Universidad Nacional Autónoma de México
Facultad de Contaduría y Administración
Sistema Universidad Abierta y Educación a Distancia

Licenciatura en Informática

Informática VII. Ingeniería de Software

Apunte electrónico

SUAyED

COLABORADORES

DIRECTOR DE LA FCA
Dr. Juan Alberto Adam Siade

SECRETARIO GENERAL
L.C. y E.F. Leonel Sebastián Chavarría

– – – –

COORDINACIÓN GENERAL
Mtra. Gabriela Montero Montiel
Jefe de la División SUAyED-FCA-UNAM

COORDINACIÓN ACADÉMICA
Mtro. Francisco Hernández Mendoza
FCA-UNAM

– – – –

COAUTORES
Lic. Ricardo Alberto Báez Caballero
Mtro. René Montesano Brand

DISEÑO INSTRUCCIONAL
Mtro. Joel Guzmán Mosqueda

CORRECCIÓN DE ESTILO
Mtro. José Alfredo Escobar Mellado

DISEÑO DE PORTADAS
L.CG. Ricardo Alberto Báez Caballero
Mtra. Marlene Olga Ramírez Chavero
L.DP. Ethel Alejandra Butrón Gutiérrez

DISEÑO EDITORIAL
Mtra. Marlene Olga Ramírez Chavero

OBJETIVO GENERAL

Al finalizar el curso, el alumno aplicará el proceso de desarrollo de *software* con los estándares de calidad reconocidos por la industria de *software* para garantizar la calidad del producto.

TEMARIO OFICIAL (horas 64)

	Horas
1. Fundamentos de la ingeniería de *software*	12
2. *Software*	8
3. Administración de proyectos	12
4. Verificación y validación	8
5. Métricas	8
6. Liberación y mantenimiento	8
7. Situación de la ingeniería de *software* en México	8
Total	**64**

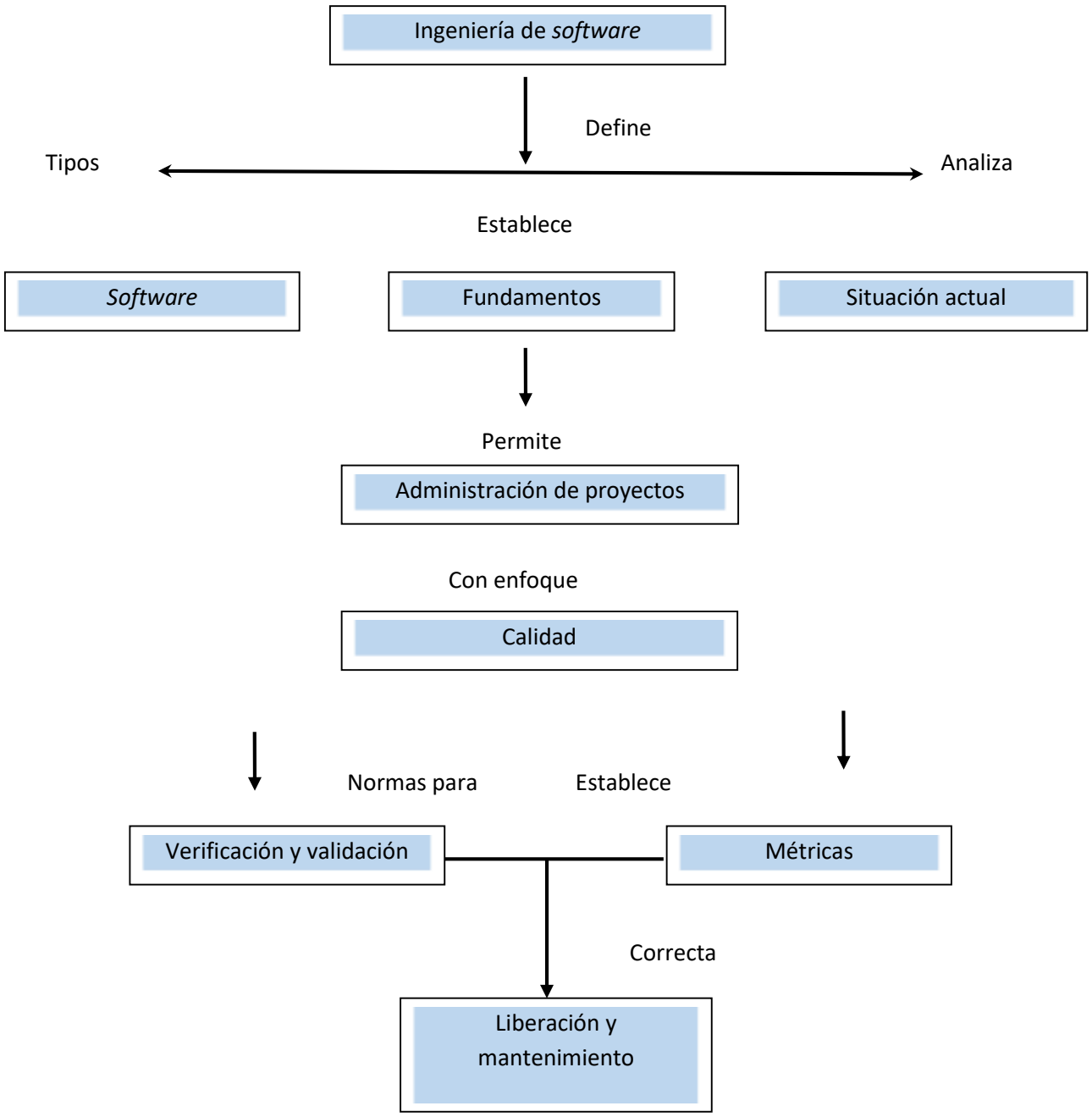
INTRODUCCIÓN A LA ASIGNATURA

A lo largo de este curso se revisarán algunos elementos importantes en el proceso de desarrollo de *software*, en donde se hace un recorrido por la evolución y fundamentos del empleo del *software* en la ingeniería.

Se define el *software* en su comprensión de sus elementos, sus estándares de calidad reconocidos por la industria y cómo éstos pueden vincularse en la administración de proyectos para garantizar la calidad del producto.

Así mismo, se revisa cómo es este proceso en el desarrollo de la informática en las diferentes instituciones y organismos en México y cuáles son las diferentes normas que los rigen.

ESTRUCTURA CONCEPTUAL



UNIDAD 1

Fundamentos de la ingeniería de software

OBJETIVO PARTICULAR

Al finalizar el curso, el alumno analizará los conceptos y principios de la ingeniería de *software*.

TEMARIO DETALLADO

(12 horas)

1. Fundamentos de la ingeniería de software

- 1.1. Crisis del *software*
- 1.2. Objetivos de la ingeniería de *software*
- 1.3. Principios
  - 1.3.1. Rigor
  - 1.3.2. Formalismo
  - 1.3.3. Modularidad
  - 1.3.4. Abstracción
  - 1.3.5. Anticipación al cambio
  - 1.3.6. Arquitectura de software
- 1.4. Personas, procesos, proyectos y productos de la ingeniería de software
- 1.5. Metodología, técnicas y herramientas
- 1.6. Código de ética
- 1.7. Modelos del ciclo de vida de sistemas
- 1.8. Procesos de desarrollo de software

- 1.8.1. Tradicionales
- 1.8.2. Ágiles
- 1.9. Estándares para la calidad del proceso
  - 1.9.1. Modelo de madurez de capacidades (CMM, capability maturity model)
  - 1.9.2. Mejora del proceso de software y determinación de la capacidad (ISO-15504 / SPICE, Software Process Improvement and Capacbility Determination)
  - 1.9.3. Proceso de software personal (PSP, personal software process)
  - 1.9.4. Proceso de software en equipos (TSP, team software process)
  - 1.9.5. Moprosoft (Norma Oficial Mexicana)

INTRODUCCIÓN

El desarrollo de *software* ha sido una necesidad creciente a partir de la creación de las computadoras. Desde la década de 1960 hasta la fecha, las empresas grandes y pequeñas requieren nuevos sistemas de información capaces de automatizar muchos de sus procesos, para ello, en la mayoría de casos, es indispensable diseñar sistemas a la medida.

La ingeniería de *software* es una disciplina que permite al desarrollador de *software* o de sistemas informáticos crear *software* con calidad que solucione las necesidades de las empresas o de particulares.

A lo largo de esta unidad, se revisarán algunos aspectos centrales de la ingeniería de *software*: sus principios y enfoques de calidad, así como los diversos modelos a seguir para construir *sotfware* de calidad.

1.1. Crisis del *software*

A finales de la década de 1960, con la computación en pleno proceso de crecimiento, había problemas de desarrollo de *software*:

- Retrasos respecto a los tiempos de planeación de proyectos
- Baja calidad y fiabilidad del *software*
- Costos elevados
- Poca satisfacción de demanda
- Inmadurez de la industria
- Costos de mantenimiento elevados

A esta situación que predominaba en esos tiempos hoy se le denomina “crisis del *software*”.

Para resolver la crisis imperante, los desarrolladores de *software* buscaron generar metodologías y procesos que permitieran crear *software* de manera más eficiente, económica y de calidad. El resultado de estos esfuerzos se conoce como ingeniería de *software*.

1.2. Objetivos de la ingeniería de *software*

De acuerdo con Sommerville (2005, p. 5), la ingeniería de *software* “es una disciplina de la ingeniería que comprende todos los aspectos de la producción de *software*”. En otras palabras, se concentra en el desarrollo de *software* desde su concepción hasta su puesta en marcha, enfocándose en cada uno de los procesos para que sea de calidad.

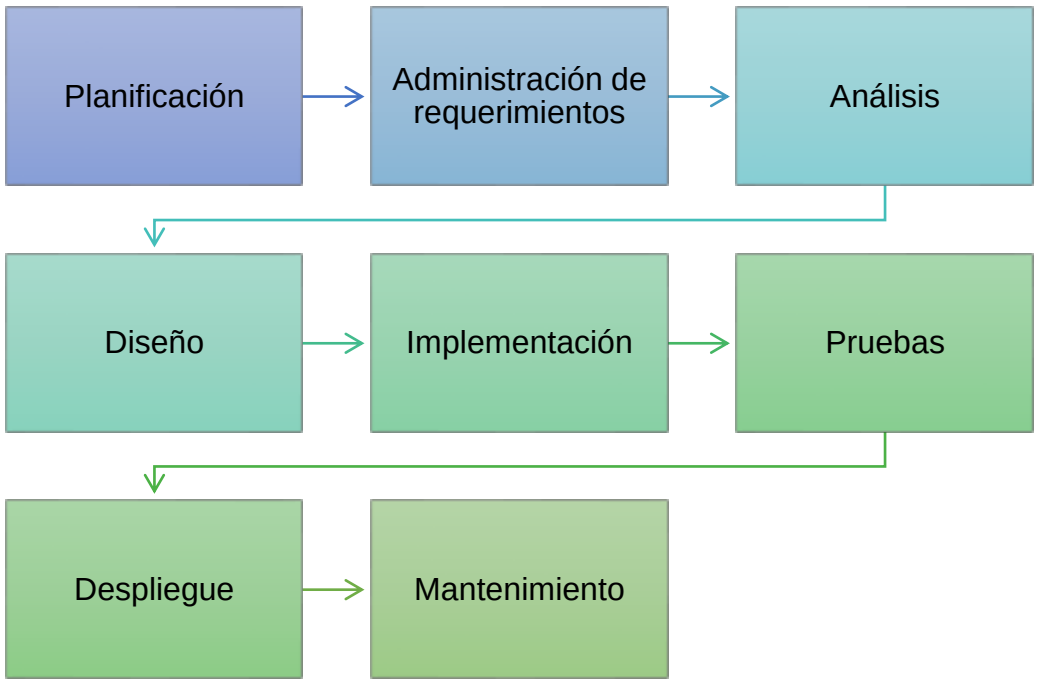
El objetivo principal de la ingeniería de *software* es desarrollar *software* de calidad que satisfaga las expectativas de los usuarios. En este caso, por calidad se entiende que el *software* cumpla con los requerimientos de los usuarios al cien por ciento, es decir, que el software realice lo que el usuario necesita para efectuar su trabajo.

1.3. Principios

Los principios de la ingeniería de *software* son los pilares sobre los que debe regirse el desarrollo de *software* de calidad, a continuación describiremos dichos principios.

1.3.1. Rigor

Todos los campos de la ingeniería siguen procesos y métodos específicos para alcanzar sus objetivos. Dentro de la ingeniería de *software*, existen diversos métodos para desarrollarlo; en todo caso, se ajustan a los siguientes pasos:



El rigor, entonces, es la realización de cada paso anterior con base en cualquier metodología para el desarrollo de *software* de calidad, documentando todo el proceso de forma minuciosa.

1.3.2. Formalismo

La formalidad es el punto más elevado del rigor, pues requiere que el proceso de desarrollo del *software* sea guiado y evaluado de forma cuantitativa. Así, el formalismo consiste en establecer métricas que permitan identificar si la metodología seleccionada está dando resultados.

1.3.3. Modularidad

El *software* puede ser muy simple o muy complejo, y entre más complejo sea, resultará necesario dividirlo en fracciones más simples o módulos. La división del *software* en módulos más pequeños permite evaluar más facilmente su cumplimiento de los requisitos y funcionalidad.

1.3.4. Abstracción

"La abstracción es un proceso mediante el cual se identifican los aspectos relevantes de un problema ignorando los detalles"[1]. Hace que los desarrolladores de *software* creen un modelo conceptual a partir de los requerimientos del *software*, lo que les permite identificar las funciones y módulos principales que lo integrarán.

[1] *Principios de la ingeniería de software.* A partir de *Fundamentals of software engineering*, Carlo Ghezzi, Mehdi Jazayeri y Dino Mandrioli. Pontificia Universidad Javerlana. Cali, Colombia. Disponible en http://cic.puj.edu.co/wiki/lib/exe/fetch.php?media=materias:pis:principios_de_la_ingenieria_de_software.pdf. Consultado el 18/12/2014.

1.3.5. Anticipación al cambio

El *software* cambia de forma constante, ya sea por sus requerimientos, adición de nuevas funciones, modificaciones en el sistema operativo o simplemente por la corrección de fallas en sus componentes. Por eso, uno de los principios de la ingeniería de *software* dispone diseñar el *software* de manera flexible pero robusta, es decir, que sea funcional y a la vez deje agregar funciones o realizar mantenimiento en sus componentes de forma transparente para los usuarios; para lograr esta flexibilidad y robustez se requiere un diseño donde se emplee distintos patrones de un diseño.

Lo mencionado se alcanza mediante una documentación rigurosa de todas las fases de formación del *software*, desde su planeación hasta sus pruebas. Esto para que los desarrolladores puedan identificar fallas y realizar modificaciones de manera más sencilla a partir de esa documentación.

1.3.6. Arquitectura de software

La arquitectura de *software* se refiere al establecimiento de una estructura base que sirva como guía a los desarrolladores de *software*. Proporciona una línea de trabajo común, con patrones para la generación de *software* que ayudan a establecer la estructura, funcionamiento e interacción entre sus diversos componentes.

Se refiere a sistemas complejos compuestos de subsistemas que interactúan bajo el control de un diseño de sistema Arquitectura de Software
Los subsistemas que componen el sistema, las interfaces y las reglas de interacción entre ellos.

Una arquitectura de software para un sistema es la estructura o estructuras del sistema, que consisten en elementos, sus propiedades externamente visibles y las relaciones entre ellas.

Las ventajas de diseñar y documentar explícitamente una arquitectura de software se reflejan en:
Comunicación entre las partes interesadas
Decisiones tempranas de diseño
Reúso a gran escala
Los patrones de diseño se agrupan en tres tipos

Estilos arquitectónicos: Soluciones de organización a nivel del sistema
Un estilo arquitectónico expresa un esquema de organización estructural para sistemas de software.
Provee un conjunto de tipos de elementos predefinidos, especifica sus responsabilidades e incluye reglas y guías para organizar las relaciones entre ellos
Patrones de diseño: Soluciones a problemas detallados de diseño de software, Un patrón de diseño provee un esquema para refinar los elementos de un sistema de software o las relaciones entre ellos.
Describe una estructura recurrente de elementos de diseño interconectados que soluciona un problema general de diseño dentro de un contexto particular.

Idioms: Soluciones útiles para problemas específicos en algún lenguaje de programación, Un idiom es un patrón de bajo nivel, específico para un lenguaje de programación.

Describe cómo implementar aspectos particulares de elementos o de las relaciones entre ellos usando las características de un lenguaje particular.

1.4. Personas, procesos, proyectos y productos de la ingeniería de software

Las Cuatro "P" en el desarrollo de software: Personas, Proceso Proyecto, y Producto, El resultado final de un Proyecto Software es un producto que toma forma durante su desarrollo gracias a la intervención de muchos tipos distintos de personas.

En el desarrollo de *software* siempre existirá la interacción de elementos que faciliten alcanzar el objetivo de la ingeniería en este campo:

Personas	Quienes participan dentro de un proyecto de *software* y sus interacciones. Los principales autores de un proyecto Software son los arquitectos, desarrolladores, ingenieros de prueba y el personal de gestión. . Herramientas: Software que se utiliza para automatizar las actividades definidas en el proceso.
Producto	Elementos que se entregan y van más allá de la aplicación de *software.* Son los artefactos que se crean durante la vida del proyecto.

Proceso	Proporciona un marco sobre el cual se puede establecer un plan claro para desarrollar *software*. Mediante este proceso, los proyectos originan productos de manera efectiva debido a que las actividades propuestas se pueden ajustar a las características del proyecto. Es una definición del conjunto completo de actividades necesarias para transformar los requisitos de usuario en un producto
Proyecto	Su objetivo es realizar un producto de *software*. Se refiere al elemento organizativo a través del cual se gestiona el desarrollo del software.

Figura 1.1. Cuadro de personas, producto, proceso y proyecto de la ingeniería de *software.*[2]

Una organizacion enfrenta una tarea esencial siempre que hace una persona pase de recurso "latente" a un puesto de "trabajador". La palabra "trabajador" la usamos para denominar a los puestos a los cuales se pueden asignar persona, el termino rol para hablar de los papeles que cumple un trabajador. Un trabajador puede asumir roles en relación con otros trabajadores en diferentes flujos de trabajos. Cada trabajador es responsable de un conjunto de actividades necesarias para el diseño de un subsistema.

En el contexto de proceso unificado, el producto que se obtiene es un sistema software. El termino producto aquí se hace referencia al sistema entero, y no solo al código que se entrega.

Un sistema son todos los artefactos que se necesitan para representarlos en una forma comprensible para maquinas u hombres, para las maquinas, los trabajadores y los interesados.

[2] *Apuntes de Ingeniería de* software. SUAyED, FCA-UNAM, Plan 2005. P. 13.

Existen dos tipos de artefactos: artefactos de ingeniería y artefactos de gestión. Los de ingeniería creados durante las distintas fases de proceso (requisitos, análisis, diseño, implementación y prueba). Los de gestión tienen un tiempo de vida corto, lo que dura la vida del proyecto; A este conjunto pertenecen artefactos como el análisis de negocios, el plan de desarrollo (incluyendo el plan de versiones e interacciones).

Un sistema posee una colección de modelos. Cada trabajador necesita una perspectiva diferente del sistema, las perspectivas recogidas de todos los trabajadores se encuentran en unidades mas grandes, es decir, modelos de modo que un trabajador pueda pueda tomar una perspectiva concreta del conjunto de modelos.

El Proceso Unificado proporciona un conjunto de modelos cuidadosamente seleccionando con cual empezar. Este conjunto de modelos hace claro el sistema para todos los trabajadores, incluyendo a los clientes, usuarios y jefes del proyecto.

1.5. Metodología, técnicas y herramientas

Como todas las ingenierías, la de *software* se basa en diferentes métodos de desarrollo de *software* que se adaptan a las exigencias del proyecto establecido.

Se podría decir que las metodologías orientadas a agentes son el resultado de una transferencia de conocimiento desde la ingeniería del software. Los investigadores de agentes son los expertos para decir si una metodología captura lo esencial del concepto de agente; sin embargo, para evaluar la capacidad de una metodología los expertos son los ingenieros de software.

Metodología

Una metodología propone una manera de resolver problemas empleando recursos, técnicas y herramientas de forma organizada. En general, se conforma de fases y procedimientos bien definidos que ayudan a los desarrolladores en la construcción del *software* de manera eficiente.

Pressman afirma que “los métodos de la ingeniería de *software* nos indican cómo construir técnicamente el *software*” (2002, p. 14).

Los diferentes métodos de la ingeniería de *software* abarcan desde la etapa de análisis de requisitos, diseño, desarrollo, pruebas, hasta la fase de mantenimiento.

Técnica

Es definida como un “Conjunto de recursos que se usan en un arte, en una ciencia o en una actividad determinada, en especial cuando se adquieren por medio de su práctica y requieren habilidad”.[3]

En la ingeniería de *software*, la técnica permite a los desarrolladores de *software* seguir una metodología y establecer un marco de trabajo para la ejecución integral del proyecto.

Herramientas

Las herramientas son instrumentos para realizar de forma más eficiente una tarea específica. Asimismo, proporcionan al desarrollador enfoques automáticos o semiautomáticos para la construcción de *software*, y le permiten reutilizar código u otras herramientas que faciliten las actividades. A este enfoque se le suele denominar ingeniería de *software* asistida por computadora (CASE, por sus siglas en inglés).

