INGENIERÍA DE SOFTWARE I

Ángel Varón

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Ingeniería de Software I
Ángel Varón
Bogotá D.C.

Fundación Universitaria del Área Andina. 2018

Catalogación en la fuente Fundación Universitaria del Área Andina (Bogotá).

Ingeniería de Software I

ISBN (impreso): **978-958-5462-99-1**

Fundación Universitaria del Área Andina
Calle 70 No. 12-55, Bogotá, Colombia
Tel: +57 (1) 7424218 Ext. 1231
Correo electrónico: publicaciones@areandina.edu.co

Director editorial: Eduardo Mora Bejarano
Coordinador editorial: Camilo Andrés Cuéllar Mejía
Corrección de estilo y diagramación: Dirección Nacional de Operaciones Virtuales
Conversión de módulos virtuales: Katherine Medina

BANDERA INSTITUCIONAL

Pablo Oliveros Marmolejo †
Gustavo Eastman Vélez
Miembros Fundadores

Diego Molano Vega
Presidente del Consejo Superior y Asamblea General

José Leonardo Valencia Molano
Rector Nacional
Representante Legal

Martha Patricia Castellanos Saavedra
Vicerrectora Nacional Académica

Jorge Andrés Rubio Peña
Vicerrector Nacional de Crecimiento y Desarrollo

Tatiana Guzmán Granados
Vicerrectora Nacional de Experiencia Areandina

Edgar Orlando Cote Rojas
Rector – Seccional Pereira

Gelca Patricia Gutiérrez Barranco
Rectora – Sede Valledupar

María Angélica Pacheco Chica
Secretaria General

Eduardo Mora Bejarano
Director Nacional de Investigación

Camilo Andrés Cuéllar Mejía
Subdirector Nacional de Publicaciones

INGENIERÍA DE SOFTWARE I

Ángel Varón

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

ÍNDICE

INGENIERÍA DE SOFTWARE I

Ángel Varón

EJE 1

Conceptualicemos

AREANDINA
Fundación Universitaria del Área Andina
MIEMBRO DE LA RED ILUMNO

Fuente: Shutterstock/432008923

INTRODUCCIÓN

La ingeniería del software es una rama de la informática que está compuesta por una serie de técnicas, herramientas y metodologías que sirven como guía para desarrollar, mantener y refinar los productos de software. En este módulo trataremos cada uno de estos elementos, con el objetivo de brindar al estudiante los aspectos más importantes, que le ayuden a adquirir las competencias necesarias para emprender un proyecto de desarrollo de software.

Introducción a la ingeniería de software

Software

Es un producto que desarrolla un equipo de profesionales (analistas, diseñadores, programadores), con el objetivo de dar respuesta a los requerimientos de un proceso dentro de una organización, el cual se basa en una serie de requisitos, cuyo objetivo consiste en agilizar la forma en que se realizan los procedimientos dentro de este. Es importante tener presente que estas operaciones solicitan una entrada de datos, estos se procesan y el resultado de este produce la información, la cual se almacena y posterior a esto, cuando se vuelve a realizar algunas operaciones, se exige la realimentación con más datos o información.

Ingeniería del software

Es un área de la informática que se encarga de utilizar herramientas y metodologías para analizar las necesidades de una organización, recolectar la información necesaria para especificar los requerimientos y, con estos, hacer el diseño de software que cumpla con todos los estándares de calidad, también se encarga de realizar toda la documentación necesaria para realizar el desarrollo, operar y mantenerlos.

Se conoce también como desarrollo de software o producción de software (Bohem, 1976).

Metodologías
Es un modelo que consta de una serie de pasos lógicamente estructurados que tiene como finalidad orientar la elaboración de un producto.

Importancia del software

El software es el que da órdenes a cada equipo y cada uno de sus elementos, siendo de gran importancia para cualquier tipo de sistema informático, ya que indica funciones según características y permite que funcione todo conjuntamente, es decir este controla y dirige al hardware; es evidente que sin este el software no serviría de nada, cuando el software no funciona bien en un hardware establecido es porque entre los dos existe incompatibilidad.

Actualmente el software se considera como pieza central en todas las áreas de nuestra vida, proporcionándonos un mundo lleno de ventajas en el trabajo, el estudio, la comunicación, etcétera.

Su importancia también reside en:

- Facilitar a usuarios la ejecución de tareas.
- Admitir comunicación máquina - usuario.
- Interacción entre los dos.

En informática los programadores, son quienes diseñan el software para que cumpla con la función que se desea. En grandes programas, como los sistemas operativos, que tienen alto grado de complejidad trabajan equipos ilimitados de personas por mucho tiempo, por esto grandes empresas como Apple, Microsoft, Google, se dedican a desarrollar software, lucrándose con esto, aunque existen otras que abogan por el software gratis o libre.

Usuario
Persona que interactúa con una computadora.

Figura 1.
Fuente: Shutterstock/531355591

Problemas del software

Cuando emprendemos un proyecto de desarrollo de software suelen presentarse algunos errores, estos se perciben más que todo en la fase inicial del proyecto tales como:

01 Cuando se hace la planificación se suele tener excesiva confianza.

02 No se estima conscientemente el alcance del proyecto (varios proyectos al mismo tiempo).

03 Crear expectativas muy altas e imposibles de realizar.

04 Mal realizada la especificación de requerimientos (a mitad del proyecto incluir nuevas condiciones).

05 La realización de pruebas se hace a medias, permeando la calidad del producto, es decir que no es del todo segura.

06 Deficiente gestión del riesgo.

07 Personal no calificado.

08 Diseño inapropiado.

09 Ausencia de apoyo para el proyecto.

10 Actividades excesivas (distintos proyectos a la vez).

11 Trastocar estimación con propósitos.

Falta de compromiso del usuario.

Errores de poca frecuencia

12 Ausencia de control automático del código fuente.

13 Incorporación de funciones nuevas.

14 Modificar elementos a mitad del proyecto.

15 Estimación eludiendo la utilización de elementos o técnicas nuevas.

16 Desarrollo orientado por la investigación.

17 Coincidencia precoz o reiterada (presión de cierre antes de que sea realizable).

18 Contrariedades con el personal.

Codigo fuente
Estructura lógica de un software creado en un lenguaje de programación.

Figura 2.
Fuente: propia

Características del software

Cuando se emprende un proyecto de desarrollo de software se selecciona un modelo o metodología, la cual generalmente contempla las siguientes fases: análisis, diseño, desarrollo, implementación, prueba y mantenibilidad.

El software se desarrolla no se fabrica: estas dos tareas solicitan construir un producto desde perspectivas distintas, qué quiere decir esto, que no se gestionan los proyectos de software, como si fuesen proyectos de fábrica.

El software no se daña se debilita: reside en que el software no es propenso a daños del ambiente que generan daños en el hardware, en software los fallos no detectados durante su creación, hacen que el programa posea fallos en las primeras fases de su ciclo de vida.

El software se construye a medida: para incluir no solo estructuras de datos, sino también algoritmos, la reutilización se expande y los elementos modernos reutilizables envuelven tanto procesos como datos, admitiendo que el ingeniero de software idee aplicaciones novedosas con base en estos elementos, como lo pueden ser las interfaces gráficas del usuario.

Para ampliar más los conocimientos adquiridos le invito a realizar la siguiente lectura complementaria:

Lectura recomendada

Introducción a la ingeniería del software
Francisco José García Peñalvo, Miguel Ángel Conde González, Sergio Bravo Martín.

Conceptos de calidad

Calidad es el grado en que un conjunto de características inherentes cumple con unos requisitos (Norma ISO 9000, 2000).

Cuando se lleva a cabo un proyecto de ingeniería del software, lo que espera es que el producto cumpla con unos estándares de calidad y den respuesta a los requerimientos que se establecieron en el análisis y los que surjan a medida que avanza el desarrollo del proyecto, por eso es necesario establecer un marco de referencia que permita definir y poner en marcha un plan SQA (aseguramiento de la calidad del software),

para reunir las características necesarias para medir y verificar el nivel de satisfacción de todos los requisitos del sistema de información.

Según Roger Pressman, calidad es el "cumplimiento de los requerimientos funcionales y de rendimiento explícitamente definidos, de los estándares de desarrollo explícitamente documentados y de las características implícitas esperadas del desarrollo de software profesional".

Instrucción

Para realizar un plan como Asesor de Calidad de Software SQA, le invito a que revise la infografía Actividades plan de calidad del software dentro de los recursos de aprendizaje del eje.

Existen otras definiciones para determinar la calidad del software, que a continuación resaltaremos:

Según la norma IEEE 1601: "La calidad de un producto de software se define como el grado en que posee una combinación adecuada de determinadas características como son rendimiento, fiabilidad y seguridad".

La norma ISO 8492 de 1994: "Totalidad de propiedades y características de un producto proceso o servicio que le confiere su aptitud para satisfacer necesidades expresadas o implícitas".

La calidad de software se determina si cumple con todos los requerimientos que se determinaron en su inicio, además de esto se debe validar su funcionalidad, rendimiento, confiabilidad, seguridad y la disponibilidad de mantenimiento y la posibilidad de que se pueda actualizar.

También es muy importante que la interfaz de usuario sea amigable para el usuario, es decir de fácil manejo.

Gestión de la Calidad de Software (Software Quality Management): para cada proyecto de desarrollo de software debe existir personal encargado de gestionar la calidad del software y esta debe fundamentarse en entender las expectativas del cliente frente a la calidad del producto, para ello se recomienda poner en práctica las acciones necesarias para cumplir con estas expectativas.

En algunas ocasiones la administración de la calidad del producto se hace según políticas de la organización, es decir aplicando las políticas de calidad de la empresa a cargo del CIO quien se encarga de determinar la calidad, los objetivos y responsabilidades.

Aseguramiento de la calidad software: se emplean una serie de técnicas que, sin duda alguna, bien ejecutadas permitirán generar la confianza necesaria para que el software satisfaga los requerimientos establecidos para determinar la calidad del producto.

Control de la calidad de software: conjunto de técnicas y actividades de carácter operativo, utilizado para verificar los requisitos relativos a la calidad, centrados en mantener bajo control el proceso de desarrollo y eliminar las causas de los defectos en las diferentes fases del ciclo de vida.

Verificación y validación de software: se aplica una serie de técnicas y protocolos para medir la calidad del producto, se basa en identificar si el software realiza las funciones que se determinaron con anterioridad, es decir si satisface los requisitos que el usuario solicitó.

Mitos del software

Instrucción

Los mitos del software son creencias engañosas y están compuestos por varias propiedades, para poder identificarlas les recomiendo acceder a la infografía Mitos del software dentro de los recursos de aprendizaje del eje.

Paradigmas de la ingeniería de software

Para hacer fácil el control durante el desarrollo de software y la calidad de este, surgen componentes que brindan varios modelos para que la construcción del software se haga de manera más eficiente y productiva; a continuación, los mencionaremos:

Métodos

Señalan técnicamente la construcción del software a través de una amplia gama de procedimientos como lo son: análisis, diseño, desarrollo, implementación, prueba, mantenibilidad.

Modelos

Un modelo es una conceptualización teórica que se figura por medio de diagramas para mostrar los efectos de los factores más significativos, con el objeto de predecir acontecimientos que no han sido observados y por medio de este plantear soluciones a problemas formulados.

En el desarrollo de software se aplican técnicas para documentar la determinación del sistema, utilizando un conglomerado de modelos del sistema; para esto conceptualizamos el modelo como una idealización que está en proceso de investigación, en lugar de una representación disyuntiva del sistema y que estos diferentes modelos se grafican para describir cómo interactúan los procesos de la organización. El enigma a determinar y el software a realizar, esta graficación permite que exista una mejor comprensión, siendo la fisonomía de este más relevante en la representación del sistema para suprimir pormenores, ya que constituye un enlace entre el proceso de análisis y el de diseño; sin embargo, se considera que el gráfico de un sistema obligatoriamente debería contener toda la información necesaria sobre la entidad que se está representando, expresando requerimientos del sistema técnicamente para que las personas que no son expertas lo comprendan fácil y rápidamente debido a las representaciones gráficas usadas, por eso se usan modelos en el proceso de análisis desde diferentes ópticas como: la óptica externa en la que se nos da a conocer el entorno del sistema, otra óptica es la de comportamiento que nos muestra obviamente el comportamiento del sistema y por último la óptica estructural en la que se modela la arquitectura de los datos del sistema.

Sommerville (2005) afirma que:

1. Un modelo de flujo de datos: muestra cómo se procesan los datos del sistema en diferentes etapas.
2. Un modelo de composición o de agregación: muestra como las entidades del sistema están compuestas por otras entidades.
3. Un modelo arquitectónico: muestra los principales subsistemas que componen un sistema.
4. Un modelo de clasificación: los diagramas de clase/herencia de objetos, muestra como las entidades tienen características comunes.
5. Un modelo de estímulo-respuesta: también llamado diagrama de transición de estados, muestra cómo reacciona el sistema a eventos internos y externos (p. 154).

Modelo de contexto

Todos los sistemas que se desarrollan son en realidad subsistemas de uno mayor, es por ello que es importante definir primeramente las interfaces entre nuestro sistema y el resto del universo, o sea el ambiente.

Para definir el ambiente utilizamos valga la redundancia, el modelo ambiental con el cual podemos modelar el exterior del sistema y su interior (modelo de comportamiento), delimitamos las fronteras entre el sistema y el ambiente y sabremos también qué información entra al sistema desde el ambiente exterior y cuál se produce como salida del sistema.

El diagrama de contexto sirve para demostrar el ambiente en el cual actúa el sistema, y el ambiente que lo rodea, identifica interfaces que permiten comunicar al sistema con su medio externo, se utilizan flujos de control de datos para mostrar las interacciones existentes entre los agentes externos y el sistema describiendo la información entrante o saliente del sistema a través de las diferentes interfaces y de materiales, que permiten distinguir lo que es en sí el sistema y su entorno y que no, pues implica aspectos sociales y organizacionales del entorno, pero mostrando de manera amplia las relaciones entre el sistema que está desarrollando y el medio, es decir no describe en ningún momento la estructura del sistema de información; es conocido como nivel 0 o DFD diagrama de flujo de datos.

Cómo elaborar un diagrama de contexto

Para saber cómo elaborar un diagrama de contexto los invito a ver el video Resumen:

Video

Cómo elaborar diagrama de contexto dentro de los recursos de aprendizaje del eje. https://vimeo.com/235379369

Modelos de comportamiento

La utilidad de modelo de comportamiento consiste en que se explica detalladamente la forma en la que procede el sistema en su totalidad, dentro de este se encuentran dos tipos de modelos: los modelos de flujo de datos y los de máquinas de estado, cada uno muestra el procesamiento y la fluidez de los datos y como el sistema reacciona a posibles eventos, estos dos modelos pueden trabajarse unidos o bien sea separados, dependiendo del tipo de sistema que se desarrolle.

Modelos de procesos

Sommerville y Galipienso (2005) definen este modelo de software como "una representación simplificada de un proceso de software, representada desde una perspectiva específica. Por si naturaleza los modelos son simplificados, por lo tanto, un modelo de procesos del software es una abstracción de un proceso real".

Para la elaboración de un modelo de flujo de datos existen varias herramientas case, la más conocida es la suite Power Designer que contiene el Process Analyst y que acompaña a otro software como data Architect que sirve para elaborar diagramas de flujos de datos.

Inicialmente seleccionamos la paleta de herramientas, según la ilustración del Power Designer, donde cada archivo se presenta inicialmente como una pantalla vacía con una paleta de herramientas.

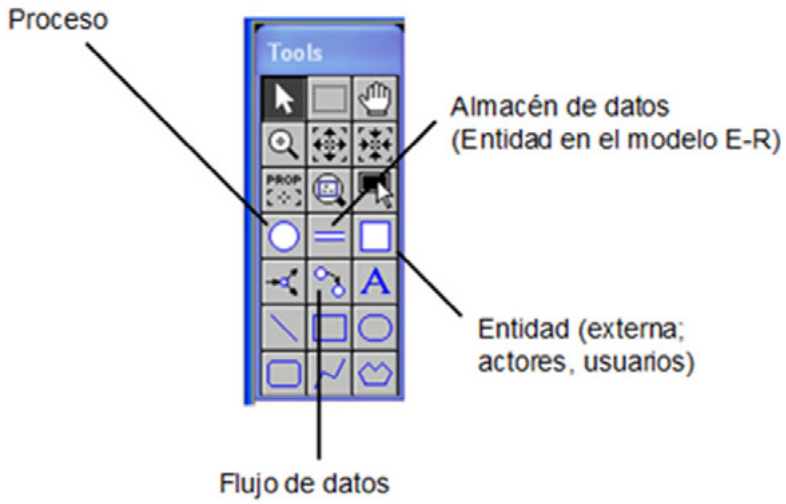


Figura 3. Paleta de herramientas de Power Designer
Fuente: Power Designer

Estas herramientas primordiales son el proceso, el almacén de dato, el flujo de datos y la entidad externa.

El proceso es, en este caso un círculo.

