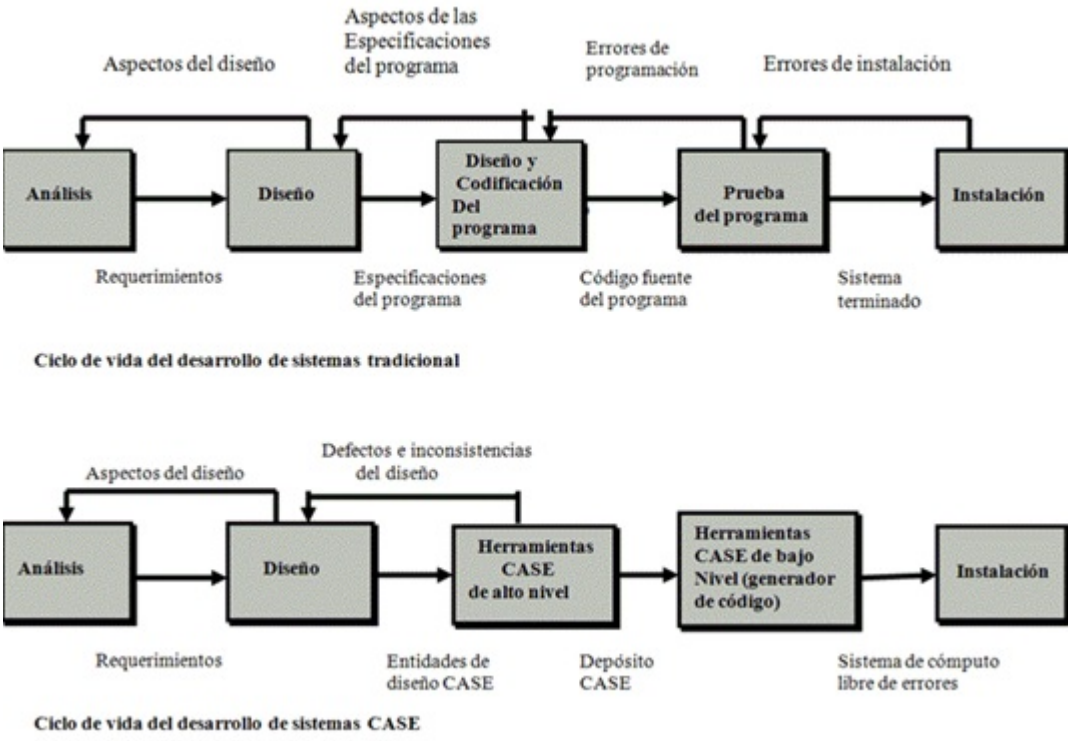


Figura. 1.13. Fuente: Kendall (2005). *Análisis y diseño de sistemas* [8ª ed.]. México: Pearson, pág. 18.

iCASE en la actualidad

A inicios del siglo XXI las herramientas CASE dejaron de desarrollarse debido a inconvenientes, sobretodo de integración, costos de mantenimiento, incompatibilidad entre herramientas y el surgimiento de nuevas tecnologías de cómputo, telecomunicaciones y lenguajes de programación. Esto no significa que las herramientas CASE hayan desaparecido, sino que evolucionaron dejando de llamarse CASE (desde mi punto de vista por cuestiones mercadológicas).

Existen herramientas tipo iCASE para el desarrollo de *software* con elementos como repositorios, herramientas de diagramación, creación automática de código, control de cambios por citar algunos.

A continuación, se mencionan algunas de las más conocidas:

EasyCASE, MetaEdit+ Workbench, Modeler, Erwin, PowerBuild, Oracle Designer, Microsoft Office Enterprise Project Management, Rational rose. Para uso personal están Visual Paradigm for UML, Poseidon, Eclipse, NetBeans, Microsoft Access, Microsoft SQL server, Microsoft FoxPro.

RESUMEN

En un principio, tener un sistema con calidad, significaba cumplir con los requerimientos de ser confiable, fácil de usarlo (amigable) y fácil de mantener. Actualmente, existen modelos como CMMi, PSP, ITIL y MOPROSOFT que, además de lo anterior, consideran otros factores como ser configurable, eficiente, económico, corregible, flexible, interoperable, reusable, íntegro, seguro, portable, certificable. Estos modelos contienen procesos de validación, estandarización y certificación, tanto en las etapas de análisis y diseño (proceso de desarrollo) como en la implantación, pruebas y mantenimiento (creación del producto).

Lograr la calidad de un sistema no es el principio del sistema, sino el fin al que se aspira llegar y para ello se deben tomar en cuenta los factores de calidad de los cuales en la unidad se mostraron diversas propuestas tanto para el proceso de desarrollo como en nivel de producto.

Para llegar a ese fin, se han hecho propuestas que indican los pasos a seguir, estas propuestas se engloban en 3 estrategias que ayudan a lograr un sistema con calidad:

- Ciclo de vida de sistemas. Define las fases para el desarrollo de *software* como son la determinación de requerimientos, diseño del sistema, desarrollo de *software*, pruebas de desempeño e implantación.
- Por análisis estructurado. Modela los componentes de un sistema y su interacción por medio de diagramas de flujo y estructuras de datos acompañado del diccionario de datos, lo que ayuda al analista a examinar el sistema.

- Por prototipos de aplicaciones. Consiste en la creación de un sistema incompleto, pero suficientemente funcional para probar ideas o suposiciones y evaluar resultados de una parte o todo el sistema lo que origina un ciclo de ensayo y error hasta lograr el resultado final o abandonar el proyecto.

Ninguna estrategia es mejor que otra, cada una es efectiva si se emplea correctamente y es más fácil de ejecutar haciendo uso de herramientas asistidas por computadora para el desarrollo de sistemas (CASE).

Como son herramientas para creación de interfaces, diagramas, prototipos y código fuente; también repositorios de información y herramientas de gestión de cambios, calidad y detección de errores.

Dada la cantidad de componentes que puede tener un sistema CASE existen varias clasificaciones; sin embargo, con el paso del tiempo muchos autores coinciden en 2 clasificaciones: Front-end o Upper-CASE para componentes que asisten en las primeras etapas de análisis y diseño y Back-end o Lower-CASE para herramientas que asisten en la terminación del producto final.

Existe una tercera clasificación llamada de integración o soporte para herramientas que asisten en la comunicación de datos y gestión entre todos los elementos de los sistemas CASE que en un principio se adquirían por separado pero que evolucionaron a herramientas integradas (iCASE), las cuales integran bases de conocimiento, aplicaciones de administración y componentes para generar código, interfaces, diagramas.

Actualmente, ya no se conocen como CASE, sino como plataformas de desarrollo de *software* menos complejas de entender y manejar e independientes del *hardware*, sistema operativo o lenguaje de programación.

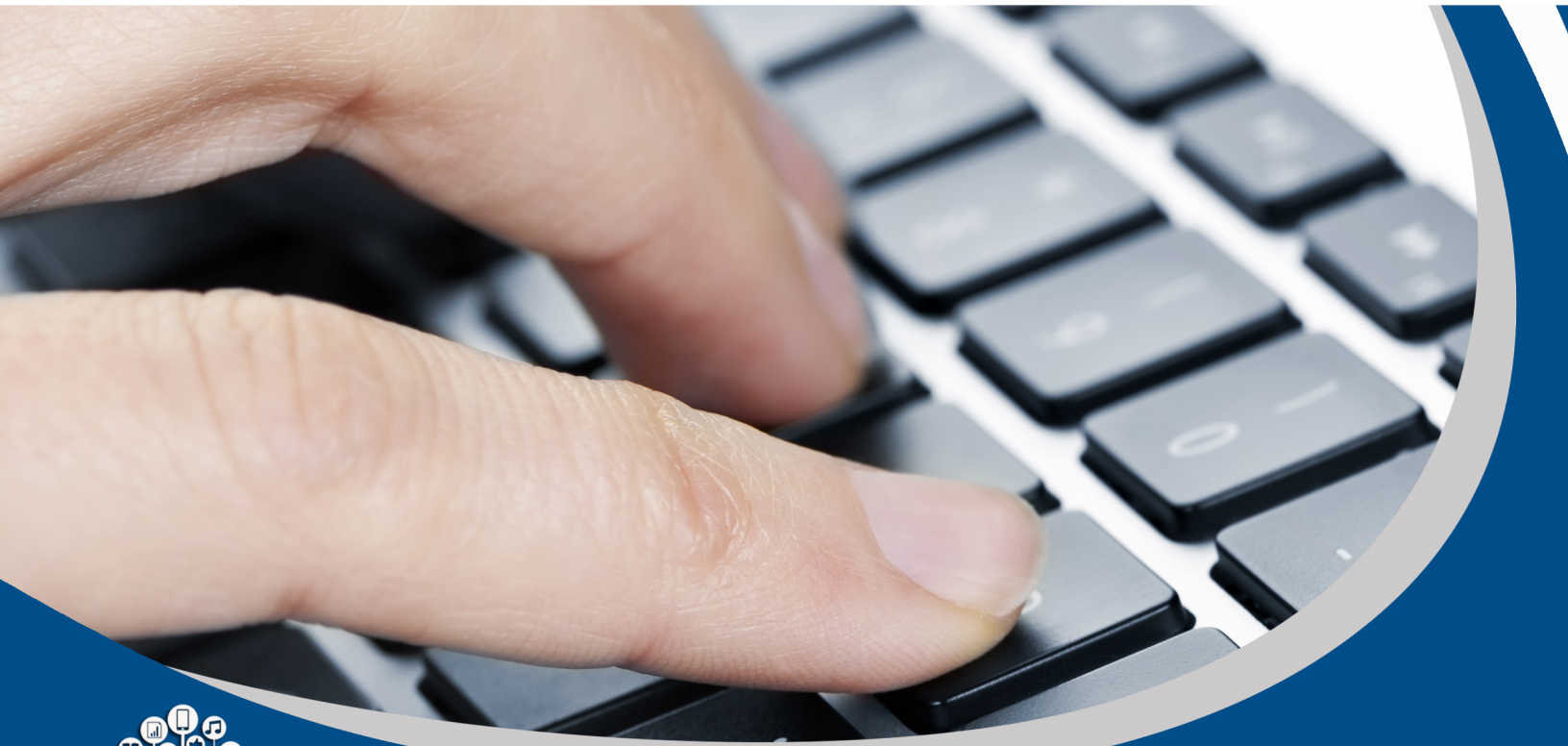
Bibliografía de la unidad

SUGERIDA

Autor	Capítulo	Páginas
Pressman, Roger	8	183 - 202
Mochi, Prudencio (2006)	3 y 4	149 - 227
McClure Carma (1992)	5	179 - 202
McClure Carma (1992)	7	227 - 239

Unidad 2

ANÁLISIS DE SISTEMAS



UNAM FCA
CEDGEC
Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

OBJETIVO PARTICULAR

Modelar el aspecto estático y el aspecto dinámico de la situación actual de la organización con el paradigma estructurado.

TEMARIO DETALLADO

2. Análisis de Sistemas

- 2.1. Identificación del problema
- 2.2. Determinación de requerimientos
- 2.3. Análisis Costo-Beneficio
- 2.4. Estudio de factibilidad

INTRODUCCIÓN

"Los intelectuales resuelven problemas, los genios los previenen."
Albert Einstein

Las TIC siempre están en constante cambio para mejorar su funcionamiento, destaca la velocidad y cantidad de procesamiento, eficiencia en la arquitectura de cómputo, incremento en la memoria, capacidad de almacenamiento, mayor compatibilidad (y a veces incompatibilidad con ciertas marcas o tecnología). Esto sucede gracias a la importancia de las TIC para el desarrollo económico, intelectual y social del mundo actual, tal como menciona Pressman:

"En la actualidad, la enorme industria del software se ha convertido en un factor dominante en las economías del mundo industrializado" (2010: 3).

Dicha evolución de las TIC ha provocado que la ingeniería de *software* sea más amigable y creativa; pero, a la vez, más compleja para un solo programador pues requiere del apoyo de más personal, no sólo de especialistas en *software*, *hardware* o tecnologías, sino también de otras áreas de conocimiento como administradores, financieros, médicos, etc. Sin embargo, las preguntas del programador siguen siendo las mismas para desarrollar los sistemas actuales: Por obsolescencia del *software* u otro motivo ¿Se considera necesario desechar la solución informática existente? ¿El mantenimiento es más caro que crear un *software* nuevo? ¿Cómo desarrollar un nuevo proyecto en el menor tiempo, menor costo y sin errores?

Para Pressman: “La práctica de la ingeniería de software tiene un solo objetivo general: entregar a tiempo software operativo de alta calidad que contenga funciones y características que satisfagan las necesidades de todos los participantes.” (Pressman, 2010: 84-85).

Los temas de esta unidad se refieren al análisis, que es una etapa importante en el desarrollo de *software*, porque se enfoca a solucionar problemas; por lo mismo, se recomienda tener un entendimiento claro de lo que se requiere para solucionar dichos problemas; por ejemplo, tener conocimiento de cuál es el comportamiento del sistema, especificar la solución sin ambigüedades tomando en cuenta las capacidades del usuario final; de forma tal que se consideren los recursos del cliente, además de validar con la calidad necesaria para evitar demasiadas correcciones y, si es factible, la supuesta solución.

Cuando se modela, se busca representar un sistema en la vida real dentro de una organización considerando dos aspectos: Estático y dinámico.

El aspecto estático, le da forma y estructura al sistema con la identificación del problema, entidades y determinación de requerimientos cuidando siempre que las reglas del negocio se cumplan y sean soportadas en un análisis Costo-Beneficio y el estudio de factibilidad.

El aspecto dinámico se refiere a tomar en cuenta las entidades del sistema u objetos de negocio y la interacción entre ellos según las reglas establecidas en el análisis, pero que toman forma en las etapas del diseño.

A este respecto, hay autores como Sommerville, que agrega una nueva perspectiva al mencionar que el proceso de desarrollo RUP (Rational Unified Process) tiene 3 perspectivas (Sommerville, 2011: 50):

- Estática, que presenta las actividades establecidas.
- Dinámica, que muestra las fases del modelo a través del tiempo.
- Actividad, que sugiere buenas prácticas a usar durante el proceso.

Cabe mencionar que, si bien RUP es muy usado actualmente y sus fases abarcan gran parte del ciclo de vida clásico del sistema, no se considera en el enfoque estructurado. Es por ello que en esta unidad no se menciona explícitamente, porque se verá con más detalle en otras asignaturas, pero es importante ilustrar que las fases de RUP para el análisis son:

- Concepción. Identificación de entidades (personas y sistemas) y sus interacciones.
- Elaboración. Comprender el problema de dominio, establecer un marco conceptual arquitectónico, diseñar el plan del proyecto e identificar los riesgos clave del proyecto (Cf. Sommerville 2011: 50).

Las fases RUP de construcción y transición son para las etapas de diseño, programación, pruebas e implementación.

Construir un sistema sin llevar a cabo un análisis, es como edificar una casa sin saber las dimensiones del terreno, tipo de suelo, número de habitaciones o pisos, conexiones de agua y drenaje, costo de los materiales, cuestiones legales, color y arquitectura que el cliente desea. Lo más grave de construir sin análisis, es tener la imperiosa necesidad de hacer reparaciones o en definitiva tirar la casa para construirla como se debe.

En un sistema de información mal diseñado pasa lo mismo que lo descrito anteriormente, al detectarse constantemente fallas, partes del sistema que usuario final no usará, inconsistencias u omisiones de funciones que el sistema no hace. Y entonces surgen las reclamaciones, reparaciones o parches necesarios (a veces costosos), pero en muchos casos evitables con un buen análisis.

El análisis no se trata sólo de determinar el tiempo para terminar el proyecto, las variables, el lenguaje de programación, la plataforma tecnológica o las interfaces, también previenen de dar solución a problemas que no son el principal objetivo y por ello se especifica qué es exactamente lo que el cliente espera que haga o resuelva el sistema, si cuenta con los recursos tecnológicos y operativos, el costo o si vale la pena la inversión para desarrollarlo o mejorar el que ya se tiene. Esto último puede suceder por las siguientes razones:

- Surgimiento de nuevas tecnologías con las que el sistema debe interactuar o ser compatible.
- Nuevas necesidades o procesos del negocio o cliente.
- Corrección de fallas u omisiones.

Después de realizar el análisis, cabe la posibilidad, que se llegue a la conclusión de que el problema se resuelve mejor sin necesidad de un sistema de información.

Las herramientas de modelado necesarias para el análisis son:

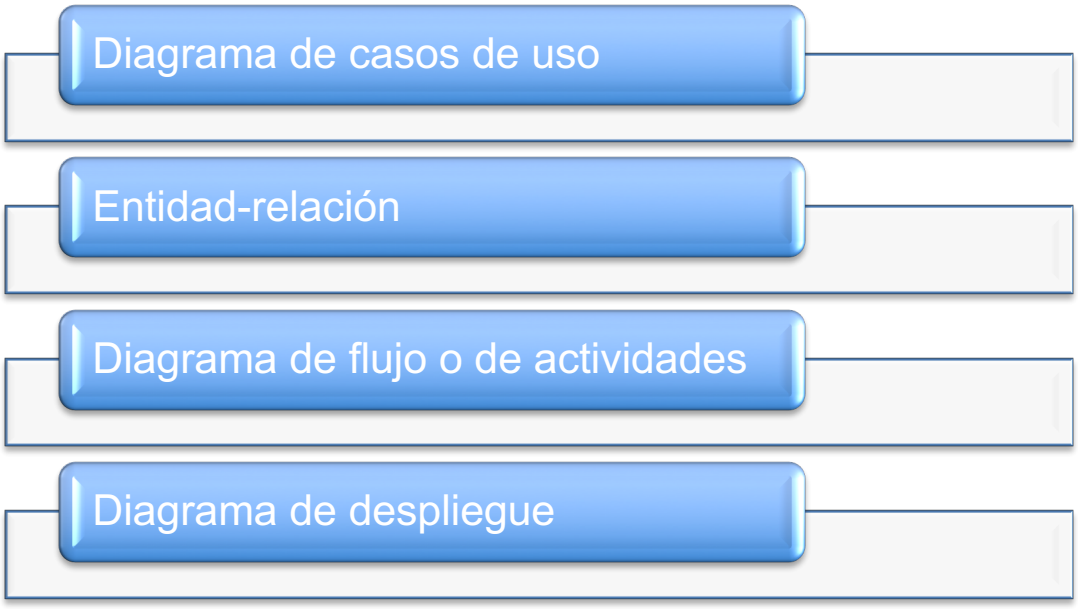


Figura 2.1. Herramientas de modelado. Elaboración propia.

Actualmente, estas herramientas forman parte del lenguaje de modelado UML (Lenguaje Unificado de Modelado), el lenguaje de modelado más usado; cabe aclarar que aquí se mencionan los diagramas UML que ayudan en el análisis, pero que existen otros más para el diseño.

Para el caso del análisis estructurado existen metodologías, nociones y simbologías de autores como Tom de Marco, Farley, Saen A. Garzon y Yourdon.

Actualmente hay otras metodologías o procesos de *software* más detallados o que requieren más documentación que otros. Además, hay métodos de ingeniería del *software* basados en desarrollo interactivo o incremental conocidos como desarrollo ágil con modelos ágiles de proceso como programación extrema (XP), scrum, MDSD, proceso unificado ágil, por citar algunos. Estos modelos omiten ciertas etapas durante análisis y por ende en el desarrollo de *software* que en muchos casos es una buena elección, pero requiere de una gran experiencia, un proceso definido y disciplina por parte del equipo de programadores. Al respecto Pressman aconseja:

“No cometa el error de suponer que la agilidad le da permiso para improvisar soluciones. Se requiere de un proceso, y la disciplina es esencial.” (2010: 57).

2.1. Identificación del problema

"Si tuviera una hora para resolver un problema y mi vida dependiera de ello, pasaría los primeros 55 minutos determinando cuál es la pregunta adecuada, pues una vez la sepa, podré resolver el problema en menos de 5 minutos".
Albert Einstein

Fuente: Hernández, Rafael (2013) "*El arte de diagnosticar la solución a un problema*" Disponible en: https://www.americaeconomia.com/analisis-opinion/el-arte-de-diagnosticar-la-solucion-un-problema. 10/09/2018)

El primer paso para implantar un sistema de cómputo, es identificar la necesidad de su elaboración y después crearlo con base en la utilidad que tendrá para resolver un problema determinado. Pero no siempre el problema se resuelve con un *software* sin tomar en cuenta la afectación que pueda tener en otras áreas del negocio o componentes de un sistema mayor, a veces la solución sólo requiere de un mejor uso o configuración del sistema actual, organización o capacitación del personal, integración de nueva tecnología y hasta corregir errores en el sistema actual por falta de entendimiento del problema.

Hay una delgada línea entre identificar el problema y definir los requerimientos para resolverlo, es decir, si al determinar los requerimientos se logra identificar el problema o viceversa. Según Pressman:

> La mayor parte de proyectos comienzan cuando se identifica una necesidad del negocio o se descubre un nuevo mercado o servicio potencial. Los stakeholders o participantes de la comunidad del negocio (por ejemplo, los directivos, personal de mercadotecnia, gerentes de producto, etc.) definen un caso de negocios para la idea, tratan de identificar el ritmo y profundidad del mercado, hacen un análisis de gran visión de la factibilidad e identifican una descripción funcional del alcance del proyecto. (2010: 102).

Identificar el problema requiere escuchar al cliente o a los que cubrirán procesos del negocio, así como conocer las necesidades, objetivos y políticas del negocio. Si se deben escuchar distintas opiniones, cada una con ideas distintas sobre cuáles son las características y funciones que debe tener el sistema para cumplir con las necesidades del negocio y usarse en las operaciones cotidianas.

Kendall sugiere tomar en cuenta ciertas señales de los stakeholders para identificar el problema:

Para identificar los problemas	Busque estas señales específicas:
Revisar la salida y compararla con los criterios de rendimiento.	• Demasiados errores • El trabajo se completa con lentitud • El trabajo se hace en forma incorrecta • El trabajo se hace en forma incompleta • No se hace ningún trabajo
Observar el comportamiento de los empleados.	• Niveles altos de ausentismo • Mucha inconformidad en el trabajo • Mucha rotación de empleados
Escuchar la retroalimentación externa de: Distribuidores Clientes Proveedores	• Quejas • Sugerencias para mejorar • Pérdida de ventas • Ventas más bajas

FIGURA 3.1
Comprobar la salida, observar el comportamiento de los empleados y escuchar la retroalimentación son todas formas de ayudar al analista a destacar los problemas y oportunidades de sistemas.

Figura 2.2. Fuente: Kendall, 2011: 57.

Algunas preguntas que se pueden tomar en cuenta para identificar el problema son:

- Para un mejor entendimiento ¿El problema puede dividirse en partes distintas entre sí, pero sin perder la interacción entre ellas?
- ¿Hay algún patrón o problema recurrente en varias partes del sistema?
- Además del equipo técnico, ¿quiénes son los participantes del proyecto (directivos, clientes o usuarios)?
- ¿Cuáles son los objetivos y funciones que debe logar el sistema?
- ¿Hay alguna solución a problemas similares que pueda reutilizarse?
- ¿Se puede modelar cada elemento sin necesidad de dar detalles del mismo?

- ¿En el futuro alguien dará mantenimiento al sistema?

El siguiente tema habla sobre la determinación de requerimientos que sirve para definir con mayor detalle el problema al obtener la información que facilite entender lo que el cliente desea, negociar la mejor solución de tantas posibles o propuestas, la función o el comportamiento, lograr que el *software* sea fácil de usar de acuerdo a los requerimientos y administrar los requerimientos a medida que se va desarrollando el *software*.

2.2. Determinación de requerimientos

"Si al franquear una montaña en la dirección de una estrella, el viajero se deja absorber demasiado por los problemas de la escalada, se arriesga a olvidar cual es la estrella que lo guía. "

Antoine de Saint-Exupéry

De acuerdo con el vocabulario IEEE 24765 un requerimiento es:

1. Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.
2. Una condición o capacidad que debe cumplir o tener un sistema o sus componentes para satisfacer un contrato, estándar, especificación u otro documento formal.
3. Una representación documentada de una condición o capacidad. Incluyen las necesidades cuantificadas y documentadas, los deseos y las expectativas del patrocinador, cliente y otras partes interesadas (Std 24765:2010, IEEE: 301).