[3] Definición consultada en Google.

1.6. Código de ética

La Association for Computing Machinery (ACM) aprobó el código[4] en noviembre de 1998 y el Institute of Electrical and Electronics Engineers (IEEE) – Computer Society, en diciembre del mismo año. Código de Ética y Práctica Profesional (Versión 5.2).

Preámbulo

La versión corta del código resume las aspiraciones a un alto nivel de abstracción; las cláusulas que se incluyen en la versión completa proporcionan ejemplos y detalles acerca de cómo estas aspiraciones modifican nuestra manera de actuar como profesionales de la ingeniería de *software*. Sin aspiraciones, los detalles corren el riesgo de convertirse en tediosos y legalistas; sin detalles, las aspiraciones pueden convertirse en altisonantes y vacías. Aspiraciones y detalles, entonces, forman un código cohesivo.

Los ingenieros de *software* deberán comprometerse a convertir el análisis, especificación, diseño, implementación, pruebas y mantenimiento de *software* en una profesión respetada y benéfica. En este orden, comprometidos con la salud, seguridad y bienestar social, seguirán los ocho principios descritos a continuación.

[4] Esta es una versión abreviada del código de ética, traducida libremente del original. La versión completa se puede encontrar en http://www.acm.org/about/se-code#full. Recuperado el 18/12/2014.

Sociedad.	• Actuarán en forma congruente con el interés social.
Cliente y empresario.	• Actuarán de manera que se concilien los mejores intereses de sus clientes y empresarios, en congruencia con el interés social.
Producto.	• Asegurarán que sus productos y modificaciones correspondientes cumplen los estándares profesionales más altos.
Juicio.	• Mantendrán integridad e independencia en su juicio profesional.

1.7. Modelos del ciclo de vida de sistemas

Ciclo de vida

Evolución de un sistema, producto, servicio, proyecto u otra entidad realizada por el hombre desde su concepción hasta el retiro.[5]

[5] ISO/IEC. *Systems and software engineering – Software Life Cycle Processes*. IEEE std 12207:2008. Disponible en línea para su adquisición en http://www.iso.org/iso/catalogue_detail?csnumber=43447. Definiciones tomadas del *Apunte de Ingeniería de* software. SUAyED FCA-UNAM, Plan 2005.

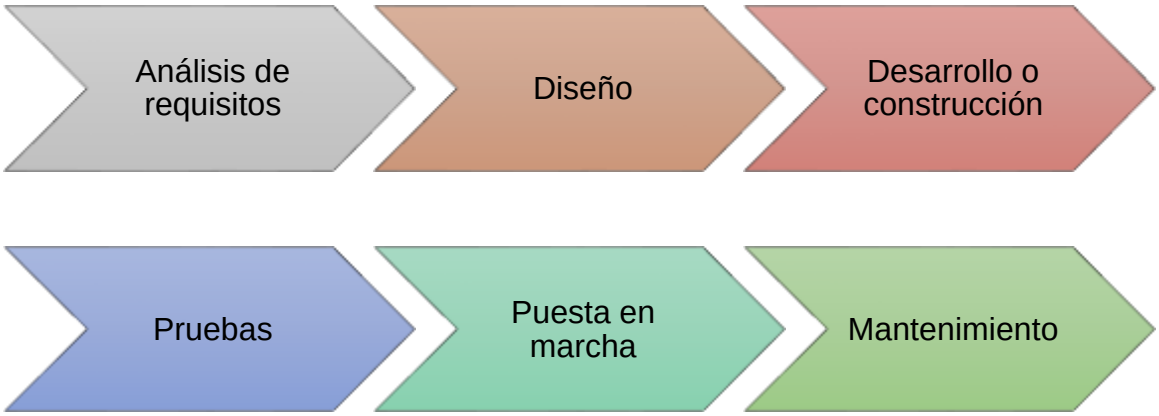
Modelo de ciclo de vida

Marco de referencia de procesos y actividades relacionados con el ciclo de vida, que puede estar organizado en fases. Actúa como una referencia común para la comunicación y entendimiento.[6]

Procesos del ciclo de vida del *software*

Marco de referencia que contiene procesos, actividades y tareas que serán aplicados durante la adquisición de un producto de *software* o servicio, y en el abastecimiento, implementación, operación, mantenimiento y disposición de productos de *software*[7].

Los ciclos de vida del *software*, conocidos también como modelos de proceso del *software*, contienen las siguientes fases:



[6] ISO/IEC. *Systems and software engineering – Software Life Cycle Processes*. IEEE std 12207:2008. Disponible en línea para su adquisición en http://www.iso.org/iso/catalogue_detail?csnumber=43447. Definiciones tomadas del *Apunte de Ingeniería de* software. SUAyED FCA-UNAM, Plan 2005.

[7] ISO/IEC. *Systems and software engineering – Software Life Cycle Processes*. IEEE std 12207:2008. Disponible en línea para su adquisición en http://www.iso.org/iso/catalogue_detail?csnumber=43447. Definiciones tomadas del *Apunte de Ingeniería de* software. SUAyED FCA-UNAM, Plan 2005.

En general, estos modelos o ciclos de vida indican de forma específica diferentes enfoques para desarrollar un proyecto de *software* (en el siguiente punto abordaremos algunos de ellos).

1.8. Procesos de desarrollo de *software*

El proceso de desarrollo de *software* o modelos de ciclo de vida hace referencia a diferentes enfoques que nos permiten desarrollar *software* de forma distinta, siempre teniendo en mente la calidad del producto final.

Es decir, un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que reúna los requisitos del cliente.

Un producto software en sí es complejo, es prácticamente inviable conseguir un 100% de confiabilidad de un programa por pequeño que sea. Existe una inmensa combinación de factores que impiden una verificación exhaustiva de las todas posibles situaciones de ejecución que se puedan presentar (entradas, valores de variables, datos almacenados, software del sistema, otras aplicaciones que intervienen, el hardware sobre el cual se ejecuta, etc.).

El proceso de desarrollo de software según Pressman,8(1997), no es único. No existe un proceso de software universal que sea efectivo para todos los contextos de proyectos de desarrollo.

[8] Pressman, R, Ingeniería del Software: Un enfoque práctico, McGraw Hill 1997.

A pesar de la variedad de propuestas de proceso de software, existe un conjunto de actividades fundamentales que se encuentran presentes en todos ellos:

1. Especificación de software	• Se debe definir la funcionalidad y restricciones operacionales que debe cumplir el software.
2. Diseño e Implementación	• Se diseña y construye el software de acuerdo a la especificación.
3. Validación	• El software debe validarse, para asegurar que cumpla con lo que quiere el cliente.
4. Evolución	• El software debe evolucionar, para adaptarse a las necesidades del cliente.

Además de estas actividades fundamentales, señaladas por Pressman (1997), tambien menciona un conjunto de “actividades protectoras”, que se aplican a lo largo de todo el proceso del software. Ellas se señalan a continuación:

- Seguimiento y control de proyecto de software.
- Revisiones técnicas formales.
- Garantía de calidad del software.
- Gestión de configuración del software.
- Preparación y producción de documentos.
- Gestión de reutilización.
- Mediciones.
- Gestión de riesgos.

1.8.1. Tradicionales

Las metodologías tradicionales o clásicas son aquellas que están guiadas por una fuerte planificación durante todo el proceso de desarrollo; llamadas también, no ágiles donde se realiza una intensa etapa de análisis y diseño antes de la construcción del sistema.

Los procesos de desarrollo tradicionales de *software* establecen los pasos a seguir de manera formal para su construcción de *software*. A continuación se enuncian los principales.

Modelo en casacada

Modelo de procesos que contiene las actividades fundamentales para la construcción de *software*, y representa cada actividad o fase de manera secuencial y separada.

Fases del modelo

Análisis y definición de requisitos.	Se especifican las funciones que deberá tener el *software*, su entorno de trabajo, restricciones y tipos de usuario.
Diseño del software*.*	A partir del análisis de requisitos, se desarrolla un diseño conceptual del *software* que sirve como plantilla de trabajo para su construcción.
Implementación y pruebas de unidades.	Establecido, aceptado y verificado el diseño, se procede a la construcción del *software*. Durante esta fase, se construyen los módulos funcionales del *software* y se prueban de manera independiente.
Integración y pruebas del sistema.	Se integran todos los módulos construidos en un sistema completo y se realizan las pruebas de los módulos de manera conjunta.
Funcionamiento y mantenimiento.	Terminadas las pruebas de integración, se procede a instalar el *software* en su entorno de trabajo y se pone en marcha. A partir de aquí, se identifican y corrigen errores y se procede a actualizar componentes.

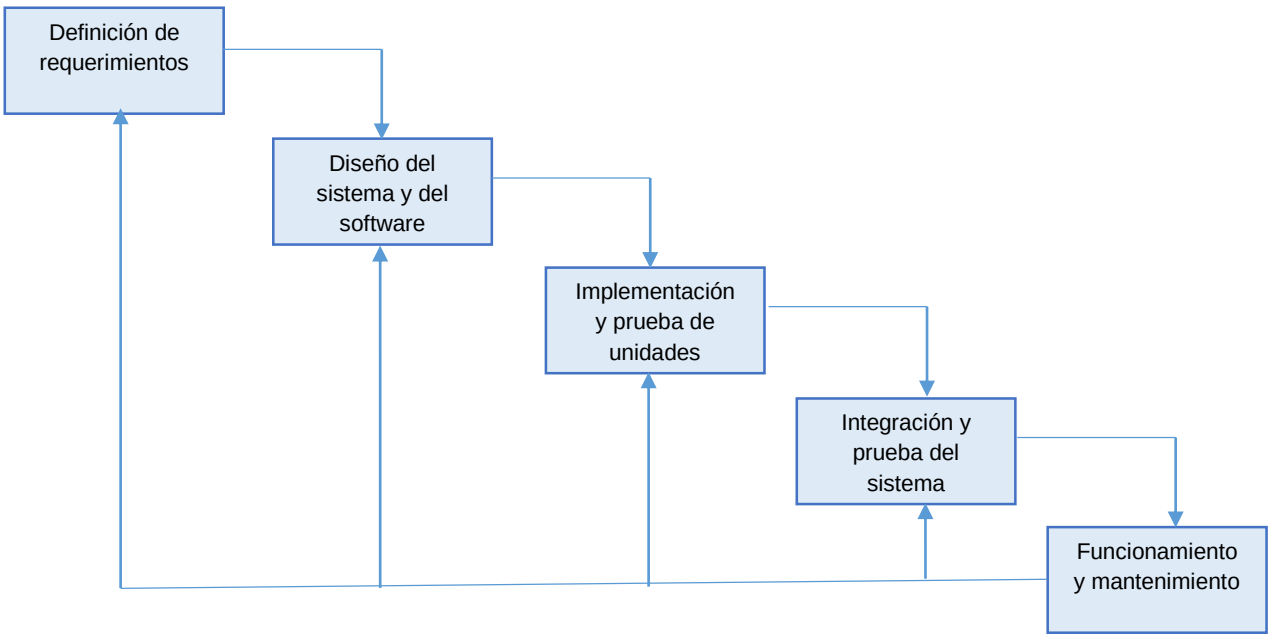


Figura 1.2. Modelo de ciclo de vida en cascada.

FUENTE: Sommerville (2005, p. 62).

Modelo evolutivo

Modelo basado en un desarrollo inicial que va siendo adaptado conforme a las modificaciones de los requisitos del *software* con el paso del tiempo. En este modelo, algunas actividades se entrelazan para ir refinando el desarrollo inicial hasta alcanzar un sistema adecuado que cumpla en su totalidad con los requerimientos del *software*.

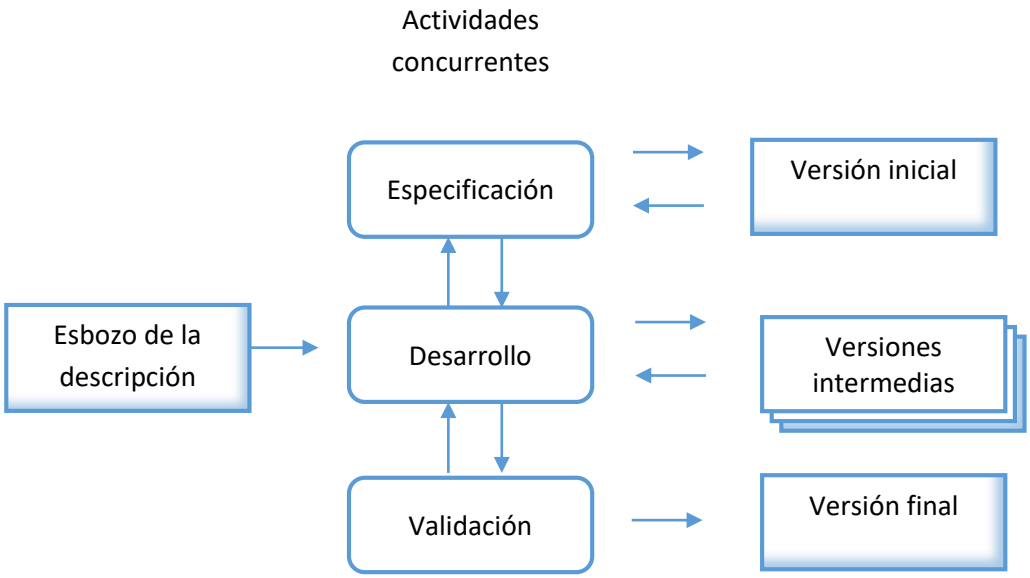


Figura 1.3. Modelo de proceso evolutivo.

FUENTE: Sommerville (2005, p. 63).

Modelo basado en componentes

Este modelo soporta su desarrollo en la reutilización de componentes de *software* previamente elaborado, o reutilizando componentes nuevos en diversos módulos del *software* a desarrollar.

En este modelo, el análisis y definición de requisitos son similares a los del modelo en cascada, pero hay procesos que lo distinguen:

Análisis de componentes.	Ubicados los requerimientos del *software*, se busca descomponerlo en elementos funcionales que permitan identificar funciones idénticas o similares en los módulos que compondrán el *software* para ser reutilizados.
Modificación de requisitos.	Se revisan de nueva cuenta los requisitos del sistema con base en los componentes identificados, a fin de ajustarlos a ellos; si no hay modificación, se puede volver a la fase de análisis de componentes.
Diseño del sistema con reutilización.	Se identifican los componentes de otros proyectos que sirven para ser utilizados en el desarrollo del *software* actual. Si no se tienen, se procede a su programación.
Desarrollo e integración.	Se da paso a la construcción de los módulos empleando los componentes reutilizables y agregando componentes nuevos. Al terminar este proceso, se integran los módulos como en el modelo de cascada.

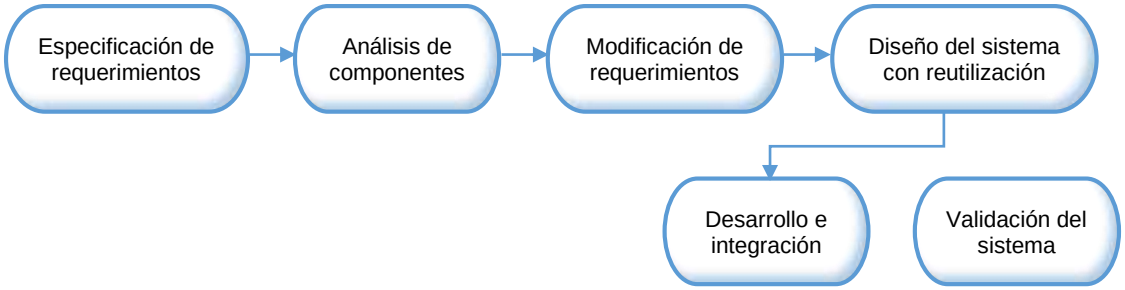


Figura 1.4. Modelo basado en componentes.

FUENTE: Sommerville (2005, p. 65).

1.8.2. Ágiles

Los metodos de desarrollo tradicionales o pesados comenzaron a ser rebasados por la creciente necesidad de nuevo *software* para las empresas pequeñas y medianas, que lo requerían a la medida y elaborado rápida y eficientemente. En este contexto, durante la década de 1990, los ingenieros de *software* comenzaron a crear modelos ágiles que cubren estas exigencias; se basan en los siguientes principios.[9]

Participación del cliente.

Los clientes deben involucrarse en todo el proceso de desarrollo y proporcionar requisitos del *software* todo el tiempo y evaluar los avances del sistema.

Entrega incremental.

El *software* se debe desarrollar sobre un enfoque incremental, donde en cada prototipo o incremento el cliente valida y provee nuevos requerimientos para el sistema.

Personas, no procesos.

Las formas extrictas y rigurosas de los modelos tradicionales son dejadas de lado; se focalizan los esfuerzos en las habilidades de cada persona y equipos de trabajo, permitiéndoles establecer sus propias formas de trabajo.

Aceptar el cambio.

Se diseña con la idea del cambio constante de requisitos, por lo que cada incremento o prototipo debe ser flexible a dichos cambios.

[9] Abrahamsson, P., Salo, O., Ronkainen, J., (2002) Agile Software Development Methods. Review and Analysis, VTT.USA. California

Mantener la simplicidad.

Se procurará mantener la simplicidad tanto en el desarrollo del *software* como en el proceso de desarrollo, dando prioridad al trabajo activo para mantener dicha simplicidad.

Entre las metodologías ágiles identificadas en:

- Extreme Programming .
- Scrum
- Familia de Metodologías Crystal.
- Feature Driven Development .
- Proceso Unificado Rational, una configuración ágil .
- Dynamic Systems Development Method.

Adaptive Software Development .

Open Source Software Development [.

1.9. Estándares para la calidad del proceso

La calidad es el objetivo fundamental de la ingeniería de *software*, y se refiere principalmente al grado en que cumple con los requisitos establecidos para su construcción. Hay diversos enfoques o estándares para evaluar la calidad del *software*.

1.9.1. Modelo de madurez de capacidades (CMM – Capability maturity model)

El modelo de madurez de *software* o CMM es una base para los desarrolladores de *software* en la mejora continua de sus procesos.

Estructura del estándar

CMM se divide en dos partes:

La representación por etapas establece una serie de niveles de madurez, con base en procesos probados, agrupados, ordenados y en la relación entre ellos.

Las etapas de madurez ayudan a los desarrolladores de *software* a construir un mapa de niveles que les facilita mejorar sus procesos. Cada etapa comprende

varias tareas de un proceso, y hacen que los desarrolladores se enfoquen a tareas críticas específicas que redunden en la mejora del proceso. Dentro de cada tarea se describen diversas actividades en términos de prácticas que contribuyen a la implementación eficiente de cada tarea y elevan el grado de madurez. Este proceso se repite continuamente hasta alcanzar todos los objetivos de las tareas del proceso.

En la representación continua, se determina una línea de trabajo base que sirve para medir el grado de madurez de cada tarea. De modo similar a la representación por etapas, se definen prácticas en favor de la mejora continua de cada tarea.

En ambas representaciones es necesario incluir metas generales y específicas que serán evaluadas a través de cada nivel de madurez, que asimismo será evaluado mediante las prácticas (también deberán contar con objetivos generales y específicos).

1.9.2. Mejora del Proceso de Software y Determinación de la Capacidad (ISO-15504 / SPICE – Software Process Improvement and Capability Determination)

ISO/IEC 15504 es un estándar internacional para la evaluación, medición y mejora continua de los procesos de desarrollo de ingeniería de *software*, basado en un conjunto de medidas de capacidad para cada proceso del ciclo de vida del *software*.

Objetivos principales del SPICE:

- Desarrollar un borrador de trabajo para un estándar de evaluación de procesos de *software*.

- Llevar a cabo los ensayos de la industria de la norma emergente.
- Promover la transferencia de tecnología de la evaluación de procesos de *software* a la industria del *software* a nivel mundial.[10]

Características del proyecto SPICE[11]:

- Establece un marco y los requisitos para cualquier proceso de evaluación de procesos y proporciona requerimientos para los modelos de evaluación de los mismos.
- Proporciona requisitos para cualquier modelo de evaluación de organizaciones.
- Ofrece guías para definir las competencias de un evaluador de procesos.
- Actualmente, tiene diez partes (de la 1 a la 7 están completas, y de la 8 a la 10 se encuentran en fase de desarrollo).
- Comprende evaluación de procesos, mejora de procesos, determinación de capacidad.
- En su parte 5, presenta un modelo de evaluación para los procesos de ciclo de vida del *software* definidos en el estándar ISO/IEC 12207, que determina los procesos del ciclo de vida del desarrollo, mantenimiento y operación de los sistemas de *software*.
- En su parte 6, brinda un modelo de evaluación para los procesos de ciclo de vida del sistema definidos en el estándar ISO/IEC 15288, que establece los procesos del ciclo de vida del desarrollo, mantenimiento y operación de sistemas.