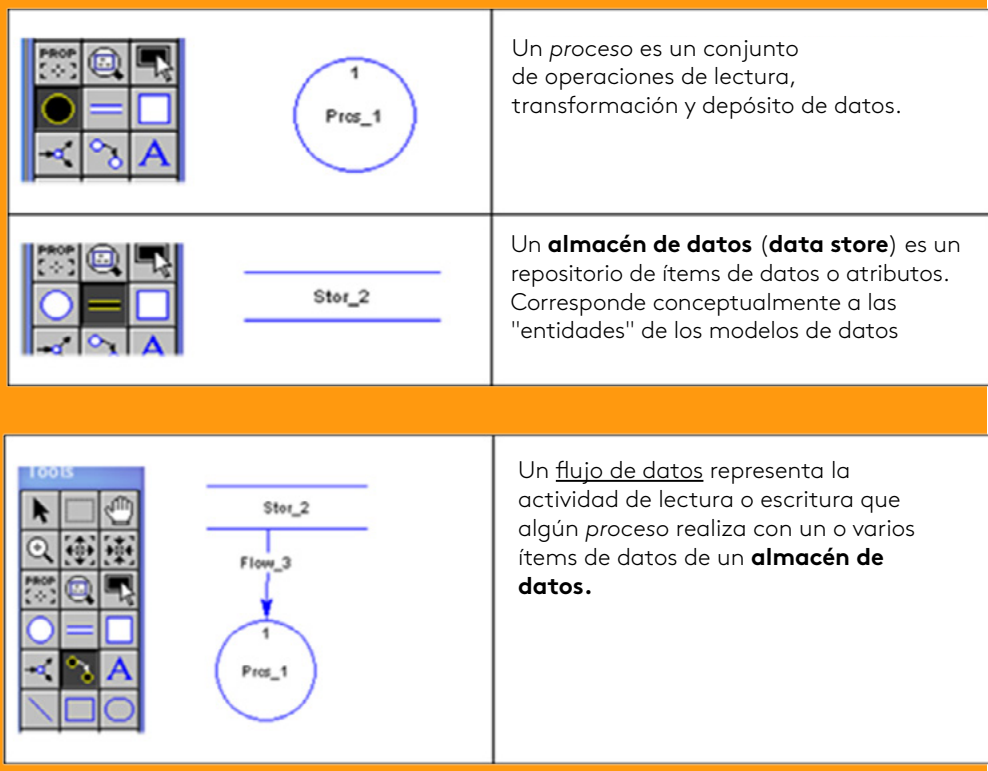


Figura 4. Herramientas para elaborar modelos de comportamiento
Fuente: Power Designer

A continuación, se siguen usando los mismos estilos tipográficos (proceso, almacén, flujo) y además los ítems de datos y las variables se imprimen en fuente Courier New, como se muestra en la figura para elaborar diagramas de comportamiento.

Modelos de máquina de estados

Es el mismo modelo de comportamiento del sistema, donde es interpretado como un grupo de estados que sirve como puente entre entradas y salidas, siendo la señal de entrada la que indica para cada instante un estado para la máquina, de modo que la salida está sometida del estado y de las entradas actuales.

Modelos de flujos de datos

Es el mismo modelo de procesos que sirve para delimitar el alcance y la serie u orden de pasos donde se muestran las acciones, que tienen lugar a partir de una entrada que se procesa hasta la salida que establece la respuesta del sistema de información.

Los modelos de flujo de datos son una forma de suponer cómo podría ser el procesamiento de los datos a través del seguimiento y la documentación de estos por el sistema, es decir, cómo fluyen los datos a través del sistema y cómo se convierten en cada paso antes de trasladarse a la siguiente fase, representados por los diagramas de flujo de datos, siendo estos un abrebocas para el diseño del sistema lógico y conceptual (idea lógica). El análisis esta herramienta nos sirve para representar gráficamente las actividades implicadas en un proceso, mostrando la relación secuencial entre ellas, se recomienda para modelar la forma y las reglas en la que los datos son procesados en el sistema existente, facilitando la obtención de un panorama claro del proceso para una mejor comprensión de las actividades, relaciones y eventos que ocurren dentro de un proceso haciendo fácil la identificación de los clientes y determinar sus necesidades, el diagrama de flujo incide para que exista una comunicación ya que se aplica un lenguaje común, estimulando la creatividad en el momento de analizar un proceso porque permite generar más ideas útiles, empleando acciones orientadas a mejorar las variables de eficacia y eficiencia.

Fase
Etapa de un proceso o actividad.

Para la construcción de los Diagramas de Flujo (DFD) se deben tener en cuenta las siguientes premisas:

- **Delimitar gráficamente el sistema.**
- **Representar el flujo de los datos y su evolución en el sistema.**
- **Diferir las limitaciones físicas de las lógicas.**

A continuación, se mencionan los elementos que intervienen en el diseño del diagrama de flujo de datos: entidad externa, proceso, almacén de datos y flujo de datos.

Modelos de datos

Un modelo es una serie de instrumentos conceptuales para describir la estructura como están conformados los datos, con referencia a las relaciones existentes entre estos, su significado y sus restricciones de consistencia. Este es uno de los elementos necesarios e importantes a la hora de emprender el desarrollo de cualquier proyecto, aunque existen varias expresiones para el modelado de datos, el modelo más utilizado frecuentemente es el Modelo Entidad Relación (MER) que se centra en las entidades y las relaciones descritas por los datos, dotando de entendimiento en el dominio de la información del problema, constituyéndose en gran ayuda para el ingeniero de software ya que le sirve de base para dar inicio al diseño, esta técnica se aplica para estructurar, organizar y documentar los datos definiendo los requerimientos de una base de datos, en el medio suele llamarse modelado de bases de datos donde básicamente se emplean tres tipos de modelados de datos.

1. Modelo conceptual: es muy similar al modelo de contexto ya que muestra de forma general y abstracta la representación global de la organización.

2. Modelo lógico: se encarga de mostrar la interpretación completa que incluye todos los detalles de los datos.

3. Modelo físico: esboza la estructura que se va a poner en marcha por medio de un Manejador de Bases de Datos (DBMS).

DBMS
Sistema manejador de bases de datos.

La arquitectura de los datos se representa mediante un modelo se representa utilizando un modelo entidad relación que debe contener entidades, atributos, dominios, relaciones, llaves y cardinalidad.

Modelos de objetos

Se utilizan básicamente para representar tanto los datos del sistema como su procesamiento, describiendo objetos los cuales son una abstracción con límites de gran ayuda para mostrar cómo se clasifican las entidades en el sistema y cómo se componen de otras

entidades, este modelo nos muestra la estructura estática del sistema, así objetos, relaciones entre objetos, atributos y operaciones utilizando en su sistema clases de modelos para detallar como:

- Modelo de objeto: representación gráfica donde los nodos son clases de objetos los cuales son relaciones entre clases.
- Modelo dinámico: representación gráfica donde los nodos son estados y los arcos son transiciones entre estados causados por sucesos.
- Modelo funcional: representación gráfica donde los nodos son procesos y los arcos son flujos de datos.

Los programadores emplean los objetos como elementos fundamentales para dar una solución, puesto que pueden construir, estructurar, agregar, cambiar, modificar o eliminar elementos y contenido para diseñar aplicaciones y programas informáticos, basados en técnicas, todas usadas para perfeccionar el modelo ya que simplifican la transición entre el diseño orientado a objetos y la programación orientada a objetos en la cual se evidencia la estructura y el comportamiento de datos mediante componentes fundamentales y secundarios así:

Flujos de datos
Recorrido que hacen los datos.

Atributos
Es una cualidad de un objeto.

Abstracciones
Tener en mente ideas que representan las cosas de forma diferente a la realidad.

Componentes fundamentales

- Abstracción: simplifica la realidad que queremos modelar para centrarnos en el comportamiento de los objetos del software y no en la implementación de su código.
- Modularidad: propiedad que posee un sistema que ha sido subdividido en un conjunto de módulos adhesivos e inciertamente vinculados.
- Encapsulamiento: proceso que restringe el acceso a los atributos y métodos de los objetos, para fijarnos en el tipo de órdenes e información que se transmite y no en su estructura y funcionamiento interno.
- Mensajes: solicitudes que se hacen a los objetos, para que ejecuten varias de sus rutinas.
- Jerarquía: categorización o clasificación de las abstracciones.

Componentes secundarios

- Polimorfismo: capacidad de distintas clases para responder a mismo llamado del método, de modo que cada una lo implementa de distinta forma.

- Persistencia: propiedad cuya existencia de un objeto se propaga en el tiempo (esto es, el objeto sigue existiendo después de que su creador deja de existir) o en el espacio (la localización del objeto cambia respecto a la dirección en la que fue creado).

- Concurrencia: característica que identifica un objeto activo de uno no activo, también permite que distintos objetos actúen al mismo tiempo, usando distintos hilos (threads) de control.

Threads
Hilos son programas pequeños o procedimientos para realizar operaciones.

Para profundizar los conocimientos adquiridos recomiendo la lectura complementaria:

Lectura recomendada

Curso de introducción a la ingeniería del software. Instituto Nacional de Tecnologías de la Comunicación.

Instrucción

Recomiendo realizar la actividad de repaso ya que es muy útil para la presentación de la evaluación del este eje.

BIBLIOGRAFÍA

Sommerville, I. M., Galipienso, A. (2005). *Ingeniería del software*. Madrid: Pearson.

Pressman, R. (2001). *Ingeniería del software un enfoque práctico*. Madrid: McGraw Hill.

Yourdon, E. (1993). *Análisis estructurado moderno*. México: Prentice Hall.

Sommervile, I. (2006). *Ingeniería del software*. Madrid: Editorial Pearson.

Whitten, J. y Bentley, D. (2008). *Análisis de sistemas*. *Diseños y métodos*. Madrid: Editorial McGraw Hill.

INGENIERÍA DE SOFTWARE I

Ángel Varón

EJE 2

Analicemos la situación

AREANDINA
Fundación Universitaria del Área Andina
MIEMBRO DE LA RED
ILUMNO

INTRODUCCIÓN

El desarrollo de un producto de software es bastante complejo, por eso es importante acudir a la ingeniería de requerimientos, la cual ofrece técnicas y tareas específicas que al realizarlas cuidadosamente permiten entender cuáles son los requisitos en detalle y con esto evitar que el desarrollo del proyecto se convierta en un caos.

Se recomienda que los equipos de desarrollo entiendan que el proceso de construcción del producto, realmente comienza durante la fase de especificación de necesidades, fase en la cual deben participar gerentes, clientes, usuarios y el equipo de desarrollo para que puedan determinar el objetivo y las funciones del software, y continuar con la fase de diseño y desarrollo.

Introducción a la ingeniería de requisitos

Un requerimiento es una característica o una restricción que debe tener el sistema.

Según la IEEE, un requerimiento es la capacidad que debe tener en un sistema o en sus componentes para satisfacer un contrato, estándar, especificación u otro documento formal.

Reside en recopilar, analizar y verificar la información necesaria para satisfacer las necesidades del usuario o cliente y las propias del sistema; en el desarrollo de software esta tarea es de mucha importancia, ya que tiene como finalidad entender y especificar los requisitos del sistema, para ello es necesario asumir que es un procedimiento técnico que suele involucrar a muchas personas, dependiendo del tamaño o magnitud del software, lo que en ocasiones hace que la actividad sea bastante compleja.

Estas acciones comprenden la obtención de información (recolección, análisis, negociación, especificación y validación de los requisitos del software) y establece una actividad de gestión de requerimientos para manejar los cambios, el mantenimiento y la rastreabilidad de los mismos.

Restricción:
Condición que se establece en un software.

Rastreabilidad:
Condición de hacer seguimiento es decir seguir el rastro.

Esto claramente indica que es lo que se debe hacer al momento de emprender un proyecto de desarrollo de software, que consiste en aplicar las técnicas necesarias para recopilar información y con esta hacer un análisis de requerimientos para identificar las funciones operativas del software, identificar plenamente las restricciones y limitaciones del mismo y a la vez poder establecer la interfaz.

Es importante establecer que el análisis de requisitos lo puede hacer un ingeniero de sistemas, ingeniero de software o un profesional en análisis o diseño de sistemas de información, lo importante es construir los requerimientos básicos, utilizando uno de los modelos que se utilizan para esta actividad, que menciona Roger Pressman y que visualizamos en la siguiente tabla.

Modelo
Modelos basados en el escenario de los requerimientos desde el punto de visto de distintos "actores" del sistema.
Modelo de datos, que ilustran el dominio de la información problema.
Modelos orientados a clases, que representan clases orientada a objetos (atributos y operaciones) y la manera en que las clases colaboran para cumplir con los requerimientos del sistema.
Modelos orientados al flujo, que representan los elementos funcionales del sistema y la manera cómo transforman los datos a medida que avanzan a través del sistema.
Modelos de comportamiento, que ilustran el modo en que se comparten el software como consecuencia de "eventos" externos.

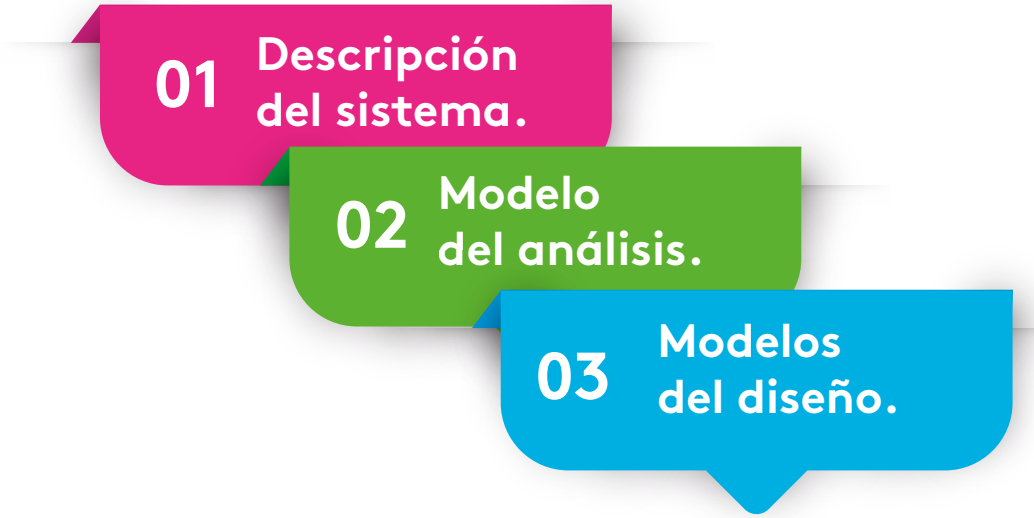
Tabla 1. Modelos para especificación de requerimientos del software
Fuente: Pressman (2010)

Independiente del modelo que se utilice para el análisis de requerimientos, este debe dar como resultado las pautas para que el desarrollador y el cliente evalúa la calidad del mismo una vez se ponga en funcionamiento, además de esto también permite al diseñador establecer los datos de la interfaz.

- El modelo basado en escenario se ha convertido en el más usado y en una buena técnica dentro de los procesos de ingeniería del software.

- El modelo basado en datos se utiliza más que todo cuando se va a desarrollar un software de mayor complejidad.

- El modelo orientado a clases lo que hace es representar el funcionamiento del sistema mediante clases orientada a objetos.



Interfaz:
Elemento que permite al usuario interactuar con una aplicación.
Complejidad:
Dícese de un software que contiene muchos procesos o subsistemas
Acoplamiento:
Adaptabilidad del sistema

Figura 1. Fases del modelo de especificación de requerimientos
Fuente: Pressman (2010)

Para el análisis de sistemas Arlow y Neudstadt recomiendan que sea muy importante aplicar protocolos para crear el modelo del análisis:

- El eje central del modelo recae en los requerimientos para que estén orientados a la solución del problema o dentro del dominio del negocio, y expliquen la funcionalidad del sistema.

- Cada componente del modelo de requerimientos debe ser claro y objetivo para que permita entender el dominio de la información, los procedimientos y el comportamiento del software con un nivel de abstracción bastante alto para que cuando llegue a la etapa de diseño sea entendible y no se retrase este proceso.

- Las relaciones entre las clases y las funciones deben representarse haciéndose un esfuerzo para simplificar la interconectividad minimizando el acoplamiento por medio del sistema.

Este modelo de requerimientos debe cumplir una funcionalidad adecuada, es decir que por medio del mismo el cliente pueda evaluar los requerimientos, el gestor de calidad lo pueda utilizar para generar un plan estratégico para realizar las pruebas y los diseñadores lo entiendan y plasmen única y exclusivamente los diagramas necesarios para el desarrollo del producto, es decir que contenga la información apropiada para no crear diagramas innecesarios.

Análisis de dominio: consiste en analizar, reconocer y determinar requerimientos a través de un dominio de aplicación definida, para que ser reutilizado en varios proyectos.

Análisis de dominio orientado a objetos: reside en establecer y analizar las competencias y reutilizarlas en condición de clases, objetos y estructuras dentro de un dominio de aplicación determinado.

Dominio de aplicación determinado o específico: su finalidad es crear o hallar patrones de análisis manejables de manera tal que se puedan reutilizar.