CARACTERÍSTICAS DE UN REQUERIMIENTO

El estándar IEEE 29148, establece que un requerimiento visto de manera individual debe tener las siguientes características:

- **Necesario**. De tal modo que si se eliminara existirá una deficiencia, que no puede ser cumplida por otras capacidades del producto o proceso. El requisito es actualmente aplicable.
- **Independiente de la implementación**. - El requisito establece lo que se requiere, no cómo se debe cumplir el requisito. No debe incluir o describir restricciones de diseño.
- **No ambiguo. -** Debe definirse de manera única, simple, fácil de entender para todos los involucrados.
- **Consistente.** - Libre de conflictos con otros requerimientos o que pueda ser confundido con otro requerimiento.
- **Completo**. - Describe completamente su funcionalidad, nivel y alcance sin necesidad de información adicional.
- **Singular**. - Una definición incluye un requisito, esto facilita su verificación. (Nota: La especificación sí pude tener varios requisitos).

- **Factible**. - Técnicamente posible dentro de las restricciones, capacidades propias del sistema y su entorno operativo, financiero y legal.
- **Recuperable. -** Codificado adecuadamente para ser identificado de manera única, estructurada y ordenada para su mención en un documento, especificación o artefacto.
- **Verificable. -** Con pruebas que puedan demostrar que cumple con las necesidades y criterios. Es mejor si el requisito puede ser medible.
  Std 29148: 2011, IEEE: 11

Fuente: ISO (2018) Recuperado el 28 de marzo de 2019 de: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29148:ed-2:v1

Algunos autores como Sommerville (2011: 240) agregan la característica de *modificable* acompañado de un historial de cambios. El estándar IEEE 29148 señala esta característica cuando los requerimientos se agrupan.

Al documentar un requerimiento (especificación), éste debe tener los siguientes atributos:

- Un código que identifique a cada requerimiento.
- Descripción que justifique su existencia.
- Cuál es el documento que lo origina (ej. procesos, objetivos, normativa, leyes o roles).
- Prioridad según la importancia dada por el cliente, riesgo del negocio o complejidad técnica.
- Quién desarrolla el producto o responsable de la gestión del requerimiento.
- Si el requerimiento es modificable, indicar cuál es la versión en el historial de cambios.
- Si el requerimiento es usado en diferentes etapas de desarrollo del *software* indicar su estado; por ejemplo, construcción, pruebas o implementación.

- Un mecanismo de trazabilidad que permita identificarlo en los diferentes artefactos.
- Tipo de requerimiento, asumiendo que los requerimientos varían en intención y en los tipos de propiedades que representan. Algunos factores para definir el tipo son: Funcionalidad, desempeño, usabilidad, interfaces, restricciones de diseño, seguridad, calidad o factores humanos.

TIPOS DE REQUERIMIENTOS

Los requerimientos se clasifican en:

- **Requerimientos de negocio.** Describe la visión del producto (lo que el sistema representa para la organización), alcance del producto, las reglas de negocio (objetivos, políticas, condiciones, restricciones, conocimientos, estándares o leyes). (Basado en Torres M. y colabs., 2017:24).

  Ejemplo de regla de negocio en una biblioteca:

  - El usuario debe tener una credencial de alumno oficial o una credencial de biblioteca válida.
  - Un usuario vigente es aquel alumno inscrito o profesor que imparte clase en el semestre en curso.

  De las anteriores reglas se derivan los siguientes requerimientos:

  - El préstamo a domicilio es por un periodo de 7 días.
  - El préstamo de libros a domicilio es sólo para usuarios vigentes.

- **Requerimientos de usuario**

  De acuerdo con Sommerville:

"Son enunciados, en un lenguaje natural junto con diagramas, acerca de qué servicios o comportamiento esperan los usuarios del sistema, y de las restricciones con las cuales éste debe operar" (2011:93). Incluye procedimientos de los *stakeholders* que cumplen las reglas de negocio.

Para Torres se entiende que el objetivo de recolectar los requerimientos de usuario es:
"Entender cuáles son las clases de usuarios del sistema, sus roles y qué es lo que el sistema debe permitirles hacer, consultar, etc., para que se puedan apoyar y cumplir los requerimientos de negocio" (Torres, 2017:26).

En el ejemplo concreto de requerimientos de usuario en una biblioteca se puede observar que:

- El bibliotecario debe ingresar al sistema el número de cuenta del alumno o el número de trabajador del académico y el número de adquisición del ejemplar.
- El usuario puede consultar cuántos libros tiene en préstamo a domicilio.

- **Requerimientos de sistema.** Sommerville dice que "son descripciones más detalladas de las funciones, los servicios y las restricciones operacionales del sistema de software." (2011: 93).

  Ejemplos de requerimientos de sistema en una biblioteca:

  - El sistema debe registrar un historial de préstamos por usuario y por ejemplar.
  - El sistema permite el registro de préstamo a domicilio sólo a usuarios vigentes.

Los requerimientos de sistema se dividen en:

- Requerimientos funcionales.

> Son enunciados acerca de servicios que el sistema debe proveer, de cómo debería reaccionar el sistema a entradas particulares y de cómo debería comportarse el sistema en situaciones específicas (salidas). En algunos casos, los requerimientos funcionales también explican lo que no debe hacer el sistema. (Sommerville, 2011: 85).

Ejemplos de requerimientos funcionales en una biblioteca:

- El sistema debe bloquear el servicio de préstamo a domicilio cuando un usuario solicite una carta de no adeudo de libros.
- El sistema debe enviar un email al usuario 2 días antes de la fecha de vencimiento del préstamo a domicilio.

- Requerimientos no funcionales

"(…) surgen a través de necesidades del usuario, debido a restricciones presupuestales, políticas de la organización, necesidad de interoperabilidad con otro software o sistemas de hardware, o factores externos como regulaciones de seguridad o legislación sobre privacidad". (Sommerville, 2011: 85).

Ejemplos de requerimientos no funcionales en una biblioteca:

- El sistema debe permitir la catalogación de libros en formato MARC 21.
- El sistema debe permitir el registro de libros con la clasificación tipo LC. Según lo establecido por la Dirección General de Bibliotecas de la UNAM.

Los requerimientos no funcionales son también conocidos como cualidades de *software* (Guevara, 2005:50) o atributos de calidad (Torres M. y colabs., 2017:28); sin embargo, también están otros factores que pueden afectar la determinación de requerimientos como factores operacionales propios de la organización y factores

externos como leyes y estándares. El siguiente esquema, muestra las categorías en que se dividen los requerimientos no funcionales según Sommerville:

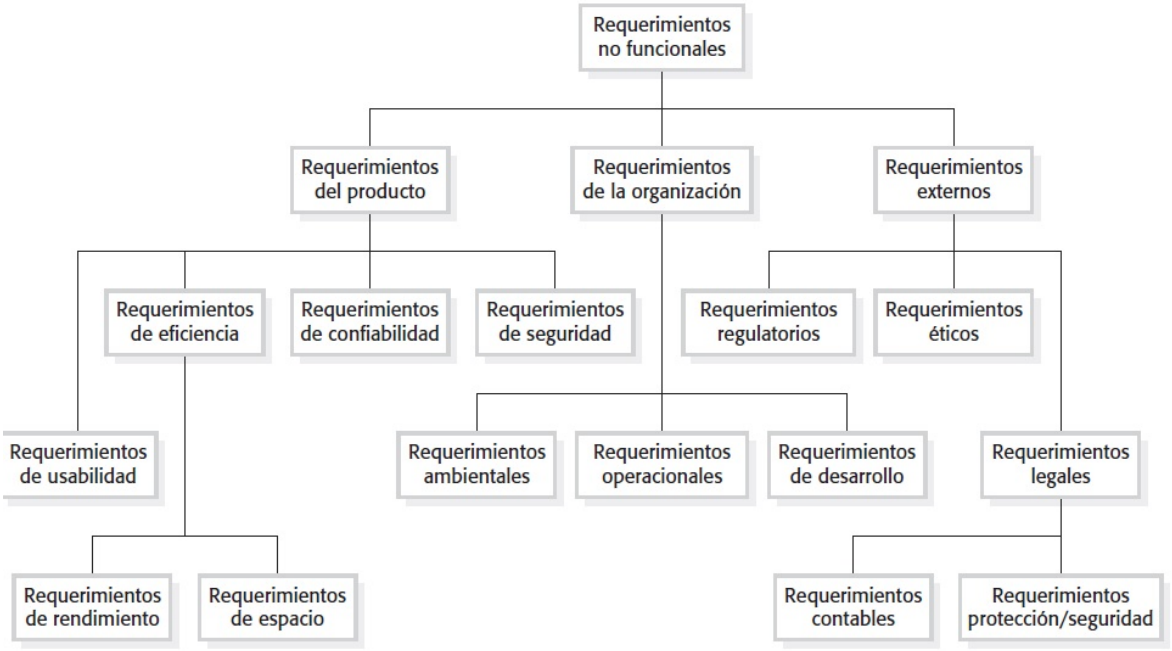


Figura 2.3. Clasificación de requerimientos no funcionales. Fuente: Sommerville, 2011: 88.

- ✓ *Requerimientos del producto.* Describen o restringen el comportamiento del *software*, su clasificación está basada en algunos atributos de calidad de Cavano y McCall, aunque pueden variar según el autor u organismo (véase el tema 1.3 Factores de calidad del *software* de este material didáctico). Los siguientes atributos son considerados para los requerimientos del producto por Gabriel Guevara (2005: 50-58), Ian Sommerville (2011:88), los estándares 25010:2011 según Torres Moreno (2017:29) e ISO 9126 (ahora remplazado por ISO/IEC 25000:2005 SQuaRE). Sobre este tema, Pressman (2010: 343), muestra la opinión de varios autores:

- Usabilidad**.** **"**Un sistema es amigable si un usuario humano lo encuentra fácil de utilizar (…) capacidad que debe tener el sistema de ser comprendido, aprendido y utilizado bajo las condiciones de uso estipuladas en los requerimientos del sistema." (Guevara 2005:52).
- Eficiencia. Uso de la menor cantidad de recursos y un tiempo de respuesta aceptable.
- Confiabilidad. Debe ser capaz de evitar fallos (madurez), mantener un óptimo nivel de rendimiento al presentarse una falla (tolerancia a fallas) o recuperar datos y rendimiento después de una falla (recuperable).
- Seguridad. Protección del sistema para asegurar su integridad física en caso de daños al *hardware* por caídas, descargas eléctricas o robo y seguridad lógica por modificaciones o destrucción del *software*. También seguridad del usuario si el sistema activa dispositivos industriales o médicos. (Basado en Torres, 2017: 30-32).
- Mantenibilidad. Capacidad del sistema para ser modificado, Guevara propone dividir el mantenimiento del sistema en tres categorías que son corrección, adaptación y mejora (Guevara 2005: 53).
- Portabilidad. El sistema debe adaptarse o coexistir con otros ambientes, consiste básicamente en poder ser ejecutado en diferentes ambientes, en plataformas distintas y en determinados sistemas operativos, como lo menciona Guevara (Guevara 2005: 56).
- Funcionalidad. Corresponde al "Grado en el que el software satisface las necesidades planteadas" (Pressman, 2010: 343).

  Este atributo puede confundirse con un requerimiento funcional, sin embargo, es considerado aquí en el contexto de atributo de

calidad del software con las funciones necesarias para cumplir las tareas requeridas (idoneidad) y dando los resultados esperados (precisión). (Basado en Torres, 2017: 29).

Ejemplo de requerimiento del producto:

Deben ser posibles todas las comunicaciones necesarias entre el sistema y el usuario expresadas en el estándar de caracteres ANSI.

- *Requerimientos de la organización.* Son requerimientos basados en objetivos, procedimientos y políticas de la organización que delimitan el funcionamiento del sistema. En esta clasificación considera tres tipos:

Ambiental. Infraestructura física donde operará el sistema.
Operacional. La manera en que se usará el sistema en el negocio.
Desarrollo. Lenguaje de programación, estándares y proceso de desarrollo.

Tabla. 2.1. (Sommerville, 2011: 88)

Mientras que Torres considera:

Entrega. Debe especificarse la fecha límite para la entrega del sistema y la documentación respectiva.
Implementación. Debe especificarse el lenguaje de programación a utilizar, así como el método de diseño más pertinente para utilizarse.
Provenientes de las normas. Describe los estándares de calidad, de documentación y de los procesos de desarrollo.

Tabla. 2.2.(2017:32)

Ejemplo de requerimiento de la organización:

El proceso de desarrollo del sistema y el documento entregable se ajustarán al proceso y los entregables definidos en XYYbd-FCA-STAN-18.

- ✓ **Requerimientos externos**. Factores sin una relación directa con el sistema, pero que influyen en su ciclo de vida, se presentan a continuación algunos ejemplos de estos requerimientos que dan a conocer varios autores:
- ✓
  - **Interoperabilidad.** Que consiste, como lo dice Guevara, en la habilidad que tiene un sistema de lograr la interacción y cooperación con otros sistemas (2005:56).
  - **Regulatorios.** Sujetos a la aprobación de un regulador (ej. Secretaría de Hacienda para la generación de factura electrónica). (Basado en Sommerville).
  - **Éticos.** Se refiere a garantizar que el sistema sea aceptado por todos los usuarios finales.
  - **Legales.** Garantizar que el sistema no infrinja ninguna ley (Ej. Ley Federal de Protección de Datos Personales). (Basado en Sommerville, 2011: 88).
  - **Seguridad.** Como lo dice de manera textual Torres y colaboradores:

> Especificarán cuál es la protección que debe tener el sistema en caso de modificaciones o destrucción. Además, debe asegurar la integridad física de los usuarios que interactúan con el sistema en caso que el sistema controle algún dispositivo de hardware (por ejemplo, un torno automatizado) (2017: 33).

Ejemplo de requerimiento externo:

Con base en el ejemplo que se expone anteriormente, en el caso de una biblioteca, el sistema no deberá dar a conocer al bibliotecario ninguna información de tipo personal de los usuarios que no sean el nombre y número de cuenta.

Otras cualidades de calidad del *software* como requerimientos mencionados son:

Figura 2.4. Fuente: (Guevara, 2005: 50-58)

No olvidemos que, tanto los requerimientos funcionales como los no funcionales, son la base para el desarrollo del sistema, muchos de ellos se interrelacionan, por lo que sería muy difícil entenderlos por separado.

Lo más importante en el desarrollo de un sistema, es que cumpla con las necesidades del cliente. Pressman advierte sobre el dilema de la calidad citando a Bertrand Meyer:

> Si produce un sistema de software de mala calidad, usted pierde porque nadie lo querrá comprar. Por otro lado, si dedica un tiempo infinito, demasiado esfuerzo y enormes sumas de dinero para

> obtener un elemento perfecto de software, entonces tomará tanto tiempo terminarlo y será tan caro de producir que de todos modos quedará fuera del negocio [*Y aconseja:]* Cuando se enfrente al dilema de la calidad (y todos lo hacen en un momento u otro), trate de alcanzar el balance: suficiente esfuerzo para producir una calidad aceptable sin que sepulte al proyecto (Pressman, 2010: 345).

PROCESOS DE INGENIERÍA DE REQUERIMIENTOS

Con la determinación de requerimientos, se afina la identificación del problema y su solución al describir lo que debe hacerse y sin considerar en esta fase como debe implementarse. Esta etapa forma parte de la ingeniería de requerimientos definida en el estándar IEEE 24765 como "ciencia y disciplina relacionada con el análisis y la documentación de los requisitos" (Std 24765:2010, IEEE: 302). Según la IEEE, la ingeniería de requerimientos consiste en:

Software requirements elicitation (SRE). Proceso a través del cual el desarrollador del sistema y los participantes o *stakeholders* (personal involucrado: clientes, usuarios finales, administradores, encargados del mantenimiento del sistema y expertos) descubren, revisan, articulan, y comprenden las necesidades de los usuarios y las limitaciones del *software* y el desarrollo de actividades, Cabe mencionar que, según el diccionario de la Real Academia Española, la palabra elicitación[13] no existe; en inglés *elicit* significa obtener algo con dificultad.

Software requirements analysis (SRA). Proceso de analizar las necesidades de los clientes y usuarios que permita llegar a una determinación de requerimientos.

[13] Aunque la RAE no considera el término *elicitación*, en el ámbito informático es explicado en el siguiente sitio: Innova-T (1987), "*Análisis de Requerimientos de los Sistemas de Información*" Recuperado el 28 de marzo de 2019 de: **http://innova-t.co/lessons/en-que-consiste-la-elicitacion-de-requerimientos/**

Software requirements specification (SRS). El desarrollo de un documento donde se registre de manera clara y precisa cada uno de los requerimientos del sistema.

Software requirements verification (SRV). Proceso de garantizar que la especificación de requerimientos cumple con los requisitos del sistema, de acuerdo con la documentación estandarizada en la fase de requerimientos, y es una base adecuada para la fase de diseño.

Software requirements management (SRM). Planificación y control de las actividades de "elicitation", especificación, análisis y verificación de requerimientos (Thayer, R. H., 2000: 1).

Para Pressman, la ingeniería de requerimientos tiene que ver con el mecanismo apropiado para comprender lo que el cliente desea; considera analizar las necesidades, factibilidad, negociación de una solución razonable, precisar una solución concreta y sin ambigüedades, validar la especificación y administrar los requerimientos en la medida que se convierten en un sistema útil y funcional. Menciona, además, que deben considerarse las siguientes siete tareas:

Concepción. Consiste en definir el alcance y la naturaleza del problema.
Indagación. Se fundamenta en definir lo que quieren los participantes.
Elaboración. Esto es, refinar y modificar los requerimientos.
Negociación. Consiste en afinar la definición del problema de acuerdo con las prioridades, objetivos y políticas.
Especificación. Tener en cuenta los requerimientos aprobados.
Validación: Consiste en garantizar lo que quiere el cliente y las propuestas del analista.
Administración. Se encarga de gestionar todo el proceso.

Tabla. 2.3. (Pressman, 2010: 102)

Por la importancia del tema, es preciso citar textualmente a Kendall en lo que plantea respecto a la determinación de requerimientos:

> El analista de sistema debe conocer los detalles sobre las funciones del sistema actual: el quién (las personas involucradas), el qué (la actividad de la empresa), el dónde (el entorno en el que se lleva a cabo el trabajo), el cuándo (la coordinación) y el cómo (de qué manera particular se realizan los procedimientos actuales) de la empresa a la que está estudiando. Y agrupa la ingeniería de requerimientos en 3 fases (con sus herramientas o técnicas):
>
> **Recopilación de información** (métodos interactivos (entrevistas, cuestionarios) y métodos discretos (muestreo, investigación, observación).
> **Determinación de requerimientos** (diagramas de flujos de datos (físicos y lógicos)).
> **Especificación de los procesos** (español estructurado, tablas de decisiones y árboles de decisión.) (2011:10).

Otros autores como Loucopoulos, Sommerville o Herrera integran en la fase de *elicitation*, las tareas de concepción e indagación con el fin de conseguir la información útil para entender la necesidad del cliente, y en la fase de análisis las tareas de elaboración y negociación para establecer los requerimientos.

En conclusión, la ingeniería de requerimientos es un puente entre el diseño y la construcción, que define las actividades para determinar los requerimientos; es decir obtener, analizar y especificar los requerimientos para luego verificar que con esas especificaciones se logre el sistema que el cliente necesita.

Obsérvese el siguiente esquema:

Figura 2.5. Fuente: Tecnología de Sistemas de Información (2017) "Gestión de requerimientos. Proceso metodológico" Recuperado el 13 de marzo de 2019 de: http://www.tsitecnologia.com.co/outsourcing-ti-servicios-gestion-requerimientos/.

Saber qué es lo que pretende resolver el cliente y lo que se requiere para ello, es una de las tareas más cruciales para los responsables de crear el sistema o *software* porque muchas veces el cliente no sabe lo que quiere, y los profesionales del *software* no conocen los procedimientos, objetivos y políticas del negocio. A lo largo del desarrollo del *software* puede haber cambios radicales y diferentes ideas o contextos, sobre todo, entre el cliente y el usuario final.

En conclusión y considerando las ideas de los diferentes autores, para este apunte se define la determinación de requerimientos con las fases de:

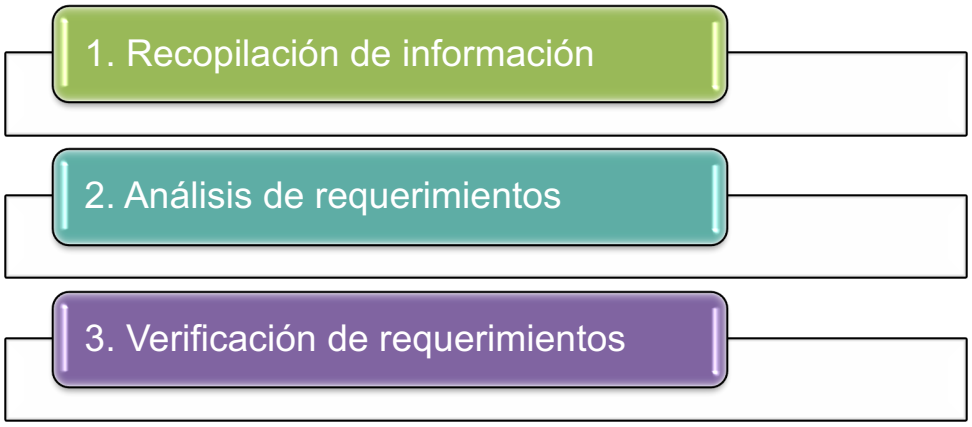


Figura 2.6. Fases de determinación de requerimientos. Elaboración propia.

I. Recopilación de información

Para recopilar la información precisa, completa y adecuada que determine los requerimientos, se debe tomar en cuenta lo siguiente:

1. Identificar al personal involucrado en el proyecto (*stakeholders*) como pueden ser clientes, usuarios finales, administradores, encargados del mantenimiento del sistema y expertos de otras áreas diferentes a la informática.
2. Reconocer las fuentes de información que permitan conocer, comprender y evaluar los asuntos del negocio.
3. Realizar las preguntas adecuadas a los participantes mediante entrevistas y cuestionarios, para conocer las necesidades que el sistema resolverá.
4. Entender las necesidades que el cliente y el usuario final consideren para resolver el problema.
5. El usuario final no sabe o no quiere dar información sobre los procesos operacionales.

6. Ser consciente que un requerimiento no siempre se define al primer intento entre un grupo de personas con ideas diferentes y será necesaria una negociación.
7. Reconocer la infraestructura tecnológica y restricciones del sistema (sin realizar diseño ni implementación).
8. Todos los requerimientos deben quedar especificados por escrito.
9. Durante el proceso de desarrollo podrían encontrarse omisiones o surgir nuevas necesidades que cambien un requerimiento u obliguen a especificar un nuevo requerimiento.

- Problemas de alcance. Los participantes especifican de manera confusa detalles técnicos sobre los objetivos del sistema.
- Problemas de entendimiento. Los participantes no entienden del todo el problema ni las capacidades de su infraestructura tecnológica y omiten información por considerarla obvia. Esto pude generar conflictos o ambigüedad en los requerimientos.
- Problemas de volatilidad. Es probable que los requerimientos cambien (Pressman, 2010: 103).

Técnicas de recopilación de información

El principal objetivo de las técnicas de recolección de información es mejorar la comunicación entre usuario, clientes y desarrolladores de *software* para especificar más claramente los requisitos que debe cumplir el *software*.

Según Kendall (2011:10), la recopilación de información se hace por:

- **Métodos discretos** (observación de los stakeholders, procedimientos y el entorno).
- **Métodos interactivos** (entrevistas, muestreos, investigación y cuestionarios).

Se recomienda utilizar primero los métodos discretos para tener una mejor comprensión de lo que se hace y cómo se hace en la organización antes de presentarse con los participantes (*stakeholders*).

Entrevista

La entrevista es un instrumento de recopilación de datos, consiste en un conjunto de preguntas que se hacen de manera directa a una persona con la intención de conocer su punto de vista sobre determinado tema. Las preguntas que se hacen, por lo regular, son abiertas porque de lo que se trata es de recuperar información cualitativa y no cuantitativa ni manipulable para fines estadísticos. Para algunos autores como Kendall, también existe la posibilidad de que se puedan incluir preguntas cerradas, con la finalidad de obtener datos muy precisos.

Para hacer una entrevista con buenos resultados, se hacen las siguientes recomendaciones:

a) Antes de la entrevista

- Estudiar la información obtenida en la investigación documental, esto evita hacer preguntas generales sobre la organización y se ahorra tiempo.
- Tener preguntas abiertas y cerradas establecidas de manera previa con la intención de no llegar a la entrevista a hacer improvisaciones.

b) Durante la entrevista

- Tomar en cuenta el aspecto del vocabulario con la finalidad de contextualizar la entrevista y no divagar ni enfrentar malos entendidos e interpretaciones erróneas.
- Es preciso que se escuche al entrevistado con atención, paciencia y sencillez. Es probable que el participante no sepa expresarse en términos de sistemas de información, pero en todo momento deberán tomarse en cuenta las opiniones.
- Evitar interrupciones.
- Centrarse en un procedimiento y evitar distraerse con otra problemática ajena al problema a resolver.
- Para registrar la información se recomienda tomar notas de lo importante, de ninguna manera deberá tratar de escribir todo lo que dice el entrevistado.
- Los esquemas o dibujos mejoran la comunicación cuando algo no está claro.
- Si es oportuno, se recomienda dar una breve explicación técnica cuando el usuario mencione algún término erróneamente o lo desconozca.
- En la entrevista puede utilizarse la grabación de audio o vídeo, siempre y cuando el entrevistado acepte ser grabado.

c) Después de la entrevista

- Reunir la información recabada.
- Seleccionar la información.
- Presentar la información por escrito en un informe.