[10] Norma ISO/IEC 15504. Disponible para su adquisición en http://www.iso.org/iso/catalogue_detail.htm?csnumber=37458. (Recuperado el 20/12/2014).
[11] Norma internacional ISO 9001. Sistema de gestión de calidad. Requisitos. Traducción disponible en http://farmacia.unmsm.edu.pe/noticias/2012/documentos/ISO-9001.pdf

- En su parte 8, ofrecerá un modelo de evaluación para los procesos de servicios TIC, que serán definidos en el estándar ISO/IEC 20000-4, que fijará los procesos contenidos en la Norma ISO/IEC 20000-1.
- Equivalencia y compatibilidad con CMMI. ISO forma parte del panel elaborador del modelo CMMI, y SEI mantiene la compatibilidad y equivalencia de esta última con 15504. Sin embargo, CMMI-DEV aún no es un modelo ajustado a esta norma (según lo requiere la Norma ISO 15504 para todo modelo de evaluación de procesos).[12]

1.9.3. Proceso de software personal (PSP – personal software process)

El Instituto de Ingeniería de *Software* (SEI) desarrolló una nueva metodología enfocada a la calidad del *software* denominada personal software process (PSP), enfocada a la planeación, costos y productividad dentro del desarrollo de *software*.

Características

PSP emplea documentos, denominados *scripts*, que detallan las actividades y tareas que cada desarrollador debe realizar en el proceso de desarrollo de *software*. Los *scripts* son la parte medular del PSP.

Los *scripts* empleados en el PSP generan datos estadísticos que permiten identificar las fortalezas y debilidades del proyecto de *software*, y explotarlas y solventarlas para mejorar la calidad del producto final.

PSP basa su metodología en la estimación. Facilita a los desarrolladores establecer periodos para efectuar actividades y tareas de un proceso en

[12] Tomado del *Apunte de Desarrollo de* software *empresarial.* SUAyED, FCA-UNAM, Plan 2005.

particular. Esto genera un mejor control del proyecto, su costo y tiempo de desarrollo.

La información de las estimaciones se emplea para evaluar los procesos actuales y realizar las mejoras necesarias para los procesos futuros. Lo anterior permite a los desarrolladores identificar sus propias carencias y pulir sus habilidades.

La metodología PSP requiere un análisis de requisitos previo, puesto que los requisitos son la base para evaluar las tareas y actividades definidas en los *scripts*.

1.9.4. Proceso de software en equipos (TSP – team software process)

Team software process (TSP) es una metodología enfocada al trabajo efectivo en equipo para el desarrollo de *software*. Su objetivo es la mejora de la calidad y productividad de un proyecto de desarrollo de *software*, a partir de la definición de acuerdos para optimizar costos y tiempos al ejecutar el proyecto.

La metodología TSP exige que los integrantes de los equipos de trabajo sean previamente capacitados en la metodología PSP, de modo que estén familiarizados con el empleo de planes de trabajo detallados.

TSP define los pasos para establecer los equipos de trabajo y fomentar un ambiente adecuado, mientras PSP aporta los elementos para que cada miembro de los equipos sea responsable de la calidad de sus tareas. En consecuencia, TSP establece y mantiene equipos autodirigidos.

TSP posibilita lo siguiente[13]:

[13] Metodología TSP/PSP, por Grupo de Tecnologías Kernel. Disponible en http://www.kernel.com.mx/documentos/psp_tsp.pdf. (Recuperado el 18/12/2014).

Con PSP, los desarrolladores utilizan procesos definidos y medibles. Se toma información del tamaño, tiempo y defectos al momento de realizar el trabajo. Se utilizan los datos para planear y monitorear el trabajo, administrar la calidad de los productos que se producen, y medir y mejorar el desempeño.
TSP ha permitido resolver problemas típicos de un negocio, como predicciones de costo y tiempo, aumento de productividad y ciclos de desarrollo, y mejora de calidad de productos.
PSP/TSP acrecienta la calidad del desempeño tanto de equipos como de individuos, es disciplinado y ágil, provee beneficios inmediatos y medibles, y acelera las iniciativas de mejora de procesos organizacionales.
Con TSP, los equipos encuentran y reparan defectos en etapas tempranas del proceso de desarrollo, lo que reduce significativamente el tiempo de pruebas.

Con una etapa de pruebas más corto, se reduce el ciclo completo.[14]

1.9.5. Moprosoft (Norma Oficial Mexicana)

El modelo de procesos de *software* hecho en México (Moprosoft) es una iniciativa de la Secretaría de Economía y de empresarios mexicanos dedicados al desarrollo de *software*, encabezados por Hanna Oktaba –de la Facultad de Ciencias de la UNAM–, quien se ha ocupado en fomentar la estandarización de operaciones en los procesos de desarrollo de *software* en México, a través de la incorporación de las mejores prácticas en la gestión e ingeniería de *software.*

El modelo Moprosoft promueve la mejora en la calidad y eficiencia de las empresas dedicadas al desarrollo de *software*, para aumentar su capacidad para alcanzar niveles internacionales de competitividad. De igual forma, puede ser

[14] Retomado de los *Apuntes de Diseño de* software *empresarial*. SUAyED, FCA-UNAM, Plan 2005.

aplicado a las áreas de informática y sistemas de las empresas, para la generación de sistemas internos de calidad.

MoProSoft se define como un modelo de procesos para el desarrollo y mantenimiento de software dirigido a la pequeña y mediana industria y a las áreas internas de desarrollo de software-, su objetivo principal es incorporar las mejores prácticas en gestión e ingeniería de software.

Moprosoft fue desarrollado por expertos mexicanos que recopilaron las experiencias exitosas de la industria de software a nivel mundial, y las adaptaron a las necesidades y características de las pequeñas y medianas industrias mexicanas (PYMEs) desarrolladoras de software.

MoProSoft está dividido en 9 procesos, llamados también prácticas, organizados por categorías de acuerdo a sus respectivas áreas de aplicación. Las categorías de procesos coinciden con los tres niveles básicos de la estructura de una organización: alta dirección, gestión y operación.

Moprosoft determina el nivel de madurez de la capacidad de cada proceso a través de una evaluación, que permite colocar a la empresa en uno de los siguientes 5 niveles.
Nivel 1: Proceso Realizado
Nivel 2: Proceso Administrado
Nivel 3: Proceso Establecido
Nivel 4: Proceso Predecible
Nivel 5: Optimización del proceso

También existe el nivel 0, que indica que el proceso está incompleto (caos). El nivel de una empresa corresponde al nivel máximo al que están todos sus 9 procesos. Para pasar de un nivel al siguiente, la empresa debe cumplir todos los requisitos de los niveles anteriores más los del nuevo nivel.

Las 4 partes de la norma son:

Parte 01: Definición de Conceptos y Productos

Parte 02: Requisitos de procesos (Moprosoft)

Parte 03: Guía de Implementación de Procesos

Parte 04: Directrices para la Evaluación de Procesos (EvalProsoft)

Entre otras actividades, NYCE se encarga de verificar si una organización cumple con los requisitos de alguna de las normas a su cargo. La verificación de una norma es la confirmación mediante la aportación de evidencia objetiva que se han cumplido los requisitos especificados.

La verificación consiste en determinar el nivel de madurez de los 9 procesos en las organizaciones que tienen como referencia el modelo

Las Características del modelo Moprosoft:

- Es específico para el desarrollo y mantenimiento de *software*.
- Es sencillo de entender y adoptar.
- Facilita el cumplimiento de los requisitos de otros modelos como ISO 9000:2000, CMM y CMMI.
- Se enfoca a procesos.
- Se le considera práctico en su aplicación, principalmente en organizaciones pequeñas, con niveles de madurez bajos.
- Comprende un documento de menos de 200 páginas, lo que, a diferencia de otros modelos y estándares, lo hace práctico.
- Es acorde con la estructura de las organizaciones mexicanas de la industria de *software*.
- Está orientado a mejorar los procesos para contribuir a los objetivos del negocio; no se limita a ser un marco de referencia o certificación.

- Tiene un costo bajo, tanto para su adopción como para su evaluación.

¿Para qué sirve Moprosoft?

- Mejora la calidad del *software* producido por la empresa que adopta el modelo.
- Eleva la capacidad de las organizaciones para ofrecer servicios con calidad y alcanzar niveles internacionales de competitividad.
- Integra todos los procesos de la organización y mantiene la alineación con los objetivos estratégicos.
- Inicia el camino a la adopción de los modelos ISO 9000 o CMMI.
- Sirve para implantar un programa de mejora continua.
- Permite reconocer a las organizaciones mexicanas por su nivel de madurez de procesos.
- Facilita la selección de proveedores.
- Permite obtener acceso a las prácticas de ingeniería de *software* de clase mundial.[15]

El modelo se convirtió en Norma Oficial Mexicana el 15 de agosto de 2015, tras su publicación en el Diario Oficial de la Federación, con el registro NMX-059-NYCE-2005 y título “Tecnología de la información-*software*-modelos de procesos y evaluación para desarrollo y mantenimiento de *software*”.

[15] Ventura, T. y Peñaloza, M. (2006). “Moprosoft, Modelo de procesos de *software* hecho en México”. Disponible en http://www.enterate.unam.mx/Articulos/2006/marzo/moprosoft.htm. (Consultado el 18/12/2014).

RESUMEN

La ingeniería de *software* surgió como respuesta a la “crisis del *software*” de finales de la década de 1960, ante la necesidad de crear *software* a la medida que satisficiera las exigencias de las empresas que comenzaban a solicitar *software* especializado para sus procesos internos.

La ingeniería de *software* es una rama de la ingeniería enfocada al desarrollo de *software* de calidad. Ofrece modelos que permiten a los desarrolladores de *software* crear proyectos de *software* de calidad que cumplan con los requisitos de sus clientes, siguiendo principios fundamentales como rigor, formalidad, abstracción, modularidad, anticipación al cambio y arquitectura.

Los modelos de desarrollo de *software,* o modelos de ciclo de vida, proveen a los desarrolladores diversos enfoques que pueden ser abordados para la generación de *software* de calidad. Estos modelos se clasifican en pesados y ágiles; los primeros se enfocan más a sistemas complejos, y los segundos se basan en el principio de la reutilización de componentes para la creación de *software* a la medida, de calidad y de manera más rápida que los modelos tradicionales o pesados.

Los desarrolladores adoptan diversos enfoques para garantizar la calidad del *software*, como la Norma ISO-15504 o la NMX-059-NYCE-2005, a fin de garantizar procesos de desarrollo de calidad.

MESOGRAFÍA

Bibliografía recomendada

Autor	Capítulo	Páginas
Pressman	1, 2	3-33
Sommerville	1, 4, 17	3-12, 59-79, 357-378

Bibliografía básica

Pressman, R. (2002). *Ingeniería del* software*. Un enfoque práctico* (5.ª ed.). México: McGraw-Hill.

Sommerville, I. (2005). *Ingeniería de* software (7.ª ed.). España: Addison Wesley / Pearson Educación.

Bibliografía complementaria

Booch, G., Rumbaugh J. y Jacobson, I. (2000). *El proceso unificado de desarrollo de* software. Madrid: Addison Wesley.

Braude J. E. González Osuna, M. (trad.). (2003). *Ingeniería de* software*: una perspectiva orientada a objetos*. México: Alfaomega.

Bruegge, B. y Dutoit, A. Ruiz Faudón, S. (trad.). (2002). *Ingeniería de software orientado a objetos*. México: Prentice Hall / Pearson Educación.

GhezzI, C. *et al.* (1991). *Fundamental of software engineering*. EUA: Editorial Prentice-Hall.

McConnell, S. *Desarrollo y gestión de proyectos informáticos*. McGraw-Hill.

Weitzenfeld, A. (2004). *Ingeniería de* software *orientada a objetos con UML, JAVA e Internet*. México: Thomson.

Sitios electrónicos

Sitio	Descripción
http://www.enterate.unam.mx/Articulos/2006/marzo/moprosoft.htm	Artículo. Moprosoft. Por Ma. Teresa Ventura & Marcela Peñaloza. Entérate en Línea. DGTIC. UNAM.
http://www.acm.org/about/se-code/	Código de Ética y Práctica Profesional del Ingeniero de *Software*, emitido por la Association for Computing Machinery (ACM).
http://www.swebok.org/	Base de conocimientos para la ingeniería de *software*, creada por la comunidad de desarrolladores de *software* a nivel mundial.

UNIDAD 2

Software

OBJETIVO PARTICULAR

Al finalizar la unidad, el alumno analizará la definición de *software* y las características que se espera que tenga.

TEMARIO DETALLADO

(8 horas)

2. *Software*

2.1. Concepto
2.2. Clasificación
2.3. Calidad del *software*
2.4. Estándares para la calidad del producto

INTRODUCCIÓN

El *software* permite interactuar con la computadora, y presenta diversos tipos enfocados a distintos mercados y personas. Su calidad es fundamental, en tanto determina si cumple con los requisitos solicitados por los clientes, o si es de utilidad para los usuarios al realizar sus actividades.

En esta unidad, se analizará el concepto de *software* y sus clasificaciones, así como los aspectos y normas relacionados con su calidad.

2.1. Concepto

El *software* consiste en “programas de computadora y la documentación asociada…” (Sommerville, 2005, p. 5). Los productos de Software pueden ser desarrollados para un cliente particular ó un mercado en general.

Un buen software debería ofrecer la funcionalidad y el rendimiento requeridos por el usuario y debería ser mantenible, confiable y usable.

La Ingeniería de software es la disciplina o área de la Ingeniería que ofrece métodos y técnicas para desarrollar y mantener software.

Las actividades fundamentales de la Ingeniería de Software.

Especificación de Software,

Desarrollo de software,

Validación de software

Evolución del software.

La diferencia principal entre la Ingeniería de Software y las ciencias de la computación radica en que, las ciencias de la computación se centran en la teoría y los fundamentos; La Ingeniería de software tiene que ver con los aspectos prácticos de desarrollo y distribución de software útil.

Por otro lado, la diferencia entre la ingeniería de software y la ingeniería en sistemas reside en que la Ingeniería de sistemas se refiere a todos los aspectos del equipo de desarrollo basado en sistemas, incluyendo la ingeniería de hardware, software y procesos. La Ingeniería de software es parte de este

proceso más general. Desde un punto de vista tradicional, son los programas que permiten darle utilidad a una computadora y que van acompañados de una serie de documentos que describen su proceso de creación (manual técnico) y se explica al usuario cómo instalar y operar el programa (manual de usuario), además de un archivo donde se encuentran almacenados los códigos de programación empleados (programa fuente) y el programa final del usuario (programa ejecutable). A partir de sus diversos métodos, la ingeniería de *software* nos lleva paso a paso en la creación de cada uno de estos elementos que forman parte del *software*.

2.2. Clasificación

El *software* puede ser creado ya sea de forma genérica, para aplicaciones comunes; o de forma personalizada, para satisfacer las necesidades de las empresas y sus procesos.

Por función o de aplicación	Se emplea de tal manera que la computadora llega a ser una herramienta útil para el usuario final en alguna de las actividades que realiza.
De utilería	Permite dar mantenimiento al equipo de cómputo.
De entretenimiento	Hace posible utilizar al equipo de cómputo como medio de entretenimiento.
Integrado	Reside solamente en la memoria y es utilizado para controlar productos y sistemas. Puede realizar funciones limitadas.

Para lenguaje de programación	Se enfoca a un lenguaje de programación. Genera todo un ambiente de desarrollo que facilita la generación y ejecución de código.
Por sistema operativo	Se encarga de la administración de los recursos de un equipo de cómputo.
Personalizado	Permite personalizar las funciones según los gustos o tipo de actividades que realiza con en el equipo de cómputo.
De negocios	Se utiliza para soportar las operaciones más importantes de un negocio en funciones como almacenamiento, análisis y procesamiento.

Figura 2.1. Clasificación del *software*. Pressman, (2002).

FUENTE: Pressman, R. (2002). *Ingeniería del* software*. Un enfoque práctico* (5.ª ed.). México: McGraw-Hill.

2.3. Calidad del *software*

La calidad es un concepto difícil de definir. Algunos autores lo hacen de la siguiente manera.[16]

- "La calidad es cumplir especificaciones. Cero defectos" (P. Crosby).
- "La calidad es el cumplimiento de los propósitos. Adecuación para el uso satisfaciendo las necesidades del cliente" (J. M. Juran).

[16] *Apuntes de Ingeniería de software*, SUAyED FCA-UNAM, Plan 2005.

- “La calidad es un grado predecible de uniformidad y fiabilidad a bajo costo, adecuado a las necesidades del mercado” (E. W. Deming).
- “La calidad es el costo que un producto impone a la sociedad desde el momento de su concepción” (G. Tagushi).

De acuerdo con estas definiciones, concluimos que la calidad reside en que un producto o servicio cumple con las expectativas de una persona. Si se trata, por ejemplo, de un celular, debe realizar y recibir llamadas, enviar y recibir mensajes de texto, tener cobertura en todas partes, buena duración de pila, etcétera. Pero no todos conciben de igual modo la calidad para el celular, algunos querrán que tenga conexión a Internet; otros, que sea de cierto color, etcétera. En consecuencia, la calidad es relativa a cada persona.

En el caso del *software*, pasa algo similar. Sin embargo, en general, un buen *software* presenta las características siguientes.

- *Mantenibilidad*. El *software* debe escribir de tal forma que prevea las modificaciones que puedan sufrir los requerimientos del cliente. De esta forma, se cumple con el principio de anticipación al cambio: el *software* se adapta a los cambios de su entorno.

- *Confiabilidad*. Un *software* confiable funciona acorde con las especificaciones de los clientes sin causar daños al *hardware* o al *software* con el que va a interactuar; tampoco provoca pérdidas económicas por su uso. Algunas características asociadas a la confiabilidad son fiabilidad, protección y seguridad.

- *Eficiencia*. El *software* debe hacer un uso adecuado de los recursos del sistema donde se encuentra instalado, sin desperdiciar recursos. De igual modo, responderá dentro un tiempo adecuado a las peticiones de los

usuarios. Algunos aspectos asociados con la eficiencia del *software* son tiempo de respuesta y de procesamiento, y uso de memoria.

- *Usabilidad*. Un buen *software* no genera problemas al usuario. Esto implica que debe ofrecer un manual de usuario para ser interpretado por los usuarios a los que está dirigido; tener una interfaz de usuario sencilla e intuitiva que genere una interacción simple.

Estos cuatro puntos describen de modo panorámico los principios de la ingeniería de *software* estudiados en la unidad anterior.

El concepto de calidad se extiende para abarcar las características que influyen en la calidad hacia el interior de las empresas o hacia el consumidor. También hay otros factores y características que determinan la calidad de software:

Funcionabilidad: que el usuario pueda utilizar el software

Confiabilidad: que los datos sean íntegros

Usabilidad: fácil de usar, fácil de aprender a usar

Portabilidad: compatible con otras plataformas

Compatibilidad: visible y ejecutable en la plataforma que corra

Corrección: capaz de darle mantenimiento

Eficiente: hace lo que debe bien, lo hace a tiempo y no derrocha recursos

Robustez: que se mantenga en un rito que debe

Oportunidad: fácil de acceder, en cualquier momento

2.4. Estándares para la calidad del producto

Según Pressman (2002, p. 135), la calidad del *software* es la "concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente".

La definición anterior, implica tres aspectos principales:

1. Los requisitos del *software* son la base de su calidad, ya que el *software* debe cumplirlos de la mejor manera.

2. Es fundamental seguir los estándares especificados como la base de trabajo para desarrollar el *software*; la falta de seguimiento o establecimiento de los mismos dará por resultado un *software* de mala calidad.

3. El *software* cumplirá tanto con los requisitos explícitos como implícitos. Es decir, existen factores que no necesariamente son expuestos por los clientes, pero que han de ser considerados al desarrollar el *software*: entorno de trabajo, tipo de usuario final, plataformas en donde puede ser instalado, etcétera. De omitir estos aspectos, el *software* sería de dudosa calidad.

Dentro de los estándares para garantizar la calidad del *software*, los más importantes son los establecidos por la Organización Internacional de Estandarización (ISO), en su serie de normas ISO-9000.

Normas ISO relevantes para la industria del *software*[17]:

ISO 9001.	•Quality Systems- Model for Quality Assurance in Design, Development, Production, Installation and Servicing. Estándar para describir el sistema de calidad empleado para el desarrollo del producto en su parte de diseño.
ISO 9000-3.	•Guidelines for Application of ISO 9001 to the Development, Supply and Maintainance of Software. Documento que sirve de guía para los desarrolladores de *software* para aplicar la Norma ISO 9001.
ISO 9004-2.	•Quality Management and Quality System Elements. Part 2. Documento que brinda los criterios para administrar e implementar los elementos necesarios para establecer servicios de calidad, por ejemplo, el soporte de usuarios.

Los requisitos se agrupan en 20 títulos:

- Responsabilidad de la gestión
- Inspección, medición y equipo de pruebas
- Sistema de calidad
- Inspección y estado de pruebas

[17] Puedes consultarlos en los anexos de esta unidad.

- Revisión de contrato
- Acción correctiva
- Control de diseño
- Control de producto no aceptado
- Control de documento
- Tratamiento, almacenamiento, empaquetamiento y entrega
- Compras
- Producto proporcionado al comprador
- Registros de calidad
- Identificación y posibilidad de seguimiento del producto
- Auditorías internas de calidad
- Formación
- Control del proceso
- Servicios
- Inspección y estado de prueba
- Técnicas estadísticas

RESUMEN

El *software* es un conjunto de documentos y programas generados a partir del proceso de desarrollo del mismo. Estos documentos registran cómo se construyó el *software* y la forma adecuada como debe ser empleado; mientras que los programas contienen el ejecutable final, que es instalado en la computadora del usuario, y el código fuente de ese ejecutable.

Es posible clasificar el *software* de acuerdo con su utilización: desde el genérico, de uso promedio, hasta el especializado, desarrollado a detalle para un cliente específico.