Frecuentemente se hallan patrones de análisis reutilizable en un dominio establecido, los cuales hacen que el modelo del análisis sea practicable a la hora de clasificarlos y determinarlos, de manera que se logren identificar y emplear para dar respuesta a problemas normales.

Cuando se introducen patrones de diseño y elementos de software ejecutable, estos generan una mejora, tanto en el tiempo como en el coste de desarrollo llevando a que sean mínimos.

Para ampliar más nuestro conocimiento los invito a ver la siguiente videocápsula:

Video

Requerimientos funcionales y no funcionales.

http://bit.ly/1oL6gaJ

Requerimientos funcionales y no funcionales

Figura 2. Diagrama de entradas y salidas para el análisis de dominio
Fuente: Pressman (2010)

Análisis y negociación de requerimientos

Esta etapa es la más importante para el desarrollo de un sistema de información, ya que cumple un papel relevante, pues es el punto de partida y su objetivo principal consiste en la determinación correcta de especificaciones o características con que debe contar el sistema y se deben plantear de forma clara, compacta y consistente para el comportamiento del sistema y con esto minimizar los problemas relacionados con el desarrollo del software.

Para determinar los requerimientos se deben realizar las siguientes tareas:

Comprender el problema que se desea resolver siendo necesario estudiar el contexto o entorno en el que el sistema va a operar, especificar lo que se quiere hacer, conocer las necesidades del cliente y los usuarios, ya que el problema será resuelto mediante la construcción de un producto o sistema de software. Para poder entender el problema hay que establecer cuáles son los objetivos que persiguen el cliente y su organización y cuál es su visión del producto a desarrollar.

Buscar y recolectar información del sistema a implementar, pues se hace importante contar con esta desde varias instancias que le permita conocer la organización, los estándares y las políticas que describen los diferentes flujos de información que tendrá la aplicación, los procesos, los procedimientos y las normas de la empresa que se puedan obtener del sistema actual, información sobre el dominio de la aplicación y regulaciones nacionales e internacionales.

Definir los límites y las restricciones del sistema será la tercera actividad a desarrollar para determinar con precisión qué es lo que el sistema hará, especificar lo que no hará y hasta dónde llegará, los límites que definen el contexto de operación del sistema y su medio de operación, las condiciones de confiabilidad, disponibilidad, desempeño, seguridad e integridad y, en general, los requerimientos no funcionales que le exigirán al sistema. Los desarrolladores pueden iniciar definiendo un bosquejo general del sistema su funcionamiento básico y estableciendo su alcance e identificando a las personas o usuarios interesados en el sistema ya que ellos conocen el ambiente en que operará el sistema y pueden ayudar describiendo sus necesidades así se recolectan y clasifican requerimientos.

Contexto:
Entorno en el que va a residir el sistema.

Sistema de Software:
Todo aquello que tiene una entrada, un proceso y una salida, cuenta con realimentación.

Bosquejo:
Diagrama, modelo o representación gráfica.

Requerimientos básicos

Son aquellos que nos permiten identificar cuál es el proceso básico de la organización y su funcionalidad, para ello nos enfocamos en los usuarios del sistema mediante la observación o entrevistas, con esto se obtiene información detallada sobre qué tipos de datos utiliza o produce este proceso, el paso a paso de cada procedimiento que realiza cada usuario. Además, se podrá evidenciar el tiempo que tarda en ejecutarse cada actividad y con qué frecuencia la hacen, el flujo de datos y quiénes emplean la información resultante, lo que permite describir la actividad (funciones, procedimientos y qué información se obtiene), el analista de requerimientos después de la descripción del problema por el cliente debe realizar en un documento inicial una lista de requerimientos específicos, para mencionar en detalle el problema planteado por el cliente, este documento debe ser analizado para darle consistencia durante la fase de análisis de requerimientos.

Otra clasificación de los requerimientos del sistema puede darse mediante la identificación de los elementos que la componen como: el proceso básico de la empresa, la finalidad de esta actividad, para llevarla a cabo, dónde se realizan estos pasos, personas que lo realizan, el tiempo que tarda en efectuarlo, cuál es la frecuencia en que se hacen, personas que emplean la información resultante, los datos utilizados o producidos en este proceso y los límites impuestos por el tiempo y la carga de trabajo (causa-frecuencia), los controles de desempeño que se utilizan (debilidad, estándares, errores).

Muchas de las características de calidad de los requerimientos han sido conocidas desde muchos años; sin embargo, aún hoy día muchos proyectos carecen de algunas de estas, por lo cual producen productos de software con poca calidad, para mejorar las características de calidad de los requerimientos es necesario detallar los conceptos que permiten optimizar la calidad de los requerimientos de software la cual debe estar dada de acuerdo a sus características, permitiendo garantizar a los desarrolladores y clientes su entendimiento y utilización.

Instrucción

Tengan en cuenta que también deben estructurar su investigación según los aspectos que encontrarán en la galería Aspectos clave del software de los recursos de aprendizaje.

Instrucción

Para saber cuáles son las características de calidad de los requerimientos debe ingresar a la nube de palabras Características de calidad de los requerimientos.

Especificación y validación

Para hacer una buena validación de requerimientos es imprescindible que se haga un listado de estos, luego se clasifique y se evalúan para determinar en realidad cuales son los que se necesitan, estos deben identificarse de la siguiente forma.

Figura 3.
Fuente: Shutterstock/563400433

Requerimientos de las transacciones de los usuarios

Los proyectos de software se basan en requerimientos del usuario los cuales se recolectan y se categorizan en grupos para llegar a un acuerdo y definir el alcance del proyecto, esta clase de requerimientos capturan, procesan y almacenan información, para dar al usuario la facilidad de navegar en el sistema y utilizar sus funciones, por eso es importante saber la forma en que se lleva a cabo las transacciones con el objetivo de realizar un análisis de los procesos del sistema, en los que se requiere una interacción entre el usuario y el sistema, con el fin de crear una interfaz que satisfaga todos los requisitos establecidos.

Estos se convierten en formatos de pantallas, cuadros de diálogos, menús e informes, para determinar el modo de llevar a cabo oportunamente este proceso en su construcción, emplearemos una activación de las distintas operaciones de las herramientas mediante menús y submenús, botones de opciones, listas de chequeos, cuadros de listas, íconos, formularios y ventanas de la aplicación, en donde se configuran las opciones que pueden ser utilizadas y las que no se necesiten en un momento determinado, a su vez se tendrá una aplicación que muestre los resultados de los procesos de identificación mediante un árbol, donde los diferentes nodos representan las categorías, por esto se exigirá la confirmación del usuario para cualquier transacción y operación puede ser de cancelación o cierre de la ventana.

Para llevar a cabo esta actividad deben reunirse el analista y el grupo de usuarios para determinar con mayor precisión los elementos que conforman cada una de las interfaces es recomendable resolver los siguientes cuestionamientos:

- ¿Cómo se inician las transacciones?
- ¿Cuáles son los datos de origen, entrada y almacenamiento?
- ¿Quiénes hacen parte de las transacciones?
- ¿Quién inicia las transacciones y cuál es su propósito?
- ¿Cada cuánto ocurre?
- ¿Qué volumen de información maneja la transacción?
- ¿Cuáles son las personas clave?
- ¿Qué tipo de controles se usan y qué métodos aplica?

Requerimientos de decisión de los usuarios

En un sistema de información es muy importante atender las solicitudes que hagan los usuarios, pues son ellos los que realizan las actividades cotidianamente e interactúan con el entorno, por eso son los llamados a determinar la funcionalidad del sistema y a la vez evaluar si es amigable.

Es decir, fácil de aprender, fácil de utilizar y que responda a todas las necesidades de la organización, el software debe atender las necesidades requeridas en cuanto a transacciones se refiere, deben capturar, procesar y almacenar los datos necesarios evitando perder información importante que pueda incidir en la toma de decisiones, sobre todo porque estas se adoptan con base en la información obtenida de forma integral para que los gerentes sepan cómo actuar frente al tema. Las acciones concernientes a las decisiones no tienen un procedimiento enmarcado ni un estándar específico de acciones, por eso la importancia de que los sistemas capturen la información apropiada que sirva como soporte al momento de tomar una decisión, para ello se requiere de procedimientos adecuados cuando se llevan a cabo las transacciones, por eso muchas veces el trabajo que se hace en una dependencia afecta directamente a las otras.

Para evitar esta incertidumbre se debe involucrar a todos los usuarios del sistema para encontrar, con sus aportes, estos casos inusuales, es allí donde resalta la importancia de la aplicación de técnicas de recolección de información como son los cuestionarios, entrevistas, lluvia de ideas, revisión de registros, la observación directa y otras técnicas que se puedan emplear y que permitan solucionar este conflicto.

Otra estrategia es acudir a los manuales de funciones de los usuarios de la dependencia, las políticas institucionales, la estandarización de procedimientos operacionales que utiliza la mayoría de los empleados y que sirven de guía a los gerentes, estos documentos y técnicas pueden facilitar el trabajo del analista para comprender mejor las acciones que debe realizar el sistema y determinar qué funciones requieren apoyo con las relaciones consecuentes dentro de la organización.

Amigable:
En desarrollo de software el término amigable hace referencia a que es fácil de entender y de usar.

Incertidumbre:
Sentir temor porque no se sabe que evento pueda ocurrir

En este caso la observación directa pasa a ser una herramienta fundamental porque se captura información real de la forma en que se llevan a cabo las actividades y da un indicio de que buscar y cómo evaluar lo que está buscando, por eso los analistas deben obtener información pertinente que lo lleva a resolver algunas inquietudes como:

- **¿Cuál es la información útil para tomar decisiones?**
- **¿Qué fuente es la que provee mayor información?**
- **¿Qué otros datos se requieren, pero el sistema no los captura?**
- **¿De qué forma se deben procesar los datos para que produzcan la información que apoyen la toma de decisiones?**
- **¿Qué tipo de información proveen las fuentes externas?**

Video

Especificación de requisitos.
http://bit.ly/2wTbhY4

Requerimientos de la organización

En el proceso de desarrollo de software se aplica la ingeniería de requerimientos, donde el analista debe obtener información de múltiples fuentes y debe clasificar en varias categorías de acuerdo a la información que recibe, no solamente se recaba la información para analizar el proceso, también se deben evaluar los demás subprocesos con los que nuestro sistema va a interactuar. Para identificar los requerimientos que permitan la fluidez de la información, es importante identificar a todos los involucrados en el proceso a fin de obtener y validar los datos de forma que estos posteriormente puedan ser analizados eficientemente, esto implica que siempre debe existir una comunicación fluida y continua entre clientes-usuarios y desarrolladores, aplicando técnicas que permitan recopilar la información necesaria con el objeto de reunirlos, para seleccionar los requerimientos.

Los requerimientos obtenidos deben de estar de acuerdo con los objetivos y planes de la organización, y aquellos requerimientos que no ayuden a lograr estos objetivos no deben ser incluidos. Las fuentes de donde se obtienen los requerimientos deben ser fiables, ya que de la información obtenida depende la naturaleza del producto a desarrollar y del ambiente de desarrollo.

Los interesados deben clasificarse de acuerdo con su actividad y perfil en el sistema así:

Usuario final: son las personas que usarán el sistema, están relacionados con la usabilidad, la disponibilidad y la fiabilidad del sistema; se relacionan con los procesos específicos que se deben realizar en el software dentro de los parámetros de su ambiente laboral, serán los que utilizan las interfaces y los manuales de usuarios.

Usuario líder: estos individuos comprenden el ambiente del sistema o el dominio del problema, donde será empleado el software desarrollado, proporcionan al equipo los detalles y requerimientos de las interfaces del sistema.

Analistas y programadores: son los responsables del desarrollo del producto en sí, ellos interactúan directamente con el cliente.

Personal de pruebas: se encarga de elaborar y ejecutar el plan de pruebas para asegurar que las condiciones presentadas por el sistema son adecuadas, son quienes van a validar si los requerimientos satisfacen las necesidades del cliente.

Según el PMI (Project Management Institute) los requerimientos de un proyecto pueden dividirse en dos categorías: 1) requerimientos de negocio y 2) requerimientos técnicos. Los primeros definen las necesidades y deseos de la organización en relación a la consecución del proyecto, mientras que los segundos se centran en las soluciones que harán posible la obtención de dichas metas. Todos son igual de importantes de satisfacer e imprescindibles para finalizar el proyecto con éxito.

Este tema es de mucha delicadeza y se recomienda ser intuitivo, ya que se debe determinar elementos claros que vayan direccionados a suplir estas necesidades.

- Permitir y procesar los datos para producir la información necesaria.
- La forma en que debe presentarse la información.
- Qué datos se originan en fuentes externas de la organización.
- Las rutinas deben ser claras para controles precisos.
- Información apropiada para direccionar la toma de decisiones.
- Determinar la fuente de información.

Lectura recomendada

Introducción a la ingeniería de requisitos

García, Conde y Bravo.

Gestión de requisitos

Herramientas

Para la gestión de requisitos, primero debe hacerse la recolección de información que permita definir los requerimientos, se acostumbra a utilizar varias técnicas, entre ellas tenemos: cuestionarios, entrevistas, lluvias de ideas y la observación directa.

Cuestionario: sirve para reunir información de varias personas a la vez, en ocasiones permite agilizar el proceso de recolectar información ya que se utiliza un formato que debe diseñarse cuidadosamente para poder obtener buenos resultados. El objetivo de esta técnica es obtener información de grupos de personas de forma acelerada y se pueden hacer preguntas redactadas y estructuradas de forma coherente a los usuarios del sistema actual o al propuesto, las personas que nos propician la información deben estar implicados en el sistema que se va a emprender, son administrativos o empleados y proporcionan datos útiles para el sistema.

Entrevista: se realiza para obtener información de forma verbal por medio de preguntas que formula el analista para que obtener datos del sistema. El analista debe ser bastante cauteloso ya que son opiniones que dan los usuarios mediante un diálogo donde se hacen preguntas concretas, profundas y abstractas sobre el flujo de información, procesos y procedimientos que se realizan, con la ventaja de que pueden ampliarse para obtener una mayor cantidad de datos, esta actividad se realiza al inicio del proyecto, para obtener información sobre aspectos globales del problema lo que permitirá obtener soluciones potenciales.

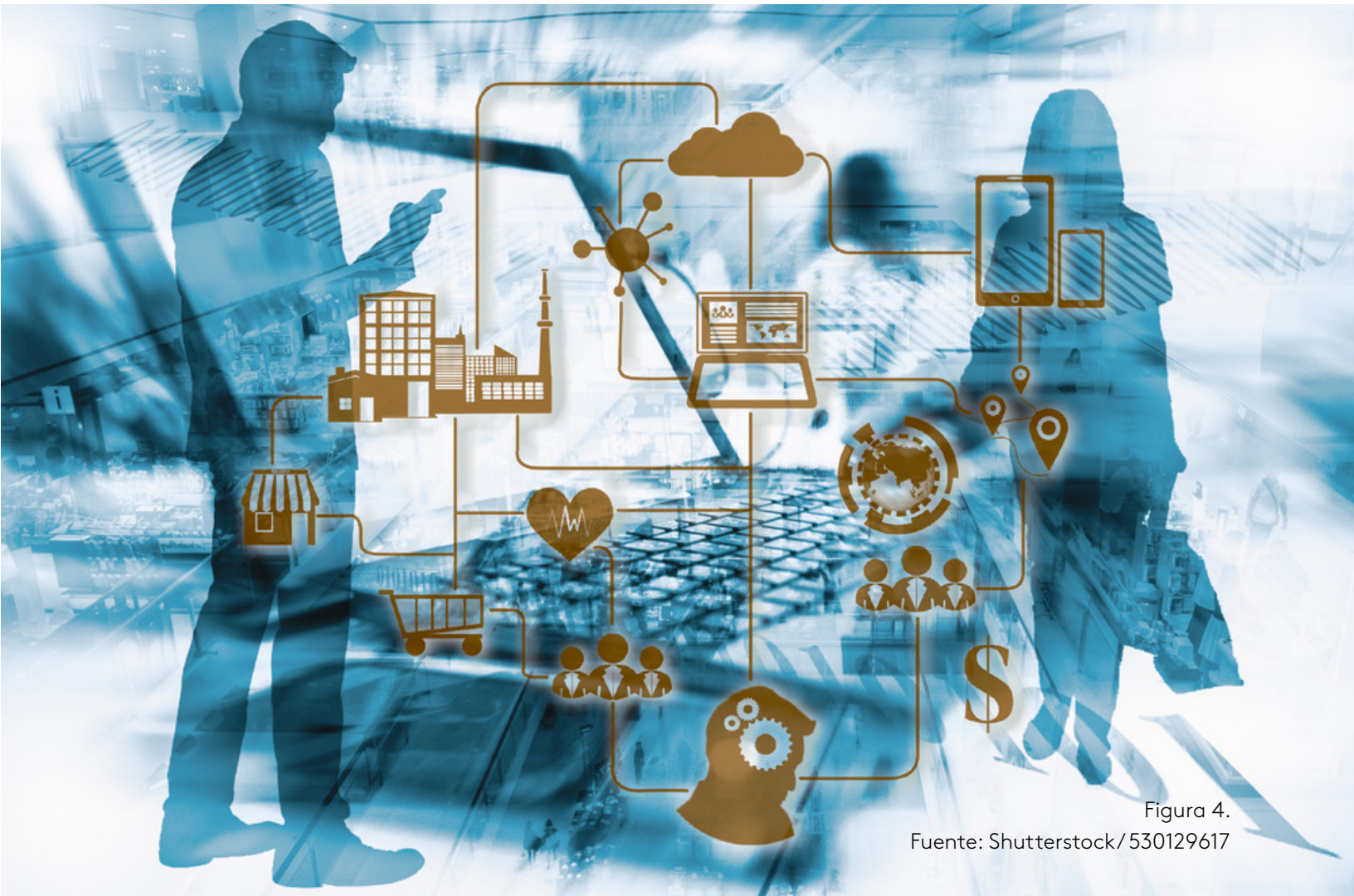


Figura 4.
Fuente: Shutterstock/530129617

El entrevistador debe tener habilidad y contar con una muy buena preparación para conducir una charla armoniosa que le permita indagar a profundidad al entrevistado para lograr el éxito de esta técnica. De igual forma, el entrevistador debe saber cómo tratar con problemas complejos y complicados, ya que debe consignar la información en documentos para posteriormente hacer un análisis que le ayude a tomar determinaciones sobre los requisitos del sistema.