Un aspecto que es importante en la entrevista, es el proceso de comunicación, Pressman lo expone claramente en los siguientes puntos:

Tercer semestre

Elementos del proceso	Descripción
Escuchar	Poner atención en las palabras del hablante y evitar las interrupciones. No es conveniente, en ningún momento, manifestar distracción o desaprobación de lo que se dice.
Prepararse antes de comunicarse	De manera previa, deben estructurarse las preguntas y tener información de lo que se va a preguntar.
Debe haber un facilitador en la comunicación	Debe haber un líder que oriente las preguntas y que tenga funciones de mediador.
Comunicación cara a cara	En la entrevista cara a cara pueden utilizarse, por ejemplo hacer algún dibujo como apoyo a la discusión.
Toma de notas y documentar las decisiones	Es importante que se escriban las ideas centrales para mantener un registro de las decisiones importantes.
Búsqueda de la colaboración	La colaboración y consenso se muestran cuando el equipo de trabajo describe las características del sistema.
Formación de módulos de conversación	Cuando hay muchas personas en la conversación, se corre el riesgo de dispersión, por lo que se sugiere que el mediador genere módulos de conversación para que no quede nada inconcluso.
Elaboración de un esquema o	Para no hacer uso únicamente de la

dibujo sobre lo que se dice	expresión verbal, pueden usarse algún esquema o dibujo como apoyo de lo que dice.
Avanzar en la comunicación	En la comunicación de ingeniería de *software* se requiere de tiempo; se sugiere agilizar la comunicación.
Importancia de la negociación	Se deben negociar, por ejemplo, funciones y características, prioridades y fechas de entrega.

Tabla. 2.4. Elementos del proceso de comunicación. Elaboración con base en Pressman, (2010) *Ingeniería del software. Un enfoque práctico*, pp. 86-87. Recuperado el 28 de marzo de 2019. Vista previa en: https://vdocuments.mx/ingenieria-del-softwareunenfoquepractico.html

Muestreos

El muestreo es una herramienta utilizada en la investigación científica, tiene como función determinar una parte del todo para hacer estudios y, con base en los resultados, hacer generalizaciones de toda la población. Existen varios tipos de muestreo: aleatorio simple, sistemático, estratificado y por conglomerados. Para efectos de esta asignatura no es el fin profundizar en cada uno de ellos, ya que los verás con mayor detalle en la asignatura de estadística.

Investigación

Entre las fuentes de investigación documental pueden citarse los reglamentos, manuales, documentos gubernamentales, los proveedores, etc. (Guevara, 2005: 45).

Las organizaciones tienen (o deberían tener) documentos como oficios, manuales de procedimientos, reportes, oficios, memorándum o correos electrónicos que

pueden proporcionar datos muy útiles para describir los procesos en los que el sistema estará involucrado. Estos documentos deben catalogarse u organizarse según la importancia para el sistema y la determinación de requerimientos, de tal manera, que sea fácil de ubicar cuando se especifique y verifique un requerimiento.

Cuestionarios

Los cuestionarios son instrumentos escritos para la recopilación de información, pueden elaborarse con preguntas que pueden ser abiertas o cerradas, estas últimas, tienen la característica que pueden ser de opción múltiple, o falso-verdadero.

II. ANÁLISIS DE REQUERIMIENTOS

Para el análisis de requerimientos se identifican cinco tareas o etapas:

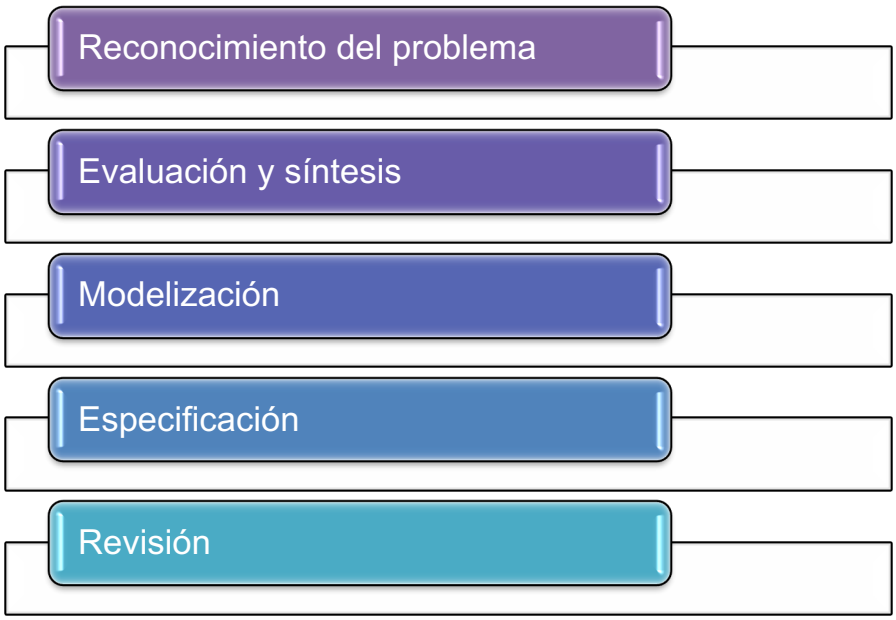


Figura 2.7. Tareas o etapas para el análisis de requerimientos. Elaboración propia.

Con la finalidad de complementar las etapas anteriores puedes consultar el

Anexo 1. Análisis de requerimientos. http://fcaenlinea.unam.mx/anexos/1348/1348_anexo_1.pdf

Negociación de requerimientos

La negociación se realiza por medio de la documentación que se genera en el momento de crear un sistema entre los clientes y desarrolladores. Dichos documentos son tanto de carácter técnico como jurídico y quedan estipulados en el contrato inicial.

Esta negociación trata de integrar la aceptación del sistema, con las condiciones de funcionalidad adecuadas y los cambios y adecuaciones que se generen durante el uso del mismo. En esencia, se negocian los aspectos de costos, condiciones de licencia y posibles cambios al sistema (Sommerville, 2011: 277).

III. Verificación de requerimientos

Una verificación de requerimientos garantiza que el sistema, producto o *software* cumpla con las necesidades del cliente y/o del negocio. Las múltiples interpretaciones o ideas de cómo resolver el problema obtenidas en la recolección de información quedan definidas en el análisis de requerimientos, la especificación establece formalmente los requisitos que debe cumplir el *software* con la posibilidad de realizar pruebas que verifiquen que los requerimientos cumplen con

las necesidades. Y con ello quedan establecidas las bases para el diseño y desarrollo del sistema.

Para la verificación del sistema se utiliza el estándar ISO/IEC/IEEE 29148 de 2011, que describe los procesos que se ejecutan en el contexto de ingeniería de sistemas y describe tres documentos que corresponden a dichos procesos:

1. Documento de especificación de requerimientos de *stakeholder* (StRS). Básicamente, describe los requerimientos de negocio y la motivación después de la construcción del sistema.

   A continuación, se muestra su contenido:

1. Introduction
1.1 Business purpose
1.2 Business scope
1.3 Business overview
1.4 Definitions
1.5 Stakeholders
2. References
3. Business management requirements
3.1 Business environment
3.2 Goal and objective
3.3 Business model
3.4 Information environment
4. Business operational requirements
4.1 Business processes
4.2 Business operational policies and rules
4.3 Business operational constraints
4.4 Business operational modes
4.5 Business operational quality
4.6 Business structure
5. User requirements
6. Concept of proposed system
6.1 Operational concept
6.2 Operational scenario
7 Project Constraints
8. Appendix
8.1 Acronyms and abbreviations

Tabla. 2.5. Documento de especificación de requerimientos de stakeholder (StRS). Fuente: IEEE 29148:43.

2. *Documento de especificación de requerimientos de sistema (SyRS),* que describe el sistema, su entorno y las formas de interacción con los usuarios y otros sistemas. Contiene elementos que se asocian a restricciones, requerimientos no funcionales como la usabilidad, el desempeño, la seguridad, entre otros.

1. Introduction
1.1 System purpose
1.2 System scope
1.3 System overview
1.3.1 System context
1.3.2 System functions
1.3.3 User characteristics
1.4 Definitions
2. References
3. System requirements
3.1 Functional requirements
3.2 Usability requirements
3.3 Performance requirements
3.4 System interface
3.5 System operations
3.6 System modes and states
3.7 Physical characteristics
3.8 Environmental conditions
3.9 System security
3.10 Information management
3.11 Policies and regulations
3.12 System life cycle sustainment
3.13Packaging, handling, shipping and transportation
4. Verification
(parallel to subsections in Section 3)
5. Appendices
Assumptions and dependencies
Acronyms and abbreviations

Tabla 2.6. Documento de especificación de requerimientos de sistema (SyRS). Fuente: IEEE 29148:44.

1. *Documento de especificación de requerimientos de software*: es el más común. Describe el software, sus restricciones, operaciones y requerimientos.

1. Introduction
1.1 Purpose
1.2 Scope
1.3 Product overview
1.3.1 Product perspective
1.3.2 Product functions
1.3.3 User characteristics
1.3.4 Limitations
1.4 Definitions
2. References
3. Specific requirements
3.1 External interfaces
3.2 Functions
3.3 Usability Requirements
3.4 Performance requirements
3.5 Logical database requirements
3.6 Design constraints
3.7 Software system attributes
3.8 Supporting information
4. Verification
(parallel to subsections in Section 3)
5. Appendices
5.1 Assumptions and dependencies
5.2 Acronyms and abbreviations

Tabla. 2.7. Documento de especificación de requerimientos de software Fuente: IEEE 29148:45.

Este documento reitera la definición de requerimientos en cuanto a los términos técnicos.

Es la contrapartida técnica al documento de definición de requerimientos y es escrito por los analistas.

De acuerdo con los planteamientos de Kendall, en cuanto a los requerimientos, se deben establecer los objetivos de los usuarios y la organización. Se recomienda la metodología arriba-abajo, que consiste en que todas las decisiones en la organización se relacionan con los objetivos de la organización. El propósito de la especificación de requerimientos es útil para evaluar los resultados del sistema, además de ser una guía para los diseñadores del mismo.

A continuación, se presenta la estructura de un documento de sistemas de acuerdo con Sommerville:

Capítulo	Descripción
Prefacio	Debe definir el número esperado de lectores del documento, así como describir su historia de versiones, incluidas las causas para la creación de una nueva versión y un resumen de los cambios realizados en cada versión.
Introducción	Describe la necesidad para el sistema. Debe detallar brevemente las funciones del sistema y explicar cómo funcionará con otros sistemas. También tiene que indicar cómo se ajusta el sistema en los objetivos empresariales o estratégicos globales de la organización que comisiona el software.
Glosario	Define los términos técnicos usados en el documento. No debe hacer conjeturas sobre la experiencia o la habilidad del lector.
Definición de requerimientos del usuario	Aquí se representan los servicios que ofrecen al usuario. También, en esta sección se describen los requerimientos no funcionales del sistema. Esta descripción puede usar lenguaje natural, diagramas u otras observaciones que sean comprensibles para los clientes. Deben especificarse los estándares de producto y proceso que tienen que seguirse.
Arquitectura del sistema	Este capítulo presenta un panorama de alto nivel de la arquitectura anticipada del sistema, que muestra la distribución de funciones a través de los módulos del sistema. Hay que destacar los componentes arquitectónicos que sean de reutilización.
Especificación de requerimientos del sistema	Debe representar los requerimientos funcionales y no funcionales con más detalle. Si es preciso, también pueden detallarse más los requerimientos no funcionales. Pueden definirse las interfaces a otros sistemas.
Modelos del sistema	Pueden incluir modelos gráficos del sistema que muestren las relaciones entre componentes del sistema, el sistema y su entorno. Ejemplos de posibles modelos son los modelos de objeto, modelos de flujo de datos o modelos de datos semánticos.
Evolución del sistema	Describe los supuestos fundamentales sobre los que se basa el sistema, y cualquier cambio anticipado debido a evolución de hardware, cambio en las necesidades del usuario, etc. Esta sección es útil para los diseñadores del sistema, pues los ayuda a evitar decisiones de diseño que restringirían probablemente futuros cambios al sistema.
Apéndices	Brindan información específica y detallada que se relaciona con la aplicación a desarrollar; por ejemplo, descripciones de hardware y bases de datos. Los requerimientos de hardware definen las configuraciones, mínima y óptima, del sistema. Los requerimientos de base de datos delimitan la organización lógica de los datos usados por el sistema y las relaciones entre datos.
Índice	Pueden incluirse en el documento varios índices. Así como un índice alfabético normal, uno de diagramas, un índice de funciones, etcétera.

Tabla. 2.8. Estructura de un documento de requerimientos. Fuente: (Sommerville,2011:93).

VERIFICACIÓN

Es el proceso que determina si la especificación es consistente de acuerdo con las necesidades del cliente.

Por lo regular, se trabaja con un bosquejo del documento y se realizan los siguientes requerimientos:

A) **Validez**. Es el compromiso que se contrae con el usuario y de esta manera validar lo que más quiere.
B) **Consistencia**. No debe haber contradicciones.
C) **Realismo**. Que sea factible implementarlo de acuerdo a la tecnología, presupuesto y calendario.
D) **Verificabilidad.** Debe diseñarse un conjunto de pruebas para demostrar que el sistema cumple esos requerimientos.

Validación

Este proceso incluye dos pasos:

- Debe asegurar que cada especificación pueda ser rastreada hasta su requerimiento en el documento de definición.
- Examinar la definición para ver si cada requerimiento es rastreable hasta la especificación.

Este proceso no es únicamente un rastreo de traza, ya que pretende garantizar que el sistema hará lo que los usuarios finales esperan y se debe validar la satisfacción del cliente, por lo regular para realizar la validación se organizan reuniones para realizar revisiones.
Los participantes en estas revisiones son parte del cliente:
Operadores, gerentes, etcétera.

Las revisiones incluyen revisar los objetivos del sistema, evaluar los requerimientos con dichos objetivos y revisar el ambiente de operación, las

interfaces con otros sistemas, funcionamiento completo, restricciones, evaluación de riesgos, etcétera.

Son difíciles de verificar.

Se deben expresar de manera cuantitativa utilizando métricas que se puedan probar de forma objetiva.

Propiedad	Medida
Rapidez	Transacciones por segundo
Tamaño	KB
Fiabilidad	Tiempo promedio entre fallas
Robustez	Probabilidad de datos corruptos después de la falla
Portabilidad	Número de sistemas
Facilidad de uso	Tiempo de capacitación

Para los usuarios es difícil especificarlos de forma cuantitativa.

La determinación de requerimientos puede variar en enfoques diferentes al estructurado; tal es el caso del método ágil o la metodología RUP para orientado a objetos.

2.3. Análisis Costo-Beneficio

"Si buscas resultados distintos, no hagas siempre lo mismo."
Albert Einstein
"El optimista encuentra una respuesta para cada problema. El pesimista ve un problema en cada respuesta."
Anónimo

Durante el desarrollo de *software* se presentan múltiples decisiones que se deben tomar al momento de iniciar un sistema determinado. Dichas decisiones en ocasiones no se toman con fundamento en una confiable base de datos histórica y en ocasiones hay deficiencias en la capacitación del personal para realizar las estimaciones correspondientes y los factores implicados no están bien interrelacionados.

La estimación del costo del sistema impacta sobre la calidad de las decisiones de inversión, donde se toman en cuenta las posibles pérdidas o ganancias que pudieran obtenerse en la organización que invierte sus recursos para la producción del *software*. De esta manera se estiman el esfuerzo, la complejidad y costos del proyecto, esto es de suma importancia para planificar y programar el presupuesto asignado al desarrollo de cualquier sistema que sea redituable para la empresa y sus usuarios.

Modelo para mejorar los procesos de estimación de costos de software.

La estimación de costos toma en cuenta los datos estadísticos que se obtuvieron en la calificación y valoración del *software* diseñado.

Tanto la opinión como la realimentación por parte de los expertos es muy valiosa, ya que dan una orientación sobre cómo se realizar las estimaciones de costos de un proyecto de *software*, se identifican, por cada uno de los agentes del modelo

una lista de variables críticas en el desarrollo del *software*. El siguiente esquema muestra un ejemplo del proceso de estimación de costos.

Figura 2.8. Fuente: Mercado Caruso. Nohora (2015) "Mejora de los procesos de estimación de costos de software. Caso del sector de software de Barranquilla". En *Revista Ingenierías,* Universidad de Medellín. Recuperado el 15 de septiembre de 2018 de: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1692-33242015000200013.

El modelo de análisis de mejora de los procesos de estimación de costos consta de varios elementos como lo muestra el esquema siguiente desarrollado por Nohora Mercado Caruso y colaboradores:

Modelo conceptual de mejora de procesos de costos de *software* para las empresas desarrolladoras.

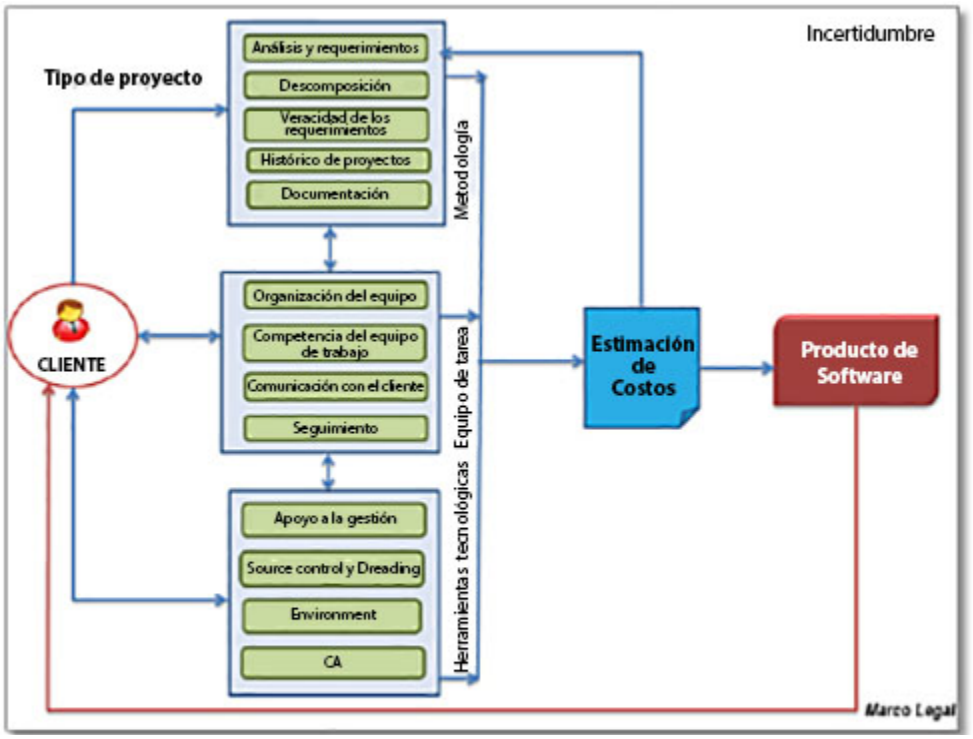


Figura. 2.9. Fuente: Mercado Caruso, Nohora y otros (2015). "Mejora de los procesos de estimación de costos de software. Caso del sector de software de Barranquilla" en *Revista Ingenierías,* Universidad de Medellín. Recuperado el 15 de septiembre de 2018, disponible en: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1692-33242015000200013

MÉTODO DE ANÁLISIS COSTO – BENEFICIO

Durante la realización de un proyecto, el principal objetivo es determinar una medida de costos, la cual nos permita llevar a cabo una comparación de dichos costos con sus beneficios esperados; para la realización del proyecto, se deben tener en cuenta los siguientes puntos:

1. Ver la oportunidad para la realización del proyecto.
2. Comparar entre varias opciones para ver cuál es la más redituable y que beneficie verdaderamente a los usuarios finales.
3. Estimación de recursos tanto económicos, de recursos humanos, así como de otro orden para la realización del proyecto.
4. Realización del procedimiento de costo-beneficio debidamente documentado, para posteriormente evaluar el proceso de inicio a fin.

En el desarrollo del proyecto se analizan los factores que intervienen en la elaboración del *software*, los beneficios que se tendrán a futuro, ya sea tangibles o intangibles, así como los costos que implica dicho desarrollo.

La dirección debe tener toda la información adecuada respecto al proyecto: cómo se va a implantar el *software* para poder realizar el análisis y poder distribuir los recursos necesarios para poder realizar la ejecución y los siguientes costos:

- Equipo para realizar el proyecto.
- Infraestructura: (ambiente adecuado, mobiliario, etc.).
- Personal. (Técnico, administrativo, de capación).
- Materiales que se utilizarán de diversa índole.
- Gastos en consultoría. (Aspectos de garantía de funcionamiento del sistema).

La segunda lista nos referimos a los beneficios que podemos conseguir en el proyecto de forma objetiva y acorde con los objetivos que se quieren alcanzar de esta manera. Daremos un ejemplo.

- Aumento de las utilidades debido al servicio por parte de los clientes.
- Mejora en la toma de decisiones para un mejor soporte operativo.
- Optimización de los procedimientos administrativos.
- Recuperación del capital que podría perderse sin un sistema adecuado en la operación.

Determinar la vialidad del proyecto y su aceptación

Para llevar a cabo la ejecución de un proyecto, es conveniente realizar un estudio en donde se determine si el proyecto que se está realizando es factible o no; por lo tanto, el análisis de costo beneficio se debe utilizar completamente con un método de valor presente.

Como conclusión, se considera que un proyecto es rentable cuando los beneficios exceden a los costos; por lo tanto, el primer paso en el análisis es comprobar la relación costo-beneficio. En general, los bienes que llega a tener un propietario se manifiestan en el dinero que percibe. Por otro lado, los costos son los gastos anticipados para construcción de operación, mantenimiento, etcétera.

La relación convencional Costo – Beneficio se calcula como:

$$B/C = \frac{\text{BENEFICIOS} - \text{DESBENEFICIOS}}{\text{COSTOS}} = \frac{B-D}{C}$$

La relación B/C mayor o igual a 1, indica que el proyecto evaluado es económicamente ventajoso. En los análisis de B/C, los costos no van precedidos por el signo menos.

En resumen, en relación con el método valor presente y estableciendo la TIR de un proyecto, se puede concluir que:

$$\frac{\text{VP del Beneficio}}{\text{VP del Costo}} = 1$$

- Esta fórmula fundamenta el cálculo para realizar el análisis económico conocido como el método de costo- beneficio, si la razón es menor que 1 el proyecto no es viable y se debe rechazar.

2.4. Estudio de factibilidad

"Plantear nuevas preguntas, nuevas posibilidades, considerar los viejos problemas desde un nuevo ángulo, requiere imaginación creativa y marca un avance real en la ciencia."
Albert Einstein

Cuando se va a comenzar con un desarrollo de sistema, se debe realizar un análisis de lo que se pretende desarrollar y deben tenerse en cuenta los aspectos económicos, operativos y técnicos para poder realizar el proyecto. Una de las características de las organizaciones es aplicar un estudio de la factibilidad en un tiempo determinado. Por ejemplo: en una organización que contrata su personal por proyecto; es decir, bajo determinado tiempo, le es indispensable hacer su estudio de factibilidad económica.

El estudio de factibilidad no es un estudio de sistema completo, los usuarios que están a cargo del análisis del proyecto deben recopilar los datos burdos para la administración y una vez que hayan revisado a detalle, tomarán la decisión de seguir o no con el estudio del *software*.

La información que se puede recolectar, es por medio de entrevistas con el usuario y esto se lleva de una manera práctica, con base en la búsqueda de solución a los problemas de la organización.

El analista comúnmente entrevista a aquellos que requieren ayuda y a los que están directamente involucrados con el proceso de toma de decisión, que por lo general son los administradores.

Aunque es importante solucionar el problema de manera adecuada, el analista de sistemas no debe gastar mucho tiempo haciendo estudios de factibilidad, debido a que muchos proyectos serán solicitados y solamente unos cuantos deberán ser

ejecutados. El estudio de factibilidad debe ser elaborado en poco tiempo, y debe comprender varias actividades en corto tiempo.