En general, la calidad se refiere a cumplir con ciertas características y funcionalidad, y cuya percepción varía de persona a persona. En lo que se refiere al *software*, la calidad puede ser medida por su cumplimiento con cuatro exigencias: mantenibilidad, confiabilidad, eficiencia y usabilidad.

Algunas normas internacionales, por ejemplo, ISO-9000, dan un marco de referencia que asegura la calidad del *software* a través de sus procesos de desarrollo, como ISO-9001, ISO 9000-3 e ISO 9004-2.

MESOGRAFÍA

Bibliografía recomendada

Autor	Capítulo	Páginas
Pressman	8	131-150
Sommerville	1	3-12

Bibliografía básica

Pressman, R. (2002). *Ingeniería del* software*. Un enfoque práctico* (5.ª ed.). México: McGraw-Hill.

Sommerville, I. (2005). *Ingeniería de* software (7.ª ed.). España: Addison Wesley / Pearson Educación.

Bibliografía complementaria

Booch, G., Rumbaugh J. y Jacobson, I. (2000). *El proceso unificado de desarrollo de* software. Madrid: Addison Wesley.

Braude J. E. González Osuna, M. (trad.). (2003). *Ingeniería de* software*: una perspectiva orientada a objetos*. México: Alfaomega.

Bruegge, B. y Dutoit, A. Ruiz Faudón, S. (trad.). (2002). *Ingeniería de software orientado a objetos*. México: Prentice Hall / Pearson Educación.

GhezzI, C. *et al.* (1991). *Fundamental of software engineering*. EUA: Editorial Prentice-Hall.

McConnell, S. *Desarrollo y gestión de proyectos informáticos*. McGraw-Hill.

Weitzenfeld, A. (2004). *Ingeniería de* software *orientada a objetos con UML, JAVA e Internet*. México: Thomson.

Sitios electrónicos

Sitio	Descripción
https://law.resource.org/pub/us/cfr/ibr/002/asqc.q9001.1994.pdf	Documento ISO 9001.
http://www.iidia.com.ar/rgm/tesistas/scalone-tesis-maestria-ingenieria-en-calidad.pdf	Estudio comparativo de modelos de calidad para el desarrollo de *software.* Tesis de maestría de Fernanda Escalone. Universidad Tecnológica Nacional, Buenos Aires, Argentina.
http://www.uv.mx/personal/jfernandez/files/2010/07/8_Calidad.pdf	Presentación sobre calidad del *software.* Juan Manuel Fernández Piña, Universidad Veracruzana.

UNIDAD 3

Administración de proyectos

OBJETIVO PARTICULAR

Al finalizar la unidad, el alumno reconocerá el proceso de administración de proyectos para la construcción de *software*.

TEMARIO DETALLADO

(12 horas)

3. Administración de proyectos

3.1. Definición de proyecto
3.2. Proceso de administración de un proyecto
3.3. Integración del proyecto
3.4. Administración del alcance
3.5. Administración de tiempos
3.6. Administración de costos
3.7. Administración de calidad
3.8. Administración de riesgos
3.9. Administración de recursos humanos
3.10. Administración de la comunicación
3.11. Administración de adquisiciones

INTRODUCCIÓN

El desarrollo de un proyecto de ingenierìa de *software* no difiere mucho al de un proyecto en otro campo. En todo caso, es necesario comenzar con una correcta planificación, de modo que se tengan los recursos necesarios para que el proyecto sea completado cabalmente; además de planificar las actividades. Sin embargo, dentro de la ingeniería de *software*, a diferencia de otros proyectos, el producto a desarrollar es intangible, es decir, resulta imposible notar su avance de manera sencilla, hasta que los procesos se comienzan a integrar para dar forma al producto final.

Para la correcta administración de un proyecto de ingeniería de *software,* se deben tomar en cuenta múltiples aspectos, como su planificación general –que visualiza el alcance del proyecto total–, definición y planificación de tareas, tiempos, costos, etcétera, a fin de conformar un producto de calidad.

En esta unidad, se revisarán los conceptos fundamentales de la administración de proyectos de ingeniería de *software*. Se partirá desde la definición misma del proyecto, hasta la administración de los diversos aspectos a considerarse para alcanzar un producto final de calidad.

3.1. Definición de proyecto

En general, un proyecto es una serie de acciones realizadas en un periodo finito con el objetivo de desarrollar un producto o servicio nuevo. En el caso del *software*, un proyecto son las acciones a llevar a cabo para desarrollar un nuevo *software* en un periodo determinado.

Según Pressman (2002, p. 39), la administración de un proyecto de *software* exige considerar cuatro aspectos:

Personal.

Todo proyecto conlleva personal especializado para su realización: elegir al personal adecuado es indispensable para ejecutar el proyecto.

Producto.

Aunque el *software* es el objetivo final de proyecto, es necesario considerar todo lo que va a envolver al producto, como usuarios, medio ambiente de trabajo, funciones, etcétera, para prever alternativas y soluciones a problemas que puedan presentarse.

Proceso.

Los procesos definen el camino para desarrollar de buena forma el producto, por lo que la planificación detallada de cada proceso, sus actividades, el personal, tiempos y costos es crucial para alcanzar las metas y objetivos planteados en el proyecto.

Proyecto.

La integración de los elementos anteriores, la planeación, dirección y control de cada una de las actividades, costos y tiempos asignados a cada proceso, son fundamentales para alcanzar el éxito del proyecto.

A continuación, revisaremos los aspectos principales a considerar y administrar en un proyecto de *software*.

3.2. Proceso de administración de un proyecto

La administración de proyectos es una de las partes medulares de la ingeniería de *software*, toda vez que permite determinar los tiempos de entrega, costos asociados al proyecto, personal requerido, actividades a realizar, etcétera.

Una administración de proyectos correcta conduce a cumplir con lo acordado entre los clientes y los desarrolladores de *software*. En cambio, la mala administración puede llevar al incremento de costos o al incumplimiento de los requerimientos o requisitos.

La administración de proyectos de ingeniería de *software* es similar a la gestión de proyectos en general, aunque su principal diferencia según Sommerville (2005, p. 86), radica en los siguientes puntos, para considerar su aplicación o no:

1. Producto intangible. En comparación con otros proyectos de ingeniería, donde es posible ver e inspeccionar el avance del proceso y detectar las fallas de forma directa, en los proyectos de ingeniería de *software* es imposible realizar lo anterior. Como el *software* es algo intangible, se debe esperar a que los diversos integrantes del proyecto terminen sus actividades para documentarlas y poder realizar una validación conforme a los requerimientos establecidos.

2. *No existen procesos de* software *estándar*. A pesar de que existen diversas metodologías o modelos de desarrollo de *software*, no hay una metodología de desarrollo única; según la complejidad del proyecto y los tiempos de desarrollo, es posible emplear una metodología al cien por ciento, o emplear varias de ellas para llegar al objetivo del proyecto.

3. A menudo los proyectos grandes son únicos. En el 90% de los casos, los proyectos de ingeniería de *software* son desarrollados de manera única para una organización, para cubrir necesidades específicas propias, por lo que los requerimientos, procesos y metodologías empleadas de un proyecto a otro siempre son cambiantes. Lo anterior implica un reto particular para los administradores de proyectos de ingeniería de *software*.

Dentro de las actividades principales de administración de proyectos de ingeniería de *software*, encontramos las siguientes (Sommerville, 2005, p. 87):

- Redacción de la propuesta
- Planificación y calendarización del proyecto
- Estimación de costos del proyecto
- Supervisión y revisión del proyecto
- Selección y evaluación del personal
- Redacción y presentación de informes

3.3. Integración del proyecto

La correcta planificación de un proyecto de *software* permite realizar una integración puntual de todas las actividades y procesos que conforman al proyecto de ingeniería de *software*.

Los proyectos de ingeniería de *software* comienzan con una planificación general que considera la selección de un modelo de desarrollo a partir del análisis de los requerimientos. Con todo, para lograr una integración adecuada de cada fase o proceso del modelo seleccionado, los administradores han de considerar la realización y ejecución de otros planes (Sommerville, 2005, pp. 88-89):

Plan de calidad.	Describe los procedimientos, estándares o buenas prácticas a emplear en el desarrollo del proyecto.
Plan de validación.	Dentro de este plan, se describen las técnicas, tiempos y recursos empleados en la validación del proyecto con respecto a los requerimientos del mismo.
Plan de administración de configuración.	Expone la forma como se estructurará o configurará cada proceso del proyecto, así como las formas en que serán solicitados, reportados y realizados los cambios sobre ellos.
Plan de mantenimiento.	Permite predecir costos, tiempos y recursos requeridos para realizar el mantenimiento del sistema una vez terminado.
Plan de desarrollo personal.	Describe la forma de capacitar y evaluar los recursos humanos necesarios para el proyecto; así como la experiencia que será adquirida durante el desarrollo del mismo.

3.4. Administración del alcance

De acuerdo con el Instituto para la Calidad de la Pontificia Universidad Católica del Perú, el alcance de un proyecto es el proceso de subdividir los entregables principales en componentes administrables con el objetivo de:

- mejorar la exactitud de los estimados de costo y tiempo.
- definir una línea de base para medición y control del proyecto.
- facilitar una clara asignación de roles y responsabilidades.[18]

Con base en lo anterior, dentro de los proyectos de ingeniería de *software*, es necesario definir hitos o puntos finales de cada proceso que conforma al proyecto, en los que debe existir un entregable formal que incluya los avances y logros alcanzados al concluir el proceso.

Para fijar un hito, es indispensable dividir los procesos que integran el proyecto en actividades básicas con salidas bien definidas. La figura siguiente ilustra el establecimiento de hitos dentro del proceso de especificación de requerimientos.

[18] Definición del alcance de un proyecto. Instituto para la Calidad de la Pontificia Universidad Católica del Perú. Disponible en http://calidad.pucp.edu.pe/wiki-calidad/que-es-el-alcance-de-un-proyecto#sthash.tBZ47LSG.dpbs. (Consultado el 17/06/2015).

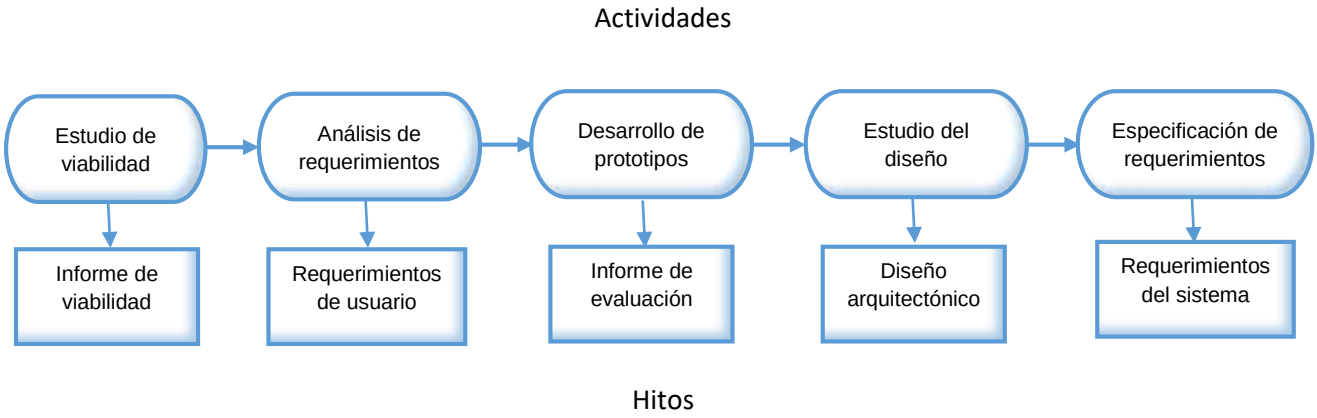


Figura 3.1. Establecimiento de hitos en el proceso de especificación de requerimientos.

FUENTE. Sommerville (2005, p. 91).

El establecimiento de las actividades básicas de cada proceso, los hitos y los entregables, facilitan la planificación de cada proceso y, por tanto, favorecen la correcta administración del proyecto.

3.5. Administración de tiempos

Una de las actividades más complicadas en cualquier proyecto es la estimación de tiempos o calendarización del proyecto. El objetivo de la calendarización es establecer los tiempos en que cada actividad debe ser realizada y la sucesión coherente de cada actividad, lo que ayuda a una asignación de recursos acorde con cada actividad, para completarla en tiempo y forma.

La calendarización del proyecto consiste, principalmente, en identificar las actividades que serán parte del proyecto, establecer sus entradas y salidas, los

recursos involucrados y las dependencias entre cada actividad. Esto último permitirá definir si una o más actividades se pueden realizar de forma paralela a otras, lo que llevará a optimizar el tiempo de desarrollo.

Para facilitar la calendarización, hay herramientas, como gráficas de Gantt, análisis de ruta crítica o redes de actividades, que llevan a dimensionar el proyecto en su totalidad y establecer los tiempos de cada actividad.

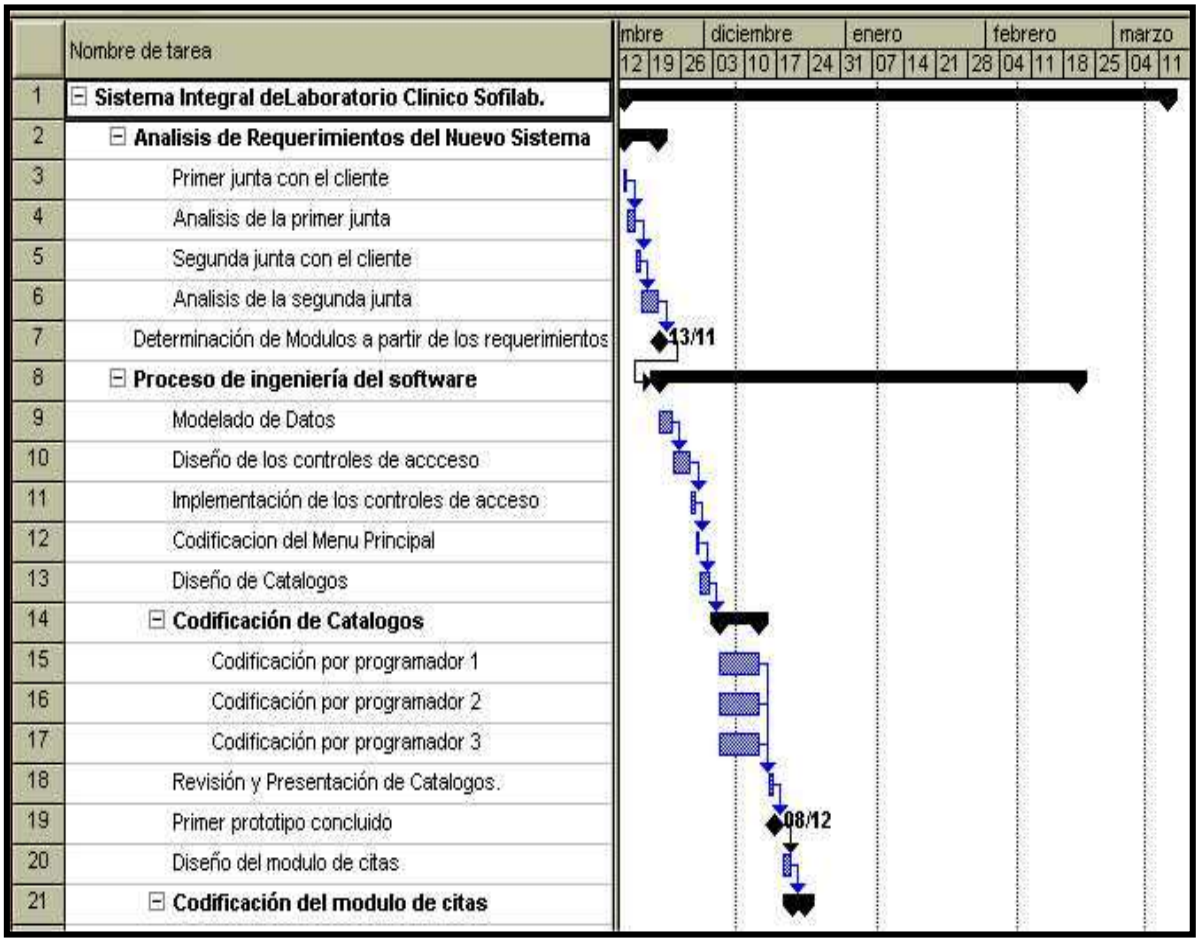


Figura 3.2. Ejemplo de gráfica de Gantt.

FUENTE: Manzano Vega, M. y Montesanos Brand, R. Tesis de licenciatura: Sistema Integral de Laboratorio Clínico Sofilab. Facultad de Ingeniería, UNAM. 2002. P. 37.

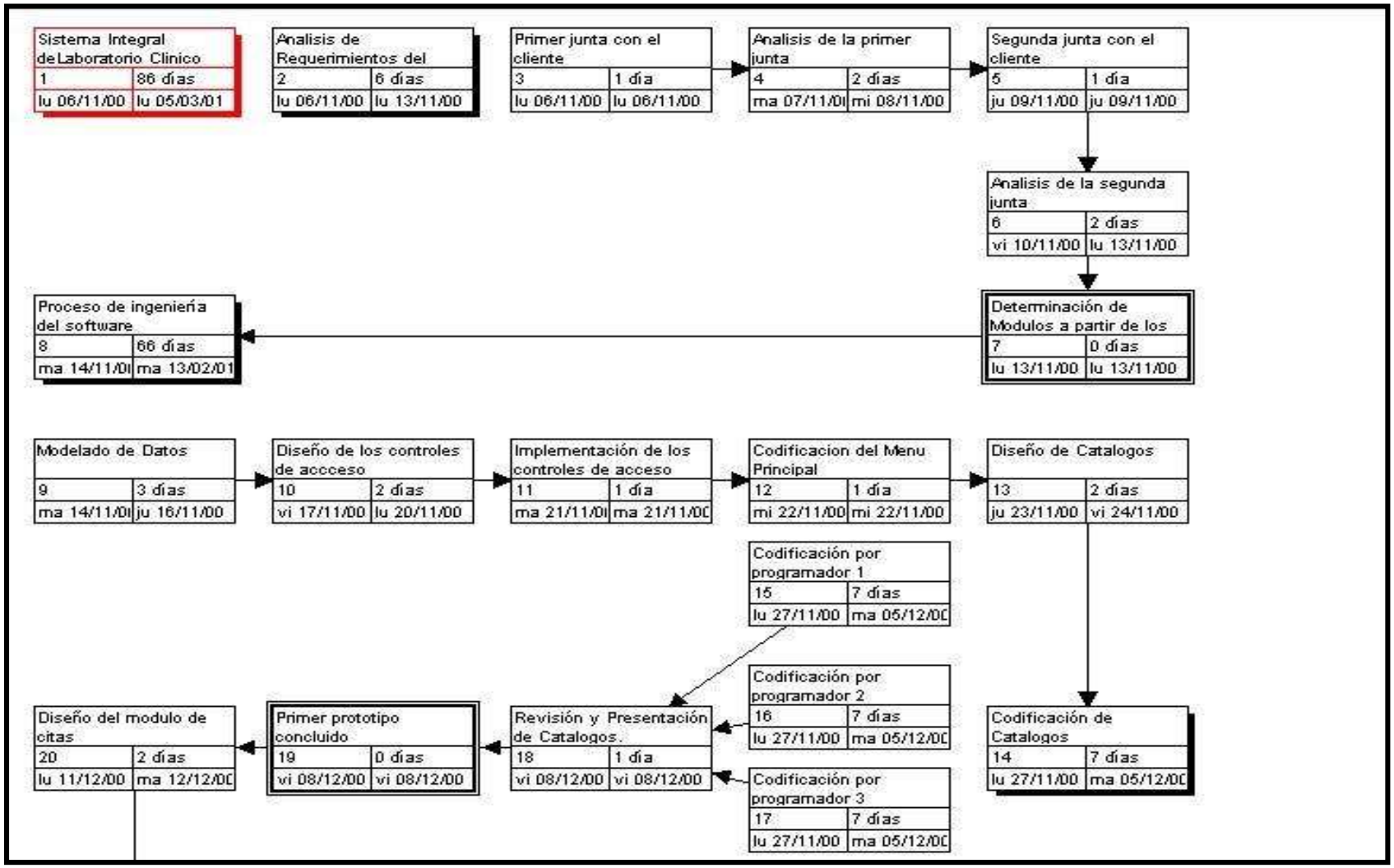


Figura 3.3. Ejemplo de red de tareas.

FUENTE: Manzano Vega, M. y Montesanos Brand, R. Tesis de licenciatura: Sistema Integral de Laboratorio Clínico Sofilab. Facultad de Ingeniería, UNAM. 2002. P. 41.

3.6. Administración de costos

Los costos de un proyecto de ingeniería de *software* es otro punto difícil de administrar: en muchas ocasiones, los proyectos suelen alargarse más del tiempo estimado o requieren de más personal, o incluso regresar y diseñar parte o todo el proyecto.