Lluvia de ideas: consiste en reunir un grupo de personas las cuales lanzaran ideas para visionar posibles soluciones del problema, esta técnica permite desarrollar el pensamiento creativo a todos los niveles; básicamente se busca que los involucrados en el proyecto participen ya que entre más ideas surjan se podrán obtener y producir mayor calidad en los resultados, las mejores ideas se seleccionan y de estas se realizan filtros hasta escoger la mejor y sobre esta hacer un análisis para determinar los requisitos del sistema, se recomienda que el encargado de liderar la sesión sea el (gerente o director de área), quien debe lograr que todos participen y den ideas, además de esto hará las veces de juez para que conceda la palabra y de por finalizada la sesión.

Bernd y Allen (2002) afirman que "una vez que el cliente y los desarrolladores convergen en una idea en común, define las formas y fronteras del sistema y se ponen de acuerdo en un conjunto de términos estándar, comienza la solidificación" (p. 84).

En la aplicación de cualquiera de estas técnicas para llevar a cabo la recolección de datos se pueden utilizar preguntas abiertas que permitan opiniones, sugerencias y experiencias generales, o para conocer el paso a paso de un procedimiento o problema. Este tipo de preguntas permiten evidenciar lo que el usuario percibe y se puede enfocar a un componente del sistema tales como procesos, procedimientos, actividades etc.

Este es un ejemplo de algunas de las preguntas abiertas que se les pueden hacer a los usuarios:

- ¿Qué importancia tiene resolver el problema?
- ¿Por qué se quiere resolver el problema?
- ¿Qué solución recomienda el usuario?
- ¿Quién es el usuario?
- ¿Cuáles son sus necesidades?
- ¿Cuáles son sus expectativas para los conceptos de usabilidad, confiabilidad, y rendimiento?
- ¿Qué tecnología se tiene para implementar el sistema?
- ¿Cuánto tiempo lleva realizando el procedimiento?
- ¿Cuáles son sus habilidades, capacidades y experiencia?
- ¿Cuál es el valor de una solución exitosa?
- ¿Qué inconvenientes podría tener el desarrollo del producto?
- ¿Qué problemas podría causar este producto en el negocio?
- ¿En qué ambiente se usará el producto?

Observación directa: en esta técnica lo que se hace es que el analista dedica tiempo para observar directamente cómo las personas realizan las actividades concernientes al sistema de información para determinar que se está haciendo, la forma en que se hace, quién lo hace, con qué frecuencia se hace, cuánto tiempo dura realizando la actividad y dónde lo hace; esta técnica permite clasificar y documentar hechos reales.

Para llevar a cabo esta actividad se recomienda seguir estos pasos:

- Determinar el procedimiento o actividad a observar.
- Cuáles son los objetivos por los cuales se procede a hacer la observación.
- Identificar la forma de consignar la información obtenida.
- Llevar a cabo la observación de forma cuidadosa y crítica.
- Documentar los datos obtenidos.
- Analizar e interpretar la información.
- Realizar las conclusiones pertinentes.

Para mejorar los conocimientos adquiridos recomiendo la lectura complementaria:

Lectura recomendada

Análisis de requisitos.

Instrucción

Y para adquirir habilidades en la especificación de requerimientos recomiendo desarrollar la actividad de repaso Analicemos la situación dentro de los recursos de aprendizaje.

Metodologías de desarrollo de software

El proceso de desarrollo de software es el mismo que se denomina ciclo de vida del desarrollo de software. Este proceso consiste en emprender un proyecto aplicando una de las diferentes metodologías existentes, estas se componen de diversas fases: análisis, diseño, desarrollo, implementación y prueba.

Para que el producto cumpla con estándares de calidad utilizamos las métricas que son medidas cuantificables y sirve como herramienta para medir la eficacia del proceso del desarrollo de software, estos deben ser revisados y analizados por el director del proyecto para obtener evidencias buenas o malas para realizar estimaciones y determinar cuándo se deben llevar a cabo mejoras y proporcionar una visión profunda del proceso y la comprensión del proyecto.

La medición del proyecto y sus recursos asociados son el elemento principal sobre el que se basa el estudio de las métricas del proceso de software. Cuando se mide el proyecto, el objetivo fundamental que se pretende es de reducir el coste total y el tiempo de desarrollo del mismo.

La norma ISO 9000 ISO/IEC 9126 indica los atributos que deben contener las características de:

Lectura recomendada

El proceso del software

- Funcionalidad: adaptabilidad, exactitud, interoperabilidad y seguridad.
- Usabilidad: comprensibilidad, aprendizaje, operabilidad y atractivo.
- Mantenimiento: análisis, cambio, estabilidad y prueba.
- Confiabilidad: madurez tolerancia a fallos y recuperabilidad.
- Eficiencia: comportamiento del tiempo y uso de los recursos.
- Portabilidad: adaptabilidad, instalación, coexistencia y reemplazo.

Ciclo de vida del software

Algunas de las actividades que más se dificulta al momento de abordar un proceso de desarrollo de software es la complejidad que lleva al comprender plenamente lo que es el ciclo de vida del software. En este apartado, entonces, se hace una descripción e incursión en el estándar que se emplea para el desarrollo de procesos y revisaremos algunos de los modelos del ciclo de vida del software y en qué consiste la administración de actividades y productos para que se facilite la comprensión y especificación de estas actividades en el desarrollo de un producto.

El ciclo de vida del software es una estructura aplicada al desarrollo de un producto y define las fases y el estado de cada una de ellas, a través de las cuales se mueve un proyecto e intenta determinar el orden y la transición de estas, permitiendo a los desarrolladores concentrarse en la calidad del software, pues posibilita que se detecten errores a tiempo.

El modelo de ciclo de vida del software detalla las fases principales de desarrollo de este, como lo evidenciamos en la siguiente figura, con el fin de ordenar las actividades y tareas del proyecto; también suministra un marco técnico para la administración del desarrollo y las fases primarias esperadas a ser ejecutadas durante estas actividades, provee un espacio de trabajo para la definición de un descriptivo proceso de desarrollo de software y respectivo mantenimiento permitiendo estimar recursos, puntos del sistema, desarrollo de software, prueba del sistema e implementación.

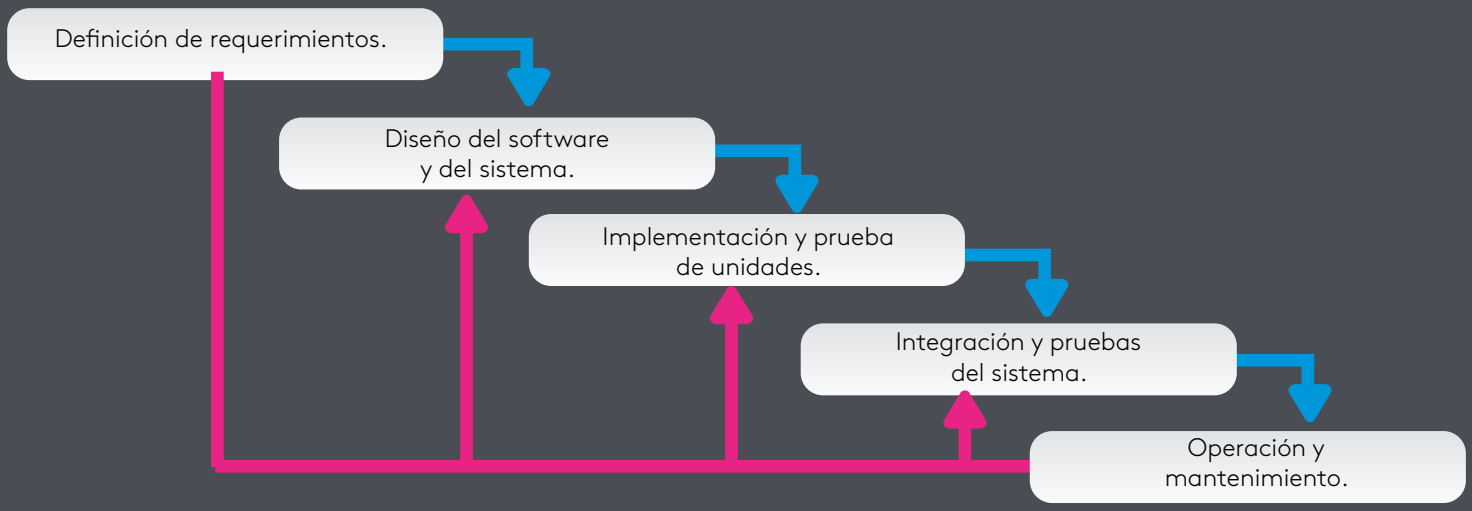


Figura 4. Ciclo de vida
Fuente: Rogger Pressman 2002.

El estándar para el desarrollo de procesos del ciclo de vida del software

El objetivo principal del estándar es el de proporcionar una estructura común para que todas las personas como compradores, proveedores, desarrolladores, personal de mantenimiento, operadores, gestores y técnicos involucrados en el desarrollo de software usen un lenguaje común que se establece en forma de procesos bien definidos, los cuales se clasifican en tres tipos:

Procesos principales: adquisición, suministro, desarrollo, explotación y mantenimiento.

Procesos de soporte: documentación, gestión de la configuración, aseguramiento de calidad, verificación, validación, revisión conjunta, auditoría y resolución de problemas.

Procesos de la organización: gestión, infraestructura, mejora y formación.

Para establecer un proceso de ciclo de vida para el software la norma ISO/IEC 12207 contempla procesos y actividades que se aplican desde la definición de requisitos, pasando por la adquisición y la configuración de los servicios del sistema, hasta la finalización de su uso.

El protocolo considera que la estructura del estándar se implementó de manera flexible para que pueda ser adaptada según las necesidades de cualquiera que lo use y se basa en dos principios fundamentales: 1) la modularidad que permite adaptar procesos con facilidad de adaptación y una máxima adherencia y 2) la responsabilidad, que busca asignar un apoderado para cada proceso, facilitando la aplicación del estándar en proyectos en los que participan varias personas u organizaciones involucradas.

Modelos del ciclo de vida del software

Un modelo del ciclo de vida del software especifica las fases y define las etapas primarias para ser aplicadas durante su proceso, ayuda a administrar el progreso del desarrollo del proyecto.

Los modelos del ciclo de vida más conocidos son:

Modelo clásico o de cascada: creada por Winston Royce a finales de 1970, define como la secuencia ordenada de etapas, cada una revisada cuando ha finalizado para determinar si se está listo para la siguiente.

Ha sido el más utilizado y ofrece una velocidad de desarrollo aceptable, siendo sometido a numerosas críticas, debido a que es restrictivo y riguroso, lo que dificulta el desarrollo de proyectos de software moderno; desde entonces muchos equipos de desarrollo han seguido este modelo que modificándolo funciona mejor; sin embargo, han surgido nuevos modelos, incluyendo los que pretenden desarrollar software más rápidamente, algunos de ellos son:

Modelo espiral: consiste en dividir un proyecto en miniproyectos orientado a riesgos, pero en su desarrollo se puede mezclar con otros modelos, siendo este un modelo meta e interactivo donde utiliza cuatro pasos en cada ejecución del desa-

rrollo como fijar objetivos, analizar riesgos, desarrollar y verificar y por último planificar.

Modelo de desarrollo de prototipos: es utilizado para que el usuario tenga una vista preliminar de parte del software, este se inicia elaborando un prototipo del producto final, donde se definen objetivos, luego se recolectan y se refinan los requisitos y las áreas del esquema donde se haga necesaria más definición; de esta manera tanto el ingeniero de sistemas como el usuario entienden de mejor manera cuál será el resultado de la construcción cuando los requisitos sean favorables. Este modelo reduce costos y aumenta la probabilidad de éxito.

Este modelo puede ser usado como parte de la fase de requerimientos (determinación de requerimientos) o justo antes de la fase de requerimientos (como predecesor de requerimientos).

Modelo incremental: es la unión de las mejores funcionalidades del modelo de cascada y de prototipo; este proceso es de construcción, siempre incrementando subconjuntos de requerimientos del sistema, el cual no demanda una forma específica de observar el desarrollo de algún otro incremento, sino que está orientado a ser operacional en cada incremento y no ser solo una vista previa de cómo sería el sistema en su versión final.

Entre otros modelos se encuentran: el modelo extremo, el modelo de desarrollo concurrente, el modelo espiral win win (ganar & ganar), modelo scrum, etcétera.

Para obtener mayores conocimientos los invito a leer la siguiente lectura complementaria:

Lectura recomendada

Modelos del ciclo de vida

BIBLIOGRAFÍA

Pressman, R. (2001). *Ingeniería del software un enfoque práctico*. Madrid: McGraw Hill.

Senn, J. (1997). *Análisis y diseño de sistemas de información*. Madrid: McGraw Hill.

INGENIERÍA DE SOFTWARE I

Angel Varon

EJE 3

Pongamos en práctica

AREANDINA
Fundación Universitaria del Área Andina
MIEMBRO DE LA RED
ILUMNO

INTRODUCCIÓN

El presente eje, con énfasis en la puesta en práctica, centra su atención en el diseño orientado a objetivos y a la forma de desarrollarlo. El estudiante podrá encontrar recursos y actividades diversas y una actividad de evaluación individual.

Diseño orientado a objetos (DOO)

El diseño orientado a objetos reside en determinar un proceso como un sistema de objetos que operan entre sí, tomando como referencia el diseño estructurado.

Un software que opera con métodos orientados a objetos necesita primero establecer los objetos del programa, estos objetos son instancias de clases.

Un software orientado a objetos está compuesto de fases como:

- DOO: Diseño orientado a objetos.
- POO: Programación orientada a objetos.
- AOO: Análisis orientado a objetos.

Al emplear un modelo orientado a objetos, en las fases de análisis de un proyecto genera grandes utilidades, por esto la programación orientada a objetos posee ventajas aplicables en todo el ciclo de vida de un proyecto como lo son:

- Modificabilidad: se pueden emplear omisiones o agregados a programas, es decir los programas se pueden cambiar fácilmente.
- Mantenibilidad: hace que los programas sean claros en su lectura, comprensión y control de la complejidad del programa, es decir reside en la facilidad del mantenimiento.
- Credibilidad: suelen ser más confiables los programas orientados a objetos ya que estos se encuentran establecidos y han superado las pruebas de calidad.
- Reutilización: si el diseño de los objetos ha sido el indicado este se puede utilizar en diferentes proyectos y las veces que se quiera.

Conceptos básicos de la orientación a objetos

El modelado orientado a objetos reside en depurar reiteradamente, de forma tal que, en la fase de implementación, se introduzca un diseño adecuado para que no se creen problemas inesperados.

La orientación a objetos se establece por conceptos como:

Clase: es un conjunto de elementos que conforman un objeto que posee características propias que se denominan atributos, que contiene variables con métodos lógicamente estructurados para operar dichos datos, según el comportamiento de la clase. Ejemplo:

```
Clase vehiculo {
    Atributos o caracteristicas // estas son las variables
variables
        Placa
        Tipo de vehiculo
        Color
        Modelo
        Ejes
            }
Operaciones a realizar
Asignar placa
Asignar tipo de vehiculo
Asignar un color
Asignar un modelo
Asignar ejes
```

Figura 1.
Fuente: propia

Métodos
Serie de pasos para realizar una actividad.

Objeto: Es un elemento que ocupa espacio en memoria y que contiene unos datos agregados y que puede modificar otros objetos. Ejemplo:

Objeto taxi

Placa ICB 314

Figura 2.
Fuente: propia

Atributo: es una característica propia de una clase e identifican el estado:

Clase vehiculo

Objeto Taxi Placa ICB 314

Atributos (Numero de puertas, tipo de combustibles, modelos)

Figura 3.
Fuente: propia

Métodos: son las acciones u operaciones que realiza una clase. Ejemplo:

Clase vehiculo

Objeto Taxi Placa ICB 314

Atributos (Numero de puertas, tipo de combustibles, modelos)

Operaciones (Arrancar, Parquear; Frenar, etc)

Figura 4.
Fuente: propia

Instancia: se denomina instancia a los objetos que proceden de otro, es decir que todos los objetos son instancias de algún otro.