Definición de objetivos

Durante el proceso de desarrollo de un proyecto, se debe tomar en cuenta la determinación de la factibilidad; esto se refiere a que se deben definir cuáles son los objetivos del proyecto, así como determinar a detalle el proyecto para revisar si realmente el negocio continúa con el objetivo planteado, por otra parte se deben realizar entrevistas con el personal, los grupos o departamentos que proporcionen las entrevistas para llevar a cabo el proyecto; además, revisar todos los trabajos o procesos escritos que estén relacionados con el análisis y requerimientos del proyecto solicitado.
Los puntos que deben considerarse son los siguientes:

- Eliminación de errores.
- Reducción de costos de salida del sistema por medio de agilización y eliminación de reportes.
- Unificación de subsistemas del negocio.
- Proposición de una mejora de servicios al cliente.
- Aceleración de la entrada.
- Disminución del tiempo para el proceso de datos.
- Automatización de procesos, en cuanto a cero errores, aumento de velocidad, precisión, optimización de tiempo, etcétera. (Kendall, 2005: 52).

Determinación de recursos

El estudio de factibilidad cuenta con tres áreas que permite revisar, evaluar y estudiar el *software* para tomar la determinación de los recursos para el estudio de dicho proyecto.

Figura. 2.10 Áreas de estudio de factibilidad. Elaboración propia.

Factibilidad técnica

La factibilidad técnica se refiere a la determinación de recursos de un proyecto, por ejemplo, algún *scanner*, impresora, cámara u otro dispositivo de entrada o salida. El analista debe tenerlos presente para mejorarlos o, si es el caso, incorporar nuevos recursos al proyecto, siempre y cuando sean necesarios; sin embargo, las adiciones a los sistemas actuales son costosos y a veces innecesarios. Se debe pensar siempre en optimizar y adecuar los recursos, de acuerdo con el *software* que se está utilizando.

Entre los aspectos que deben tenerse en cuenta para la factibilidad técnica son los que se muestran en la siguiente figura a manera de ejemplo:

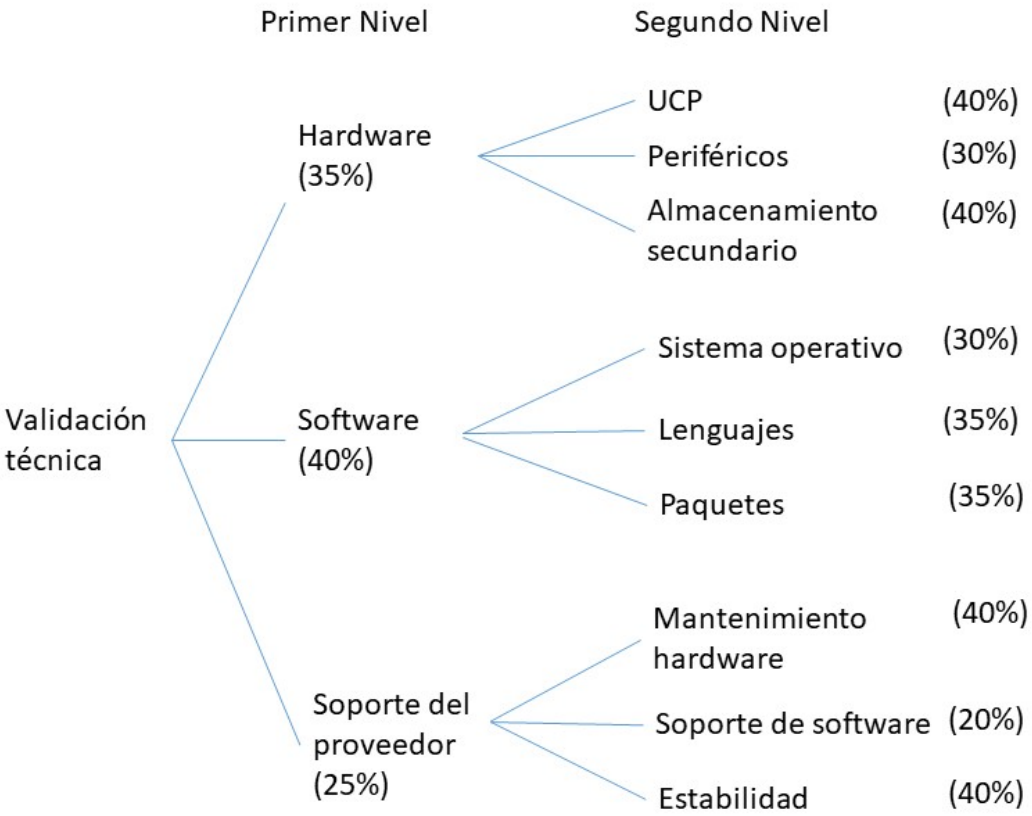


Figura 2.11 Aspectos a considerar para la factibilidad técnica. (Márquez, 1990: 35).

Para la factibilidad técnica, es preciso que se defina algún tipo de cuestionario para la recopilación de información, puede ser específico dependiendo de las características e intereses de la organización.

A manera de ejemplo, consulta el **Anexo 2**. ***Ejemplo de un cuestionario***, http://fcaenlinea.unam.mx/anexos/1348/1348_anexo_2.pdf para que tengas una idea general del cómo elaborarlo[14].

14 Nota importante: el cuestionario no se retoma de manera completa del autor consultado, pero resulta útil para que conozcas qué tipo de información se solicita y puedas adaptarlo a tus necesidades específicas.

Factibilidad económica

Este tipo de factibilidad da una determinación de los recursos básicos del proyecto, así como una lista que a continuación se muestra:

- ***Costos de personal.*** Incluye sueldos y salarios de personal, materiales, tiempo de máquina, etc., requeridos para analizar diseñar, desarrollar, implantar, operar y evaluar el nuevo sistema.

- ***Costos del usuario.*** Incluye el tiempo del usuario para determinar sus requerimientos, así como los de la fase de implantación y mantenimiento del nuevo proyecto.

- **Costos de equipos.** Incluye costo de los equipos utilizados durante el desarrollo del proyecto, así como el costo de adquisición de un nuevo equipo (si se requiere), y los costos de operación una vez que el sistema se encuentre instalado.

- ***Otros costos.*** La introducción de un nuevo equipo implica la previsión de costos de instalación, así como gastos imprevistos.

  (Márquez, 1990: 57)

Antes de comprometerse al estudio de un *software* completo, se recomienda analizar los costos a corto, mediano y largo plazo para el negocio, de modo que no sobrepase el presupuesto. También se verifican los costos de operación, para saber si es factible económicamente y si el proyecto que se está desarrollando debe continuar o debe suspenderse.

Factibilidad operacional

La factibilidad operacional, requiere del aspecto humano para poder continuar con el proyecto, ya que el usuario es el que se involucra y proyecta el sistema que opera diariamente ya instalado. Por otro lado, el analista debe considerar la

factibilidad operacional con los recursos técnicos y económicos para que pueda continuar con el proyecto sin ninguna limitación.

Los usuarios son los principales actores que realizan y ven los problemas del sistema, ya que a diario están en contacto directo con dicho sistema y observan su correcto funcionamiento.

Los analistas de sistemas, deben tener creatividad, desempeño e imaginación; así como también deben escuchar a los usuarios cuidadosamente para conocer lo que necesitan y qué es lo que van a utilizar dentro del sistemas para poder determinar la factibilidad operacional, de tal manera que permita que los usuarios deben saber qué tipos de interfaces son posibles y cuáles satisfacen las necesidades de la organización.

RESUMEN

En esta unidad se desarrollan varios temas en los que podemos identificar los problemas de una organización o empresa para poder dar una solución a los diferentes aspectos que puedan dar una buena toma de decisiones para la organización; además, es conveniente saber aplicar qué tipo de requerimientos se van a utilizar y saber la definición de un requerimiento, sus características principales y de qué manera aplicarlos; sin embargo, es importante saber los tipos de requerimientos funcionales y no funcionales y cómo podemos identificar cada uno de éstos y cuál es la función principal.

Otro punto que se menciona y que es de gran utilidad es el proceso de ingeniería de requerimientos, por su utilidad y cómo aplicarlo en diferentes situaciones en que se llegue a requerir.

Por otro lado, es importante la recopilación de información al momento de estar desarrollando un *software* o sistema para precisar, completar y adecuar los requerimientos que se necesiten para el diseño del *software*, se exponen de manera breve las técnicas de recopilación de información para una mejor comunicación entre el usuario, cliente y los desarrolladores de *software*.

Se sugiere realizar entrevistas al usuario o al personal que tenga contacto con el *software* para una buena recopilación de información que sea de utilidad al momento del desarrollo*;* por ejemplo, costos, inversión, beneficios, etcétera.

Por último, se muestran los procesos en cuanto a la factibilidad, para tener un mejor estudio que nos dé buenos resultados y aplicar los diferentes tipos que hay

dependiendo del área en donde se requiera para un mejor procedimiento y una buena toma de decisiones para un buen desarrollo de *software*.

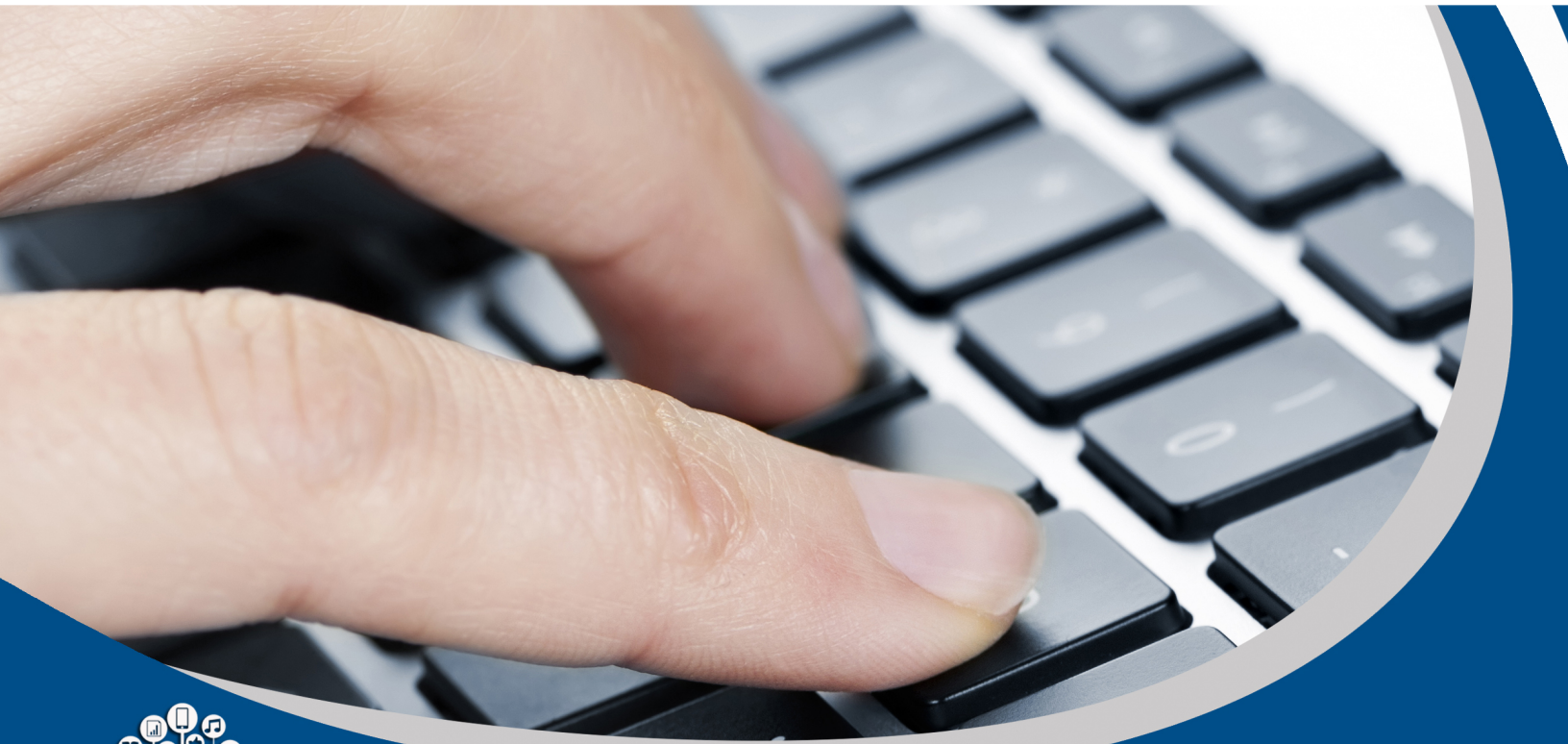
Bibliografía de la unidad

SUGERIDA

Autor	Capítulos	Páginas
Pressman (2005)	1, 2, 7, 12	24, 105, 135, 205, 214, 227
Braude (2003)	1, 6, 9, 10, 11, 12	30,107-123, 175 – 208, 260
Lawrence (2002)	2, 4, 6, 12	72, 157, 107, 201 - 226
Piattini (2000)	2, 3, 9, 17, 26	26, 136, 157, 352, 561
Pressman (2001)	4	51-77

Unidad 3

DISEÑO DE SISTEMAS



UNAM FCA
CEDGEC
Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

OBJETIVO PARTICULAR

Modelar el aspecto estático y el aspecto dinámico de la propuesta de solución para la organización con el paradigma estructurado.

TEMARIO DETALLADO

(16 horas)

3. Diseño de sistemas

3.1. Diseño de salidas de sistema
3.2. Diseño de entradas y controles
3.3 Diseño de archivos
3.4 Diseño de interfaces con bases de datos
3.5 Diseño de interfaces con otros sistemas
3.6 Documentación técnica y de usuario
3.7 Pruebas del sistema
3.8 Liberación
3.9 Mantenimiento

3.1. Diseño de salidas de sistema

Dentro del diseño de salidas de sistemas de información, ésta se debe entregar al usuario final utilizando el intranet.

I. Diseño de la salida para satisfacer un propósito específico.

Toda salida de sistema debe tener un objetivo, no es suficiente en poner a disposición a los usuarios un informe sobre este tipo de diseño, una pantalla o una página web, sólo porque actualmente la tecnología lo permite realizar. En el proceso de la fase de determinación de los requerimientos de información, la función del analista de sistemas debe averiguar qué propósito debe satisfacer, a continuación, se diseña la salida de un sistema con base en un propósito en común.

El usuario debe ver que tiene ciertas oportunidades para proporcionar salidas de sistemas simplemente porque la aplicación lo permite. Sin embargo, si la salida no es exitosa no se debe desarrollar, debido a que la salida del sistema presenta costo, tiempo y recursos para la organización.

II. Diseño de la salida para satisfacer al usuario.

Cuando se tiene un sistema de información de gran tamaño para que atienda a muchos usuarios con actividades diferentes, es más complicado personalizar la salida. Esto se debe a las entrevistas, las observaciones, los costos y algunas veces los propósitos.

En términos generales, es más práctico crear una salida de sistema específica para el usuario y que él mismo pueda personalizarlo. Ésta se diseña para un sistema de apoyo para la toma de decisiones u otras aplicaciones interactivas, tales como el desarrollo web.

Por otra parte, es posible realizar salidas que satisfagan al usuario por medio de una función en la organización para llevar a un objetivo.

III. Entrega de la cantidad adecuada de salida.

Cuando se menciona sobre la cantidad de salida del usuario no siempre es la mejor porque se debe tener una decisión sobre la cantidad de salidas para ver cuál va a ser la correcta y la tarea del diseño de la salida. La salida debe ser útil para cada persona que necesita completar su trabajo, también la información que ocupa el usuario al principio y después acceder fácilmente a la información.

Existen problemas cuando hay sobrecarga de información y esto es una preocupación válida para el usuario, porque no se da un buen servicio y se puede dar información excesiva, se debe tomar en cuenta siempre a los tomadores de decisiones; esto es, que a menudo no necesitan grandes cantidades de salida si existe una manera de acceder a la información por medio de hipervínculos u otras características de extracción de información.

IV. Asegúrese de que la salida esté donde se necesite

Dentro del procesamiento de datos se produce una salida, el aumento de salida en línea debe desplegarse en la pantalla, se puede acceder de forma individual, el analista de sistemas tiene un objetivo en común sobre las salidas del sistema, toda salida se debe presentar al usuario correcto para no entrar en conflictos, ni mucho menos en errores, los informes que se vayan realizando no debe importar

qué tan bien están diseñados, si no llegan a manos de los tomadores de decisiones, no tienen ningún valor estos informes.

V. Suministro la salida de tiempo

Durante el desarrollo de un sistema, una de las quejas más comunes de los usuarios es cuando aún no reciben la información adecuada a tiempo para poder tomar decisiones, pues las salidas a tiempo dan la oportunidad al usuario de revisar y realizar pruebas y tomar las decisiones necesarias que solucionen en parte el problema de la distribución a tiempo de la salida.

VI. Elección del método de salida correcto

Las salidas, como se dijo anteriormente, pueden tener muchas formas; se puede incluir información de pantalla, informes, audios con sonidos digitalizados que simulan la voz humana, micro formas y documentos web con el objetivo de elegir el método correcto para el usuario.

En la actualidad, durante las salidas de sistemas, aparece un menú en la pantalla de las computadoras y el usuario tiene la opción de realizar la impresión. Durante el método de salida que elige el analista de sistemas tiene que ver los pros y contras, los costos para el usuario, pues para éste hay diferencias en cuanto a la accesibilidad, flexibilidad, durabilidad, distribución, posibilidades de almacenamiento y recuperación, transpirabilidad e impacto global de los datos.

Fuente: Gomes Cristian (2019) *Diseñar una salida eficaz.* Recuperado el 29 de marzo de 2019 de: **http://analsisde.blogspot.com/2017/03/diseno-de-la-salida-para-satisfacer-un.html**

RELACIÓN DEL CONTENIDO DE SALIDA CON EL MÉTODO DE SALIDA.

En lo general, las salidas de sistemas se deben pensar de manera que cualquier información producida por el sistema de cómputo debe ser útil para cualquier usuario. La salida se clasifica como externa (la que sale de la empresa), tal como la información aparece en la web y la interna es la que permanece adentro de la empresa, tal como el material disponible en intranet.
Kenneth y Kendall lo explican de la siguiente manera:

> La salida externa difiere de la interna en su distribución, diseño y apariencia. Muchos documentos externos deben incluir instrucciones para el receptor con el fin de que éste los use correctamente. Muchas salidas externas se ponen en formularios impresos previamente o en sitios Web que llevan el logotipo y colores de la compañía.
> Las salidas internas incluyen diversos tipos de informes para los tomadores de decisiones, que van desde breves informes de resumen hasta informes largos y detallados. Un ejemplo de un informe de resumen es aquel que resume los totales de las ventas mensuales. Un informe detallado podría proporcionar las ventas semanales de cada vendedor.
> Otros tipos de informes internos incluyen informes históricos e informes de excepción que sólo se manifiestan como salida en el momento en que ocurre una situación ocasional (2005: 361).

Tecnologías de salida

En las tecnologías se necesita diferentes tipos de salidas. La de impresora necesita una gran variedad de modos de impresión; para las salidas en pantalla se incluyen monitores, video proyector; la salida en audio puede ser un altavoz, bocinas y para las salidas electrónicas se desarrolla una herramienta de *software* especial. Como se puede notar, hay varias opciones de salidas.

Impresión

Toda organización o empresa requiere de salidas de impresión, la tendencia de impresión debe ser con mayor flexibilidad, esta opción nos sirve para la ubicación

de sitios de impresión, dar cabida a diferentes cantidades de caracteres por página, podemos incluir diferentes tamaños de letra, estilos, gráficos, la reducción de almacenar cantidades de formularios pre impresos, etc.

El analista de sistema es el que determina la impresora, por tal motivo tiene que tomar en cuenta tres factores principales de las impresoras como son:

Figura 3.1 Factores de las impresoras. Elaboración propia.

Monitores con dispositivos de salida

El monitor o pantalla es un ejemplo de tecnología de salida más popular, en la actualidad, también son utilizados para la entrada de datos.

Las pantallas constituyen una tecnología factible para varios usos conforme al tamaño, precio y son compatibles con otros componentes de sistemas.

Hay una ventaja al utilizar este tipo de tecnología: la flexibilidad que tienen al permitir al usuario que pueda cambiar la información de salida en tiempo real; todo esto a través de eliminación, modificación o incorporación de información, otra de

las funciones que podemos ver en las pantallas es la revisión de la salida almacenada y el despliegue de la información de una base de datos.

Videos, audio y animación

Cuando se está trabajando, se facilitan opciones de herramientas y paquetes de aplicación para tener una salida de información, el video es un medio por el cual nos facilita la salida, incluye sonido, voz y música con un buen canal visual, existen diferentes usos para incluir un video en las pantallas de los usuarios y son útil para lo siguiente:

- Complementar la salida estática e impresa.
- Colaboración a distancia que conecta a personal que no se ven a menudo. Por ejemplo, las conferencias virtuales, equipos de personas que trabajen en equipo y se reúnen con otras personas.
- Mostrar cómo desempeñan una acción, cómo se debe instalar un *hardware* o un *software*.
- Realizar cursos de capacitación.
- Grabar un evento real para su análisis posterior.
- Conservar una ocasión importante para agregar a la memoria una organización.

La salida de audio va dirigida al usuario, la música y los efectos de sonido hacen que se puedan acceder con facilidad, hay programas que pueden agregar sonido, por ejemplo, el *software* libre *Impress*[15] donde es posible agregar sonidos, música y videos, estos archivos de sonido están en diferentes formatos y los más comunes son WAV, MP3, y WMA que se pueden reproducir en Windows.

[15] Si deseas ver un ejemplo de cómo reproducir sonidos en *Impress*, consulta el siguiente video: https://www.youtube.com/watch?v=x_MYauhy0kQ Recuperado el 25 de marzo de 2019.

Otro tipo de salida que nos permite tener es la animación y esto es para mejorar un sitio web o una presentación, la animación es una presentación de varias imágenes a la vez y algunos con movimientos; las imágenes de animación son símbolos elementales pueden ser objetos abstractos o fotografías reales, y se pueden usar en diferentes colores, tipos, texturas, la animación se usa para apoyar toma de decisiones y resulta mejor la calidad por las imágenes abstractas y las transiciones animadas así es como pueden tomar mejores decisiones. Existen programas diversos para hacer presentaciones, por ejemplo: Prezy, Genially, Canva, Emaze y Slides que tienen opciones para trabajo en línea gratuito y de pago.

CD-ROM y DVD.

Uno de los medios para una salida de manera progresiva es la salida de multimedia por medio del CD-ROM, este es más rápido que algunos otros métodos, son menos vulnerables a los daños por el usuario que otros tipos de salidas. Al CD-ROM se le puede agregar texto, gráficos, música, videos, etc.

El DVD (disco versátil digital). es una tecnología de salida útil, El DVD tiene mayor capacidad que un CD-ROM y una unidad de DVD pude leer tanto un CD-ROM como DVD, éstos no son únicamente de salida, sino también para almacenar información.

Salida electrónica

En la actualidad, muchos sistemas basados en la web tienen la capacidad de enviar la salida electrónicamente en forma de correo electrónico, fax. El correo electrónico se puede operar internamente en las organizaciones a través de la intranet, también otras de las formas que pueden tener comunicación es por *América Online* (AOL). Los correos electrónicos son un gran apoyo para los grupos de trabajo porque son más útiles y flexibles para su uso. En la actualidad, se están diseñando nuevas plataformas de diseño web, en donde las organizaciones pueden tener acceso en grupo a la web con facilidad y seguridad para una buena salida de información, por este medio permite a las organizaciones enviar periódicamente información; estas tecnologías de salida se llaman tecnologías de demanda (*puli Technology*) y tecnologías de actualización automática (*push technology*), esto refleja la manera de cómo los usuarios y las organizaciones van a trabajar por medio de la web para buscar la información.
Otras herramientas de comunicación que actualmente se utilizan por varias empresas son las redes sociales, por ejemplo, Facebook, Youtube, Pinterest, Google +, Linkedin, Instagram, Livejournal, Twitter, entre otras.

Tecnología demandada

Cuando necesitamos tener información de la web, ya sea por los vínculos o por medio de descargas es un medio de salida que podemos obtener, al utilizar una tecnología de demanda tiene varias ventajas al momento de enviar la salida en una hoja de papel simple. En el futuro, los programadores, utilizando *software* de agente inteligente, se pueden ayudar para lo que necesitan de la web.

Tecnología de actualización automática

Existe otro tipo de salida que permite al analista diseñar el contenido inalámbrico y de web mediante tecnología actualmente automática, ésta se puede manejar en la comunicación externa para la actualización automáticamente, toda información solicitada por el usuario se puede usar dentro de la organización para su revisión y hacer una buena toma de decisiones. El término de tecnología de actualización automática se describe como cualquier información o contenido enviado a los usuarios en momentos específicos.