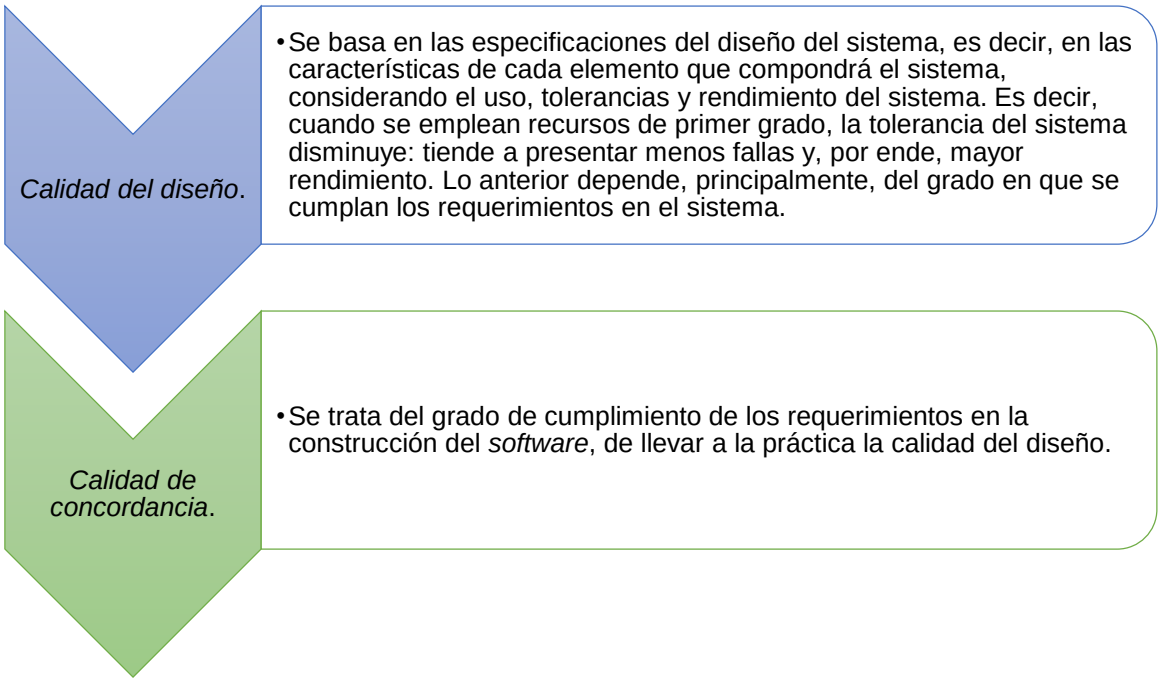
La estimación de costos de un proyecto de *software* puede abordarse a través de cinco modelos descritos a continuación.[19]

Modelos con base histórica.	Son los modelos antiguos y se fundamentan en la estimación de costos a partir de la experiencia previa, ya que basan su metodología en la comparación de proyectos con otros similares para calcular el costo de los mismos.
Modelos con base estadística.	Se apoyan en información histórica, pero tomando como base los datos duros generados por otros proyectos y estableciendo unidades de medida como líneas de código y el esfuerzo (horas-hombre-mes) empleado para su desarrollo. A partir de los datos recabados, se hacen correlaciones que ayudan a determinar el costo del proyecto a partir de fórmulas que relacionan las unidades antes mencionadas.
Teóricos.	Parten de las ideas que se generan para la construcción del proyecto –como el modelo a seguir, estándares a emplear, tiempo y recursos– para establecer las métricas utilizadas en fórmulas que permitan la estimación.
Compuestos.	Combinan el modelo estadístico y el teórico. Inician con la estimación de forma teórica y posteriormente la ajustan mediante datos estadísticos.
Basados en estándares.	Se sustentan en el esfuerzo empleado directamente para el sistema. Establecen una serie de métricas, denominadas puntos de función, que se derivan de la funcionalidad que tendrá el sistema, y a partir de ellas se emplean fórmulas que facilitan la estimación del costo del sistema.

[19] Sevilla Jarquin, P. "Planificación y estimación de proyectos de *software"*. Universidad de Managua, Nicaragua. Disponible en http://sevillajarquin.udem.edu.ni/wp-content/uploads/2014/01/Planificacion-y-Estimacion-de-Proyectos-de-Software.pdf. (Consultado el 17/06/2014).

3.7. Administración de calidad

Para administrar o gestionar la calidad de un proyecto de *software*, Pressman (2002, p. 132) propone dos enfoques:



La calidad del sistema se afirma principalmente en las fases de análisis, especificación de requisitos y diseño del sistema; mientras que la calidad de concordancia, en las fases de desarrollo, pruebas e implementación.

Para que un proyecto de *software* se desarrolle con calidad, son fundamentales una especificación de requerimientos correcta y su incorporación precisa al diseño del sistema.

3.8. Administración de riesgos

Parte de la administración de cualquier tipo de proyectos es la administración o prevención de riesgos que se puedan presentar durante su ejecución. En este orden, el primer paso es la identificación y evaluación de los posibles riesgos.

Sommerville (2005, p. 96) define el riesgo como "una probabilidad de que una circunstancia adversa ocurra". Es, entonces, la estimación de la probabilidad de que se presente cualquier tipo de incidente que incida en el desarrollo del proyecto de ingeniería de *software*.

Hay tres modalidades de riesgos del proyecto (Sommerville, 2005, p. 96):

Riesgos del proyecto.

- Afectan directamente el cumplimiento de las actividades en los tiempos marcados, o los recursos asignados para la realización de los procesos y tareas. Por ejemplo, la salida de alguno de los diseñadores del proyecto o la falta de dinero para la realización de alguna actividad.

Riesgo del producto.

- Impactan el rendimiento del sistema o su calidad. Por ejemplo, la omisión de requerimientos al codificar un módulo.

Riesgos del negocio.

- Factores que perjudican al equipo de desarrollo o a la organización que desarrolla el *software*. Por ejemplo, que otro equipo o empresa trabajen el mismo producto de forma más rápida y con menos recursos.

Aunque los riesgos dependen en gran medida del tipo de proyecto y su entorno de desarrollo, existen algunos que podemos denominar universales, como lo presenta la siguiente tabla.

Rotación de personal	Proyecto	Personal con experiencia abandona el proyecto antes de que finalice
Cambio de gestión	Proyecto	Habrá un cambio de gestión organizacional con diferentes prioridades
No disponibilidad del hardware	Proyecto	El hardware esencial para el proyecto no será entregado a tiempo
Cambio de requerimientos	Proyecto y producto	Habrá más cambios en los requerimientos de lo esperado
Retrasos en la especificación	Proyecto y producto	Las especificaciones de las interfaces esenciales no estarán a tiempo
Subestimación del tamaño	Proyecto y producto	El tamaño del sistema se ha subestimado
Bajo rendimiento de la herramienta CASE	Producto	Las herramientas CASE que ayudan al proyecto no tienen el rendimiento esperado
Cambio de tecnología	Negocio	Un producto competitivo se pone en venta antes de que el sistema se complete
Competencia del producto	Negocio	La tecnología fundamental sobre la que se construirá el sistema se sustituye por nueva tecnología

Tabla de posibles riesgos del *software.*

FUENTE: Sommerville (2005, p. 96).

Etapas para la gestión de riesgo (Sommerville, 2005, p. 96):

Identificación de riesgos.

- Ubica los posibles riesgos en el desarrollo del proyecto, según las tres clasificaciones mencionadas anteriormente.

Análisis de riesgo.

- Se asigna una probabilidad de ocurrencia y se valoran las posibles consecuencias de la ocurrencia de cada riesgo identificado.

Planificación de riesgos.

- Clasificados y evaluados los riesgos, se procede a elaborar estrategias que permitan prevenir o minimizar las consecuencias de los riesgos en caso de que ocurran.

Supervisión de riesgo.

- Es un proceso de mejora continua: se identifican, valoran o revaloran los riesgos conforme se tiene más información de ellos, a fin de mejorar las estrategias de prevención y mitigación.

3.9. Administración de recursos humanos

Los sistemas de información están asociados al factor humano: cualquier sistema de *software* se desarrolla con el factor humano en mente, para facilitar y hacer más eficientes las actividades y procesos dentro de una organización. El desarrollo de un proyecto de ingeniería de *software* es igual, debe ser realizado por personas que tienen diversos roles dentro del proyecto, desde el líder de proyecto, que administra y supervisa el cumplimiento de la planificación, hasta el programador o el *tester*, encargado de construir y probar cada módulo diseñado.

Cuando se trabaja un proyecto de ingeniería de *software*, es necesario considerar los factores tanto humanos como organizacionales que influyen en el entorno de desarrollo del sistema. De acuerdo con Sommerville (2005, p. 32), los principales factores en este aspecto son los siguientes.

Cambios en el proceso.	*Cambios en el trabajo.*	*Cambios organizacionales.*
• Cuando el proyecto requiere que se realicen cambios en uno o varios procesos en el entorno, es importante capacitar a los integrantes del equipo de desarrollo para entender y adaptarse a los nuevos procesos.	• Si el sistema realiza cambios en la forma de ejecutar un proceso o inhabilita a los usuarios, generalmente se presenta resistencia a la aceptación del sistema. En muchas ocasiones, los usuarios, por miedo a perder su trabajo o autoridad, son la principal causa de problemas para implementar el sistema dentro de la organización.	• Cuando el sistema afecta directamente la estructura organizacional, o sea, si le resta peso o importancia a ciertos directivos o áreas, quienes controlan el sistema tienden a adquirir mayor peso político en el esquema organizacional.

La consideración y balance de los factores mencionados dentro de la realización del proyecto de ingeniería de *software* son necesarios y críticos para el cumplimiento de los objetivos del proyecto.

3.10. Administración de la comunicación

Al considerar al recurso humano dentro de un proyecto de ingeniería de *software*, se tomará en cuenta el flujo de comunicación que deberá existir entre los diversos entes participantes en el proyecto.

Los flujos de comunicación son las diversas formas a emplear para integrar a cada uno de los individuos que componen el proyecto. Este flujo puede darse en tres formas básicas:

Descendente.	• Tipo de flujo de comunicación que se da desde el nivel más alto del organigrama del proyecto hacia el nivel más bajo. Son las indicaciones que da el líder del proyecto a los encargados de cada proceso, y los mensajes de los jefes hacia sus subordinados. La función principal de este tipo de flujo de comunicación es coordinar, informar, evaluar y sugerir.
Ascendente.	• Es el flujo contrario al ascendente: la información fluye desde los subordinados hasta el líder de proyecto. Su intención principal es confirmar la recepción del flujo descendente, informar y sugerir.
Horizontal.	• El fin central de este flujo de comunicación es intercambiar ideas, apoyar a compañeros y coordinar actividades. Se da principalmente entre personal de una misma jerarquía o rol en el proyecto.

Para que los procesos de comunicación sean efectivos, debe existir una retroalimentación o confirmación de la recepción de la información.

3.11. Administración de adquisiciones

De acuerdo con el manual PMBOK para la gestión de proyectos, "Administrar las adquisiciones es el proceso que consiste en gestionar las relaciones de adquisiciones, supervisar el desempeño del contrato y efectuar cambios y correcciones según sea necesario. Tanto el comprador como el vendedor administran el contrato de adquisición con finalidades similares. Cada uno debe asegurar que ambas partes cumplan con sus respectivas obligaciones contractuales y que sus propios derechos legales se encuentren protegidos"[20]. En otras palabras, administrar las adquisiciones consiste en establecer un contrato entre un cliente y un proveedor de servicios o materias primas, que debe ser supervisado entre ambas partes para su cumplimiento en tiempo y forma, desde un estricto apego a la legalidad de los términos del contrato.

La administración de adquisiciones permite que el proyecto disponga durante todo su desarrollo de los recursos necesarios para completarlo de forma correcta. Y debe estipular quiénes son los proveedores, así como los tiempos y formas de entrega y pago a los mismos, todo en un marco legal estipulado por los contratos de compra-venta y servicios.

[20] Guía de fundamentos para la administración de proyectos. La Guía PMBok. Gestión para las adquisiciones del proyecto. Sitio en español. Disponible en línea en http://uacm123.weebly.com/9-gestioacuten-de-las-adquisiciones-del-proyecto.html. (Consultado el 20/06/2015).

RESUMEN

La administración de proyectos es una práctica inherente a las actividades de ingeniería de *software*. A través de ella, es posible visualizar la magnitud del proyecto, las actividades y procesos que involucran su desarrollo, tiempos, costos, recursos y personal requeridos para completarlo en tiempo y forma.

La administración de un proyecto comienza con la definición del alcance del mismo. En este momento se definen los procesos que considerará el proyecto a partir de los requerimientos obtenidos y de la metodología de desarrollo a emplear.

Por naturaleza, los proyectos de ingeniería de *software* no cuentan con estándares para su desarrollo. Según el tamaño del proyecto y sus requerimientos, siempre son cambiantes y la metodología empleada no es la misma.

Definidos los procesos y actividades del proyecto, se procede a la planificación temporal. Se identifican los hitos de cada actividad y se ubican las que se pueden realizar de forma simultánea. Luego, se procede a la asignación de recursos y personal, lo que conlleva a la estimación de costos del proyecto.

Otra actividad importante es la administración de riesgos –ningún proyecto está exento de ellos–, a fin de prevenir o minimizar su impacto en el desarrollo del proyecto.

Todas las actividades de la administración de un proyecto de *software* deben enfocarse a la consecución de un producto de calidad que cumpla a cabalidad los requerimientos de los clientes.

MESOGRAFÍA

Bibliografía recomendada

Autor	Capítulo	Páginas
Pressman	5, 25, 26, 27	85-104, 541-604
Sommerville	3-9	37-164

Bibliografía básica

Pressman, R. (2002). *Ingeniería del* software*. Un enfoque práctico* (5.ª ed.). México: McGraw-Hill.

Sommerville, I. (2005). *Ingeniería de* software (7.ª ed.). España: Addison Wesley / Pearson Educación.

Bibliografía complementaria

Booch, G., Rumbaugh J. y Jacobson, I. (2000). *El proceso unificado de desarrollo de* software. Madrid: Addison Wesley.

Braude J. E. González Osuna, M. (trad.). (2003). *Ingeniería de* software*: una perspectiva orientada a objetos*. México: Alfaomega.

Bruegge, B. y Dutoit, A. Ruiz Faudón, S. (trad.). (2002). *Ingeniería de software orientado a objetos*. México: Prentice Hall / Pearson Educación.

GhezzI, C. *et al.* (1991). *Fundamental of software engineering*. EUA: Editorial Prentice-Hall.

McConnell, S. *Desarrollo y gestión de proyectos informáticos*. McGraw-Hill.

Weitzenfeld, A. (2004). *Ingeniería de* software *orientada a objetos con UML, JAVA e Internet*. México: Thomson.

Sitios electrónicos

Sitio	Descripción
http://uacm123.weebly.com/	Manual de PMBOK para la administración de proyectos en español.
http://sevillajarquin.udem.edu.ni/wp-content/uploads/2014/01/Planificacion-y-Estimacion-de-Proyectos-de-Software.pdf.	Pablo Sevilla Jarquín. "Planificación y estimación de proyectos de *software*". Universidad de Managua, Nicaragua.
http://www.gestiopolis.com/estimacion-de-costos-de-desarrollo-de-software/	Roque del Valle Dianelys. Universidad de Camagüey. "Diseño de métricas para la estimación de costos del *software"*.

UNIDAD 4

Verificación y validación

OBJETIVO PARTICULAR

Al finalizar la unidad, el alumno analizará los métodos y técnicas para asegurar la calidad del *software.*

TEMARIO DETALLADO

(8 horas)

4. Verificación y validación

4.1. Estándares para evaluar la madurez de un proceso de desarrollo

4.2. Estándares para evaluar la calidad de un producto de *software*

INTRODUCCIÓN

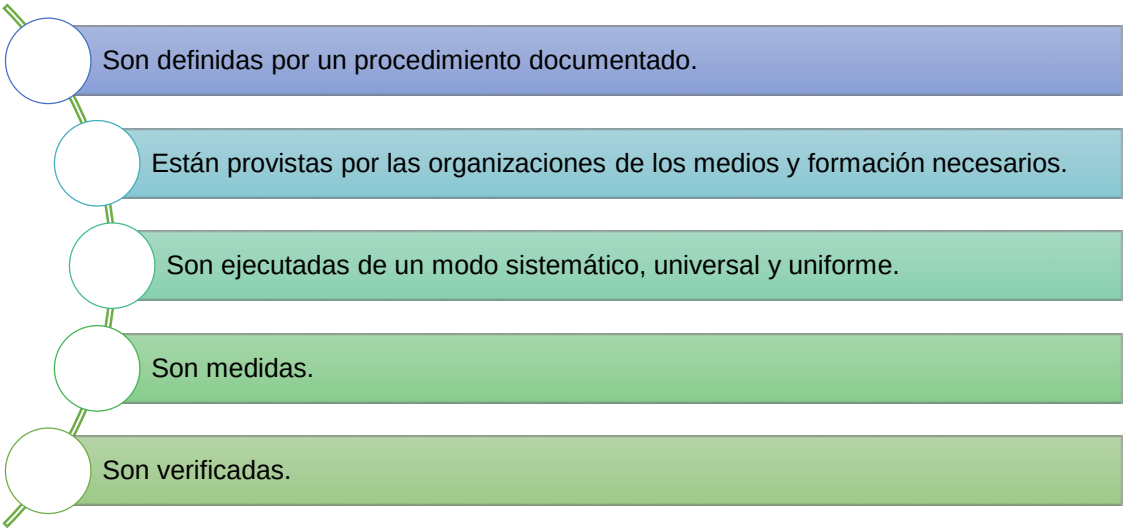
El objetivo principal de la ingeniería de *software* es el desarrollo de *software* de calidad. Recordemos que un *software* de calidad es aquel que cumple en su totalidad con los requerimientos del cliente, y para lograrlo es posible emplear estándares propuestos por diversos organismos, como la ISO, para validar y verificar si se ajusta a las características a lo largo del proceso de su desarrollo en un proyecto de ingeniería de *software*. Asimismo, es factible medir el grado de madurez de los procesos de desarrollo de *software* para garantizar la calidad del producto final.

En esta unidad, se abordarán los estándares establecidos que permiten revisar y evaluar la calidad del *software*.

4.1. Estándares para evaluar la madurez de un proceso de desarrollo

Al hablar de la madurez de un proyecto, se hace referencia a evaluar de forma continua la mejora de los procesos que lo componen. En este sentido, existe el modelo de capacidad y madurez (CMM), que establece una metodología que posibilita evaluar la madurez de los procesos de desarrollo de un proyecto de *software*.

El modelo CMM fue desarrollado por el Instituto de Ingeniería de *Software* (SEI), de la Universidad Carnegie-Mellon, en Estados Unidos, y patrocinado por el Departamento de Defensa. Establece una serie de buenas prácticas agrupadas en áreas clave del proceso, que presentan las siguientes características:



El modelo CMM establece cinco niveles de madurez para cada proceso[21]:

Inicial.	Se identifica que una organización no cuenta con elementos estables para el desarrollo y mantenimiento de proyectos de ingeniería de *software*. En este aspecto, los esfuerzos por seguir una metodología generalmente se ven truncados por la falta de experiencia de la organización y del personal asociado a ella, lo que conduce a mala planeación, retrasos en los tiempos y elevación de costos.
Repetible.	Las organizaciones ya cuentan con experiencia de gestión de proyectos y lo aplican de manera institucionaliza, pero las métricas empleadas para validarlos son básicas y el seguimiento de su calidad es aceptable.
Definido.	Las organizaciones tienen una buena gestión de proyectos, procedimientos específicos para la coordinación de grupos de trabajo, capacitación y formación del personal, técnicas de ingeniería más desarrolladas y métricas de proceso mejor definidas. Emplean técnicas de revisión de actividades entre pares: realizan revisiones entre personal de un mismo nivel en las actividades.
Gestionado.	Las métricas aplicadas en los procesos son más significativas e impactan directamente sobre la calidad del producto y la productividad. Estas métricas también son empleadas en la toma de decisiones y la administración de riesgos dentro del proyecto.
Optimizado.	Las métricas son utilizadas dentro de un proceso de mejora continua de todos los procesos. Hay un proceso de administración de la innovación.

La CMMI 1.2 es la versión actual del modelo CMM, y define las prácticas óptimas para la mejora de procesos en documentos denominados modelos, los cuales se dividen dos:

[21] Información general CMM. Soluciones Informáticas Globales, Segovia, España. Disponible en http://www.globales.es/imagen/internet/Informaci%C3%B3n%20General%20CMMI.pdf. (Consultado el 24/06/2015).

- *CMMI para el desarrollo (DEV-CMMI)*. Trata de procesos de desarrollo de productos y servicios. Liberado en agosto de 2006.

- *CMMI para la adquisición (ACQ-CMMI)*. Se enfoca a la administración de la cadena de suministros, adquisición y contratación externa para procesos de gobierno y la industria. Liberado en noviembre de 2007.

Dentro del modelo CMMI 1.2, se definen las siguientes áreas de proceso[22]:

1. Análisis de causalidad y solución
2. Administración de la configuración
3. Decisión de análisis y resolución
4. Proyecto integrado de gestión
5. Medición y análisis
6. Innovación organizacional y despliegue
7. Definición de procesos organizacionales
8. Enfoque en procesos organizacionales
9. Rendimiento de procesos organizacionales
10. Entrenamiento organizacional
11. Vigilancia y control de proyectos
12. Planificación de proyectos
13. Proceso y aseguramiento de calidad del producto
14. Integración de producto
15. Gestión de proyectos cuantitativos
16. Gestión de requerimientos
17. Requerimientos de desarrollo
18. Gestión de riesgos

[22] Información general CMM. Soluciones Informáticas Globales, Segovia, España. Disponible en http://www.globales.es/imagen/internet/Informaci%C3%B3n%20General%20CMMI.pdf. (Consultado el 24/06/2015).