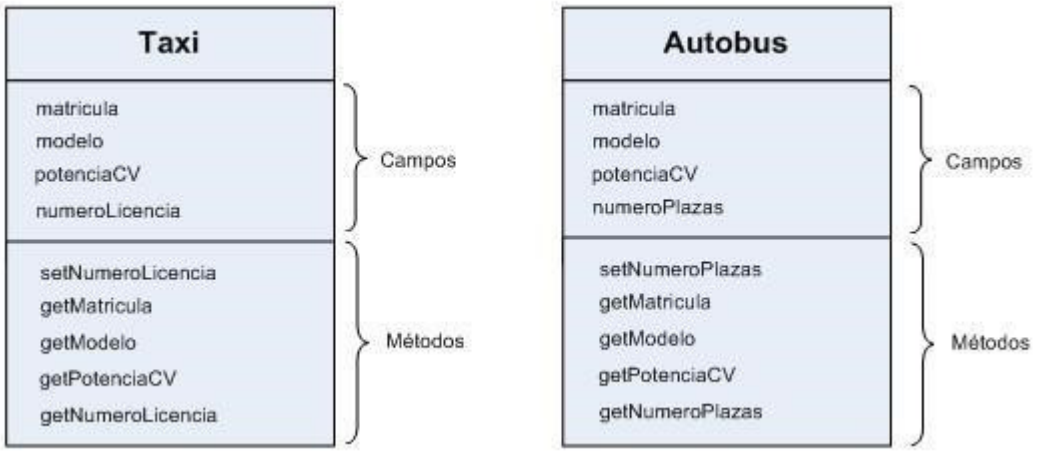


Figura 5. Instancia de clase
Fuente: http://bit.ly/2zfWR53

Herencia: acción que permite intervenir automáticamente métodos y datos entre clases, subclases y objetos.

Polimorfismo: los objetos realizan operaciones según órdenes que se les dan. Una sola orden puede ser interpretado o dar paso a distintas operaciones según qué objeto es el accionado.

Características del modelo orientado a objetos

Permite diseñar el software por módulos

El diseño debe obtenerse a partir de la especificación de requerimientos generada durante la fase de análisis y debe ser modular, esto es, el software debe estar dividido lógicamente en elementos que ejecuten funciones y operaciones específicas y debe generar módulos que exhiben niveles adecuados de independencia funcional.

Cuando se considera una solución modular para hacer frente a un problema, se puede plantear en distintos niveles de abstracción, como se indica a continuación.

Un nivel superior de abstracción supone una solución en términos extensos, utilizando un lenguaje propio del problema, un nivel más bajo de abstracción permitirá que la solución puede implementarse directamente, porque se toma una posición más procedimental, se combina una terminología orientada al problema con una orientada a la puesta en marcha del producto.

El producto asimila el encapsulamiento de forma adecuada

El encapsulamiento dentro de un software permite agregar variables y métodos dentro de una clase, conservando una sola entidad, por medio de un sistema de ocultamiento determinado, proporcionando que se conozca el comportamiento de este paquete de datos, pero no la complejidad de los mismos, evitando que sus características específicas sean vistas por quienes no han sido autorizados lo que garantiza una correcta emisión y recepción de dicha información.

El diseño permite que se agreguen módulos de forma sencilla y práctica, pues los elementos de este deben dar respuesta a instrucciones, definiciones de datos, o llamadas a otros procesos. La idea es organizar estos elementos de tal manera que tengan mayor relación entre ellos al momento de cumplir su operación, para que den respuesta a la funcionalidad del sistema.

Permite que las clases se puedan reutilizar, al momento de hacer la parte de desarrollo se puede aprovechar código creado, los objetos se pueden distribuir y la ejecución puede ser secuencial o simultánea.

A partir de lo anterior, algunas ventajas que encontramos son:

- Los módulos son robustos en lo referente a consistencia interna y poco ajuste externo (no permite variables globales).
- Permite la articulación en ambiente multiprocesador (objetos distribuidos).
- Facilita la creación de prototipos de forma ágil.
- Herramienta adecuada para aplicaciones dirigidas por eventos.
- Fácil de entender, facilita el mantenimiento.
- Utiliza variedad herramientas y bibliotecas extensas.

Introducción a UML

UML es un lenguaje unificado para modelar o diseñar software, cuyo objetivo es brindar a desarrolladores, arquitectos, ingenieros de sistemas, las herramientas para las fases de análisis, diseño e implementación del producto.

Abarca diversos componentes de dominio permitiendo seleccionar solo las partes útiles del lenguaje, facilitando el modelado de la mayoría de procesos presentándose en esquemas.

En un sistema de información es importante conocer la arquitectura, la cual inicia con el diseño que consiste en la representación gráfica mediante diagramas UML que han sido estandarizados para facilitar el trabajo de analistas, diseñadores y programadores que a su vez entiendan la estructura del sistema. Para lo anterior se han desarrollado herramientas CASE especiales para esta actividad, como Visual Paradigm, Rational Rose; Bo UML y Power Designer, entre otros.

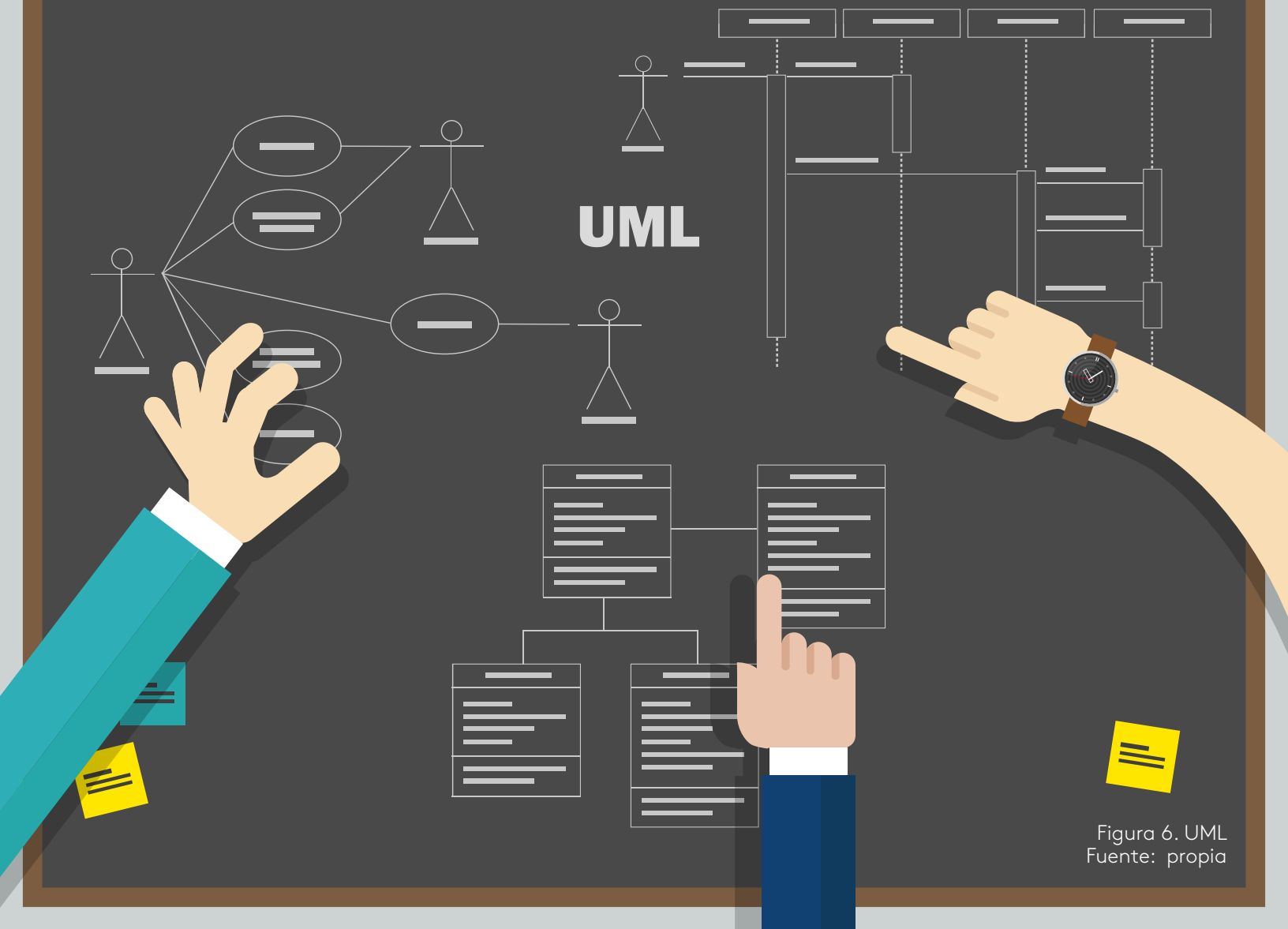


Figura 6. UML
Fuente: propia

Ejemplo

Para tener mayor claridad al respecto vamos a tomar de ejemplo el diseño de un piso de una edificación, para la construcción del software se deben tener en cuenta tres actividades fundamentales: 1) el diseño del sistema, 2) el diseño de los objetos y 3) la implementación, si comparamos estas actividades con las que se desarrolla un arquitecto observamos que después de tener el terreno y haber realizado algunos estudios de suelos y otros pertinentes a estas actividades, lo que se realiza es el plano de la construcción que va a edificar.

Si nos fijamos detalladamente podremos corroborar que todos los espacios son diseñados de forma coherente para prestar una buena utilidad a los residentes, la acometida eléctrica, la tubería de agua potable y los conductos de aguas negras se encuentran repartidos de forma adecuada para que pueda ser habitada, las habitaciones, los baños y la cocina las encontramos al alcance demostrando que los espacios se distribuyen según las normas y estándares de construcción, cumpliendo con requisitos de calidad para que quienes lo van a habitar se sientan cómodos y seguros. Pues bien, lo mismo sucede con nuestro sistema de información: debe construirse aplicando una estructura lógica y consecuente, que permita cumplir con los requisitos necesarios para su buen funcionamiento recurriendo a estándares de calidad determinados para la ingeniería de requerimientos.

En los proyectos de ingeniería de software necesariamente nos vemos abocados a utilizar (UML) lenguaje unificado de modelado a través de herramientas automatizadas CASE para realizar el modelo de requisitos, modelo de análisis, modelo de diseño, modelo de implementación, modelo de pruebas y modelo de documentación donde especificamos que el modelo de requisitos comprende el contexto del sistema delimitando su alcance.

La utilidad de los diagramas consiste en que muestran gráficamente partes del modelo, que a continuación representamos. Para profundizar más los conocimientos adquiridos recomiendo ahondar en la lectura complementaria:

Lectura recomendada

Fundamentos de ingeniería del software, cap. 12

Ros, J., Toval, A.

Para ampliar más los conocimientos adquiridos los invito a ver la videocápsula:

Video

UML Introducción.

http://bit.ly/2yoGZAz

Modelos UML

UML permite la elaboración de una serie de diagramas que nos muestra la siguiente figura.

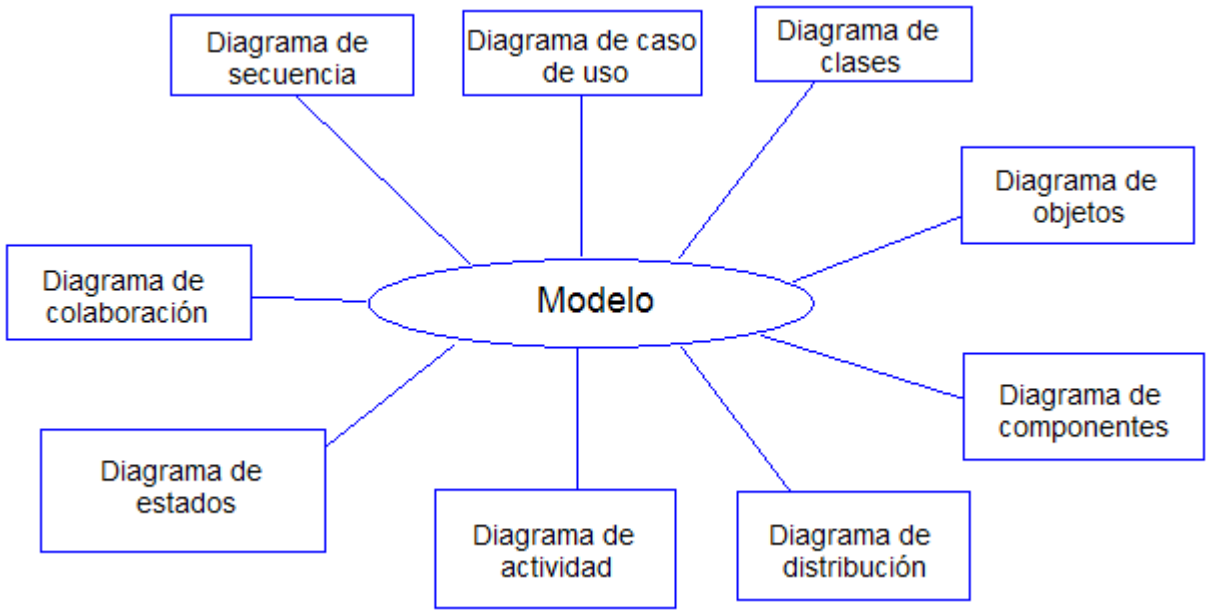


Figura 7. Diagramas UML
Fuente: propia

Diagrama de casos de uso

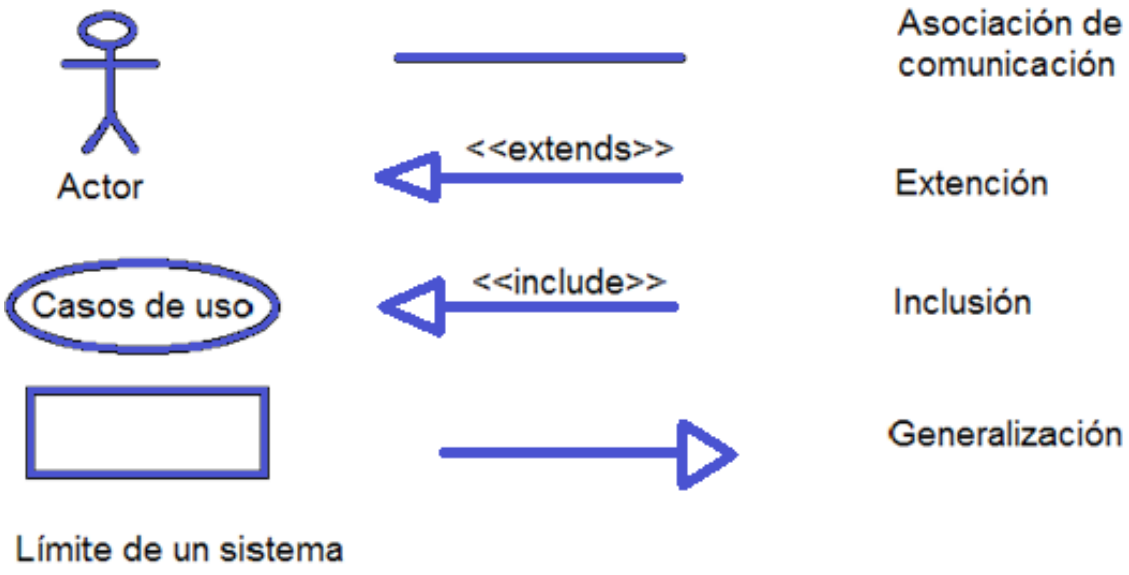


Figura 8. Herramientas para crear casos de usos
Fuente: propia

Los casos de uso son la representación de una operación/tarea/técnica para la captura de requisitos idóneos de un nuevo sistema o una actualización de software que sirve para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y otros sistemas participantes en el proceso. Es decir un diagrama que indica la concordancia entre los actores y los casos de uso de un sistema como lo muestra la ilustración diagramas de casos de uso.