Muchas organizaciones están probando por medio del internet la tecnología de actualización automática para que llegue a personas que lo necesiten y así poder transmitir la información a los empleados, de esta forma es menos caro que imprimir y después distribuirla a unos cuantos usuarios, al trabajar con esta tecnología es muy flexible con la salida de impresión, cuando la información se entrega desde una intranet a una computadora, el usuario puede tomarla y personalizar de muchas formas para tener buenos resultados.

3.2. Diseño de entradas y controles

Diseño de un buen formulario

El analista de sistemas debe tener la capacidad para crear y diseñar formularios completos y útiles. Es necesario eliminar formularios innecesarios que afecten los recursos de una organización, los formularios son herramientas muy importantes dentro de las organizaciones porque se obtienen y capturan información solicitada de los miembros de las organizaciones, por este proceso servirá con frecuencia entrada a la computadora y los formularios sirven como documentos de origen para el usuario. El diseño de formularios útiles debe seguir los cuatro lineamientos siguientes:

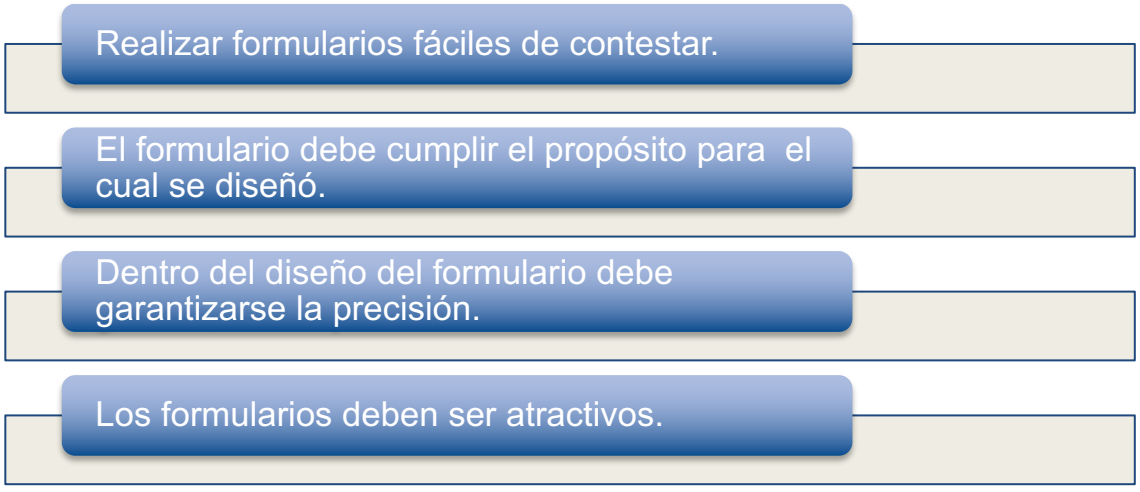


Figura 3.2. Lineamientos para para el diseño de formularios útiles. Elaboración propia.

Creación de formularios fáciles de contestar

Los tipos de formularios que existen son los siguientes[16]:

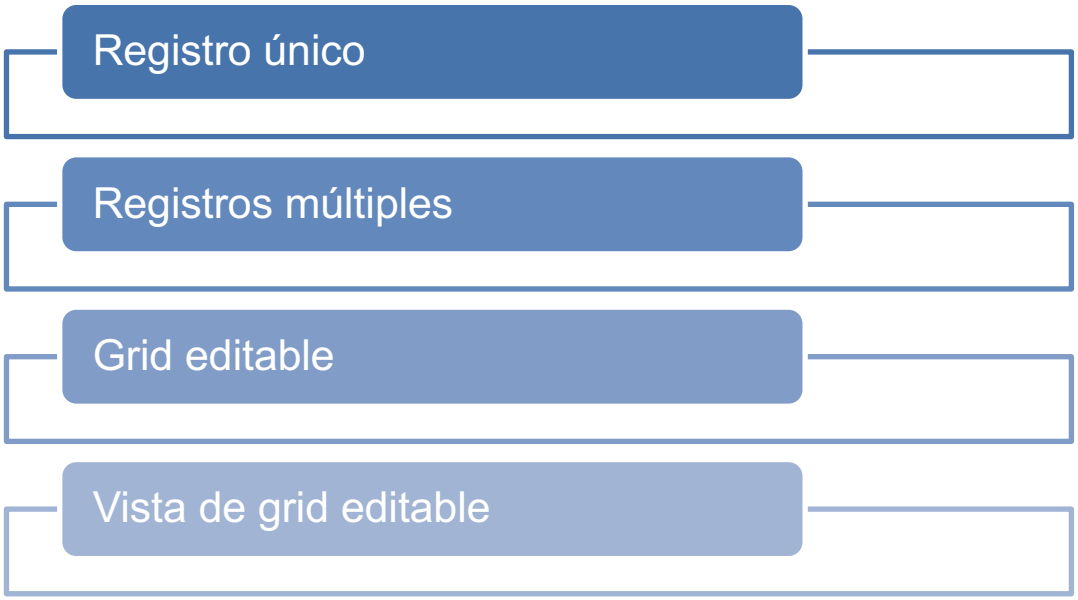


Figura 3.3. Tipos de formularios. Elaboración propia.

Dentro de la creación de los formularios, se deben reducir los errores, aumentar el llenado y tener la facilidad de la entrada de datos. El costo de los formularios es mínimo si lo comparamos con el costo del tiempo que el usuario le dedica al contestar estos formularios y agregar la información correspondiente en el sistema de información.

En un formulario es posible eliminar el proceso de transcribir al sistema de datos que se captura dentro de los formularios para recurrir al envío de medios electrónicos, de este modo, los usuarios introducen la información por sí mismos a través de los sitios web para poder realizar las transacciones con propósitos informativos.

[16] Si deseas profundizar en estos tipos de formularios, puedes ver el video disponible en: https://www.youtube.com/watch?v=m7epTa-K6xM Fecha de consulta: 25 de marzo de 2019.

Flujo de formulario

Cuando se diseña un formulario con el flujo apropiado, se puede reducir el tiempo y esfuerzo; al contestar este tipo de documento, los empleados deben fluir de izquierda a derecha y de arriba hacia abajo.

Existen siete secciones de un formulario que se mencionan a continuación:

- **Encabezado**
- **Identificación y acceso**
- **Instrucciones**
- **Cuerpo**
- **Firma y verificación**
- **Totales**
- **Comentarios**

Estas secciones se deben reflejar como aparece en la tabla siguiente.

Encabezado	**Identificación y acceso**
Instrucciones	
Cuerpo	

Firma y verificación	**Totales**
Comentarios	

Tabla. 3.1. Secciones de formulario. Fuente: Kenneth, 2005: 407.

Diseños de formularios por computadora

En la actualidad, existen muchos programas o *software* para diseñar un formulario en la computadora, algunas características son:

- Es posible diseñar formularios impresos, formularios electrónicos o formularios basados en la web usando un paquete integrado.
- Facilita el diseño de formularios mediante plantillas.
- Posibilita diseñar formularios cortando y pegando formas y objetos conocidos.
- Ofrece la posibilidad de la contestación de formularios electrónicos mediante el uso de un paquete de *software* de captura de datos.
- Permite la personalización de la contestación de formularios electrónicos con la capacidad para personalizar menús, barras de herramientas, teclados y macros.
- Soporta la integración con bases de datos populares.
- Es viable el envío y la transición de formularios electrónicos.
- Ofrece la posibilidad de la transferencia secuencial de formularios.
- Ayuda a dar seguimiento a formularios transferidos a otras áreas.

- Favorece la transmisión y el procesamiento automáticos (tecnología de actualización automática para formularios).
- Posibilita el desarrollo de base de datos que muestra las relaciones entre las personas y los tipos de información (roles).
- Establece protección de seguridad para formularios electrónicos.
- Digitaliza formularios impresos y permite su publicación en la web.
- Crea campos electrónicos automáticamente a partir de los formularios impresos digitalizados.
- Facilita la contestación de formularios en la web.
- Permite la realización automática de cálculos.

La funcionalidad del formulario se debe ampliar debido a que se crean automáticamente los nombres de los campos de los formularios que se digitalizan y que se puede modificar y publicar fácilmente en la web, donde se pueden almacenar hasta 10,000 registros en un solo sitio web; este es un servicio que tiene una ventaja dentro del comercio electrónico; por otra parte, los empleados tienen fácil acceso a los formularios de la compañía sin ningún problema, se puede aplicar al diseño de pantallas y al diseño de sitios y páginas. Hay diferencias principales sobre los formularios electrónicos, los destinados a la web y los estáticos, lo cual se debe a los diseñadores que pueden agregar o ayudar específicamente dependiendo del contexto en que se encuentre el formulario.

Los formularios se crean para poder satisfacer uno o más para su registro, el procesamiento, el almacenamiento y la recuperación de la información de las organizaciones. Durante la creación de formularios es conveniente dar la información adecuada a los demás departamentos o usuarios, aquí es donde los formularios especializados son útiles dentro de las empresas para poder tener una mejor información.

Diseño de formularios por computadora

Los formularios son creados con el *software OmniForm de ScanSoft*. Este tipo de *software* es de gran utilidad para los analistas y los usuarios que buscan automatizar rápidamente los procesos de negocios, los formularios impresos pueden digitalizarse y después publicar en la web. El analista o usuario puede usar un conjunto de herramientas en la cual prepara los campos, casillas de verificación, líneas, cuadros y varias características más.

A continuación, se muestra un ejemplo de formulario realizado por computadora:

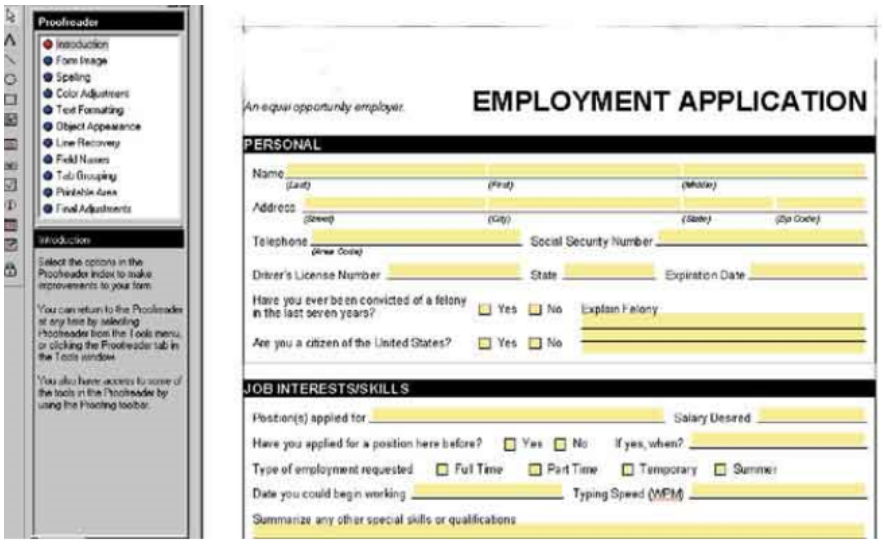


Figura 3.4. Ejemplo de formulario por computadora. Fuente: w3computing (s/f) "*Good Form Design*" Recuperado el 25 de marzo de 2019 de: https://www.w3computing.com/systemsanalysis/good-form-design/

OmniForm de ScanSoft permite al usuario tomar un formulario existente, digitalizarlo en la computadora y definir campos con el fin de que el formulario se pueda contestar fácilmente en una PC.

3.3. Diseño de archivos

El almacenamiento de información es considerado importante en los sistemas, los objetivos de la organización del diseño de archivos son los siguientes.

- **Integridad de los datos.**
- **Disponibilidad de los datos.**
- **Actualización y recuperación eficiente.**
- **Almacenamientos de datos eficiente.**
- **Recuperación de información para un propósito.**

Un archivo contiene cierto número de registros que tienen información para la operación, diseño, administración y toma de decisiones en una organización. Los tipos de archivos se describen de la manera siguiente:

Sistema de archivos manuales

Una organización propone sistemas para manejar labores esenciales de negocios. Históricamente, esos sistemas eran manuales, hechos con papel y lápiz. Estaban organizados por sistemas de carpetas de archivos y gabinetes para guardarlas, así mientras un conjunto de datos fuera relativamente pequeño y los financieros de una organización tuvieran pocas necesidades de informarse. El sistema manual cumplía con su función en forma satisfactoria como depósito de datos. No obstante, a medida que las organizaciones crecieron y que las necesidades de informar se hicieron más complejas, dar seguimiento a datos en un sistema manual de archivos se hizo difícil, por tanto, las empresas se enfocaron en el uso de la tecnología informática con las computadoras.

Sistema de archivos computarizados

Inicialmente los archivos computarizados dentro de los sistemas de archivos eran semejantes a los archivos manuales (Coronel, 2011: 11).

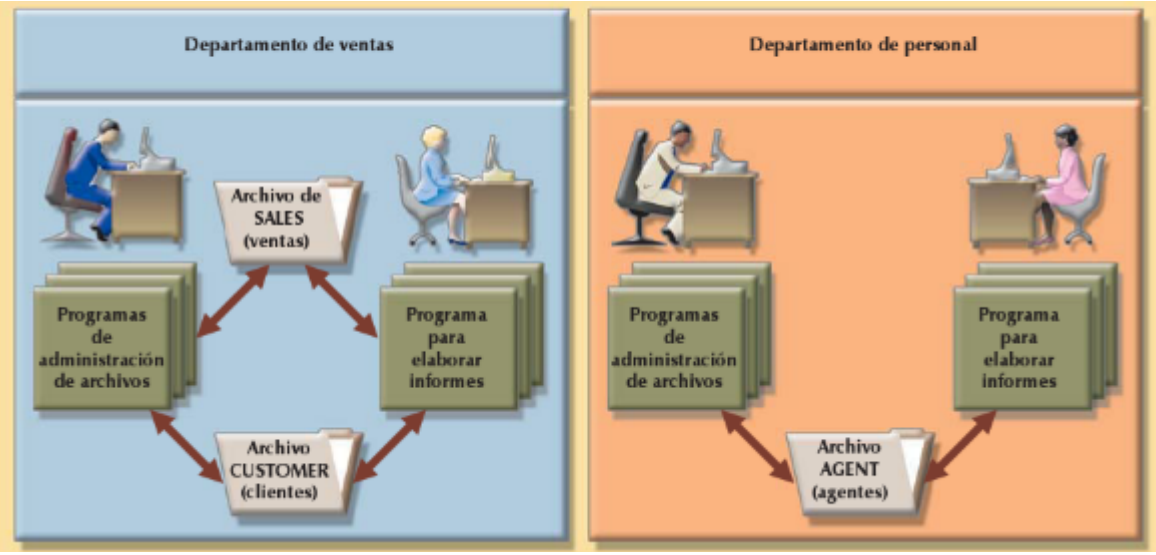


Figura. 3.5. Sistema sencillo de archivos (Coronel, 2011: 13).

Tipos de archivo. Los archivos se utilizan temporalmente para almacenar datos o información en forma indefinida, también se pueden almacenar archivos temporalmente para un propósito específico.

A continuación, se presentan algunos de los tipos más usuales:

Tipo de archivo	Características
Archivos maestros	• Contiene registros para un grupo de entidades. • Los atributos se actualizan constantemente • Los registros son permanentes no cambian. • Son propensos a tener registros grandes que contienen toda la información sobre la entidad de los datos. • Dentro de la base de datos cada registro siempre tiene una clave primaria y varias secundarias. • Se encuentran en una tabla dentro de la base de datos o archivo indexado, en su caso indexado –secuencial. Se almacena el archivo maestro utilizando métodos convencionales para lo que se reserva un área de expansión al final de cada registro. Esta misma área proporciona espacio para agregar campos al registro conforme cambian las necesidades de la empresa u organización. Si un archivo es parte de la estructura de una base de datos, no es necesario el área de expansión.
Archivos de tabla.	Contiene datos usados para calcular más datos o medidas de desempeño. Un ejemplo es una tasa de interés de un préstamo,

	normalmente los archivos de tabla se leen únicamente por *software*.
Archivos de transacción	Se utiliza para realizar cambios que actualizan los archivos maestros y producen informes. Por ejemplo, un archivo maestro de un suscriptor de revista necesita ser actualizado; el archivo de transacción contiene el número de suscriptor y un código de transacción actual.
Archivos de trabajo	En ocasiones se puede ejecutar un programa con mayor eficiencia si se utiliza un archivo de trabajo. Cuando se reordena un archivo para acceder a los registros con mayor rapidez y eficiencia para cierto tipo de proceso.
Archivo de reporte	Cuando se necesita imprimir un informe y no se tiene ninguna impresora disponible porque la impresora está atendiendo otros procesos de impresión, se necesita un archivo de reporte para que pueda enviar la salida a un archivo en lugar de enviarlo a la impresora, a esto se le llama *spooling*. Después, cuando el dispositivo está listo, el documento se puede imprimir sin ningún problema. Los archivos de reporte son muy eficientes y útiles para los usuarios, porque éstos pueden tomar los archivos de otros sistemas de cómputo y enviarlos a otros dispositivos especializados como graficadores, impresoras láser, unidades de microficha e incluso

	máquinas de composición tipográfica computarizadas.
Organización secuencial	Dentro de una base de datos, cuando los registros están físicamente en orden dentro de un archivo secuencial que se actualiza, es de suma importancia pasar por el archivo entero, esto se debe a que los registros no se pueden insertar en medio del archivo; por lo regular se copia un archivo secuencial completo durante el proceso de actualización. Los archivos maestros secuenciales se usan cuando el *hardware* lo necesita, (una cinta magnética es un dispositivo secuencial), cuando necesitamos leer o actualizar solo unos registros, es ineficaz usar una estructura secuencial, pero cuando se requiere leer o modificar muchos registros dentro de la base de datos la organización secuencial tendrá sentido; normalmente la organización secuencial se usa para todos tipos de archivos excepto en los archivos maestros.
Listas enlazadas	Cuando los archivos se almacenan en dispositivos de acceso directo tal como disco duro, los tipos opciones se extienden, los registros se pueden ordenar lógicamente, en lugar de físicamente, utilizando listas enlazadas, las listas enlazadas son un conjunto de indicadores para dirigirlo al próximo registro lógico ubicado en cualquier

	parte del archivo.
Organización de un archivo hash.	Estos dispositivos de acceso directo también permiten acceso a un registro dado, yendo directamente a su dirección. Debido a que no es factible reservar una dirección física para cada registro posible, se usa un método llamado *hashing* (reordenamiento). *Hashing* es el proceso de calcular una dirección a partir de la clave del registro.

Tabla. 3.2. Tipos de archivos y sus características. Elaboración propia.

Los archivos que se generan al momento de utilizar las computadoras

Un sistema de archivos (*file system*) es una estructura de directorios con algún tipo de organización, el cual nos permite almacenar, crear y borrar archivos en diferentes formatos. En esta sección se revisarán conceptos importantes relacionados con los sistemas de archivos.

El almacenamiento físico de datos: en un sistema de cómputo es evidente que existe una gran necesidad por parte del usuario y las aplicaciones de almacenar datos por algún medio, en unas ocasiones por un periodo largo y otras por instantes; todo usuario tiene derecho a sus datos e información, a crearlos y eliminarlos o, en su momento, cambiarlos de lugar o carpeta dentro de la computadora para tener así una mejor privacidad respecto a otros usuarios o aplicaciones.

Las operaciones básicas que la mayoría de los sistemas de archivos soportan son:

Crear (*Create*): Permite crear un archivo sin datos, con el propósito de indicar que ese nombre ya está usado y se deben crear las estructuras básicas para soportarlo.

Borrar (*Delete*): Eliminar el archivo y liberar los bloques para su uso posterior.

Abrir (*Open*): Antes de usar un archivo, se debe abrir para que el sistema conozca sus atributos, tales como el dueño, la fecha de modificación, etc.

Cerrar (*close*): Después de realizar todas las operaciones deseadas, el archivo debe cerrarse para asegurar su integridad y liberar recursos de su control en la memoria.

Leer o Escribir (*Read*, *Write*): Añadir información al archivo o leer el carácter o una cadena de caracteres a partir de la posición actual. Concatenar (*append*): Es una forma restringida de la llamada `*write'*, en la cual sólo se permite añadir información al final del archivo. Localizar (*seek*): Para los archivos de acceso directo se permite posicionar el apuntador de lectura o escritura en un registro aleatorio, a veces a partir del inicio o final del archivo.

Renombrar (*Rename*): Permite cambiarle el nombre e incluso, a veces, la posición en la organización de directorios del archivo especificado. Los subsistemas de archivos también proveen un conjunto de llamadas para operar sobre directorios, las más comunes son crear, borrar, abrir, cerrar, renombrar y leer.

3.4. Diseño de interfaces con bases de datos

Bases de datos

Un diseño de base de datos es una fuente central de datos que nos permite compartir información a muchos usuarios para una gran cantidad de aplicaciones, no es únicamente una colección de archivos. La parte más importante de una base de datos la constituye el sistema de administración de base de datos (DBMS, *database management system*), el cual nos permite realizar modificaciones y actualizaciones a la base de datos, así como la recuperación de datos, realizar informes y pantallas. Al usuario encargado de manejar la base de datos y cumplir con los objetivos se le conoce como el administrador de base de datos.

Entre los objetivos de efectividad de la base de datos están los siguientes:

1. Asegurar que los datos se puedan compartir entre los usuarios para una diversidad de aplicaciones.
2. Mantener datos que sean exactos y consistentes.
3. Asegurar que todos los datos requeridos por las aplicaciones actuales y futuras se podrán acceder con facilidad.
4. Permitir a la base de datos evolucionar conforme aumenten las necesidades de los usuarios.
5. Permitir a los usuarios construir su vista personal de los datos sin preocuparse por la forma en que los datos se encuentren almacenados físicamente. (Kenneth E, 2005: 444).

Los objetivos que acabamos de visualizar, nos proporcionan las ventajas y desventajas que tiene una base de datos. Primero, la compartición de los datos

e información, que se debe almacenar una sola vez, no puede estar almacenando varias veces, esto ayuda a conservar la integridad de los datos con mayor facilidad y confiabilidad, debido a que hay cambios constantes.

Cuando el usuario requiere información de una base de datos bien diseñada, es más probable que la información o los datos estén disponibles y más seguros en una base de datos que en un sistema de archivos, debido a que una base de datos es más flexible y segura, va evolucionando conforme a los cambios o necesidad que vaya requiriendo el usuario y las aplicaciones.

La ventaja que tiene al diseñar una buena base de datos es que permite a los usuarios obtener la propia vista de la información o datos, no tienen que preocuparse por una estructura real de base de datos o su almacenamiento físico, los usuarios extraen partes de la base de datos central desde *mainframes* y pueden descargarlas en su computadora, una vez que tienen en sus manos la base de datos, pueden realizar informes, consultas, etc.

Realidad, datos, metadatos

Debemos entender la estructura de los datos por medio de la misma información, por lo tanto, la información que se describe de los datos se le da el nombre de metadatos, en el siguiente esquema, se describe la relación entre la realidad de los datos y metadatos:

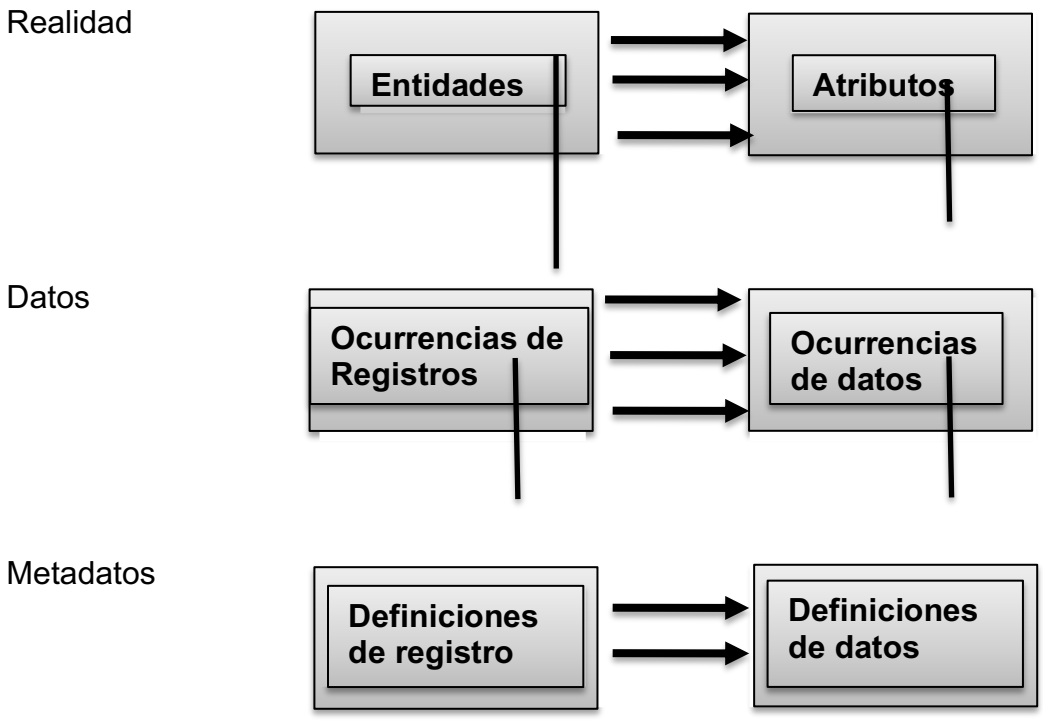


Figura 3.6. Reelaboración con base en Kenneth, 2005:445.