19. Gestión de proveedores
20. Solución
21. Validación
22. Verificación

ISO 15504 (SPICE, Software Process Improvement and Capability of Determination)

El ISO/IEC 15504 es un estándar internacional para la evaluación y determinación de capacidad y mejora continua de los procesos de la ingeniería de *software*. Su filosofía se basa tanto en el desarrollo de un conjunto de medidas de capacidad estructuradas aplicadas a los procesos del ciclo de vida del proyecto, como en los participantes, a fin de lograr un producto de calidad y mejorar las capacidades de la organización y el personal involucrado.

Características[23]:

- Establece un marco y los requisitos para cualquier proceso de evaluación de procesos y proporciona requisitos para los modelos de evaluación de los procesos.
- Brinda requisitos para cualquier modelo de evaluación de organizaciones.
- Ofrece guías para definir las competencias de un evaluador de procesos.
- Actualmente tiene 10 partes: de la 1 a la 7 completas, y de la 8 a la 10 en fase de desarrollo.
- Comprende la evaluación de procesos, mejora de procesos y determinación de capacidad.
- Proporciona, en su parte 5, un modelo de evaluación de procesos para los procesos de ciclo de vida del *software* definidos en el estándar ISO/IEC

[23] Norma ISO/IEC 15504. Sitio oficial de la Organización Internacional para la Estandarización. (ISO) http://goo.gl/calZuX. (Documento disponible para descarga en pdf. Recuperado el 24/07/2015).

12207, que establece los procesos del ciclo de vida del desarrollo, mantenimiento y operación de los sistemas de *software*.

- En su parte 6, brinda un modelo de evaluación de procesos para los procesos de ciclo de vida del sistema definidos en el estándar ISO/IEC 15288, que puntualiza los procesos del ciclo de vida del desarrollo, mantenimiento y operación de sistemas.
- En su parte 8, proporcionará un modelo de evaluación de procesos para los procesos de servicios TIC que serán definidos en el estándar ISO/IEC 20000-4, que fijará los procesos contenidos en la Norma ISO/IEC 20000-1.
- Equivalencia y compatibilidad con CMMI. ISO forma parte del panel elaborador del modelo CMMI, y SEI mantiene la compatibilidad y equivalencia de esta última con 15504. Sin embargo, CMMI-DEV aún no es un modelo ajustado a esta norma (según lo requiere la Norma ISO 15504 para todo modelo de evaluación de procesos).

Dentro de la norma, se definen seis niveles de madurez de proceso[24]:

Nivel 0 o incompleto.	*Nivel 1 o básico.*	*Nivel 2 o administrado*.
•Cuando la organización tiene fallos evidentes en los procesos o no son implementados completamente.	•Los objetivos de los procesos son alcanzados y estos son implementados de forma básica pero completa, aunque generalmente no se le da un seguimiento riguroso a los objetivos alcanzados.	•Los productos son desarrollados según los requerimientos, y los procesos son vigilados, administrados y planificados.

9 Process Quality Levels of ISO/IEC 15504, CMMI and K-Models. International Journal of Software Engineering and its Applications. 2009. Disponible en http://www.sersc.org/journals/IJSEIA/vol3_no1_2009/4.pdf. (Consultado el 25/07/2015).

Nivel 3 o establecido.	*Nivel 4 o predecible*.	*Nivel 5 u óptimo*.
•Los procesos son ejecutados y administrados de acuerdo con los estándares de la norma.	•Los procesos son ejecutados desde controles estrictos de vigilancia de cumplimiento de los límites de la norma para alcanzar los objetivos definidos de cada proceso.	•Los procesos son sometidos a mejoras continuas con el propósito de cumplir con los objetivos del negocio, respondiendo a sus necesidades actuales y futuras.

4.2. Estándares para evaluar la calidad de un producto de *software*

ISO/IEC 91261:2001

El estándar o Norma ISO/IEC 91261:2001 define un modelo de calidad que proporciona las características que ha de cumplir el *software* para ser considerado un producto de calidad. Este estándar se divide en cuatro partes principales, enunciadas a continuación.

ISO/IEC 91261-1. Define el modelo de calidad y proporciona sus características[25]:

Usabilidad.	Capacidad del producto para ser utilizado por los usuarios de forma fluida en un entorno definido de trabajo.
Funcionalidad.	Capacidad del producto de cumplir con los requerimientos de los usuarios al ejecutar sus funciones en condiciones específicas.
Confiabilidad.	Capacidad del producto de mantener un nivel mínimo de rendimiento en condiciones de trabajo específicas.
Eficiencia.	Capacidad de uso de los recursos dispuestos para el producto (tiempo de respuesta, espacio de almacenamiento, etcétera) en condiciones de trabajo específicas.
Mantenimiento.	Capacidad del producto para ser modificado y probado sin perder su función original.
Portabilidad.	Capacidad del producto para ser instalado y empleado en diferentes plataformas operativas y entornos de trabajo.

ISO/IEC TR 91261-2. Establece métricas para medir la calidad interna y externa del producto, con base en los seis aspectos definidos en la Norma 91261-1.

La calidad interna hace referencia a los módulos o funciones que forman parte del diseño del producto. El estándar 91261 propone que es posible medir y evaluar esta calidad a partir de una serie de atributos estáticos[26]:

- Especificación de requerimientos

[25] Alfonso, P. "Revisión de modelos para evaluar la calidad de productos Web. Experimentación en portales bancarios del NEA". Universidad Nacional de la Plata. Argentina, 2012. Disponible en http://sedici.unlp.edu.ar/bitstream/handle/10915/19878/Documento_completo.pdf. (Consultado el 25/07/2015).
[26] Alfonso, P. "Revisión de modelos para evaluar la calidad de productos Web. Experimentación en portales bancarios del NEA". Universidad Nacional de la Plata. Argentina, 2012. Disponible en http://sedici.unlp.edu.ar/bitstream/handle/10915/19878/Documento_completo.pdf. (Consultado el 25/07/2015).

- Diseño
- Código fuente
- Diccionario de datos

La calidad interna de un producto de *software* puede ser medida y evaluada desde los procesos iniciales del ciclo de vida del proyecto, aunque esto no garantiza la calidad del sistema.

La calidad externa se refiere a la forma como el sistema o producto ejecuta sus funciones para cumplir con los requerimientos establecidos. Esta calidad se mide y evalúa a través de atributos o propiedades dinámicas del producto, es decir, a partir de su desempeño al ser ejecutado en entornos similares al real donde será instalado, lo que permite verificar, por ejemplo, la entrega de reportes, captura de datos, etcétera.

ISO/IEC TR 91261-3.	***ISO/IEC TR 91261-4.***
Dispone métricas internas para medir y evaluar las seis características mencionadas en la Norma 91261-1.	Establece métricas para medir la calidad de uso del sistema.

La Norma 91261-4 ofrece las siguientes definiciones de la calidad de uso[27]:

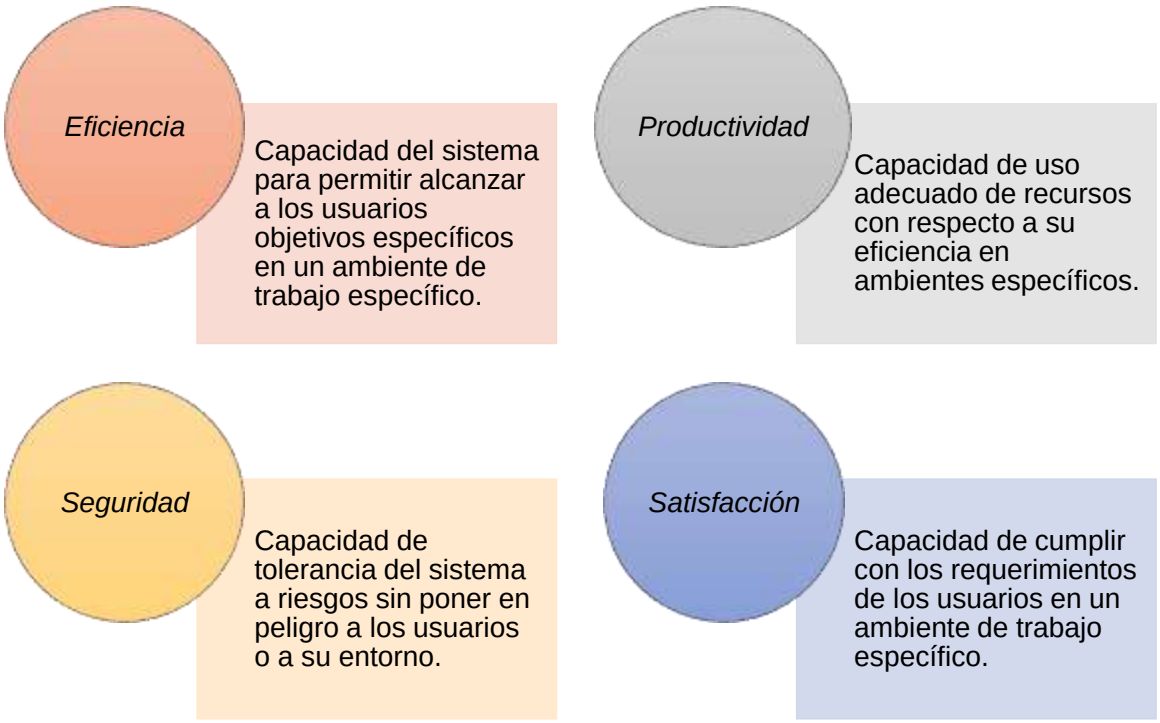
- "La capacidad de un producto de *software* de facilitar a usuarios específicos alcanzar metas específicas con eficacia, productividad, seguridad y satisfacción en un contexto específico de uso".

[27] Alfonso, P. "Revisión de modelos para evaluar la calidad de productos Web. Experimentación en portales bancarios del NEA". Universidad Nacional de la Plata. Argentina, 2012. Disponible en http://sedici.unlp.edu.ar/bitstream/handle/10915/19878/Documento_completo.pdf. (Consultado el 25/07/2015).

- “Calidad en uso es la visión de calidad de los usuarios de un ambiente conteniendo *software*, y es medida sobre los resultados de usar el *software* en el ambiente, antes que sobre las propiedades del *software* en sí mismo".

En otras palabras, es la capacidad del producto que permite a sus usuarios alcanzar sus objetivos de forma eficiente, con un producto que garantiza productividad, seguridad y satisfacción.

La Norma 91261-4 agrupa las métricas de uso en cuatro rubros[28]:



[28] Alfonso, P. “Revisión de modelos para evaluar la calidad de productos Web. Experimentación en portales bancarios del NEA”. Universidad Nacional de la Plata. Argentina, 2012. Disponible en http://sedici.unlp.edu.ar/bitstream/handle/10915/19878/Documento_completo.pdf. (Consultado el 25/07/2015).

ISO/IEC 25000:2005 – SquaRE

El estándar ISO/IEC 25000:2005 ofrece una guía para el seguimiento e implementación de los estándares 9126 y 14500 en el desarrollo de *software*. Considera la especificación y evaluación de los requisitos o requerimientos como punto focal de la calidad de *software*.

El estándar 25000 se divide en cinco familias[29]:

- ISO 2500N, enfocado a la gestión de calidad en los procesos
- ISO 2501N, establece el modelo de calidad a seguir
- ISO 2502N, establece las medidas de calidad
- ISO 2503N, define los requerimientos de calidad
- ISO 2504N, establece la evaluación de la calidad

La ventaja del estándar ISO/IEC 25000 es que explica cómo se da la transición de las normas 9126 y 14500 a la 25000. Esto permite englobar ambos enfoques en las cinco familias que componen la ISO 25000.

La siguiente imagen esquematiza los puntos que abarca la Norma 25000.

[29] Alfonso, P. "Revisión de modelos para evaluar la calidad de productos Web. Experimentación en portales bancarios del NEA". Universidad Nacional de la Plata. Argentina, 2012. Disponible en http://sedici.unlp.edu.ar/bitstream/handle/10915/19878/Documento_completo.pdf. (Consultado el 25/07/2015).

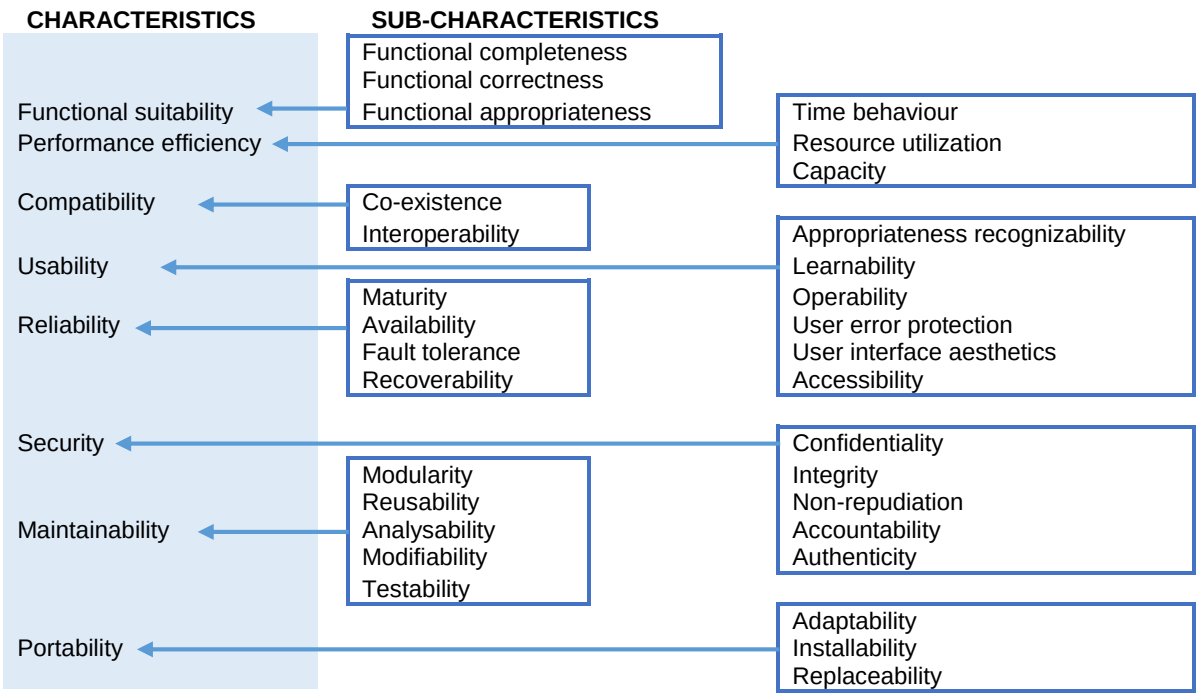


Figura 4.1. Modelo de calidad ISO/IEC 25000.

FUENTE: Polillo R. (2011). Quality Models for Web 2.0 Sites: a Methodological Approach and a Proposal". *2nd Workshop on The Web and Requirements Engineering (WeRE'11) In* (ICWE 2011).

RESUMEN

Los modelos o estándares de calidad para la ingeniería de *software* proponen una serie de buenas prácticas que los desarrolladores de *software* deben implementar en la medida de lo posible.

La calidad del *software* es alcanzable a partir de dos enfoques, el de maduración de procesos y el de evaluación de la calidad del producto terminado. En el primer caso, se mide el grado en que los desarrolladores o empresas implementan estándares y métricas que permiten evaluar la madurez de un proceso, es decir, el grado de cumplimiento e implementación de buenas prácticas en los procesos de desarrollo de *software* que componen el ciclo de vida del *software.*

Dependiendo de si se sigue el modelo CMMI o la Norma ISO 15504, es posible determinar de cinco a seis niveles de maduración de un proceso de desarrollo de *software*: desde la nula o casi nula del seguimiento de normas y métricas, hasta la total implementación, administración y mejora de ellas en los procesos encauzados a la mejora continua.

El segundo enfoque de calidad, del producto, basa sus criterios en la evaluación del *software* en diversos puntos: confiabilidad, eficiencia, satisfacción, etcétera, que llevan a establecer si el *software* cumple con los requerimientos establecidos por los clientes en las condiciones de trabajo indicadas.

MESOGRAFÍA

Bibliografía recomendada

Autor	Capítulo	Páginas
Pressman	27, 28	587 - 626
Sommerville	8	131 - 150

Bibliografía básica

Pressman, R. (2002). *Ingeniería del* software*. Un enfoque práctico* (5.ª ed.). México: McGraw-Hill.

Sommerville, I. (2005). *Ingeniería de* software (7.ª ed.). España: Addison Wesley / Pearson Educación.

Bibliografía complementaria

Booch, G., Rumbaugh J. y Jacobson, I. (2000). *El proceso unificado de desarrollo de* software. Madrid: Addison Wesley.

Braude J. E. González Osuna, M. (trad.). (2003). *Ingeniería de* software*: una perspectiva orientada a objetos*. México: Alfaomega.

Bruegge, B. y Dutoit, A. Ruiz Faudón, S. (trad.). (2002). *Ingeniería de software orientado a objetos*. México: Prentice Hall / Pearson Educación.

GhezzI, C. *et al.* (1991). *Fundamental of software engineering*. EUA: Editorial Prentice-Hall.

McConnell, S. *Desarrollo y gestión de proyectos informáticos*. McGraw-Hill.

Sitios electrónicos

Sitio	Descripción
http://sedici.unlp.edu.ar/bitstream/handle/10915/19878/Documento_completo.pdf.	Pedro Luis Alfonso. “Revisión de modelos para evaluar la calidad de productos web. Experimentación en portales bancarios del NEA”. Universidad Nacional de la Plata, Argentina. 2012.
http://www.sersc.org/journals/IJSEIA/vol3_no1_2009/4.pdf.	Process Quality Levels of ISO/IEC 15504, CMMI and K-Models. International Journal of Software Engineering and its Applications. 2009.

UNIDAD 5

Métricas

OBJETIVO PARTICULAR

Al finaliza la unidad, el alumno analizará las métricas de producto que existen para asegurar la calidad del *software.*

TEMARIO DETALLADO

(8 horas)

5. Métricas

5.1. Definición
5.2. Clasificación
5.3. Categorías
5.4. Estimación de esfuerzo y costo

INTRODUCCIÓN

La calidad en cualquier producto o proyecto es un factor que debe poder medirse. Para este fin es necesario establecer una serie de factores que permitan obtener valores medibles dentro de ellos, pues a través de estos valores determinamos si se cumple o no con un criterio de calidad.

La ingeniería de *software* facilita establecer diversas métricas que conducen a la evaluación de un producto de *software* de manera confiable, lo que implica determinar si es de calidad o no.

En esta unidad, se revisarán los conceptos principales sobre métricas que pueden ser empleadas en un producto de *software*, ya sea desde el punto de vista del sistema en funcionamiento o de sus partes estáticas, como diseño, requerimientos, etcétera.

5.1. Definición

Para determinar que un *software* es de calidad, es necesario poder medir el producto de *software* y precisar si cumple o no con los criterios de calidad establecidos. Para ello es necesario saber qué aspectos medir y cómo hacerlo.

De acuerdo con Sommerville (2005, p. 599), “La medición del *software* se refiere a derivar un valor numérico desde algún atributo del *software* o del proceso del *software*”. En otras palabras, es necesario establecer un valor numérico a los procesos o las formas como son desarrollados para poder evaluarlos; además, para llevar a cabo dicha evaluación, es indispensable comparar ese valor con los criterios o normas de calidad que se estén siguiendo en el desarrollo del *software*.

Ahora bien, para realizar una medición se requiere definir métricas. Una métrica es “cualquier medida relacionada con un sistema, proceso o documentación del *software*” (Sommerville, 2005, p. 599). La métrica es, entonces, el valor numérico asociado a un proceso, documento o al mismo producto de *software*. Y el valor numérico permite evaluar al *software* y determinar su calidad.

5.2. Clasificación

Métricas dinámicas

Son aquellos valores obtenidos a partir de la medición del *software* en ejecución.

Métricas estáticas

Son aquellos valores que es posible obtener de los aspectos del *software* que definen su estructura, como su diseño, requerimientos, documentación o el mismo código fuente.

5.3. Categorías

Dentro de las diversas métricas a emplear, encontramos cinco categorías principales.[30]

[30] Métricas de calidad del *software*. Laboratorio docente de computación (LDC), Universidad Simón Bolívar, Venezuela. Disponible en http://ldc.usb.ve/~abianc/materias/ci4712/metricas.pdf (Consultado el 24/07/2015).

Complejidad

- Métricas asociadas a la complejidad del proyecto, como volumen, tamaño, configuración, etcétera.

Calidad

- Métricas vinculadas a la calidad del *software* como modularidad, pruebas, mantenibilidad, estructura, etcétera.

Competencia

- Métricas relacionadas con la medición de las capacidades de los programadores y desarrolladores, como eficiencia, capacidad, rapidez, etcétera.

Desempeño

- Métricas enfocadas a medir la conducta del *software*, como su desempeño bajo un sistema operativo, tasa de transferencia de datos en una red, tolerancia a cambios de *hardware*, etcétera.

Estilizadas

- Métricas enfocadas a medir los estilos o formas de desarrollo del *software*, como estilo de programación, uso de convenciones, etcétera.