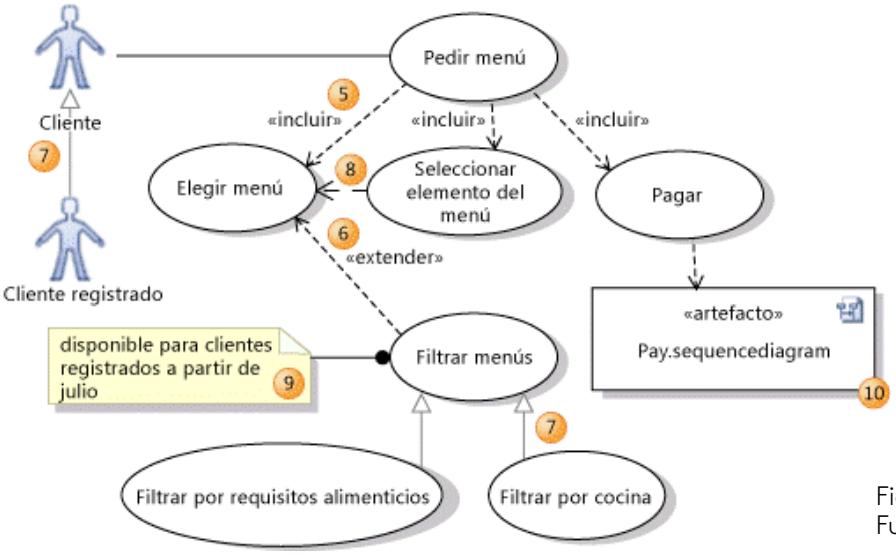


Figura 9. Ejemplo caso de uso
Fuente: http://bit.ly/2yRGFuY

Actores
Hace referencia a una entidad que interactúa con el sistema.

Diagrama de clases

Estos diagramas indican las distintas clases que integran un sistema la forma en que se vinculan con otras.

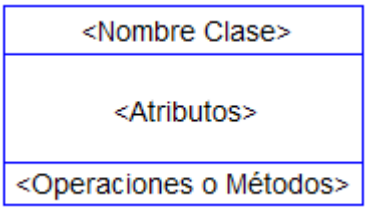


Figura 10. Diagrama de clases
Fuente: propia

Los diagramas de clase son (estáticos) porque señalan las clases, acopladas con los métodos y atributos, así como las relaciones estáticas entre ellas: qué clases (identifican) a qué otras clases o qué clases (hacen parte) de otras clases, pero no muestran los métodos mediante los que exponen entre ellas.

Para ampliar más los conocimientos los invito a ver la siguiente videocápsula:

Video

Programación orientada a objetos - Diagrama de clases y casos de uso.

http://bit.ly/2ggnJKP

Diagrama de secuencia

Estos diagramas indican los objetos y sus múltiples relaciones entre ellos, el intercambio de mensajes (es decir la forma en que se llaman) en un momento dado. Los diagramas de secuencia hacen referencia al orden y al instante en el cual se emiten los mensajes a los objetos que interpretan a través de líneas intermitentes verticales, con el nombre del objeto en la parte más alta, tal como lo evidencia la siguiente figura.

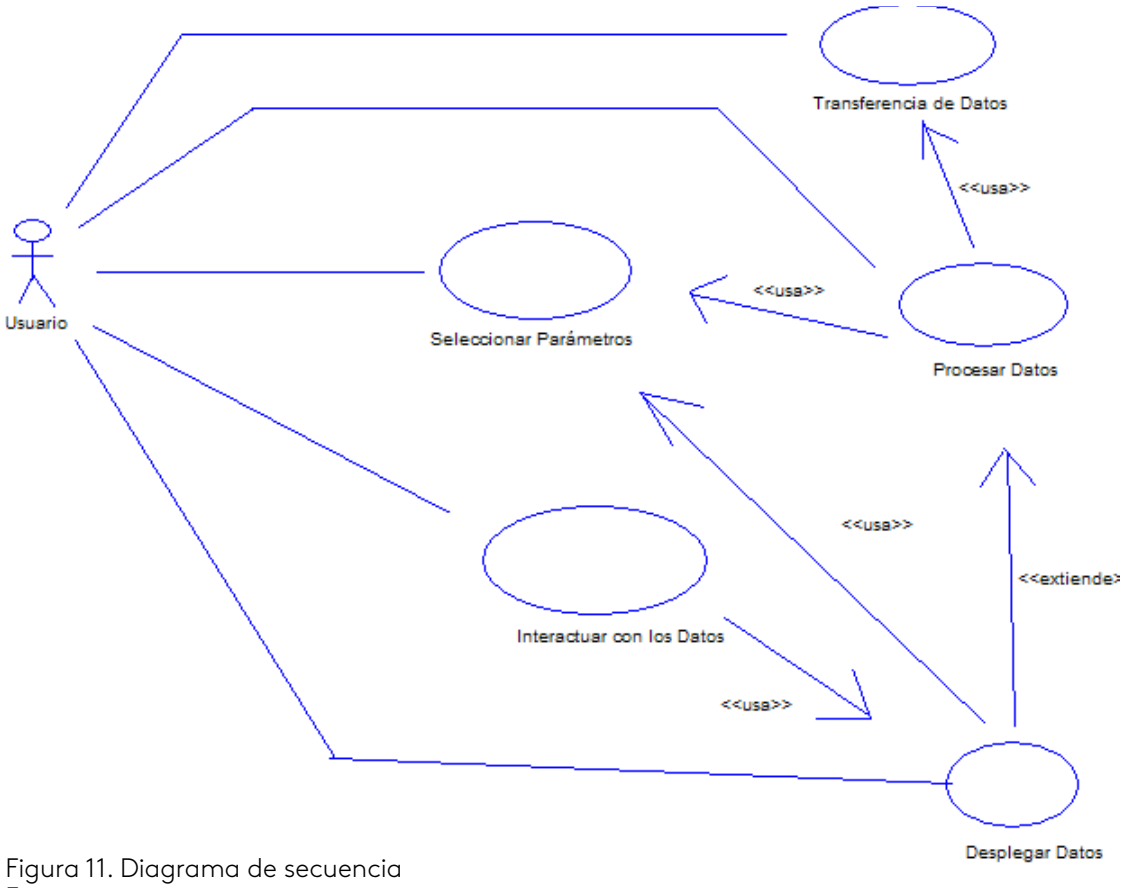


Figura 11. Diagrama de secuencia
Fuente: propia

El núcleo de tiempo también es vertical, acentuándose hacia abajo, de manera que los mensajes son emitidos de un objeto a otro en forma de flechas con los nombres de los procedimientos y parámetros.

Para profundizar más en el tema es muy importante realizar la actividad de repaso denominada caso de uso.

Diagramas de colaboración

Estos diagramas nos indican las interacciones que suceden entre los objetos y sus relaciones, que intervienen en una situación determinada, destacando los objetos que participan en el intercambio de mensajes.

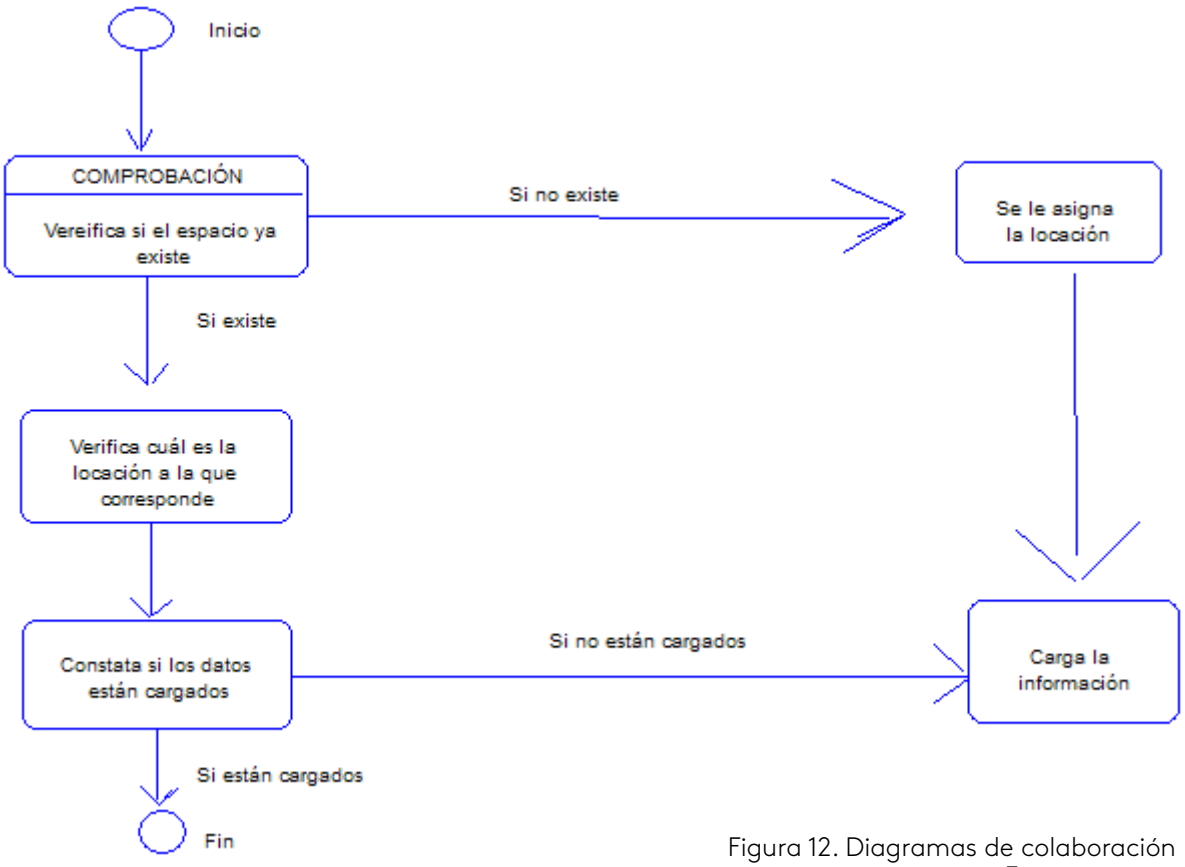


Figura 12. Diagramas de colaboración
Fuente: propia

Esta información es similar a la que nos indican los diagramas de secuencia, pero distinguiendo la manera en que las operaciones se producen en el tiempo, mientras que los diagramas de colaboración adhieren su inclinación en las relaciones entre los objetos y su topología.

En los diagramas de colaboración los mensajes emitidos de un objeto a otro se interpretan con flechas, indicando el nombre del mensaje, los límites y la secuencia del mensaje. Estos diagramas están adecuados para indicar una situación o flujo programa concreto siendo los mejores patrones para manifestar o aclarar ligeramente un procedimiento dentro de la estructura lógica del programa.

Diagrama de estado

Estos diagramas nos indican la diversidad de estados de un objeto durante su ciclo, los cambios de estado y los sucesos o en un objeto o en parte del sistema y los impulsos que inciden en los cambios de estado en un objeto, según se muestra a continuación.

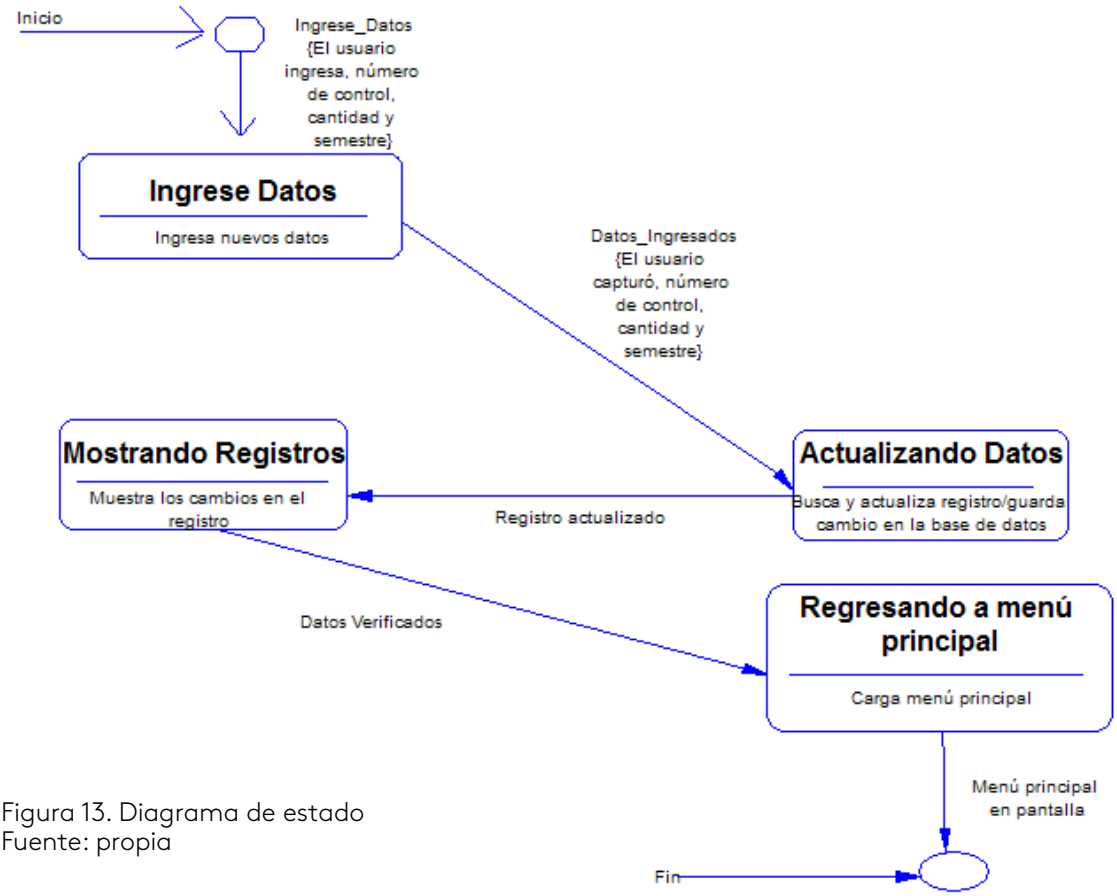


Figura 13. Diagrama de estado
Fuente: propia

Estos diagramas consideran a los objetos como máquinas de estado o androides limitados que pueden incluirse en un conjunto de estados restringidos y que probablemente cambien su estado a través de un impulso referente a un conjunto finito.

Por ejemplo, de tipo "net server" puede tener durante su ciclo uno de los siguientes estados:

- Activo.
- Atento.
- Operando.
- Suspendido.

Y los sucesos que pueden provocar que el objeto cambie de un estado son:

- Creación del objeto.
- El objeto es receptor de un mensaje.
- Un cliente solicita una conexión a través de la red.
- Un cliente concluye una solicitud.
- La solicitud se activa y se culmina.
- El objeto recibe un mensaje de detección, entre otros.

Diagrama de actividad

Este diagrama nos indica las actividades, así como las variaciones de una a otra actividad junto con los sucesos que pasan en ciertas partes del sistema. Estos diagramas detallan la serie de actividades en un sistema y son una manera singular de los diagramas de estado, que contienen exclusivamente actividades; también son parecidos a los diagramas de flujo procesales con la disimilitud de que todas las actividades están notoriamente ligadas a objetos, y siempre están incorporados a una clase, a una operación o a un caso de uso.

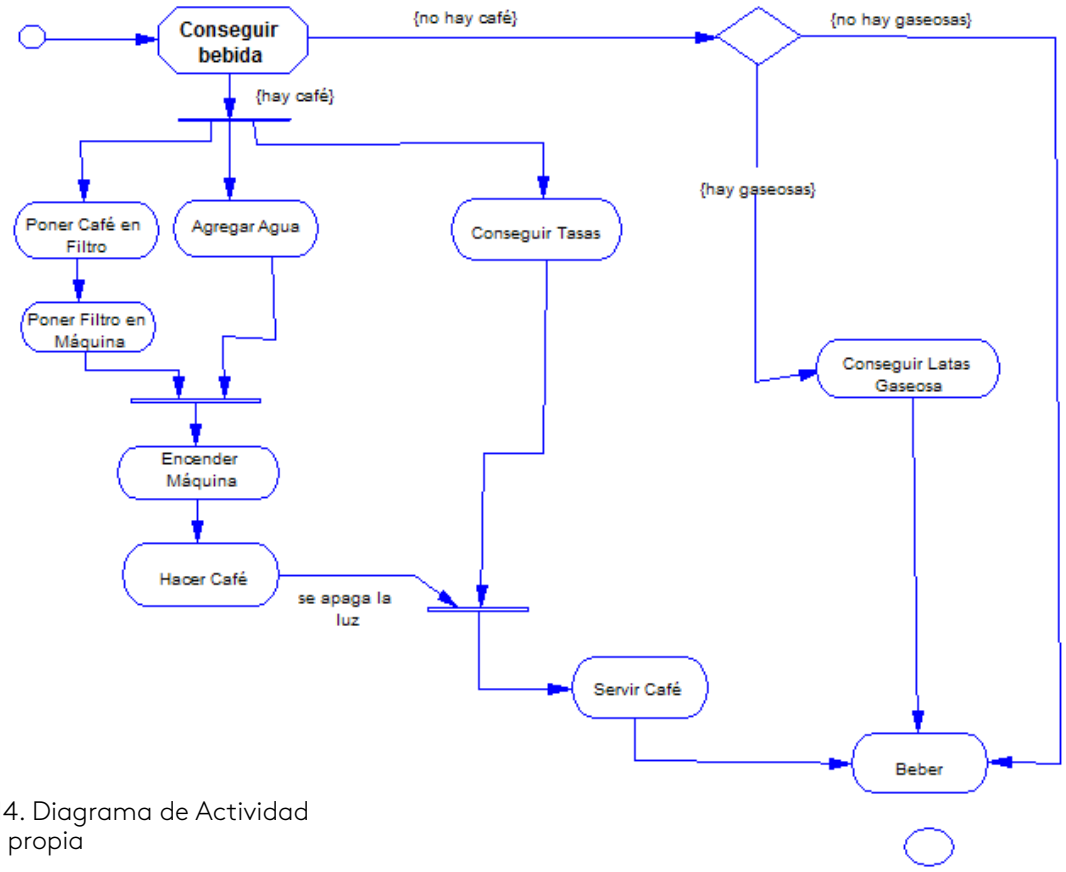


Figura 14. Diagrama de Actividad
Fuente: propia

Los diagramas de actividad sujetan actividades tanto secuenciales como paralelas; la ejecución secuencial se interpreta por medio de icono tenedor o espera, y en el caso de las actividades paralelas, no importa en qué orden sean aducidas (pueden ser ejecutadas concurrentemente o una posterior de otra).