Entidades.

podemos mencionar como un objeto o evento que por medio de este se puede recopilar datos, una entidad puede ser un usuario, lugar o cosa.

Un subtipo es una relación de uno a uno que representa atributos adicionales (campos) dentro de los subtipos de entidades que puedan tener campos nulos almacenados en las tablas de base de datos.

Los diagramas E-R son conexiones de las cuales hay 5 tipos diferentes, como a continuación lo describe Kenneth:

> más una marca corta (I] y un cero (el cual se parece más a un círculo, O] describen una relación uno a cero o uno a uno (sólo cero o uno]. Un cuarto tipo de vínculo para relacionar las entidades se dibuja con una línea recta marcada en el extremo con un cero (O] seguido por una conexión tipo pata de cuervo. Este tipo muestra una relación cero a cero, cero a uno o cero a muchos. Finalmente, una línea recta que vincula las entidades con una conexión tipo pata de cuervo en el extremo describe una relación de más de uno. Una entidad podría tener una relación que la conecte a sí misma. Este tipo de relación se llama relación recursiva; la implicación es que debe haber una forma de vincular un registro de un archivo a otro registro del mismo archivo. Un ejemplo de una relación recursiva se puede encontrar en las simulaciones de HyperCase que se mencionan en los capítulos. Una tarea podría tener una tarea precedente (es decir, una tarea que se debe completar antes de empezar la actual]. En esta situación, un registro (la tarea actual] apunta a otro registro (la tarea precedente] en el mismo archivo. Las relaciones se pueden escribir con palabras en la parte superior o al lado de cada línea de conexión. En realidad, usted ve la relación en una dirección, aunque puede escribir las relaciones en ambos lados de la línea, donde cada una representa el enfoque de una de las dos entidades (2005:448).

En las figuras siguientes se muestran los tipos de relación que pueden existir en el diseño de una base de datos:

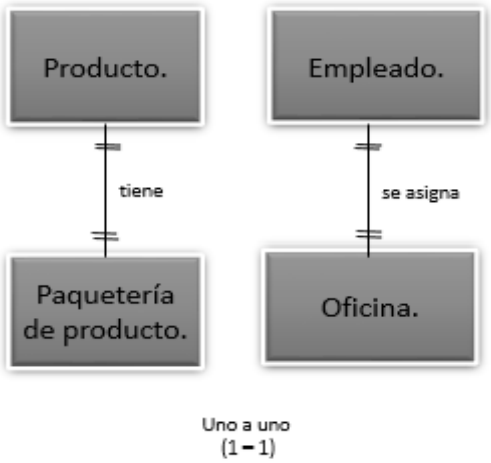


Figura 3.7. Relación 1-1.

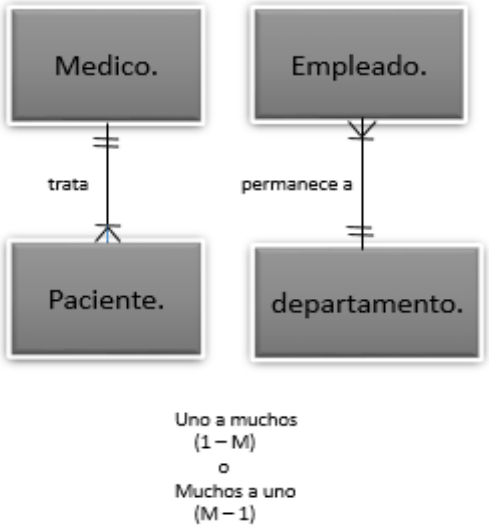


Figura. 3.8. Relación 1-M y M-1.

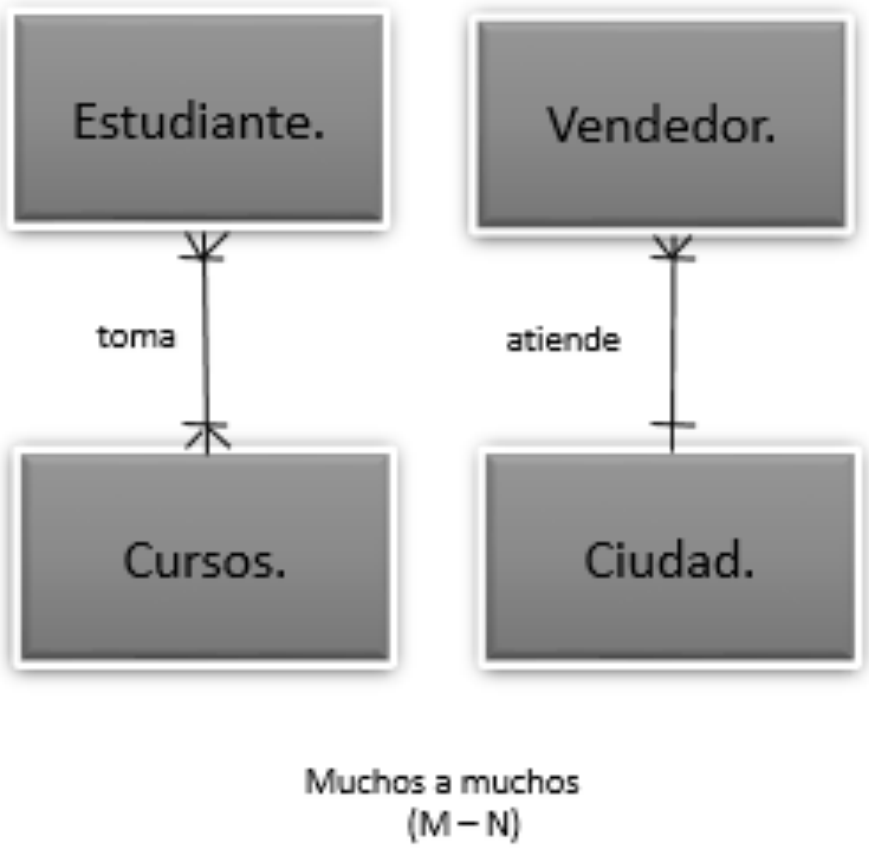


Figura 3.9. Relación M-1. (Kenneth, 2005:446).

Atributos. Un atributo es una característica de una entidad, se pude tener muchos atributos para cada entidad, por ejemplo, un paciente (entidad) puede tener muchos atributos tales como apellido, nombre, calle, ciudad, estado, etc. Cuando los archivos y base de datos se discuten, estos elementos de datos se conocen como datos, estos son las unidades más pequeñas que un archivo o base de datos puede tener.

> Los datos también pueden tener valores dentro de una base de datos, estos valores pueden ser de longitud fija o variable; también pueden ser caracteres alfabéticos, numéricos, especiales o alfanuméricos.

> A veces un dato también se le conoce como campo, sin embargo, un campo representa algo físico, no lógico en la base de datos, por lo tanto, muchos datos se pueden empaquetar en un campo, el campo se puede leer y convertir en varios datos. Un ejemplo común de esto es almacenar la fecha en un solo campo como MM/DD/AAAA. Para ordenar el archivo de acuerdo la fecha, se extraen por separado tres datos del campo y se ordenan primero por AAAA, luego por MM y finalmente por DD. (Kenneth E, 2005:448).

Registro. Es una forma de colección de datos que tiene algo en común con la entidad descrita, es una ilustración de un registro con muchos datos relacionados.

Claves. Es uno de los datos en un registro que se usa para identificar el registro, cuando una clave identifica de forma única un registro, se llama clave primaria. Por ejemplo, # COMPRA puede ser una clave primaria porque cada compra que haga cada cliente se le asigna un solo número. De tal manera se identifica la clave primaria.

Metadatos. Son datos que llegan a definir a los datos en un archivo o base de datos, los metadatos describen el nombre que se ha asignado y la longitud asignada a cada dato y la composición de cada uno de los registros.

Para poder analizar el diseño de base de datos, es necesario comprender las diferencias que hay entre datos e información. Los datos como usualmente los conocemos son elementos sin procesar; esto quiere decir, que no se pueden modificar; por ejemplo, si un lector requiere saber qué opinión tienen los usuarios sobre los servicios que ofrece un laboratorio de cómputo, por lo regular el lector empieza a observar y analizar la actuación del laboratorio y el servicio que presta a estos usuarios, el lector empieza a diseñar las encuestas que los usuarios van a responder y por medio de éste va a ver los resultados sobre dicho servicio.

La información se puede decir que es el resultado de procesar datos sin elaborar para dejar ver su significado. El procesamiento de datos es algo sencillo para organizar patrones que sean complejos, revela un significado, dicha información requiere un contexto, por ejemplo, una lectura de promedio de temperatura de 200 grados.

Administración de base de datos (*DataBase Management System*) DBMS por sus siglas en inglés).
Es un programa que controla el acceso de datos que ya están guardados en una estructura que manejan las bases de datos, es un sistema de administración de base de datos que maneja el contenido de un gabinete electrónico para tenerlo bien organizado.

Una de las funciones de este DBMS es que sirve como intermedio entre el usuario y la base de datos, dentro de la base de datos encontramos archivos en una estructura muy eficiente y la única forma de tener acceso a estos archivos es por medio de un DBMS.

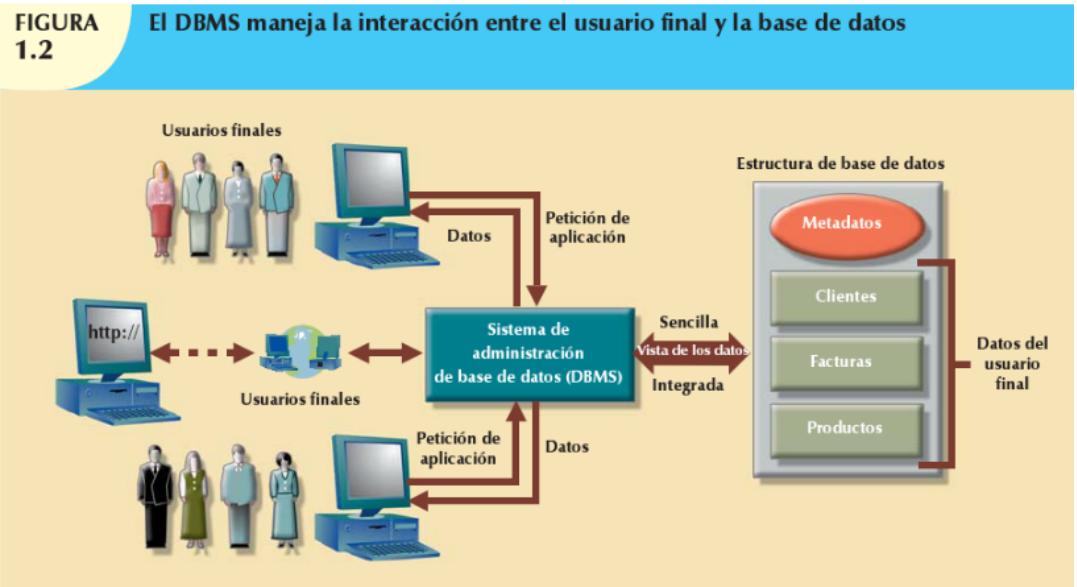


Figura 3.10. Fuente: Coronel, 2011: 8.

El contar con un DBMS entre las aplicaciones del usuario final y una base de datos es una ventaja que tenemos, el primero es que la base de datos puede ser compartida por múltiples aplicaciones y usuarios, el segundo es que podemos tener diferentes tipos de vistas de usuario a la base de datos.

3.5. Diseño de interfaces con otros sistemas

Una interfaz se entiende como un conjunto de elementos que posibilitan al usuario interactuar con el sistema computarizado; se integra de elementos *hardware* y *software*. En este tema se describen los aspectos de la relación humano–computadora. El trabajo se va enfocando al diseño de interfaces humano computadora, a partir de considerar las necesidades del usuario.

El contenido de este tema se refiere al diseño de interfaces, que son el conjunto de elementos que posibilitan la interacción entre seres humanos y las computadoras, tomando en cuenta las necesidades de los usuarios.

En las organizaciones líderes en el desarrollo de computadoras y sistemas, se tiene en cuenta que la interfaz debe ser eficiente en todo sistema, el cual debe ser capaz de proporcionar mecanismos para un uso eficiente y fácil del mismo.

El *software* de la interfaz debe tener variaciones pues los requerimientos de distintas aplicaciones muestran a los usuarios una gama de posibilidades; además de ofrecer portabilidad entre las plataformas y sistemas operativos, desde hace años se ha trabajado y realizando análisis, diseño e implementación de diferentes interfaces caracterizados por su eficiencia e interés hacia los usuarios.

En la actualidad, el desarrollo de interfaces es la base principal para poder ir creciendo, sobre todo el interés principal se centra en el diseño de interfaz gráfico. Este crecimiento es constante y se aplica en diversos sistemas como son los editores de texto y procesador de palabras que normalmente se utilizan en las oficinas y en algunos negocios. Actualmente utilizan programas de dibujo,

publicación elaborada por computadora y diseño de gráficos; correo electrónico, tableros de anuncios, conferencias por medio de computadoras, el procesamiento de imágenes, recuperación y almacenamiento de información, hojas de cálculo, estaciones de trabajo, redes sociales y apps. También en la educación y entretenimiento de tareas, la interfaz gráfica ha tenido un avanzado desarrollo con el uso de internet. La automatización doméstica y el control ambiental, los sistemas comerciales, inventarios personales, control electrónico de utilidad, las herramientas de ingeniería de *software* asistido por computadora, ambientes de programación y herramientas técnicas, entre otros, que permiten ser más rápidos.

Tipos de interfaces

Interfaz de Menú

Proporciona una lista de diferentes opciones al usuario que están disponibles en una pantalla; el usuario principal tiene la capacidad de tener varias opciones desplegadas, pero también está limitado a otras opciones de menú, por ejemplo, dentro de un documento en Word o Excel el usuario principal tiene diferentes opciones para editar, copiar, imprimir, visualizar, etcétera.

Debe tenerse en cuenta que los menús no dependen del *hardware;* sino que ya viene incluido en el *software* y eso se debe a la consistencia del diseño de la interfaz del menú, dentro del diseño del menú también se puede ocultar hasta que el usuario pueda utilizarlo y aplicarlo.

Los menús anidados son útiles para organizar los elementos en pantalla y así el desarrollador pueda anidar nuevos menús dentro de otros para, finalmente, llevar a un usuario a las opciones que presenta un programa. Además, los usuarios únicamente pueden visualizar las opciones que estén interesados en aplicar dentro del programa o documento.

Los menús de GUI se usan para controlar el *software* de PC y tienen las siguientes características:

1. Siempre se despliega la barra de menú principal.
2. El menú principal usa palabras simples para los artículos contenidos. Las opciones principales de menú siempre despliegan menús secundarios.
3. El menú principal debe tener opciones secundarias agrupadas con características similares.
4. Los menús desplegables que se presentan cuando se hace clic en un artículo, con frecuencia consisten en más de una palabra.
5. Estas opciones secundarias desempeñan acciones o despliegan artículos de menús adicionales.
6. Los artículos de menú en gris no están disponibles para la actividad actual. (Kenneth E. 2005: 500).

El usuario tiene la opción de experimentar al utilizar los menús anidados, ya sea por una línea de comandos de línea o simplemente para agilizar más los procesos; además, los usuarios utilizan métodos abreviados o las combinaciones de tecla para poder trabajar en un documento, por ejemplo, en algún procesador de palabras al que se tenga acceso.

Interfaz de Lenguaje Natural

Este tipo de interfaz es uno de los ideales de los usuarios, debido a que la interacción con la computadora se realiza por medio del lenguaje natural.

Este tipo de interfaz genera un problema de programación sumamente exigente y complejo. Estas interfaces normalmente son caras debido a que son más completas y eficientes, existe demasiada demanda para este tipo de lenguajes y

los programadores, desarrolladores e investigadores están trabajando en producir cada vez más este tipo de interfaces; por lo tanto, es un área de crecimiento y es necesaria la supervisión continua para mejorar la calidad de estas interfaces. Algunos sitios web utilizan este tipo de interfaz como, por ejemplo, el sitio *Ask Jeeves*, esta interfaz se caracteriza porque los usuarios introducen la consulta de búsqueda y el sistema responde con una lista de resultados que coinciden con la pregunta que al usuario le interesa.

Interfaz de Pregunta y Respuesta

Este tipo de interfaz trabaja dentro de la computadora y despliega en la pantalla en una pregunta para el usuario, esto quiere decir interactuar, el usuario introduce una respuesta por medio del teclado y la computadora después actúa con esa información de entrada de datos y esto es de acuerdo al programa que se está utilizando, un ejemplo se muestra a continuación en la siguiente imagen:

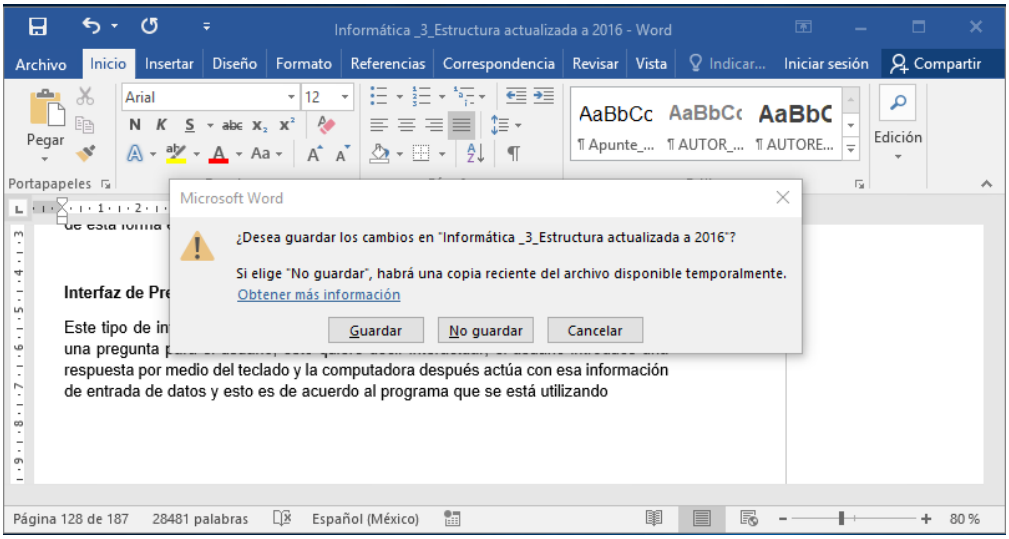


Figura 3.11. Ejemplo de interfaz. Captura de pantalla.

Este tipo de interfaz utiliza un cuadro de diálogo de pregunta y respuesta que permite al usuario elegir opciones; por ejemplo, al momento de instalar un *software* en una computadora, el usuario responde a las preguntas durante el proceso de instalación. Por otra parte, los usuarios que no estén muy familiarizados con aplicaciones de este tipo podrían encontrar interfaces de pregunta y respuesta más accesibles que puedan beneficiarlos al momento de interactuar, ganando confianza al momento de la instalación del *software* al tener éxito en dicha instalación.

Interfaz de Formulario (Formularios de Entrada/Salida)

Este tipo de interfaz está diseñada para trabajar en pantalla por medio de formularios que se puede basar en la web y despliega ciertos campos que contiene datos o parámetros que deben comunicarse con el usuario, a esta técnica de interfaz también se le conoce como el método basado en formularios de entrada/salida.

Los formularios deben configurarse para poder mostrar el tipo de información que se va introducir y en qué parte se agrega.

Estos formularios pueden simplificarse agregando valores predeterminados en los campos y de esta manera posibilitar que el usuario modifique la información predeterminada siempre y cuando sea necesario.

Una de las ventajas que tiene esta interfaz de formulario de entrada/salida es que la versión impresa del formulario que utilicen siempre muestra documentación excelente, muestra etiquetas para cada campo, así como también contextos en las entradas.

Los documentos que se utilizan en la web se pueden enviar al sistema de facturación y si llega a involucrar una transacción, se puede acudir a la base de datos para poder ver si se envía una encuesta. Los formularios que se basan en la

web hacen al usuario responsable de la calidad de los datos e información y pueden hacerlo disponible para poder completarlo y enviarlo en 24 horas, 7 días de la semana desde cualquier punto del mundo.

Una de las desventajas con los formularios de entrada/salida es que los usuarios que ya saben utilizar este tipo de formulario pueden tener poca paciencia para enviar la información y querrán formas más eficientes para la introducción de datos.

Interfaces de lenguaje de comandos

Este tipo de interfaces, permite al usuario llevar a cabo el control de la aplicación por medio del teclado, comandos, frases o alguna secuencia de estos tres métodos.

En el lenguaje de comandos no hay un significado inherente para el usuario, simplemente el hecho de utilizar comandos en una interfaz hace diferente a las demás interfaces. Este tipo de lenguajes de comando trabaja manipulando a la computadora como una herramienta que permite que los usuarios lleven el control del diálogo. Ofrece mayor flexibilidad y control cuando el usuario da las instrucciones a la computadora por medio del lenguaje de comandos, las cuales se ejecutan de inmediato por el sistema.

Las características del diseño de un *software*, se basan en tener mayor la capacidad y menor tiempo en el funcionamiento; así como los métodos y procesos en el desarrollo de los modelos y mantenimiento específicos. De acuerdo con lo anterior, se diseñan las interfaces para el usuario final con el propósito de facilitar su uso.

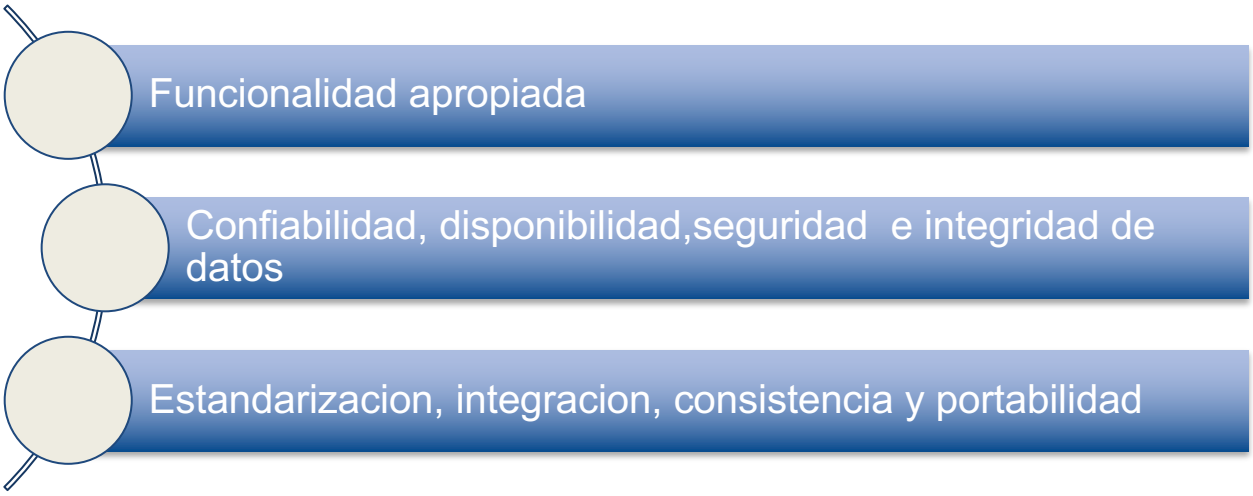


Figura. 3.12. Características de las interfaces. Elaboración propia.

Funcionalidad apropiada

Es preciso comprobar la funcionalidad necesaria de las tareas y sub-tareas que deben realizarse y efectuarse; la tarea de reparación para arreglar errores en el uso del sistema es complicado de descubrir; el sistema con la funcionalidad inadecuada frustra al usuario y hay frecuentemente rechazo o es subutilizado, si la funcionalidad es adecuada, no importa cuál interfaz sea la adecuada, simplemente se diseña de manera que el usuario se adapte al sistema. También podemos mencionar que la funcionalidad excesiva es riesgosa y es una simple equivocación de los diseñadores debido a que la confusión y la complejidad hacen que la implementación, el mantenimiento, el aprendizaje y el uso sea más difícil.