5.4. Estimación de esfuerzo y costo

Las métricas del *software* también permiten estimar el esfuerzo invertido en el desarrollo del *software* y su costo asociado. En este orden, es posible emplear dos enfoques de métricas, descritos a continuación.

Por tamaño

Las métricas por tamaño utilizan, principalmente, la experiencia de los involucrados en el desarrollo del *software* para establecer y estimar las métricas. Su base es el número de líneas de código aproximadas a emplear en el desarrollo.

En un sistema que no se ha construido, es necesario tomar como base proyectos anteriores bien documentados, donde parte de la documentación sea el número de líneas de código generadas en dichos proyectos.

Proyecto	LDC	Esfuerzo	Coste £(000)	Pag. Doc.	Errores	Defectos	Personas
Alfa	12,100	24	168	365	134	29	3
Beta	27,200	62	440	1224	321	86	5
Gamma	20,200	43	314	1050	256	64	6
•		•	•	•			
			•		•		

Figura 5.1. Ejemplo de estimación por tamaño.

FUENTE: Pressman (2002, p. 59).

En la figura anterior, el apartado LDC hace referencia a las líneas de código generadas en cada proyecto registrado por un equipo de desarrollo. A partir de ellas, es posible realizar la estimación de varios factores, como esfuerzo, costo, número de páginas por documentos, errores y defectos del sistema. Para este fin se emplean las siguientes fórmulas:

Esfuerzo = KLDC/persona-mes

Calidad = errores/KLDC

Documentación = pags. Doc/ KLDC

Costo = $/KLDC

Donde KLDC se refiere a las miles de líneas de código empleadas.

Por ejemplo, supongamos que tenemos un proyecto de tamaño mediano a tres meses de duración, con cuatro personas trabajando en él.

Considerando la tabla anterior, un proyecto mediano puede estimarse tomando 20,200 LDC como base.

Así entonces, al aplicar las fórmulas, tenemos:

Esfuerzo = 20.2/3 = 6.7 horas promedio al día

Calidad = 256/20.3 = 12.61 porcentaje de errores y defectos

Documentación = 1050/20.3 = 51.98 páginas diarias promedio

Costo = 17/20.3 = $0.83 por línea de código

Por función

El segundo enfoque no requiere experiencia, pero sí recabar mayor información para la estimación del proyecto.

Este tipo de métricas se centran en la funcionalidad o utilidad del sistema. Se inicia por obtener un sistema de valores, denominados puntos por función, los cuales se conocen al llenar la siguiente tabla:

	FACTOR DE PONDERACIÓN						
Parámetro de medición	**Cuenta**		**Simple**	**Medio**	**Complejo**		
Número de entradas de usuario		X	3	4	6	=	
Número de salidas de usuario		X	4	5	7	=	
Número de peticiones de usuario		X	3	4	6	=	
Número de archivos		X	7	10	15	=	
Número de interfaces externas		X	5	7	10	=	
Cuenta = total							

Figura 5.2. Tabla de ponderación de puntos por función.

FUENTE: Pressman (2002, p. 59).

Los valores de la tabla se llenan según el diseño del sistema, donde es posible obtener la información estimada de lo que el *software* generará durante su funcionamiento.

Luego, se deben obtener una serie de valores complementarios, denominados valores de ajuste de complejidad (fi), a partir del siguiente cuestionario:

Fi:
1. ¿Requiere el sistema copias de seguridad y recuperación fiables?
2. ¿Se requiere comunicación de datos?
3. ¿Existen funciones de procesamiento distribuido?
4. ¿Es critico el rendimiento?
5. ¿Será ejecutado el sistema en un entorno operativo existente y frecuentemente utilizado?
6. ¿Requiere el sistema entrada de datos interactivo?
7. ¿Requiere la entrada de datos interactivo que las transiciones de entrada se lleven a cabo sobre múltiples o variadas operaciones?
8. ¿Se actualizan los archivos maestros en forma interactiva?
9. ¿Son complejas las entradas, las salidas, los archivos o peticiones?
10. ¿Es complejo el procesamiento interno?
11. ¿Se ha diseñado el código para ser reutilizable?
12. ¿Están incluidos en el diseño la conversión y la instalación?
13. ¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones?
14. ¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

Figura 5.3. Cuestionario de valores de ajuste.

FUENTE: Pressman (2002, p. 59).

Las respuestas del cuestionario se ponderan a partir de la siguiente escala:

0	1	2	3	4	5
Sin influencia	Incidental	Moderado	Medio	Significativo	Esencial

Ponderación Fi.

FUENTE: Pressman (2002, p. 59).

Obtenidos ambos valores, se procede a conocer los puntos por función empleando la siguiente fórmula:

PF = CUENTA_TOTAL * [0.65 + 0.01 * SUM(fi)]

Identificados los puntos, se aplican las siguientes fórmulas, similares a las empleadas en el enfoque por tamaño:

Esfuerzo = PF / persona-mes
Calidad = Errores / PF
Costo = Dólares / PF
Documentación = Pags. Doc / PF

El enfoque por función es más útil cuando no se cuenta con experiencia previa o documentación sobre el tamaño del *software* anterior.

RESUMEN

Las métricas son valores asociados a diversos aspectos medibles de un producto de *software*, y pueden ser dinámicas o estáticas, lo que dependerá de los aspectos del *software* que se desee medir.

Al ser elementos que miden diversos aspectos de un producto de *software*, las métricas se clasifican de diversas formas. En todo caso, lo que se busca con ellas es garantizar la calidad del *software*.

Dentro de las diversas métricas, podemos encontrar dos enfoques diferentes para estimar el costo y el esfuerzo necesarios para desarrollar un nuevo producto de *software*. Estos enfoques son por tamaño, encauzados principalmente a la estimación fundamentada en las líneas de código generadas en proyectos similares; o por función, concentrados en la estimación basada en la funcionalidad del sistema. Ambos enfoques ayudan a los desarrolladores con o sin experiencia.

MESOGRAFÍA

Bibliografía recomendada

Autor	Capítulo	Páginas
Pressman	4	53-75
Sommerville	27	587-625

Bibliografía básica

Pressman, R. (2002). *Ingeniería del* software*. Un enfoque práctico* (5.ª ed.). México: McGraw-Hill.

Sommerville, I. (2005). *Ingeniería de* software (7.ª ed.). España: Addison Wesley / Pearson Educación.

Bibliografía complementaria

Booch, G., Rumbaugh J. y Jacobson, I. (2000). *El proceso unificado de desarrollo de* software. Madrid: Addison Wesley.

Braude J. E. González Osuna, M. (trad.). (2003). *Ingeniería de* software*: una perspectiva orientada a objetos*. México: Alfaomega.

Bruegge, B. y Dutoit, A. Ruiz Faudón, S. (trad.). (2002). *Ingeniería de software orientado a objetos*. México: Prentice Hall / Pearson Educación.

GhezzI, C. *et al.* (1991). *Fundamental of software engineering*. EUA: Editorial Prentice-Hall.

McConnell, S. *Desarrollo y gestión de proyectos informáticos*. McGraw-Hill.

Sitios electrónicos

Sitio	Descripción
http://ldc.usb.ve/~abianc/materias/ci4712/metricas.pdf	Métricas de calidad del *software*. Universidad Simón Bolívar, Venezuela.
http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/Metricas4.pdf	Métricas del *software*. Conceptos básicos, definición y formalización. Universidad del Valle. Colombia.

UNIDAD 6

Liberación y mantenimiento

OBJETIVO PARTICULAR

Al término de la presente unidad, el alumno analizará el proceso y los tipos de mantenimiento del *software*.

TEMARIO DETALLADO

(8 horas)

6. Liberación y mantenimiento

6.1. Implantación del proceso de desarrollo
6.2. Mejora del proceso de desarrollo

INTRODUCCIÓN

La fase de liberación y mantenimiento es la culminación del proyecto de desarrollo de *software* y, en consecuencia, el final y reinicio del ciclo de vida o modelo de desarrollo seleccionado.

Las actividades de liberación consisten, básicamente, en la revisión y prueba del producto de *software* concluido. Para esto se realizarán pruebas que permitan verificar y validar los requerimientos que fueron establecidos inicialmente por los clientes.

Luego de la liberación, se harán actividades de mantenimiento preventivo y correctivo. Esto permitirá detectar errores en el *software* una vez puesto en marcha y al mismo tiempo mejorarlo, lo que a la larga llevará a superar el mismo diseño, pensando en versiones nuevas del *software* o en la concepción de *software* similar.

En esta unidad, se analizarán los conceptos fundamentales para llevar a cabo las actividades de liberación y mantenimiento del *software*.

6.1. Implantación del proceso de desarrollo

La implantación del proceso de desarrollo consiste en la verificación y validación de los requerimientos establecidos por los clientes dentro del producto finalizado.

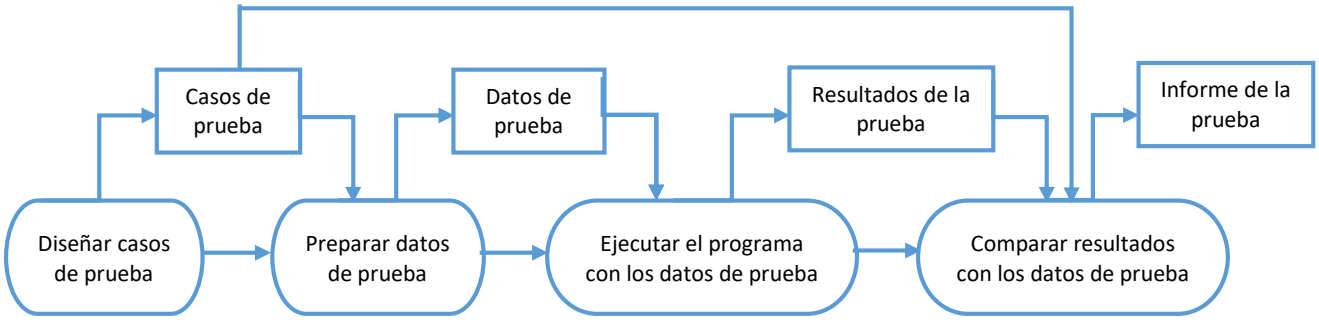
Antes de realizar la entrega del producto de *software*, es necesario realizarle diversas pruebas que permitan al equipo de desarrollo verificar y validar los requerimientos recabados en la fase de análisis e incorporados en la etapa de diseño al *software* dentro del *software* ya terminado. Lo anterior a fin de realizar los ajustes y correcciones para dar un producto de calidad que cumpla con las expectativas del cliente.

Dentro de la ingeniería de *software*, existen diversos tipos de pruebas para verificar que el *software* está en concordancia con los requisitos del cliente:

Pruebas de caja blanca.	En esta clase de pruebas se revisan los módulos o componentes del *software* para verificar que realizan sus funciones y que el proceso de la información es correcto. En estas pruebas se examina el código en ejecución con el objetivo de depurarlo y validar su funcionamiento.
Pruebas de caja negra.	Se enfocan principalmente a verificar el *software* en su conjunto. Se confirma que los datos de entrada y salida del *software* corresponden a los requerimientos; asimismo, se verifica que el *software* trabaja correctamente en el sistema o plataforma operativa especificada e interactúa correctamente con otros sistemas (si se ha requerido así). Esta clase de pruebas son comunes al aproximarse la entrega final del *software* o de un prototipo funcional cercano a la versión final (denominada también versión beta del *software*).

Como todas las actividades del desarrollo del proyecto de *software*, es necesario planificar y documentar las pruebas que serán realizadas al *software*. Según Sommerville (2005, p. 493), la planificación de una prueba llevará esta secuencia de actividades:

- Diseñar el caso de prueba
- Preparar los datos de prueba
- Ejecutar el programa con los datos de prueba
- Comprobar los resultados con los datos de prueba
- Elaborar el reporte de prueba correspondiente



Modelo de proceso de prueba de *software.*

FUENTE: Sommerville (2005, p. 495).

Un caso de prueba es un enfoque para validar los requerimientos a través de pruebas de partes específicas del *software,* y permite revelar problemas en la incorporación de requerimientos que impacten al *software* en su versión final.[31]

[31] Para mayor detalle del desarrollo de un caso de prueba, consulta el tema 4.3 del *Apunte de Informática II*.

Prueba 4. Prueba de la validez de la tarjeta de crédito
Entrada: Una cadena que representa el número de tarjeta de crédito y dos enteros que representan el mes y el año de caducidad de la tarjeta. **Pruebas:** Comprobar que todos los *bytes* de la cadena son dígitos. Comprobar que el mes se encuentra entre 1 y 12 y que el año es mayor o igual que el año actual. Con los 4 primeros dígitos del número de tarjeta de crédito, comprobar que el emisor de la tarjeta es válido consultando la tabla de emisores de tarjetas. Corroborar la validez de la tarjeta de crédito enviando el número de tarjeta y la fecha que caduca el emisor de la tarjeta. **Salida:** OK o un mensaje de error indicando que la tarjeta no es válida.

Ejemplo de un caso de prueba.

FUENTE: Sommerville (2005, p. 368).

Según el enfoque de desarrollo que se seleccione, será necesario diseñar diversos tipos de pruebas para garantizar la calidad del *software*.

Realizadas las pruebas y liberado el programa, continúa su entrega con los siguientes documentos:

Manual de usuario.	*Manual técnico.*
• Documento que describe la forma como el usuario instalará y utilizará el *software*. Por lo general, se redacta de manera simple con bajo nivel técnico, considerando siempre al usuario final que interactuará con el *software*.	• Contiene una copia de toda la documentación generada a lo largo del proceso de desarrollo. Incluye el análisis realizado, diseño del sistema, diccionario de datos, código fuente, casos de prueba documentados, etcétera.

6.2. Mejora del proceso de desarrollo

La mejora del proceso para el desarrollo del *software* se realiza siempre al final del ciclo del modelo de desarrollo seleccionado. Durante la fase de mantenimiento del producto, se evalúa y planifican las mejoras para cuando se dé el siguiente ciclo de desarrollo.

El mantenimiento presenta dos modalidades:

Mantenimiento preventivo.	Se enfoca a prevenir problemas en el funcionamiento del *software*. Algunas acciones de este tipo son, por ejemplo, respaldos de la base de datos, depuración de memoria temporal usada por el *software*, etcétera.
Mantenimiento correctivo.	Se encauza a corregir problemas detectados conforme se usa el *software*. En este proceso, se identifican problemas de código no considerados en las pruebas, se corrigen vulnerabilidades que ponen en riesgo la seguridad del *software*, entre otros.

Con base en los problemas detectados a lo largo del proceso de mantenimiento, se decide crear nuevas versiones del *software* en uso o desarrollar uno completamente nuevo. Así, la información y experiencias recabadas durante el mantenimiento son incorporadas en la estrategia de desarrollo, para mejorar el ciclo completo de forma continua.

RESUMEN

La finalización de un proyecto de desarrollo de *software* conlleva dos actividades, la liberación del *software* y su mantenimiento. La fase de liberación del *software* consiste en actividades que ayudan a verificar la correcta implementación o seguimiento del modelo de desarrollo seleccionado. Para alcanzar lo anterior, es indispensable realizar una serie de pruebas al *software* que permitan verificar la adecuada implementación de los requerimientos iníciales analizados e incorporados durante la fase análisis y diseño.

Las pruebas ejecutadas de forma general antes de liberar el producto son de caja blanca y caja negra. Es decir, se prueba integralmente la estructura interna del *software*; y posteriormente, las entradas, salidas y la interacción del *software* con su plataforma operativa y otros programas que puedan interactuar con él.

Después de la liberación, comienza la etapa de mantenimiento, que en sus dos formas (preventivo y correctivo) extienden la vida útil del *software* y ayuda a determinar si es necesario trabajar una nueva versión del mismo o uno completamente nuevo. Lo anterior ayuda a mejorar el proceso y, en consecuencia, a perfeccionar nuevos proyectos.

MESOGRAFÍA

Bibliografía recomendada

Autor	Capítulo	Páginas
Pressman	17, 23	281-304, 407-420
Sommerville	23, 28	491-518, 626-623

Bibliografía básica

Pressman, R. (2002). *Ingeniería del* software*. Un enfoque práctico* (5.ª ed.). México: McGraw-Hill.

Sommerville, I. (2005). *Ingeniería de* software (7.ª ed.). España: Addison Wesley / Pearson Educación.

Bibliografía complementaria

Booch, G., Rumbaugh J. y Jacobson, I. (2000). *El proceso unificado de desarrollo de* software. Madrid: Addison Wesley.

Braude J. E. González Osuna, M. (trad.). (2003). *Ingeniería de* software*: una perspectiva orientada a objetos*. México: Alfaomega.

Bruegge, B. y Dutoit, A. Ruiz Faudón, S. (trad.). (2002). *Ingeniería de software orientado a objetos*. México: Prentice Hall / Pearson Educación.

GhezzI, C. *et al.* (1991). *Fundamental of software engineering*. EUA: Editorial Prentice-Hall.

McConnell, S. *Desarrollo y gestión de proyectos informáticos*. McGraw-Hill.

Sitios electrónicos

Sitio	Descripción
http://ingenieria.uatx.mx/labastida/files/2011/08/MANTENIMIENTO-DE-SOFTWARE.pdf.	Mantenimiento de *software*. Facultad de Ciencias Básicas, Ingeniería y Tecnología. Universidad Autónoma de Tlaxcala.
http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/Metricas4.pdf	Pruebas, calidad y mantenimiento del *software*. Facultad de Ciencias Químicas e Ingeniería. Universidad Autónoma de Baja California.

Unidad 7

Situación de la ingeniería de *software* en México

OBJETIVO PARTICULAR

El alumno analizará el contexto pasado y actual de la ingeniería de *software* en México.

TEMARIO DETALLADO

(8 horas)

7. Situación de la ingeniería de software en México

7.1. Industria pasada y actual
7.2. Políticas y participación del gobierno
7.3. Iniciativa privada
7.4. Retos

INTRODUCCIÓN

La ingeniería de *software* es importante en el desarrollo de todos los sectores productivos de nuestro país. El *software* a la medida es empleado por empresas grandes, pequeñas y medianas: resulta fundamental en el manejo y proceso de la información de todo tipo de empresas y entidades de gobierno.

Con la dependencia cada vez mayor de las tecnologías de la información y comunicación (TIC), el desarrollo de *software* que permita explotar estas tecnologías es indispensable para las empresas. Los ingenieros y especialistas en sistemas son cada vez más demandados, por lo que es vital la formación de estos recursos humanos.

En este contexto, la presente unidad se enfoca a revisar de manera general la situación actual de la ingeniería de *software* en México.

7.1. Industria pasada y actual

La industria del *software* siempre ha estado ligada al avance de las computadoras, que demandan una plataforma o sistema operativo que permita la interacción de los humanos con el procesador central o microprocesador. Durante los primeros años de desarrollo de las computadoras y redes de telecomunicaciones, la creación de *software* recaía principalmente en manos expertas, ingenieros o científicos que dominaban la forma de trabajar los dispositivos electrónicos y, por ende, podían elaborar programas que permitieran el aprovechamiento de estos dispostivos.

Durante la mayor parte de las décadas de 1970 y 1980, la generación de *software* se limitaba a las empresas norteamericanas como Microsoft o IBM, que desarrollaban y destribuian sus sistemas operativos junto con las primeras computadoras personales. Estos sistemas se basaban en líneas de comandos que permitían a los usuarios interactuar con el equipo, pero los usuarios debían tener un nivel de manejo del sistema casi experto. Los programas de *software* eran desarrollados también en sistemas compiladores como C o Pascal, cuyo resultado era un *software* que requería de su ejecución vía comandos, y su interacción resultaba muy poco amigable.

A partir de la década de 1990, con la aparición de los primeros sistemas operativos de ambiente gráfico como Windows 95, y lenguajes de programación orientados a objetos o gráficos, los usuarios de los equipos de cómputo se multiplicaron, generando un ambiente de trabajo más amigable. Por tanto, se trabajaron aplicaciones más atractivas hacia los usuarios.

En la actualidad, las carreras enfocadas a las tecnologías de la información, como la licenciatura en Informática, la ingeniería en Computación, en Sistemas Computacionales, etcétera, incorporan en su currículo los principios de la ingeniería de *software*, en razón de su importancia en todos los sectores productivos del país.

Hoy, se tiene gran dependiencia de los sistemas de información y de las redes de telecomunicaciones. Los grandes volúmenes de información que se manejan no podrían procesarse sin *software* especializado y grandes bases de datos que les den soporte, junto con redes de datos confiables y de alta velocidad.

Los egresados de las carreras asociadas a las ciencias de la computación son más demandados y exigidos en cuanto a sus capacidades y competencias. Ya no es suficiente con la preparación que se da en las universidades; se requieren, además, certificaciones y especializaciones en áreas consideradas críticas como seguridad, desarrollo de aplicaciones bajo plataformas móviles, telcomunicaciones, etcétera.

Han surgido, entonces, empresas o consultorías especializadas en el desarrollo de sistemas informáticos a la medida. Y las mismas empresas incorporan dentro de sus áreas funcionales departamentos de sistemas y comunicaciones que trabajan sus sistemas y dan mantenimiento a su infraestructura informática.