Diagrama de componentes

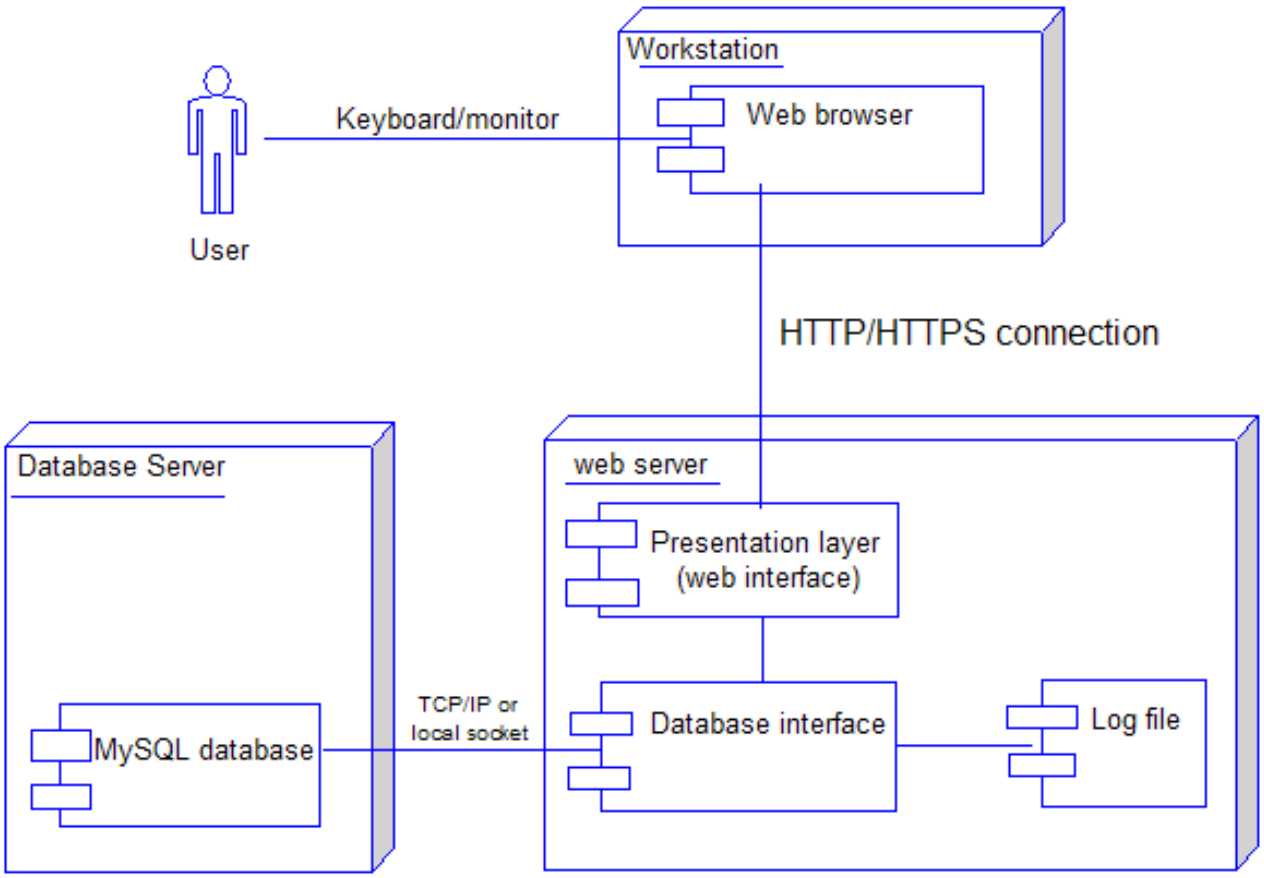


Figura 15.
Fuente: propia

Estos diagramas indican los componentes de superior escala de la programación del software (ya sea las tecnologías que lo integran como regiones, componentes CORBA, Java NetBeans o simplemente secciones del sistema visiblemente diferentes) y los mecanismos de los que está compuesto como los archivos de código de fuente, las librerías o las tablas de una base de datos; estos factores pueden poseer interfaces que admiten asociaciones entre componentes, como el ejemplo de la figura anterior.

Diagrama de implementación

Estos diagramas indican el grado de los componentes y sus relaciones al efectuarse y se interpretan los nodos que registran recursos físicos, como el ejemplo que muestra la siguiente figura.

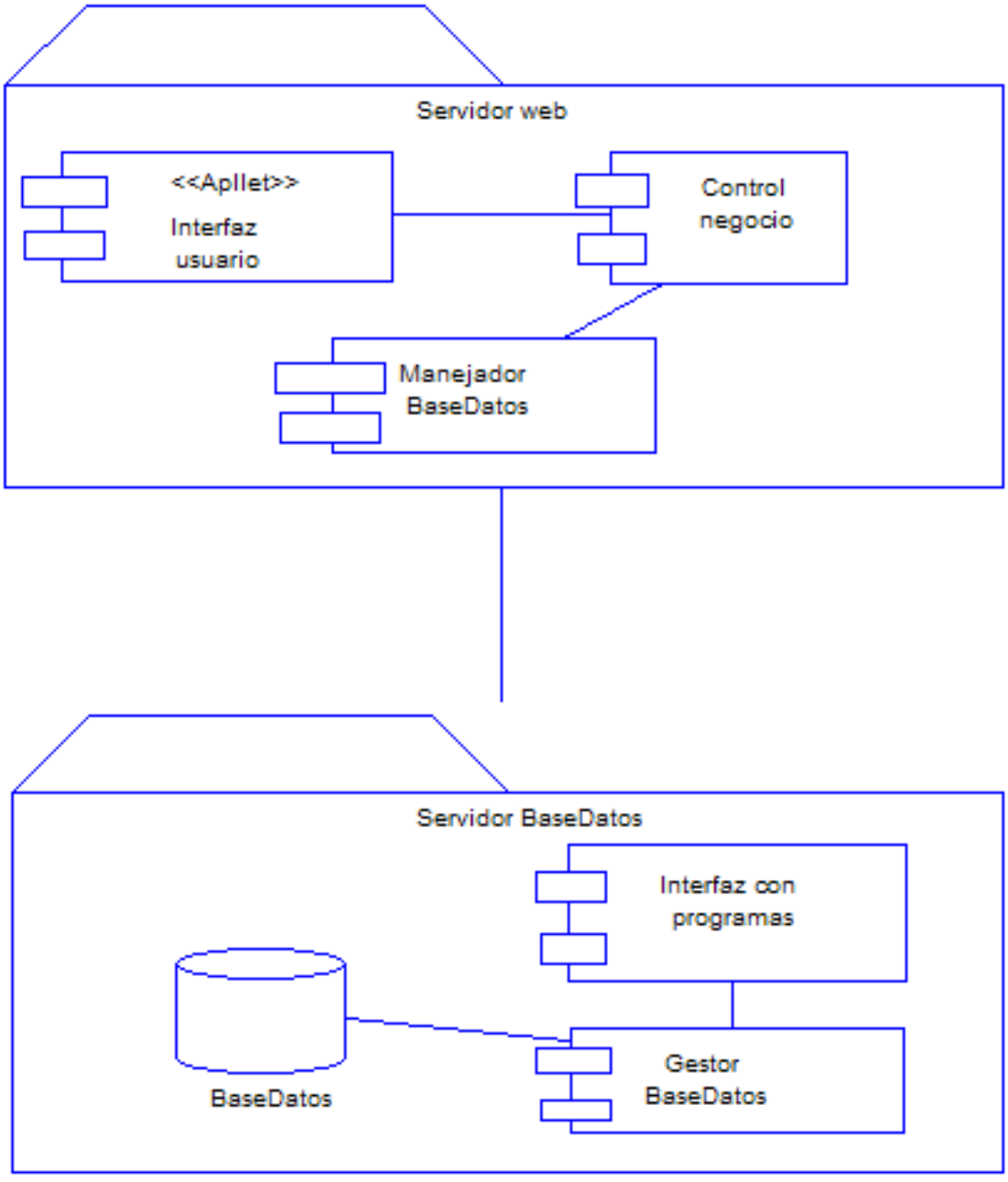


Figura 16. Diagrama de implementación
Fuente: propia

Diagrama de relaciones de entidad

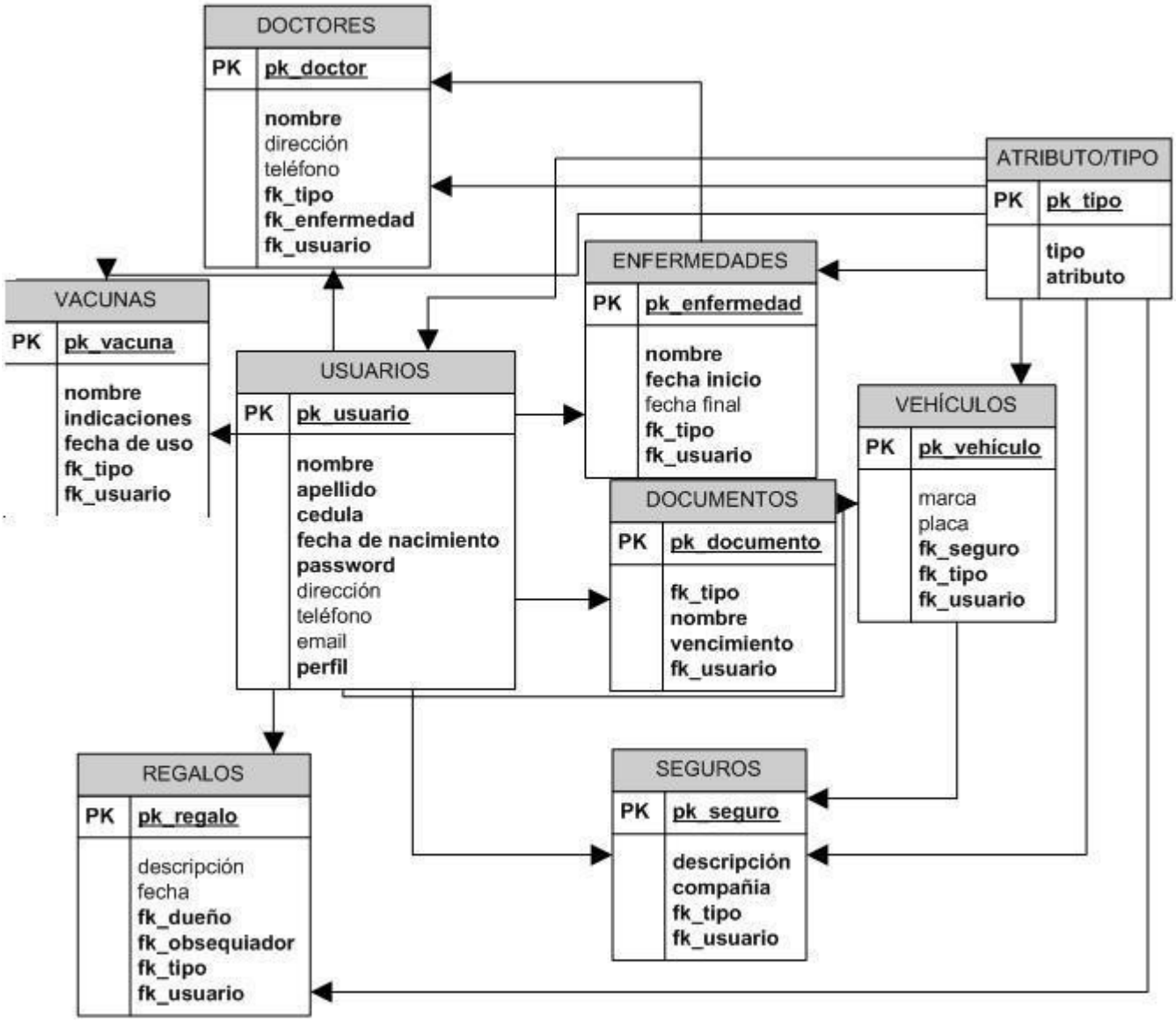


Figura 17.
Fuente: http://bit.ly/2yufJhL

Estos diagramas nos indican los datos apoyados en el diseño conceptual de las aplicaciones de bases de datos, trazando algunos conceptos en el sistema de información mostrando las relaciones y restricciones palpables entre ellos.

Un área de estos diagramas es llamada diagramas de relaciones de entidad extendida o mejorada (EER), se usa para anexar las técnicas de diseño orientadas a objetos en los diagramas ER.

Para profundizar más los conocimientos recomiendo revisar el tema 2 y el tema 6 del siguiente libro:

Lectura recomendada

Ingeniería del software

García, F., Conde, M., Bravos S.

Instrucción

De igual forma, lo invitamos a realizar la actividad de repaso del eje.

BIBLIOGRAFÍA

Whitten, J. y Bentley, D. (2008). *Análisis de sistemas. Diseños y métodos*. Madrid: Editorial McGraw Hill.

Dutoit A. y Bruegge, B. (2002). *Ingeniería del software orientado a objetos*. México: Prentice Hall.

INGENIERÍA DE SOFTWARE I

Ángel Varón

EJE 4

Propongamos

AREANDINA
Fundación Universitaria del Área Andina
MIEMBRO DE LA RED
ILUMNO

INTRODUCCIÓN

El presente eje aborda las características más importantes del proceso de diseño de un proyecto de software.

Características de seguridad del software

La seguridad informática es un tema de tendencia para cualquier usuario que utilice un sistema sea este informático o no, lo que nos ha de interesar es que se encuentre libre de amenazas y daños que puedan afectar su funcionamiento, la seguridad se divide en lógica y física, dependiendo de los ataques a los que se encuentren vulnerables.

Para determinar si un sistema es seguro debemos hallar cinco componentes incluidos en el mismo, a saber:

Confidencialidad: la información solo debe ser explícita para personal autorizado, es decir que el documento o archivo, que debe entenderse de manera comprensible, sea leído por la persona o sistema que esté autorizado.

Disponibilidad: debe encontrarse disponible la información cuando se solicite un servicio, datos o un sistema; y debe ser accesible y utilizable por los usuarios o procesos autorizados cuando lo requiere.

Integridad: es la cualidad que posee un documento o archivo que no ha sido alterado y que, además, permite comprobar que no se ha producido manipulación alguna en el documento original. La información no puede ser alterada por personas no autorizadas.

Irrefutabilidad: que no se niegue autoría (no rechazada).

Responsabilidad: deben registrarse las tareas que durante y posteriormente se realizan para establecer responsabilidades.

Instrucción

Para ratificar los conocimientos adquiridos recomiendo desarrollar la actividad de repaso del eje.

Propiedades conducentes

Estas residen en ser elementos que hacen parte de la seguridad de algún modo:

Correcto: propiedad de alcance restringido a los requisitos esperados de operación, donde la seguridad necesita que se le resguarde aún bajo circunstancias no conocidas, que son las que se presentan cuando el software se encuentra bajo un ataque.

Para que el software correcto sea seguro, se deben establecer requisitos explícitos de una conducta segura, si no esto resultaría ser un imposible.

Confianza: esta propiedad del software ratifica que este se ejecute de la forma que se espera, es decir que no contenga fallos ni defectos que generen vulnerabilidades que aumenten el riesgo en la seguridad.

Confiabilidad: dirigida a resguardar el funcionamiento predecible e indicado del software, a pesar de hallar debilidades y fallos sin intención y de alteraciones en su comportamiento no vaticinable en el entorno.

Predecible: esta radica en desplegar la seguridad del software a su ejecución bajo exigencias no previstas, en las que los intrusos tienden a rastrear fallos en el software o en su entorno.

Protección: se encuentra interrelacionada con la propiedad antes vista, pues cuando se logra una intrusión el sistema ha de parar o quedar parcialmente operativo de forma segura para evitar algún tipo de pérdida.

Metodologías vigentes

Figura 1.
Shutterstock/261389492

Una metodología de desarrollo de software que introduzca seguridad, debe brindar una referencia integrada, etapa por etapa en la totalidad del ciclo de vida para fomentar el desarrollo de la seguridad del software.

Aún no existe una metodología en especial, por lo que se han incorporado prácticas, tareas y principios durante el ciclo de vida donde se hace necesario efectuar controles y revisiones para que la calidad del software sea segura y garantizar que los parches y las actualizaciones no generen debilidades, ni fallos de seguridad.