Confiabilidad, disponibilidad, seguridad e integridad de datos

Se debe tener segura la confiabilidad del sistema, es decir los comandos que se deben desarrollar para que funcione de manera correcta; tanto la visualización de la información o datos debe reflejar los contenidos que se encuentran en la base de datos, así como las actualizaciones se deben realizar de manera correcta.

En cuanto a la arquitectura del *software* y los componentes del *hardware,* si el sistema no está bien desarrollado, mostrará errores. Para el buen funcionamiento del sistema, se debe tener previsto la seguridad, privacidad e integridad de los datos; la protección debe tenerse en cuenta, por ejemplo, para los accesos indeseados, la destrucción de datos o alteraciones maliciosas, etc.

Estandarización, integración, consistencia y portabilidad

En cuanto a la estandarización, se refiere a las características comunes de la interfaz; esto es, a través de múltiples aplicaciones. El ambiente UNIX es un lenguaje de comandos que nos permite trabajar de una manera eficiente; asimismo, UNIX es un líder de estandarización de formatos de datos para poder integrar números, programas de aplicación y herramientas de *software*, también nos permite una portabilidad a través de diferentes plataformas de *hardware*.

Durante el proceso de integración de un sistema, se toman los sub sistemas desarrollados de forma independiente y se conjuntan para crear el sistema completo; además, la integración se puede realizar reuniendo todos los subsistemas al mismo tiempo; para efectos técnicos y administrativos, es una de las mejores formas de integración creciente donde los sistemas se integran uno a uno por las siguientes razones:

1. En lo general, es imposible confeccionar una agenda para el desarrollo de todos los sistemas, de tal forma que todos terminen al mismo tiempo.
2. La integración creciente reduce el costo en la localización de errores; si varios subsistemas se integran simultáneamente, un error que surja durante las pruebas de sistema pueda estar en cualquiera de estos subsistemas; de otra manera se puede ver cuando un único subsistema se integra en un sistema de funcionamiento, los errores que se produzcan estarán

probablemente en el subsistema recién integrado o en las interacciones entre los subsistemas existentes y el nuevo sistema.

Por *consistencia* nos referimos a las acciones comunes de sucesiones en el cual los esquemas, color, tipografía, llamadas, etc., dentro de un programa de aplicación, deben tener compatibilidad a través de un rol de programa de aplicación o de sistema.

En cuanto a la portabilidad, se refiere al proceso de convertir datos y compartir la interfaz del usuario a través de diferentes ambientes de *software* y *hardware*.

3.6. Documentación técnica y de usuario

La documentación técnica y de usuario es la manera de construir documentos flexibles que ayuden a describir todas las herramientas y aplicaciones del desarrollo de *software*; de esta manera se pueden realizar soluciones eficientes con la información adecuada.

Las técnicas de documentación son auxiliares para ordenar y describir claramente la información, existen 6 tipos de técnicas de documentación que están dirigidas al entorno administrativo y otras a negocios; hay otras que están dirigidas a entornos más especializados o técnicos.

En la actualidad, las técnicas de documentación se deben dirigir al usuario especializado o al operador. Para estar seguros de tener un *software* bien desarrollado, es preciso tener bien organizados y actualizados los documentos de su desarrollo, dependiendo de las necesidades que el manejo del sistema va a resolver, así como de la información de las organizaciones que lo solicitan.

En este tema también es importante y necesario hablar de calidad, en particular dentro de un ámbito técnico-operativo de cualquier organización de las normas ISO9000; las cuales tienen como principal objetivo crear una cultura de calidad.

La técnica de documentación propuesta es una fiel muestra de trabajo de calidad e independientemente de que un sistema esté certificado con alguna norma ISO9000, se complementa con lo ya existente y puede inclusive ser suficiente para un sistema de aplicaciones, acorde con características de estandarización y eficacia dentro de la organización en que se desenvuelve (Guijosa J, 2001: 33).

Definición de ISO

Es una palabra de deriva del Griego "*isos*" que significa "igual" o "estándar", curiosamente la palabra se convirtió en el nombre de la *International Standard Organization* (ISO), es decir, la Organización Internacional de Estandarización. Esta organización es una federación de alcance mundial integrada por cuerpos de estandarización nacionales de 164 países[17]. Su objetivo es promover o dar conocer mundialmente el desarrollo de normas con el fin de mejorar la eficiencia en la manufactura, el comercio, la comunicación, la operación, la productividad, la calidad y la reducción de costos en las organizaciones de todo el mundo, para facilitar el intercambio internacional de bienes y servicios. Asimismo, se busca la cooperación de participantes en la esfera de lo intelectual, científico, tecnológico y económico.

De las normas ISO existen numerosas guías, cuatro normativas ISO9000 y una de ISO14000 de especificaciones; dependiendo al área se selecciona aquella normativa que cobra alcance de los productos, servicios y operaciones.

- ISO9001. Modelo para el aseguramiento de la calidad en el diseño y desarrollo.
- ISO9002. Modelo para el aseguramiento en calidad en producción e instalación, establecimiento, la prevención, detención y corrección de problema durante la producción e instalación.
- Iso9003 Modelo para el aseguramiento en calidad en la inspección final y prueba.
- ISO9004. Es una guía para sistemas de aseguramiento de calidad.

[17] Fuente: ISO (s/f) "*ISO: a global network of national standards bodies*" Recuperado el 25 de marzo de 2019 de: **https://www.iso.org/members.html**

- ISO14000. Serie de normas para la gestión ambiental que provee el marco de trabajo a los negocios para demostrar su compromiso con la responsabilidad ambiental.

Las técnicas de documentación son de suma importancia para las organizaciones; ya que deben contar con una documentación flexible que describa los procesos de las aplicaciones del *software*, para poder resolver los problemas al momento que se vayan dando, por lo que es conveniente que se revise a detalle la documentación técnica.

Técnicas de documentación.

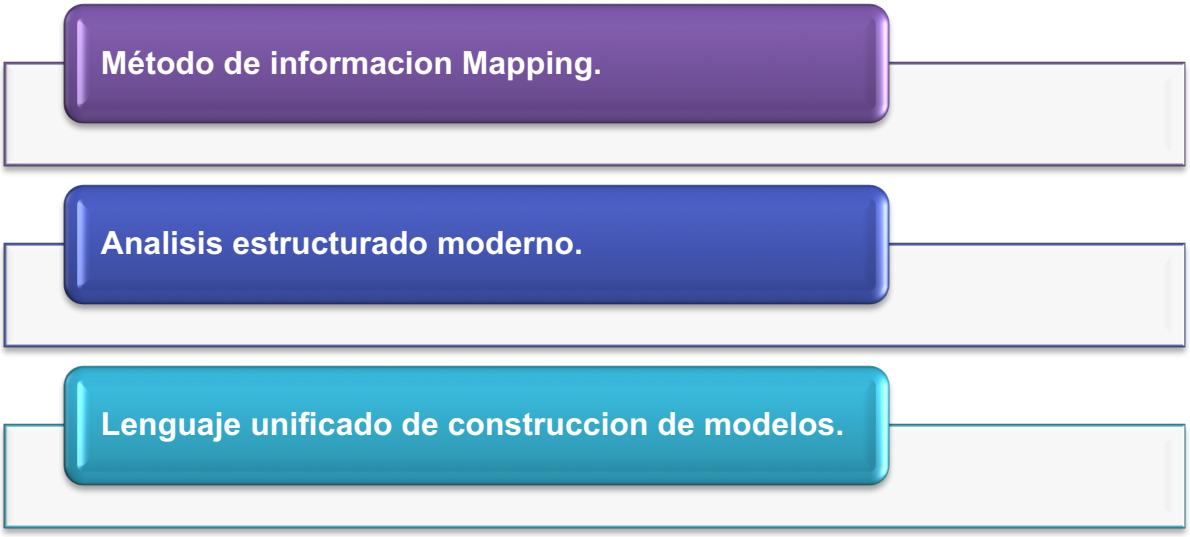


Figura. 3.13.Técnicas de documentación. Elaboración propia basado en Guijosa, 2001: 24.

Mapping. tiene un método para enseñar a pensar y escribir documentación de negocios; algunas de sus aplicaciones son métodos, entrenamientos y políticas. La documentación técnica tiene una forma de analizar y organizar ciertas cosas como pensar. Además, se trata de una herramienta que sea adecuada y flexible

para que se ajuste a las necesidades de los usuarios, teniendo un objetivo en común.

Hom señala:

> "El método de información Mapping permite al escritor integrar el uso que el lector le dará al material. (…) Se concentra en dar a la información la secuencia adecuada al uso del lector y en incluir los detalles que el lector necesita". (Guijosa, 2001: 26).

Al desarrollar cualquier documento es preciso detallar los procesos de la información con claridad y flexibilidad, pues es fundamental para cualquier usuario que la aplicación del sistema se beneficie y realice las actividades con un desempeño eficiente.

El análisis estructurado moderno. Contribuye al desarrollo de la documentación para las aplicaciones en los procesos de los modelos de la organización, de esta manera nos permite tener una mejor comunicación con el usuario de forma enfocada.

El análisis estructurado moderno. Contribuye a la mejora de la documentación para las aplicaciones en los procesos de los modelos de la organización, de esta manera nos permite tener una mejor comunicación con el usuario de manera personalizada; y así, se pueden conocer los requerimientos que tiene el usuario, ya sean correctos o incorrectos, de tal modo que se pueda modificar el sistema o reiniciarlo por completo, si fuera necesario, por lo mismo, es preciso contar con una documentación adecuada que registre todo el desarrollo.

El lenguaje unificado de construcción de modelos surgió como una notación estándar de la construcción de modelos; por esa razón será de gran utilidad, además, se hace hincapié en la estandarización de la notación dentro de la representación y esto es fundamental dentro del ambiente orientado a objetos.

La información Mapping, se caracteriza por los momentos que se esquematizan en la siguiente figura:

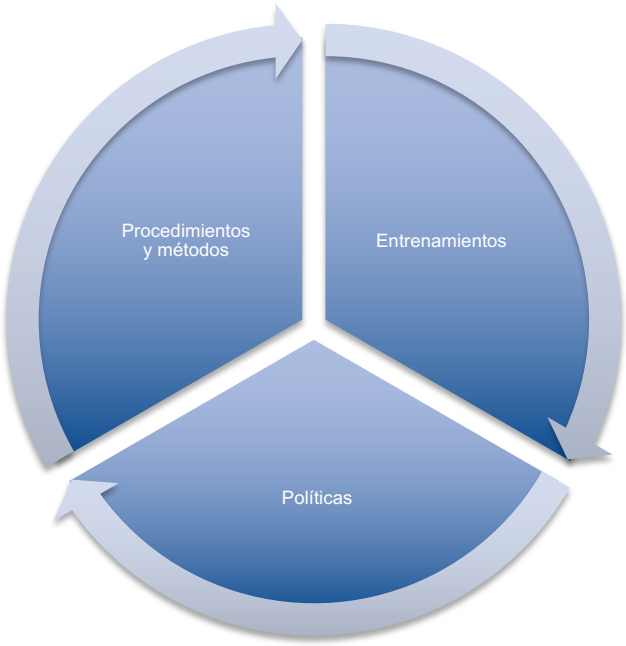


Figura. 3.14. Información maping. Elaboración propia basada en Guijosa, 2001: 27.

El método proporciona habilidades para analizar y organizar información, sus objetivos son:

- Analizar, organizar y presentar claramente la información.
- Elaborar documentos que tengan las siguientes características:

I. Que se lean con facilidad y se entienda el contenido.
II. Que facilite la información clave al momento de la búsqueda.
III. Que ayuden a las organizaciones a aumentar su productividad.
IV. Que mejoren la capacidad para la toma de decisiones.

Mapping, está dirigido a las personas que diseñan métodos y procesamientos, materiales técnicos de referencia y documentos de proyectos, todo esto se describe para cualquier programa.

A continuación, se enumeran los siete principios del método Mapping, los cuales son llamados de comunicación:

Figura. 3.15. Principios del método *maping*. Elaboración propia basado en Guijosa, 2001: 28.

Los elementos que se acaban de mencionar funcionan de manera flexible y ágil, para poder entender estas características es necesario analizar cada uno de ellos.

I. Fragmentación

Los documentos deben operar la información en ideas pequeñas y manejables, lo que depende de las investigaciones que se realicen durante las unidades de información.

II. Relevancia

Los documentos deben contener una idea dentro del mismo documento, se debe colocar la información adecuada.

III. Etiquetado

El documento debe tener etiquetas para cada unidad de la información, ubicando la información clave, por ejemplo:

Módulo I. Organización.

Módulo II. Documentación.

Módulo III. Planeación.

IV. Consistencia

El documento debe tener palabras precisas para describir la información, así como también etiquetas, formato, organización y secuencias semejantes para los temas.

V. Gráficas integradas

El documento, cuando se va a desarrollar, debe incluir diagramas, tablas, ilustración, etc. Esto se debe usar como parte del documento, no como complemento; es decir, que las gráficas que se agregan son parte de la información que se está ingresando y no un elemento más del documento.

VI. Detalle accesible.

Este tipo de documento debe ir detallado, de modo que cuando el lector lo necesite el documento sea accesible, y pueda ver la información adecuada y ser útil para cualquier persona.

VII. Jerarquía de fragmentos y etiquetas

Al momento de realizar el documento, éste debe tener elementos relevantes de información debidamente jerarquizada y etiquetada, de esta manera, el lector tendrá la oportunidad de tener una lectura más flexible y se podrá utilizar en cualquier aplicación que sea necesario; de tal manera que no debe de ser muy complejo, para que el usuario tenga la oportunidad de leer sin problemas. Otra característica que tiene el documento, es tenerlo disponible y accesible para cualquier usuario de la aplicación.

Con relación a lo anterior, Guijosa expone lo siguiente:

> El documento a desarrollar, con la técnica de documentación propuesta, se puede empezar a escribir de una aplicación que está en producción o planteado conjuramente con el análisis y diseño orientado a objetos; sin embargo aunque no se establece un momento para empezar a describirlo, precisamente por la flexibilidad que se supone es características de la técnica de documentación; es recomendable planear con tiempo el desarrollo del mismo y no cuando sea demasiado tarde y se encuentre la solución desbordada por los continuos cambios que se requieren en momentos críticos (Guijosa, 2001: 71).

3.7. Pruebas del sistema

El proceso de pruebas

El proceso de pruebas consiste en un conjunto de actividades que se realizan para identificar las posibles fallas en el funcionamiento, configuración y uso de un *software* o, en general, en un sistema informático ya desarrollado. Este proceso de pruebas se realiza en las empresas y organizaciones e incluye técnicas de diseño.

Todo proceso de prueba debe estar relacionado entre el *software* y su vinculación con el aseguramiento de la calidad a diversos tipos de sistemas de pequeña y gran escala.

Para fines demostrativos y explicativos de este tema, se muestran a continuación unas tablas con los criterios para realizar el análisis de pruebas relacionadas[18].

18 Si deseas leer y complementar lo revisado en este tema, se te sugiere leer todo el artículo disponible en el vínculo de la referencia que se encuentra al final de las tablas.

TABLA 1

CRITERIOS DEFINIDOS PARA EL ANÁLISIS DE LAS PROPUESTAS RELACIONADAS

Criterio	Definición
Tipo	Alcance: 1. Procesos de todo el ciclo de vida del *software* 2. Proceso exclusivamente de pruebas 3. Mejora de los procesos de todo el ciclo de vida del *software* 4. Mejora del proceso de pruebas 5. Pruebas de Software
Licencia	Propietario (P): Una propuesta es de tipo propietario si el usuario tiene limitaciones para usarla, modificarla o redistribuirla y requiere permiso de una organización privada. Abierto (A): Una propuesta es de tipo abierto si el usuario NO tiene limitaciones para usarla o modificarla y se puede redistribuir libremente.
Exclusivo de pruebas	Una propuesta es exclusiva de pruebas si se enfoca hacia todas las prácticas relativas a pruebas.
Completamente Definida	Una propuesta está completamente definida si todas las prácticas descritas en la misma están completamente especificadas.
Incorpora actividades	Una propuesta que define un proceso que contiene un conjunto de tareas definidas, que se llevan a cabo para cumplir los objetivos fijados.
Incorpora tareas	Son el nivel de detalle con que se desglosan las actividades.
Incorpora técnicas de pruebas	La propuesta incluye procedimientos técnicos y de gestión que ayudan a desarrollar las actividades planteadas.
Adecuada para VSEs	La propuesta es adecuada a VSEs si su implantación en este tipo de organizaciones es relativamente sencilla.

Tabla. 3.3. Criterios definidos para el análisis de las propuestas relacionadas.

TABLA 2
EVALUACIÓN DE CRITERIOS

Criterios / *Propuestas*	Tipo	Licencia	Especifica de pruebas	Completamente Definida	Incorpora Actividades	Incorpora Tareas	Incorpora Técnicas de pruebas	Adecuada para VSE'S
ISO/IEC 12207[13]	Procesos de todo el ciclo de vida del *software*	A	NO	SÍ	SÍ	SÍ	NO	NO
CMMI[14]	Mejora de los procesos de todo el ciclo de vida del *software*	P	NO	SÍ	SÍ	SÍ	NO	NO
Competisoft[15]	Mejora de los procesos de todo el ciclo de vida del *software*	P	NO	SÍ	SÍ	SÍ	NO	SÍ
ISO/IEC 29119[16]	Pruebas de Software	A	SÍ	NO	SÍ	SÍ	SÍ	NO
ISO/IEC 29110[4]	Procesos de todo el ciclo de vida del *software* de las VSEs	A	NO	SÍ	SÍ	SÍ	NO	SÍ
TMMI[17]	Mejora del proceso de pruebas	P	SÍ	SÍ	SÍ	NO	NO	NO
TPI[18]	Mejora del proceso de pruebas	P	SÍ	SÍ	SÍ	SÍ	NO	NO
Propuesta para el testeo de PYMES[19]	Mejora del proceso de pruebas	A	SÍ	NO	SÍ	NO	NO	SÍ
Minimal Test Practice Framework[20]	Mejora del proceso de pruebas	P	SÍ	NO	SÍ	SÍ	NO	SÍ
ISO/IEC 29119 Parte 2[21]	Proceso exclusivamente de Pruebas	A	SÍ	NO	SÍ	SÍ	NO	NO
Software Qualification Testing Process[13]	Proceso exclusivamente de pruebas	A	SÍ	SÍ	SÍ	SÍ	NO	NO
Proceso de pruebas de software para el modelo de referencia de Competisoft [22]	Proceso exclusivamente de pruebas	P	SÍ	SÍ	SÍ	SÍ	NO	SÍ
TestPAI[23]	Mejora del proceso de pruebas	P	SÍ	SÍ	SÍ	SÍ	NO	SÍ
Proceso de pruebas para pequeñas empresas: En un escenario brasileño [24]	Proceso exclusivamente de pruebas	P	SÍ	SÍ	SÍ	SÍ	NO	SÍ
Nuestra propuesta	Proceso exclusivamente de pruebas	A	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ

Tabla. 3.4. Evolución de criterios. Fuente de las dos tablas:

M.L. Rojas-Montes, F.J. Pino-Correa & J.M. Martínez, "Proceso de pruebas para pequeñas organizaciones desarrolladoras de software", Fac. Ing., vol. 24 (39), pp. 55-70, mayo-ago. 2015. Fecha de Recepción: 01 de Febrero de 2015 Fecha de Aceptación: 10 de Abril de 2015.
Disponible en:
https://revistas.uptc.edu.co/index.php/ingenieria/article/view/3551/4326

Para realizar una prueba de *software* se requiere considerar los siguientes puntos:

- Deberán realizarse constantemente revisiones técnicas con la finalidad de eliminar los errores que se puedan presentar.
- De acuerdo con el procedimiento, se sugiere comenzar con los componentes y operar hacia afuera, de modo que se integre todo el sistema de cómputo.
- Es conveniente que se utilicen diferentes técnicas de prueba adecuadas para distintos enfoques de ingeniería de *software* y, de preferencia, en diferentes momentos del proceso.
- Para hacer las pruebas, es preciso que las realice el desarrollador del sistema.
- Se sugiere realizar siempre la depuración en cualquier tipo de prueba.

Una estrategia para la prueba de *software* debe incluir tanto pruebas de bajo nivel, que son necesarias para verificar que un pequeño segmento de código fuente se implementó correctamente; como también pruebas de alto nivel, que validan las principales funciones del sistema, a partir de los requerimientos del cliente.

Una estrategia debe proporcionar una guía para el profesional y un conjunto de guías para el jefe de proyecto. Puesto que los pasos de la estrategia de prueba ocurren cuando aumenta la presión por las fechas límite para finalizar, el avance debe ser medible y los problemas deben salir a la superficie tan pronto como sea posible (Pressman, 2010: 384.).

Verificación y validación de *software*

La verificación se refiere al conjunto de tareas que garantizan que el *software* realice de manera correcta una función determinada, en tanto que
la validación es un conjunto diferente de tareas que aseguran que el *software* desarrollado sigue los requerimientos del cliente.

La verificación y la validación incluyen un amplio arreglo de actividades SQA: revisiones técnicas, auditorías de calidad y configuración, monitorización de rendimiento, simulación, estudio de factibilidad, revisión de documentación, revisión de base de datos, análisis de algoritmos, pruebas de desarrollo, pruebas de usabilidad, pruebas de calificación, pruebas de aceptación y pruebas de instalación. Aunque las pruebas juegan aquí un papel extremadamente importante, también son necesarias muchas otras actividades.

La calidad se incorpora en el *software* a lo largo de todo el proceso de ingeniería del *software*. La adecuada aplicación de métodos y herramientas, revisiones técnicas efectivas, y gestión y medición sólidas conducen a la calidad que se confirma durante las pruebas.

Organización de las pruebas de *software*

En todo desarrollo de *software* hay conflictos mientras se realizan las pruebas, en la actualidad a los desarrolladores de *software* se les solicita que ellos mismo realicen las pruebas para poder detectar los errores debido a que ellos son lo que conocen todo el sistema y su funcionamiento.

El desarrollador de *software* es el responsable directo de llevar a cabo las pruebas individuales del programa y debe asegurarse de que cada uno desempeñe la función adecuada; también se efectúa la prueba de integración, en la que hay una etapa de prueba que conduce a la construcción de la arquitectura completa del *software*; se involucra un grupo de prueba independiente (GPI por sus siglas).

El desempeño que realiza el grupo de prueba independiente es ver los errores del *software*; el desarrollador y dicho grupo trabajan juntos a lo largo del proyecto de *software* para garantizar que realmente se hagan las pruebas adecuadas para corregir los errores de inmediato. El GPI es parte del equipo del proyecto de un desarrollador de *software* y se involucra en el análisis y diseño; otra de sus funciones es que reporta a las organizaciones la calidad del *software*.

De acuerdo con la norma ISO-9001-2000[19], se deben considerar los siguientes elementos en todo desarrollo de sistemas:

1. EL cliente con los requisitos y la satisfacción.

2. Elaboración del sistema.

3. Las características del sistema deben considerar la calidad del proceso, calidad tanto interna como externa, calidad de uso.

19 Para complementar, se enfatiza que la ISO-9001-2000 es una norma internacional cuya función es analizar el sistema de gestión de calidad en cuanto a la responsabilidad de la dirección, la gestión de recursos, elaboración del producto, mediciones, evaluación, análisis y mejora del mismo.

4. Fases de prueba que considera la política, estrategias, planificación, seguimiento y control, diseño de las pruebas funcionales, así como el entorno de las pruebas.

Observa las siguientes figuras:

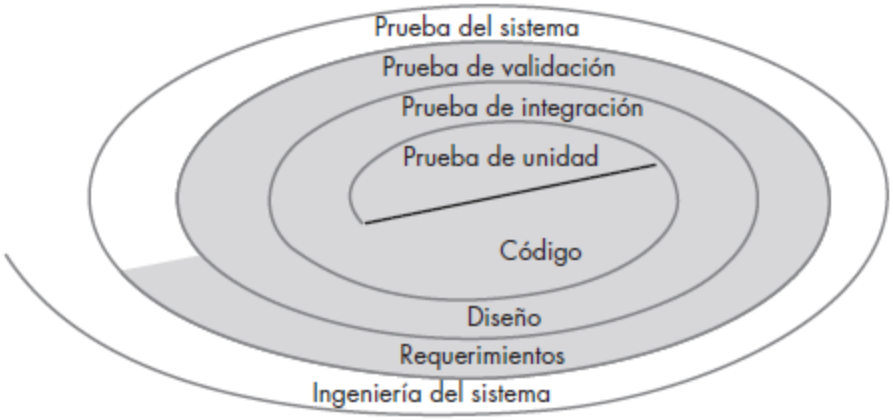


Figura 3.16. Fuente: Pressman, 2010: 386.