7.2. Políticas y participación del gobierno

En lo que respecta a las políticas públicas y a la participación del gobierno en lo que se refiere a las TIC, en 2009, se crea la iniciativa de México 1rst, la cual, acorde con su sitio web, es “una iniciativa respaldada por la Secretaría de Economía y el Banco Mundial, cuyo objetivo principal es la generación de capital humano con el fin de fortalecer la oferta laboral tanto en cantidad como en calidad, todo para facilitar el desarrollo y competitividad de las empresas mexicanas, así como la atracción de inversiones extranjeras que busquen en México un jugador de clase mundial"[32].

La iniciativa tiene como propósito central impulsar la formación de recursos humanos calificados y certificados dentro de la industria de las TIC. El sitio oficial refiere que, en 2011, la iniciativa solamente capacitó a más de 21 000 personas y certificó a más de 17 000.

[32] Página oficial de México 1st. Disponible en http://mexico-first.org

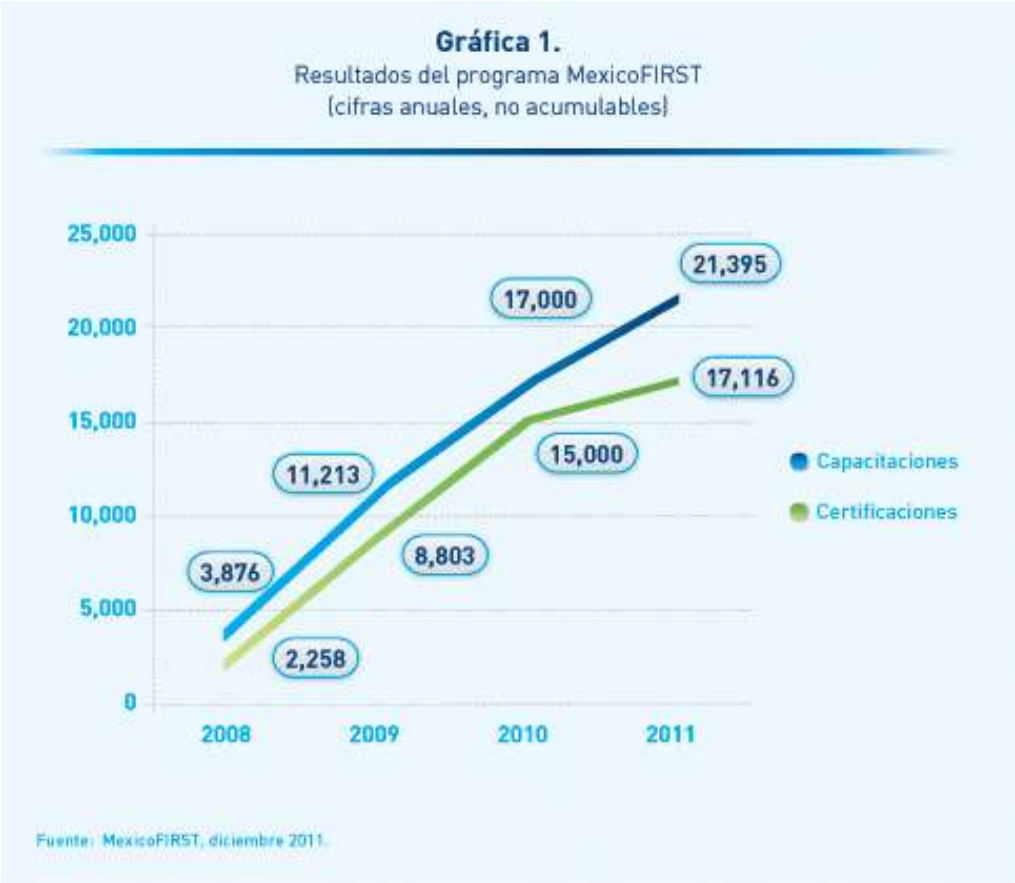


Figura 7.1. Resultados de la iniciativa MéxicoFirst.

FUENTE: Becerra Pozas, J. “MexicoFIRST hacia la cultura de la certificación en TIC”. Disponible en http://www.politicadigital.com.mx/?P=leernoticia&Article=21284&c=113. (Consultado el 27/07/2014).

Estas cifras acusan la importancia que ha tomado tanto para el sector privado como para el gubernamental el desarrollo de especialistas en el sector de las TIC.

Dentro de las certificaciones ofrecidas en el programa, encontramos las que se presentan en la siguiente imagen.

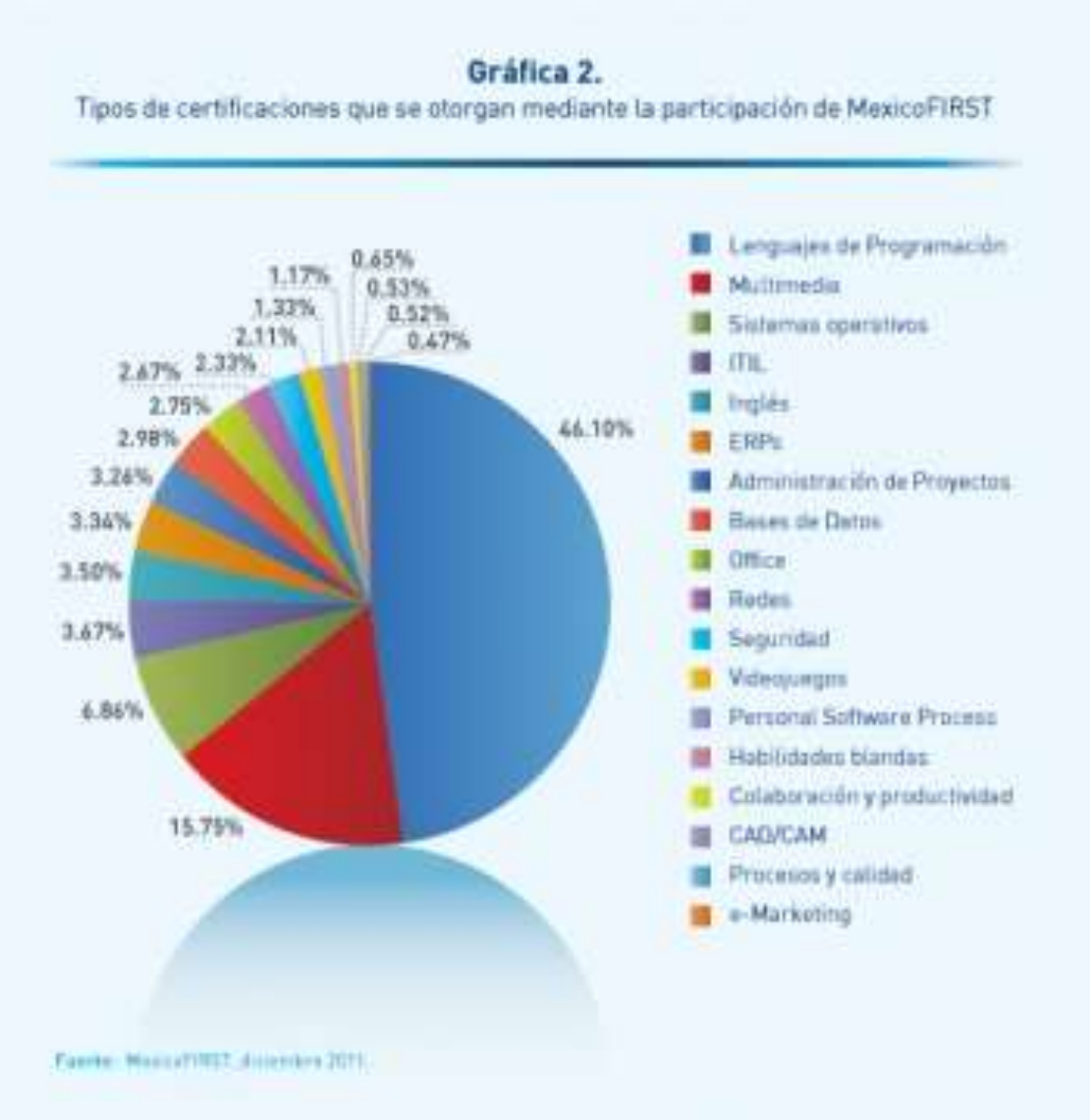


Figura 7.2. Certificaciones otorgadas por MexicoFirst.

FUENTE: Becerra Pozas, J. “MexicoFIRST hacia la cultura de la certificación en TIC”. Disponible en http://www.politicadigital.com.mx/?P=leernoticia&Article=21284&c=113. (Consultado el 27/07/2014).

Es destacable que la mayor parte de las certificaciones son en administración de proyectos, que no incluyen nada más ingeniería de *software*, sino todas las áreas de las TIC.

7.3. Iniciativa privada

En lo tocante a las empresas enfocadas a las TIC, la Asociación Mexicana de la Industria de las Tecnologías de la Información (AMITI), en su estudio de 2013 sobre negocios y comercio en las TIC, refiere los siguientes datos.

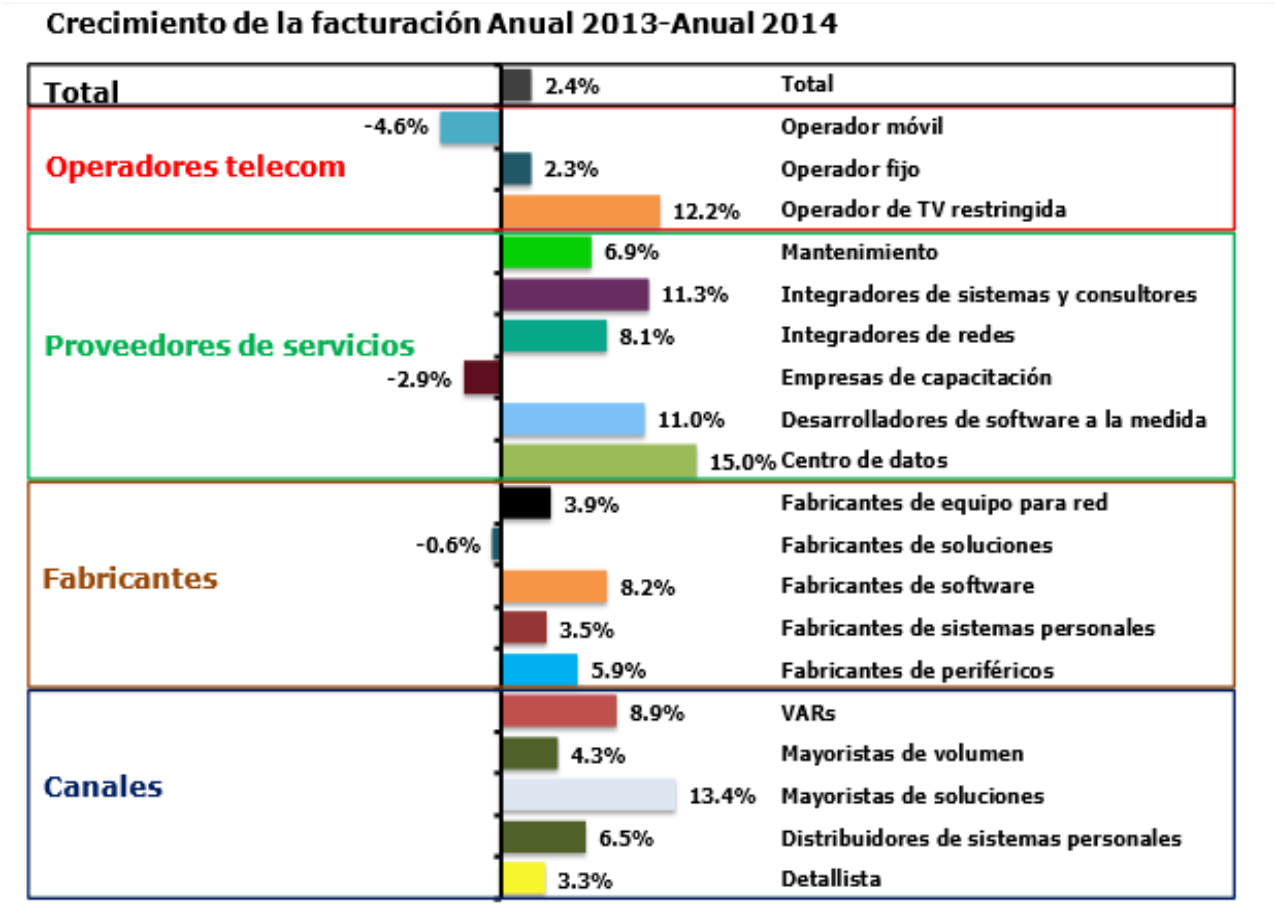


Figura 7.3. Tabla de crecimiento de las industrias relacionadas a las TIC en 2013 y 2014.

FUENTE: "Perspectivas de negocios y mercados de TIC en México, AMITI. Febrero de 2015". Disponible en http://amiti.org.mx/wp-content/uploads/2015/02/Perspectivas_negocios_-mercados_TIC_2015.pdf. (Consultado el 26/07/2015).

En la tabla anterior, se observa que el desarrollo de *software* a la medida tuvo un avance del 11%, una tasa mayor al crecimiento del PIB en el país. Esto refleja la importancia de que las empresas ofrezcan servicios de desarrollo a la medida. Adicionalmente, el requerimiento de centros de datos que permitan el almacenamiento de altos volúmenes de datos creció 15%, indicativo también de que se necesitarán aplicaciones que permitan procesar la información contenida en ellos.

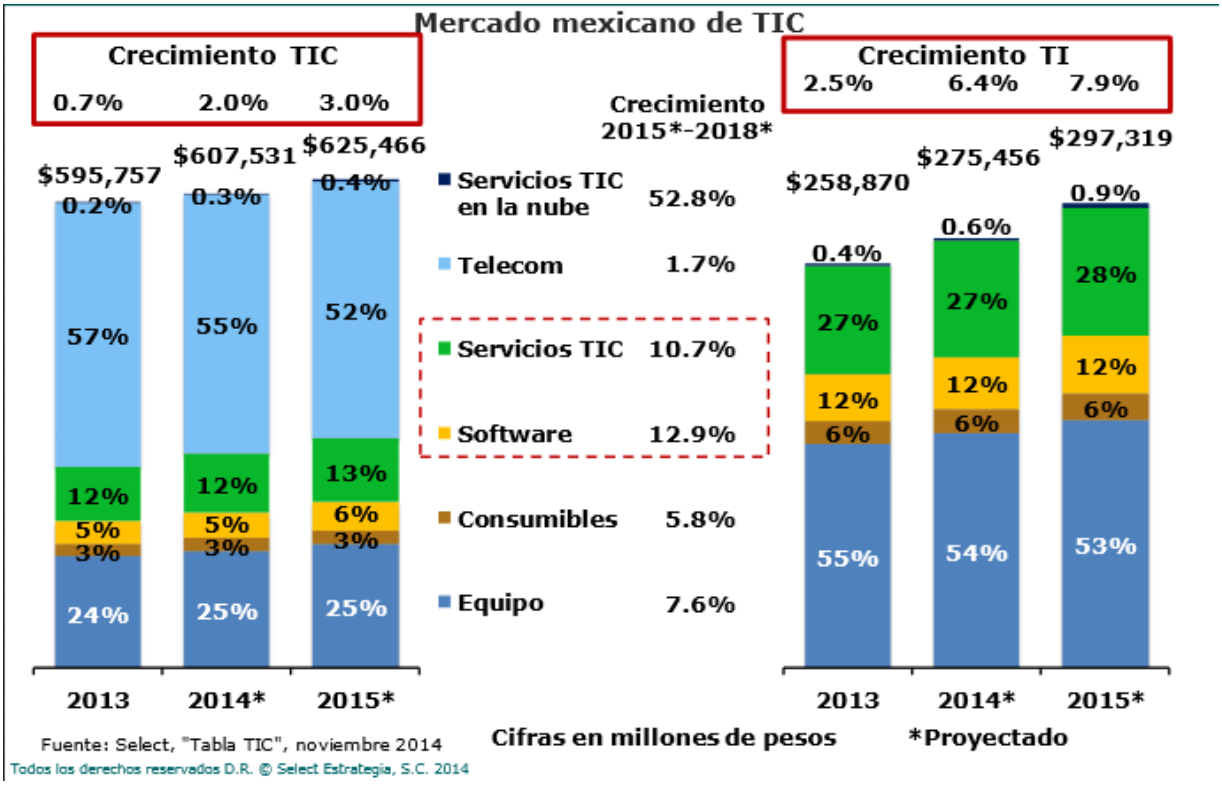


Figura 7.4. Tabla de crecimiento del mercado en TIC de México en 2013 y 2014.

FUENTE: “Perspectivas de negocios y mercados de TIC en México, AMITI. Febrero de 2015”. Disponible en http://amiti.org.mx/wp-content/uploads/2015/02/Perspectivas_negocios_-mercados_TIC_2015.pdf. (Consultado el 26/07/2015).

Así, la dependencia cada vez mayor de las TIC ha llevado a que el mercado mexicano demande mayores y mejores sistemas y servicios, por lo que el estudio de la AMITI poyecta un crecimiento sostenido de *software* de 12%, y recalca la necesidad de un proceso confiable de la información y el uso de la infraestructura de TIC.

Lo anterior presenta un panorama favorable para los egresados de las licenciaturas asociadas a las TIC, de quienes se exige mayor capacitación y experiencia.

7.4. Retos

En el futuro cercano y lejano, los mercados de servicios de TIC se enfocarán cada vez más a las aplicaciones innovadoras, especialmente en los sectores que enuncia la AMITI en la siguiente tabla.

Figura 7.5. Tabla de info-estructura. AMITI.

FUENTE: “Perspectivas de negocios y mercados de TIC en México, AMITI. Febrero de 2015”. Disponible en http://amiti.org.mx/wp-content/uploads/2015/02/Perspectivas_negocios_-mercados_TIC_2015.pdf. (Consultado el 26/07/2015).

Lo anterior sugiere que las aplicaciones especializadas en dispositivos móviles tendran una demanda creciente, al igual que las soluciones para hacer negocios en Internet y dentro de la nube, además de las aplicaciones que permitan la interacción de usuarios y empresas en las redes sociales. Por tanto, la ingeniería de *software* tiene un campo de oportunidad muy grande para explotar y desarrollar este tipo de aplicaciones, siempre con un enfoque de calidad.

RESUMEN

El desarrollo de la ingeniería de *software* y en general las empresas enfocadas a las TIC son cada vez de mayor importancia tanto para el gobierno como para los diversos sectores productivos del país.

La creciente dependencia de las TIC para las empresas, ya sea en sus operaciones cotidianas como en sus operaciones críticas, exige servicios de tecnología de soporte y desarrollo.

En su estudio de perspectivas de mercado y negocios de TIC, de febrero de 2015, la AMITI afirma que, pese al poco crecimiento del PIB en México, los negocios y mercados de TIC han presentado crecimientos de arriba del 10%, y las empresas de *software* en particular han mantenido un crecimiento de 12%. Indicador importante de la necesidad de recursos en ingeniería de *software*.

El gobierno ha puesto en marcha, en conjunto con el sector privado, iniciativas como MexicoFirst, enfocado a la capacitación y certificación de recursos humanos en las diversas áreas de las TIC.

El panorama es claro, cada vez se requieren más y mejores recursos humanos capaces de solventar las crecientes necesidades del mercado en las TIC, para lo cual deben mantenerse siempre a la vanguardia y habilitados en las diversas áreas de oportunidad.

MESOGRAFÍA

Bibliografía recomendada

Autor	Capítulo	Páginas
Mochi	NA	NA
AMITI	NA	NA

Bibliografía básica

Mochi Alemán, P. La industria del software en México. Portal de revistas científicas y arbitradas por la UNAM. Disponible en línea en http://www.revistas.unam.mx/index.php/pde/article/view/7533. Recuperado el 27/07/2014.

AMITI. Perspectiva de negocios y mercado de TIC en México. Febrero de 2015. Disponible en línea en http://amiti.org.mx/3557/perspectivas-de-negocios-y-mercados-tic-en-mexico. Recuperado el 27/07/2014.

Bibliografía complementaria

Booch, G., Rumbaugh J. y Jacobson, I. (2000). *El proceso unificado de desarrollo de* software. Madrid: Addison Wesley.

Braude J. E. González Osuna, M. (trad.). (2003). *Ingeniería de* software*: una perspectiva orientada a objetos*. México: Alfaomega.

Bruegge, B. y Dutoit, A. Ruiz Faudón, S. (trad.). (2002). *Ingeniería de software orientado a objetos*. México: Prentice Hall / Pearson Educación.

GhezzI, C. *et al.* (1991). *Fundamental of software engineering*. EUA: Editorial Prentice-Hall.

McConnell, S. *Desarrollo y gestión de proyectos informáticos*. McGraw-Hill.

Pressman, R. (2002). *Ingeniería del* software*. Un enfoque práctico* (5.ª ed.). México: McGraw-Hill.

Sommerville, I. (2005). *Ingeniería de* software (7.ª ed.). España: Addison Wesley / Pearson Educación.

Sitios electrónicos

Sitio	Descripción
http://ldc.usb.ve/~abianc/materias/ci4712/metricas.pdf	AMITI. Asociación Mexicana de la Industria de Tecnologías de la Información.
http://www.politicadigital.com.mx/?P=leernoticia&Article=21284&c=113.	MexicoFirst hacia la cultura de la certificación en TIC. José Luis Becerra Pozas.
http://mexico-first.org/	Sitio oficial de la iniciativa MexicoFirst.

Facultad de Contaduría y Administración
Sistema Universidad Abierta y Educación a Distancia