Instrucción

Se recomienda implementar un programa para incorporar seguridad en el desarrollo de software, para conocerlo le invito a ver el videoresumen que se encuentra en los recursos de aprendizaje.

Últimamente se han llevado a cabo intentos por determinar los modelos más eficaces a la hora de realizar el ciclo de vida del software, de los cuales se destacan, el modelo en espiral, en cascada, modelo unificado y el iterativo e incremental.

Aunque también encontramos algunas metodologías que se están utilizando con diferentes niveles de éxito, en proyectos de desarrollo pilotos o de productos como lo son:

Microsoft Trustworthy Computing SDL: metodología activa, en evolución y disponible, tiene como finalidad, minimizar el número de errores de diseño y de código vinculados a la seguridad de los productos Microsoft y reducir deficiencias.

Oracle Software Security Assurance Process: metodología de proceso interno, externamente no propagado, aun se conoce muy poco sobre esta.

Comprehesive, Lightweight Application Security Process (CLASP): incluye datos completos sobre las tareas en cada fase del ciclo de vida, los riesgos de no incorporarlas y una lista de debilidades disponible como plugin para Rational Unified Process y para seguridad de software, se encuentra bajo licencia open source 7 touchpoints.

TSP Secure: se constituyó como una presentación hecha en el 2006 por Software Engineering Institute SEI, mostrándose como una extensión a Team Software Process TSP.

Plugins
Indica que es un conector

SEI
Instituto de ingeniería del software

Se encuentran también proyectos de ámbito académico sobre metodologías, cuya eficacia no ha sido comprobada en proyectos de desarrollo a través de su aplicación, algunas de estas son:

- AEGIS Appropriate an Effective Guidance In Information Security.
- Waterfall Based Software Security Engineering.
- RUPSec Secure Software Engineering Process Model Rational Unified Process Secure.
- SSDM Secure Software Development Model.
- Extensiones de Seguridad propuestas a MBASE.

Haciendo énfasis en estas metodologías y realizando comparaciones en las tareas asociadas al ciclo de vida del software se incorporan otra serie de tareas transversales durante las etapas de planificación, ejecución, soporte, monitoreo y mantenimiento.

Para la realización de muchas de estas tareas se proponen los siguientes elementos:

- Árbol de amenazas.
- Consecuencias del proceso de análisis de riesgo.
- Codificación segura.
- Patrones de ataque.
- Pruebas de intrusión.

Pruebas de intrusión: Evaluación para verificar que el software puede resistir ataques por parte de personas mal intencionadas

Para profundizar más en el tema lo invito a leer la lectura complementaria:

Lectura recomendada

Metodologías para desarrollar software seguro.

J. Brito

Distribución del esfuerzo en un proyecto de software

La distribución del esfuerzo de un proyecto de programación es un proceso que se utiliza para gestionar el desarrollo de un producto de software, dentro de un plazo previsto y con recursos establecidos. La administración del proyecto involucra no solo a la organización técnica, sino que requiere de dirigir recursos humanos y habilidades organizativas, apoyadas por herramientas de trabajo para incrementar la productividad contando con planes estratégicos, tácticos y operacionales con el fin de alcanzar las metas establecidas a través del ciclo de vida del proyecto, el cual ha de estar conformado por dos niveles de actividades que son: 1) las actividades de gestión que se encargan de la administración, personas, procesos y procedimientos que planifican y construyen el sistema; y 2) actividades de desarrollo, centradas en aplicar cada fase de la metodología seleccionada, agrupadas en áreas funcionales mediante estructura de trabajo para el análisis, diseño y desarrollo.

La administración de proyectos se basa en la planificación del proyecto, en una estructura que contiene un conjunto de actividades como los elementos organizativos:

- El proceso administrativo con responsabilidades y supervisión de participantes.

- El proceso de desarrollo como métodos, herramientas, lenguajes, documentación y apoyo, por último, el proceso con distribución organizada de tiempos en los que se realizan los trabajos.

Los grupos para la administración de actividades deben ser pequeños, máximo ocho personas, si el proyecto es muy complejo se deben dividir estos equipos en subsistemas y se definen adecuadamente la interfaz y los estándares de calidad siendo el grupo de administración el responsable de la planificación, desarrollo, supervisión, control de tareas y aseguramiento de cumplir con estándares y tiempo, ejecutando tareas como la generación de propuesta, estimación de costos, proyección, monitoreo, seguimiento, revisiones, selección y evaluación de personal y documentación.

Proyección:
Planificar una serie de actividades en un lapso determinado de tiempo

xEl otro grupo, el de desarrollo, se encarga del diseño, desarrollo, mejoras, soporte a producción, mantenimiento, pruebas, instalación de paquetes.

Administración de proyectos de software

La administración de proyectos de software se encarga de la administración de los recursos destinados donde se deben aplicar cuatro conceptos fundamentales para alcanzar el propósito por el cual se va a desarrollar el software, los cuales se presentan en la ilustración Conceptos fundamentales sobre gestión de proyectos

Figura 2.
Fuente: propia

Instrucción

Para conocer los conceptos fundamentales sobre administración de proyectos los invito a ingresar a la infografía Administración de proyectos de software

Estimación de coste del software

Figura 3.
Fuente: Shutterstock

La estimación de coste de software implica la aplicación de técnicas y procedimientos para conocer el valor del desarrollo del software, donde se identifican los recursos en términos de dinero, esfuerzo, capacidad, conocimientos, tiempo de duración y capital humano para llevar a cabo el proyecto, dando garantía a la eficiencia, excelencia, calidad y competitividad del mismo.

Las técnicas de estimación permiten obtener una idea aproximada del costo real del proyecto, donde aplican métricas y modelos clasificando algunos así: métricas de productividad, de calidad, técnicas de estimación, métricas orientadas al tamaño, orientadas a la función y orientadas a la persona, medidas relacionadas con el tamaño del código (LOC – Lines of Code) líneas de código, medidas relacionadas con la función (FP – Puntos de Función), los puntos de objetos (PO), (Cocomo) modelo constructivo de costo: meses/hombres a aplicar al proyecto y Cocomo II y modelos algorítmicos de costes en la planificación; todos estos construidos para agilizar tareas.

Así que antes de emprender el proyecto se recomienda construir modelos y tener información suficiente, pues el cálculo del costo en el desarrollo de software y la exactitud de este se ha convertido en un factor clave tanto para la entidad que desarrolla el producto de software como para el cliente, el cual espera que el costo del producto coincida con el estimado.

Técnicas de estimación

Las técnicas de estimación consideran varias alternativas para calcular los costos al empezar y al completar la elaboración de plan de proyecto de software, para que su costo sea adecuado, cuando se va a iniciar un proyecto, tanto el gestor de este como el equipo de software primero deberán estimarlo para saber cuánto tiempo y con qué recursos (humano, de software reutilizable y del entorno) cuentan desde el inicio hasta el fin.

Aunque nunca serán exactas por las variables que intervienen en su cálculo, se hace complejo estimar; sin embargo, las técnicas de estimación de costos ayudan a conseguir los objetivos propuestos, este requiere de buena información histórica y confianza tanto en las métricas como en la experiencia.

Para esto se cuentan con algunas técnicas y modelos de estimación como:

Estimación análoga: esta técnica se centra en la estimación de costos basada en la comparación de los proyectos registrados.

Estimación juicio experto: se basa en la experiencia profesional para mejor comprensión de los riesgos, los problemas las limitaciones y supuestos que se afrontan, y prestar estimaciones precisas en las que se combinan opiniones de varios expertos para obtener estimaciones, como por ejemplo el método Delphi.

Estimación por descomposición: consiste en distribuir un proyecto en otros más pequeños o en temas de nivel inferior estimando el esfuerzo para lograr cada una de ellas.

Estimación paramétrica: sirve para establecer el costo de implementación de una aplicación basada en casos verificados.

Estimación de tres puntos: proceso estadístico y analítico llamado Programa de Evaluación y Revisión Técnica (PERT) se usa mediante la identificación de tres estimaciones independientes basada en los escenarios optimistas, probables y pesimistas.

Estimación por puntos de función y líneas de código: como mecanismos para determinar la complejidad y el tamaño del proyecto a desarrollar.

Modelo de Cocomo

Este modelo está orientado al volumen del producto final, el modelo constructivo de costes (Constructive Cost Model) Cocomo con su primera versión en 1981 (Barry W. Boehm) fue un modelo de tres niveles: simple, moderado y empotrado desde su inicio hasta ahora ha sido el más utilizado, ya que no solo usa el cálculo de la ecuación del esfuerzo para estimar el número de personas, de meses y costos adecuados para el progreso, equipamiento y mantenimiento del proyecto, sino que también mide el tamaño de este, basado en líneas de código principalmente expresados en SLOC así: ESFUERZO = A * TAMAÑO B siendo A y B constantes dependientes del modo de desarrollo.

Este nos aporta tres modelos diferentes para la estimación, cada uno ofrece un nivel de detalle y aproximación:

1. **El modelo básico** con una primera estimación poco refinada, el intermedio que incrementa la precisión mediante modificadores opcionales y por último el detallado que es más complejo y establece una jerarquía que influye en el software obteniendo mejores resultados en las estimaciones; estos tres modelos a la vez, se dividen en modos que representan el tipo de proyecto:

- **Modo orgánico:** un pequeño grupo de programadores experimentados que desarrollan software en un entorno familiar

- **Modo semilibre o semiempotrado:** esquema intermedio entre el orgánico y el rígido; donde se incluye una mezcla de personas experimentadas o no.

- **Modo rígido o empotrado:** con firmes restricciones, que pueden estar relacionadas con la funcionalidad siendo o no técnicas, aquí el problema es singular siendo complicado basarse en la experiencia, puesto que puede no haberla.

2. **Modelo intermedio:** cálculo del esfuerzo en función del tamaño toma como entre líneas de código (KLOC) y un multiplicador (m(x)) e integra conductores de coste (medidas) que permiten valorar el entorno de desarrollo del proyecto para tenerlo en cuenta en la estimación.

3. **Modelo avanzado:** modificación del modelo intermedio para considerar el impacto de las guías de coste. Tanto el modelo intermedio como este, introducen un multiplicador que depende de 15 puntos m(x).

A partir de este modelo fue desarrollado Cocomo II, el cual permite la estimación de coste, tiempo y esfuerzo, este modelo ofrece un framework (marco) completo para determinar constantes de productividad a partir de datos como el plazo y el esfuerzo de proyectos anteriores, estando ligados a ciclos de vida modernos, donde se incluye actualizaciones y nuevas extensiones correspondientes a los requerimientos de los ingenieros de software. Cocomo II proporciona modelos de estimación muy detallados, estos modelos se dividen en tres: 1) el modelo de composición de aplicaciones, para proyectos de construcción de interfaces graficas de usuarios que se aplicara en las primeras fases o ciclos en espiral; 2) el modelo de diseño preliminar que sirve para tener estimaciones sobre el coste de un proyecto, antes de que esté determinada por completo su arquitectura e incluye la exploración de arquitecturas alternativas o estratégicas de desarrollo incremental y, 3) el modelo postarquitectura, proporcionando información aún más precisa de los controladores de coste de entradas y se utiliza una vez determinada la arquitectura por completo.

Cocomo II nos muestra cuatro niveles:

1. Nivel de construcción de prototipos.
2. Nivel de diseño inicial.
3. Nivel de reutilización.
4. Nivel de postarquitectura.

En la siguiente figura se referencian algunas de las fórmulas más usuales.

Fórmulas:

E = Esfuerzo = a KLDC * FAE (persona* mes)

T = Tiempo de duración del desarrollo = c Esfuerzo^{q}(meses)

P= Personal = E/T(personas)

Dónde

Figura 4.
Fuente: propia

Modelos algorítmicos de costes en la planificación

El modelo de algoritmo de costes se utiliza principalmente para el desarrollo de software y tiene como objetivo esencial predecir los costes del proyecto fundamentándose en la valoración del tamaño del producto, la cantidad del personal que interviene y otros componentes participantes en el curso del proyecto, debido a que permite hacer análisis cuantitativo y para ello se emplea una fórmula que expresa:

$$\textbf{Esfuerzo} = A * \textbf{Tamaño}^{B} * M$$

Donde

A= es un valor constante

Tamaño = se refiere a la dimensión código del software

Exponente B: generalmente oscila entre 1 y 1,5 M que lo que hace es multiplicar algunos factores como: vinculación de requerimientos del software, la experticia del personal de desarrollo, teniendo en cuenta que este factor varía de una persona a otra dependiendo de su competencia y su pericia, según Boemh, et ál. (2000) este modelo puede prestar otro tipo de utilidades como estimación para investigadores en desarrollo de software, evaluar los riesgos y valoración de la resolución de determinaciones.

B y **M** son relativas ya que dependen de la pericia y el conocimiento de la persona.

Existen tres elementos a tener en cuenta en el coste de un proyecto:

1. El costo de equipamiento (hardware).

2. El costo de la infraestructura tecnológica (computadores y más dispositivos) para el desarrollo del software.

3. El coste de la labor requerida para la producción de software.

La precisión en la estimación de costo del software a desarrollar, en un modelo algorítmico está condicionado a la cantidad de información que se tenga del sistema, a medida que el proyecto progresa la estimación es más precisa y es menos complicada cuando se conoce el tipo de software a desarrollar, cuando se ha cotejado el modelo utilizando datos locales y cuando se tiene definido el lenguaje de programación a utilizar y el hardware.

Es recomendable tener presente que en algunos proyectos se requiere adquirir software/hardware y talento humano especializado para el desarrollo que se debe incluir, lo que nos permitirá obtener una aproximación más real de la inversión a realizar; además, el modelo algorítmico permite valorar los posibles riesgos del proyecto, como se refleja en la gráfica, no obstante, se aconseja utilizar más técnicas y comparar los resultados, hacer combinaciones que permitan obtener una mayor precisión.

Modelo	Ventajas	Inconvenientes	Aplicación Correcta
Algorítmico	• Entradas y parámetros concretos • Objetividad • Eficiencia en cálculos	• No presta atención a circunstancias excepcionales • Rechazan opiniones subjetivas	• Proyectos en escasas alteraciones accidentales, con desarrollo estable y productos sencillos

Tabla 1. Modelo algorítmico
Fuente: Sommerville y Alfonso (2005)

Duración y personal del proyecto

Figura 5.
Shutterstock/381343372

Al momento de emprender un proyecto de desarrollo de software se debe hacer la estimación de costes del personal que se requiere para la ejecución, el analista, diseñador, arquitecto de software, el programador y demás personal técnico que dará soporte posterior del mismo, la asignación de recursos consiste en asociar a cada una de las actividades en el proyecto a las personas participantes según la especialidad de cada uno, además; es necesario asociar equipos y materiales necesarios para que estas se puedan realizar.

Para hacer una estimación apropiada se involucra el esfuerzo durante el desarrollo del proyecto, se determina la labor que se llevará a cabo por parte de cada participante, utilizando métricas de personal, lo que consiste en evaluar la competencia, la especialidad y la experiencia de cada profesional con respecto a la plataforma, manejo del lenguaje determinado y las herramientas necesarias para la ejecución.

Por experiencia se puede asegurar que en el coste total del proyecto lo que más requiere presupuesto es el recurso humano, por lo general oscila en un 50% mientras que el hardware, instalaciones y software básico completan el resto del recurso, sin olvidar que en la estimación se tiene que incluir el tiempo que dedica cada persona interviniente. Una vez determinado el esfuerzo requerido para el desarrollo de software, se hace necesario conocer los límites temporales del proyecto donde se aplican (puntos de fusión - tareas).

Somerville (2005) afirma lo siguiente:

> El modelo COCOMO incluye una fórmula para estimar el tiempo de calendario (TDEV) requerido para completar un proyecto. Esta fórmula es igual para todos los niveles de COCOMO:
>
> $TDEV = 3 \times (PM)^{(0,33+0,2*(B-1,01))}$
>
> PM es el cálculo del esfuerzo y B es el exponente calculado (B es 1 para nivel inicial de construcción de prototipos). Sin embargo, la duración prevista del proyecto y la requerida por el plan de proyecto no son necesariamente la misma. La duración planificada es más corta o más larga que la duración media prevista. Sin embargo, existe un límite obvio para los cambios en la duración y el modelo COCOMO II predice esto como:
>
> $TDEV = 3 \times (PM)^{(0,33+0,2*(B-1,01))} \times SCEDPercentage/100$
>
> SCDEPercentage es el porcentaje de incremento o decremento en la duración nominal. Si la cifra prevista difiere significativamente de la duración planificada, esto indica que existe un alto riesgo de que surjan problemas para entregar el software como se planeó (p. 582).

BIBLIOGRAFÍA

Project Management Institute. (2004). *Guía de los fundamentos de la dirección de proyectos*. Pensilvania: Project Management Institute.

Sommerville, I. Alfonso, G. M. (2005). *Ingeniería del software*. Madrid: Pearson.

Esta obra se terminó de editar en el mes de Septiembre 2018
Tipografá BrownStd Light, 12 puntos
Bogotá D.C,-Colombia.

AREANDINA

Fundación Universitaria del Área Andina

MIEMBRO DE LA RED

ILUMNO