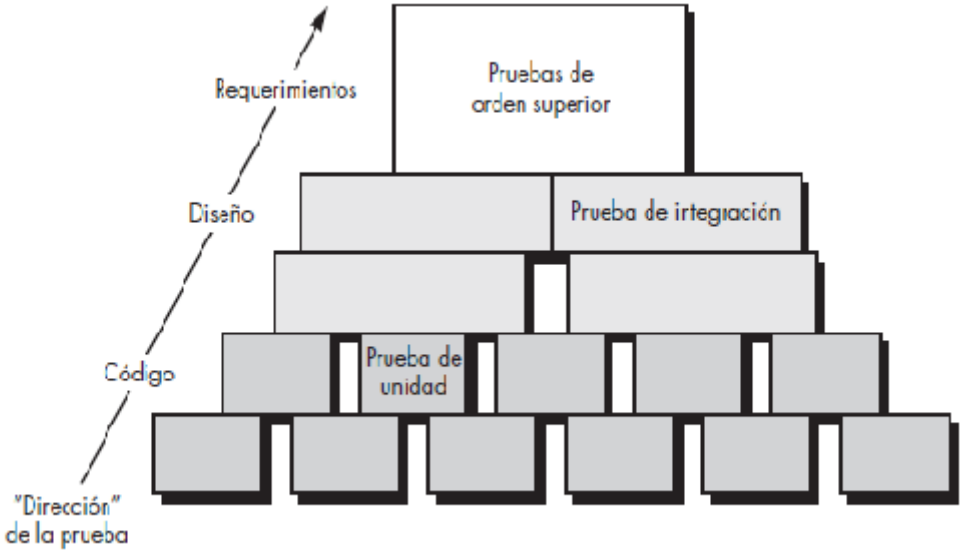


Figura 3.17. Fuente: Pressman, 2010: 387.

La figura **a**, está enfocada a los modelos TMM y TMMI para determinar la calidad del producto. Estas dos tienen los cinco niveles de madurez, todas las pruebas se realizan de manera formal, se realiza un seguimiento y control de lo planificado para tomar correcciones y se verifica si el producto cumple con los requisitos.

Sus procesos son política y estrategia, planificación, seguimiento y control, diseño (pruebas funcionales), ejecución y entorno de pruebas. Los procesos de verificación y validación (actividad o etapa que pertenece al proceso de construcción del *software*) son muy importantes (distintos el uno del otro) para determinar la calidad de un producto.

Sobre la base de los antecedentes, en este modelo se articularon los estándares de calidad ISO-9001-2000 e ISO-9126 y los modelos TMMI y TMM; posteriormente, se realizó un diagnóstico del nivel de TMMI que la empresa tiene, a fin de conocer el nivel en el que se encuentra, para luego presentar una

propuesta de un modelo formal de pruebas que permita alcanzar el nivel de madurez integrado 2 de TMMI.

Estrategia de prueba del *software*. Visión general

Durante el proceso de *software* se lleva a cabo el análisis de requerimientos del mismo, en donde se deben establecer los criterios de dominio, función, comportamiento, desempeño, restricciones y validación de información para el *software*; durante el avance que se va dando, se llega al diseño y al final a la codificación, para poder desarrollar un *software* de computadoras se debe avanzar en espiral hacia adentro.

En el proceso de prueba se tienen en cuenta varios procedimientos comenzando por la prueba de unidad, posteriormente la prueba de integración, le sigue la prueba de validación y finalmente la prueba del sistema, como se muestra en la siguiente figura:

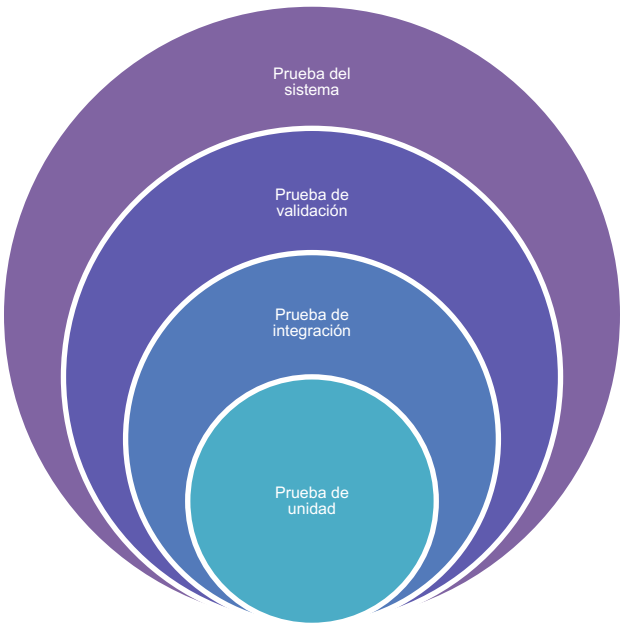


Figura 3.18. Proceso de prueba. Elaboración propia.

Para probar el *software* de cómputo, se avanza en espiral hacia afuera en dirección de las manecillas del reloj, a lo largo de las líneas que alcanzan las pruebas con cada vuelta.

Esta prueba se enfoca en entradas y salidas, aunque también pueden usarse técnicas que ejercitan rutas de programa específicas para asegurar la cobertura de las principales rutas de control. Después de integrar (construir) el *software*, se realiza una serie de pruebas de orden superior.

Deben evaluarse criterios de validación (establecidos durante el análisis de requerimientos). La prueba de validación proporciona la garantía final de que el *software* cumple con todos los requerimientos informativos, funcionales, de comportamiento y de rendimiento.

El último paso de la prueba de orden superior cae fuera de las fronteras de la ingeniería de *software* y en el contexto más amplio de la ingeniería de sistemas de cómputo. El *software*, una vez validado, debe combinarse con otros elementos del sistema (por ejemplo, *hardware*, personal, bases de datos, etc.). La prueba del sistema, verifica que todos los elementos se mezclan de manera adecuada y se logra el funcionamiento/rendimiento global del sistema.

3.8. Liberación

La liberación del *software* se hace cuando ha sido terminado o actualizado de manera completa y ha sido distribuido para su uso en las organizaciones.

Etapas de liberación de *software*[20]

- **Alfa:** es una versión preliminar en donde el equipo desarrolló el *software* con todas las funcionalidades y requerimientos básicos.
- **Beta:** consiste en tener la primera versión completa del *software* desarrollado, por lo general, se realizan pruebas y se pone a disposición del público usuario para encontrar los errores que vayan detectando y así poder mejorarlo.
- **Candidata a versión definitiva:** consiste en la versión del *software* previa a la versión final, en esta versión se detectan los últimos errores.
- **Dorada:** Es la versión del *software* que se destina al usuario final, libre de errores.

20 La liberación de un *software* sucede cuando ya está listo para utilizarlo porque ha cumplido todas las fases y requisitos para ser liberado o autorizado para su uso.

FASES DE LIBERACIÓN DEL SOFTWARE

Para liberar un *software*, es recomendable seguir el análisis y estudio de los componentes que lo integran, como son los desarrolladores, licencias, la preparación del código para distribución, etcétera.
Para la liberación se deben tener en cuenta tres fases:

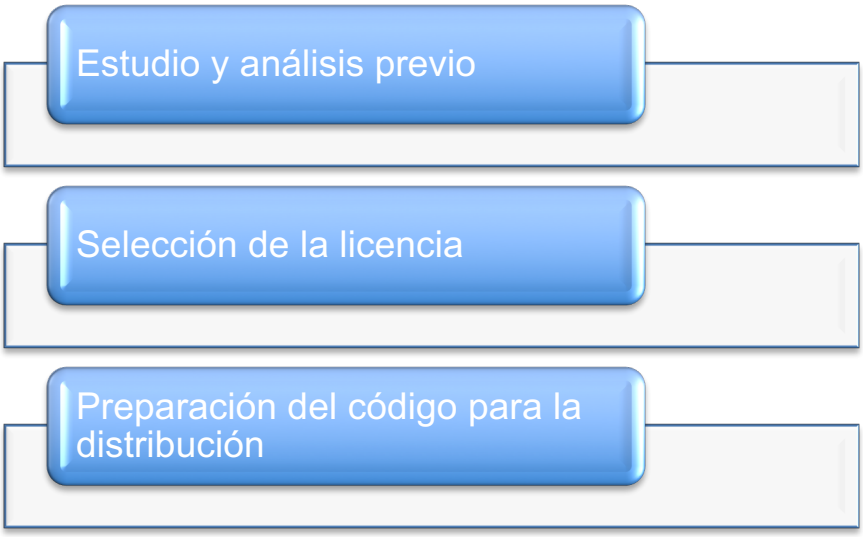


Figura 3.19. Fases para liberación de *software*. Elaboración propia.

A. Estudio y análisis previo

En este apartado se analizan los elementos clave del *software* o de los componentes que lo conforman; por ejemplo, autor o autores y sus implicaciones jurídicas en cuanto a derechos de propiedad intelectual.

Se recomienda que cuando se trata de derechos de autor debe tenerse conocimiento de la producción intelectual y las normas legales vigentes para la protección de obras en México, para estos aspectos se encuentra El Centro Mexicano de Protección y Fomento de los Derechos de Autor. Sociedad de

Gestión Colectiva, que puede consultarse en el siguiente vínculo: https://cempro.com.mx

En lo que se refiere a los derechos morales y patrimoniales puedes consultar el siguiente vínculo:

https://cempro.com.mx/sitio/derecho-moralderecho-patrimonial/

Del mismo modo, si se requiere conocer el registro por el desarrollo de un *software*, se puede consultar el siguiente vínculo para conocer los procedimientos, formatos, etcétera.

https://indautor.gob.mx/tramites-y-requisitos/registro/obra_computo.html

B. La selección de licencia

Deben considerarse los siguientes puntos:

- Las libertades que incluye.
- Considerar el régimen de cesión de derechos.
- Verificar si es irrevocable o no lo es.
- Tener presente el aspecto del *copyright* y sus implicaciones legales.
- Características de la licencia en cuanto a acceso remoto a través de redes digitales.
- Tener en cuenta los derechos internacionales, si existieran.
- Si puede o no ser posteriormente privatizable.
- Tener presente los acuerdos de contenido comercial, así como el soporte técnico o consultoría.

C. Preparación del código para la distribución

Después de ver los aspectos de la licencia es preciso tener presentes los siguientes preparativos para realizar la distribución:

- Incluir documentos de código fuente original;
- El *copyright, notice* u otro aviso de titularidad y autoría;
- Indicar cuál licencia utilizar.

Para el caso de software de fuente abierta que posibilita la modificación, se debe considerar los siguientes puntos:

- Las características y naturaleza de las modificaciones.
- La fecha de modificación.
- Autor o autores.
- Debe indicarse de manera clara si se trata de un proyecto colaborativo, una web o e-mail de contacto.
- Debe elaborarse un documento especificando las cuestiones jurídicas del software distribuido, que sea de fácil acceso al usuario en donde se mencionen los datos de identificación, los componentes, las licencias, marcas, contactos, firmas, etcétera.

3.9. Mantenimiento

En los últimos años existe gran variedad de sistemas de *software* desarrollados por consultorías, ingenieros, programadores. El mantener un *software* funcionando, por lo general, es difícil, debido a que los sistemas están sujetos a cambios constantes; en algunos casos, cuando se llega a presentar la falla en un *software,* se debe corregir inmediatamente; el encargado del sistema es la persona que debe realizar los reportes constantemente y observa cómo va funcionando el *software* y qué tipo de errores se presentan para poder dar un reporte constante y así poder realizar cambios que mejoren el *software*.

El mantenimiento de *software* se refiere a la forma de cambiar un sistema cuando este producto esté terminado y entregado. Durante su funcionamiento, los cambios que se van realizando, deben de ser sencillos para corregir errores de código; los cambios que se lleguen a necesitar, más extensos, como errores de diseño o para mejorar el *software* añadiendo nuevos componentes al sistema donde sea necesario.

Existen tres tipos diferentes de mantenimiento de software

De acuerdo con lo que presenta Sommerville, existen tres tipos de mantenimiento de software[21]:

1. Para reparar defectos del *software*.
2. Adaptación ambiental.

21 Si deseas profundizar sobre este tema, puedes consultar la obra del autor Sommerville (2011) *Ingeniería de software.* México: Pearson, pp. 242, 243. Recuperado el 27 de septiembre de 2018, disponible con vista previa en: http://www.academia.edu/15366832/Ingenieria-de-Software-Ian-Somerville-9-edicion-espa%C3%B1ol_1_

3. Adición de funcionalidad.

(Sommerville, 2011: 242, 243)

Durante el desarrollo, el *software* debe ser más sencillo y fácil de entender, para que así se puedan reducir los costos de mantenimiento; por ello se deben aplicar buenas técnicas de ingeniería de *software,* tales como una especificación precisa, el uso de programación orientada a objetos, con el fin de tener mejores y menores costos de mantenimiento.

Según Sommerville (2005: 455), las razones importantes del mantenimiento son:

1. Estabilidad en el equipo.
2. Responsabilidad contractual.
3. Habilidades del personal.
4. Antigüedad y estructura del programa.

En muchas organizaciones, las actividades de mantenimiento se ven como de baja categoría; sin embargo, es muy importante considerar que dichas actividades son valiosas porque previenen de hacer gastos excesivos si los sistemas llegan a fallar.

En general, el mantenimiento consiste en la reparación de la falla que llega a tener el *software* desarrollado, el personal encargado de reportar este tipo de fallas son los operativos que utilizan los equipos, las reparaciones son rápidas y sencillas. El mantenimiento que se da en este caso es de tipo correctivo.

Figura 3.20. Ejemplo de mantenimiento de software aboración basada en: Ian Sommerville, 2005: 455.

Proceso de evolución de mantenimiento.

En la evolución del *software* en el proceso de desarrollo de una organización, se deben realizar propuestas de cambio de sistema, que consisten en requerimientos implementados en el *software* entregado; además de las reparaciones de errores por parte del desarrollador y de agregar nuevas ideas y cambios para su mejora.

ACTIVIDADES DE MANTENIMIENTO

Como es sabido, un sistema es un conjunto de partes integradas que funcionan todas entre sí para lograr un propósito concreto.

El mantenimiento de sistemas informáticos tiene como finalidad conseguir que todos los componentes sean operativos, el mayor tiempo posible, para los usuarios en una organización determinada. El objetivo principal es que esté disponible para los usuarios cuando soliciten el mantenimiento.

Los niveles de mantenimiento son:

Nivel de mantenimiento de hardware, que se encarga básicamente del buen estado y funcionamiento de los equipos.

Nivel de mantenimiento de software, que se refiere a los programas, aplicaciones y datos que conforman el sistema.

Nivel de mantenimiento de documentación, para todo sistema informático debe existir la documentación que explique el correcto funcionamiento y los propósitos que tiene cada una de sus partes; para esto, se requiere elaborar previamente los documentos y manuales necesarios, que deben ser debidamente conocidos por los operadores del sistema.[22]

Para que funcione todo el sistema, es necesario que todos los niveles de mantenimiento estén perfectamente bien coordinados, organizados y comunicados; así se logra la rápida atención del equipo de soporte técnico.

[22] s/a (s/f) *Mantenimiento de software***.** Recuperado el 29 de marzo de 2019 de: https://swcb37.files.wordpress.com/2013/08/mantenimiento-de-software.pdf

El mantenimiento se clasifica en cuatro tipos que son:

1. Perfectivo

Este tipo de mantenimiento altera el funcionamiento con la finalidad de mejorarlo, se realiza cuando cambian los requerimientos del sistema.

2. Adaptativo

Se refiere a una adaptación en el sistema, específicamente en las entradas y salidas, o bien en las especificaciones de los programas y dispositivos periféricos.

3. Correctivo

Se centra en detectar y corregir cuando existen salidas incorrectas y ciertas anormalidades en el sistema.

4. Preventivo

Este tipo de mantenimiento se centra en predecir y anticipar problemas que pudieran presentarse posteriormente; mejora los programas, la calidad, y la documentación del sistema, sin alterar su funcionamiento.[23]

Es conveniente tener presente que el mantenimiento debe tener una debida planificación que consiste en:

a) Asignar prioridad a las peticiones de mejoras básicas o informes de errores que denotan problemas del sistema.
b) Estimar el costo de volver a desarrollar el sistema remplazándolo en comparación al costo de mantenerlo.
c) Establecer una colección significativa de estos cambios tratados como puntos funcionales para la siguiente versión.
d) Añadir, si los recursos lo permiten, pequeñas mejoras, y más localizadas para la siguiente versión.

23 Rodríguez Martínez, Rita Carolina (2000) *Mantenimiento para el sistema de control del proceso de software.* Tesis de grado de Maestría, Facultad de Ingeniería UNAM, pp. 4-5. Recuperado el 27 de septiembre de 2018 de: http://132.248.9.195/pd2000/285421/Index.html

e) Gestionar administrativamente la aceptación de las modificaciones, cambios o propuestas para ser incluidas como parte de la evolución de una versión posterior.[24]

24 Rodríguez Martínez, Rita Carolina. *Loc. cit.*

RESUMEN

Dentro del diseño de sistemas, es muy importante saber los requerimientos necesarios para poder diseñar un sistema de manera efectiva.

Para poder realizar un sistema, es necesario conocer todo sobre el buen diseño del mismo, principalmente los tipos de salidas de sistemas y las características de cada salida, así como las entradas y controles que va teniendo dicho sistema para mejorar su calidad al momento de ingresar la información.

Para poder desarrollar una base de datos, conocer la interfaz, las características, el diseño de sistemas de forma eficiente y compatible, a fin de efectuar los registros y datos, es importante saber qué tipo de diseño de sistema nos facilita el trabajo diario. También, se debe comparar con otros sistemas la factibilidad, portabilidad y sobre todo que sean compatibles en los sistemas operativos requeridos.

Una vez que el sistema se consolida, es viable tener un documento técnico para el usuario en donde el contenido esté completo y sea viable para el sistema y su uso óptimo; ahora bien, una vez que ya está el diseño del sistema, se deben realizar las pruebas necesarias para poder diagnosticar que no tenga problemas; pero el usuario debe saber qué tipo de éstas se han de realizar para estar seguro que el material o diseño final está en buenas condiciones y listo para liberarlo.

Cuando se ha liberado, se pone a prueba y se verifica para un buen uso del sistema, además todo sistema debe llevar un mantenimiento en donde se van reparando los errores internos del sistema, con lo que se va mejorando la eficiencia para el uso diario; por tal motivo, se deben conocer los tipos de

mantenimiento que se deben de realizar a un sistema completo y en buen estado para que no se tenga ningún problema al momento de ejecutarlo y utilizarlo.

Bibliografía de la unidad

SUGERIDA

Autor	Capítulo	Páginas
Bruegge (2001)	3, 9, 10,13, 14 y 15, 28	177-300, 626
Gómez y Suárez (2007)	10,11, 25	166, 217, 264-266
Johansen (2006)	16	331

REFERENCIAS BIBLIOGRÁFICAS

BIBLIOGRAFÍA SUGERIDA

Braude, Eric J. (2003). *Ingeniería de Software: una perspectiva orientada a Objetos*. México: Alfaomega.

Bruegge, Bernd. (2001). *Ingeniería de software orientada a objetos.* México: Prentice Hall.

Gómez Vieites, Álvaro y Suárez Rey, Carlos. (2007). *Sistemas de Información: Herramientas prácticas para la gestión empresarial*. (2ª ed.) México: Alfaomega.

Johansen Bertoglio, Oscar. (2006). *Introducción a la Teoría General De Sistemas*. México: Limusa.

Kendall, Kenneth E. y Kendall, Julie E. (2005). *Análisis y diseño de sistemas*. (6ª ed.) México: Pearson Educación.

McClure, Carma (1992). *CASE La automatización del software*. Delaware: RA-MA.

Mochi, Prudencio (2006). *La industria del software en México en el contexto internacional y latinoamericano.* Morelos, México: UNAM, Centro Regional de Investigaciones Multidiciplinarias.

Piattini Velthuis, Mario G., et al. (2000). Análisis y diseño detallado de Aplicaciones Informáticas de Gestión. México: Alfaomega/Ra-Ma.

Pressman, Roger (2010). *Ingeniería del software. Un enfoque práctico* (7ª ed.). México: McGraw-Hill.

BIBLIOGRAFÍA BÁSICA

Beynon-Davies, P. & Alegre, E. (2014). *Sistemas de información: introducción a la informática en las organizaciones*. Barcelona: Reverté.

Gómez, J., Gil, F., Villar, E. & Méndez, F. (2014) *Administración avanzada de sistemas informáticas*. México: Alfaomega

Kendall, K. E., Kendall, J. E., Romero, A., & Kendall, K. E. (2011). *Análisis y diseño de sistemas*. México: Prentice Hall.

Naranjo, F. (2012). *Control lineal moderno: análisis y diseño en el espacio de estados*. Colombia: Universidad Autónoma de Occidente.

Pantaleo, G., & Rinaudo, L. (2015). *Ingeniería de software*. Argentina; México Alfaomega/ Grupo Editor Argentino.

BIBLIOGRAFÍA COMPLEMENTARIA

Amaya, J. (2010). *Sistemas de información gerenciales: hardware, software, redes, Internet, diseño*. Colombia: Ecoe Ediciones.

Flórez, H. A. (2014). *Sistemas digitales: principios, análisis y diseño*. Colombia: Ediciones de la U.

Gómez, A., & Suárez, C. (2012). *Sistemas de información: herramientas prácticas para la gestión empresarial.* México: Alfaomega.

Morfín, J. A. (2013). *Sistemas digitales: una perspectiva de diseño*. México: Universidad Iberoamericana.

SITIOS ELECTRÓNICOS

(VIGENCIA AL 8 DE ABRIL DE 2019)

Sitio	Descripción
http://www.qcin.org/nabcb/	National Accreditation Council of Certification Bodies
http://cordis.europa.eu/esprit/home.html https://cordis.europa.eu/en	Procesos dependientes e independientes del ciclo de vida.
http://www.isixsigma.com	Isixsigma se enfoca a apoyar a las empresas en la obtener productos sin defectos con información esencial y conocimientos prácticos.
http://www.efqm.es/	Fundación Europea para la Gestión de la Calidad
https://www.iso.org/obp/ui/#iso:std:is	ISO (2018)

o-iec-ieee:29148:ed-2:v1	
http://jazzfantastic.blogspot.mx/	Díaz Guzmán, Yazmín (2013). “Simbología y significado”
http://churriwifi.wordpress.com/2010/05/03/16-1-identificacion-origenes-de-datos-utilizando-data-profiling/	Espinosa Roberto (2010) “*Identificación orígenes de datos. Utilizando Data Profiling*”
https://diagram-ideas.com/ejemplo-diagrama-de-flujo-proceso-pdf/	*Ejemplo diagrama de flujo proceso PDF*
https://www.americaeconomia.com/analisis-opinion/el-arte-de-diagnosticar-la-solucion-un-problema	Hernández, Rafael (2013) “*El arte de diagnosticar la solución a un problema*”
https://www.educaweb.com/profesion/analista-sistemas-informaticos-362/	Educaweb. Información sobre conceptos básicos de analista de sistemas informáticos.
http://www.tsitecnologia.com.co/outsourcing-ti-servicios-gestion-requerimientos/	TSI. Información sobre sistemas de requerimientos.
https://vdocuments.mx/ingenieria-del-softwareunenfoquepractico.html	Vista previa del libro *Ingeniería del software. Un enfoque práctico* del autor Pressman.
http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1692-33242015000200013	Mercado Caruso. Nohora (2015) “Mejora de los procesos de estimación de costos de software. Caso del sector de software de Barranquilla”. En *Revista Ingenierías,* Universidad de Medellín.
http://analsisde.blogspot.com/2017/03/diseno-de-la-salida-para-satisfacer-un.html	Información sobre Análisis de sistemas. Gomes Cristian (2019) *Diseñar una salida eficaz.*
https://www.w3computing.com/systemsanalysis/good-form-design/	Información sobre Análisis de sistemas.
https://revistas.uptc.edu.co/index.php/ingenieria/article/view/3551/4326	M.L. Rojas-Montes, F.J. Pino-Correa & J.M. Martínez, "Proceso de pruebas para pequeñas organizaciones desarrolladoras de software", Fac. Ing., vol. 24 (39), pp. 55-70, mayo-ago. 2015.
https://cempro.com.mx	Información que presenta el Centro Mexicano de Protección y Fomento de los Derechos de Autor. Sociedad de Gestión Colectiva.
https://cempro.com.mx/sitio/derecho-moralderecho-patrimonial/	Información del Centro Mexicano de Protección y Fomento de los Derechos de Autor que se refiere a los derechos morales y patrimoniales.
https://indautor.gob.mx/tramites-y-requisitos/registro/obra_computo.html	Información sobre el procedimiento de registro de desarrollo de un software y procedimientos formatos.

Plan 2012

2016

actualizado