Daena: International Journal of Good Conscience. 3(1) : 701-750. Marzo 2008. ISSN 1870-557X.

Modelo para la exposición de la materia de ingeniería de software I

(Model for teaching the software engineering I class)

Rayas, L. y J. L. Abreu

Resumen. Con este proyecto se diseñó un modelo de enseñanza que cuenta con una herramienta útil para guiar al docente para impartir la materia de Ingeniería de Software I, de una manera estructurada y tomando en consideración las metodologías vigentes para el proceso de la documentación del desarrollo del software, desde sus fundamentos más elementales, como lo son la adecuada recopilación de requerimientos, tanto funcionales como no funcionales, así como la administración de proyecto identificando la ruta crítica y determinar los tiempos de entrega, calendarización del proyecto total y su monitoreo.

Palabras claves. Modelo de enseñanza, manual, ingeniería de software, metodología de software

Abstract. With this Project was designed a teaching model that counts on a useful tool to guide the teacher to facilitate the engineering software I class, in a structured way and taking into consideration the actual methodologies for the process of documentation of the development of software, from its most elemental foundations, as the adequate recompilation of requirements, such as functionalism and the administration of projects identifying the critical route and to determine the delivery times, calendarization of the total project and its monitoring.

Key words. Teaching model, manual, software engineering, software methodology

Introducción

Debido a los avances tecnológicos y la adopción de la globalización, las universidades del mundo, se han visto en la necesidad de dar soporte y apoyo tecnológico dentro de los procesos de enseñanza – aprendizaje, al mismo tiempo, implementar la educación a distancia, para cubrir las necesidades que la propia globalización requiere.

Esto ha provocado que las universidades de México adopten estas nuevas tendencias, sobre todo, las de la educación a distancia, por lo que en la actualidad hay un sin número de carreras profesionales, diplomados y hasta postgrados que utilizan esta nueva metodología.

Por lo que dentro del Centro de Estudios Universitarios, se están adoptando algunas metodologías para llevar a cabo una mejor práctica docente y favorecer el aprendizaje de los alumnos, y al mismo tiempo, facilitar la tarea del catedrático y ayudarlo a que lleve un orden cronológico dentro de la asignatura, por tal motivo, se pretende realizar un modelo del cual surja un manual, y este establezca los temas de estudio a llevarse a cabo en la carrera de Ingeniero en Sistemas Computacionales, dentro de la asignatura de Ingeniería de Software I.

Para cumplir con las actividades, que ayuden dentro del proceso de enseñanza-aprendizaje en la citada materia, en la actualidad, el docente tiene que buscar en las bibliotecas de la institución y en la que se encuentran fuera de esta, en páginas de Internet y hasta en revistas especializadas; material necesario para cumplir con los objetivos especificados dentro del plan de estudios establecido por la institución

educativa, por lo que en muchas de las ocasiones, se ven en la necesidad de adquirir con recursos propios, este material de apoyo para cubrir dichos requerimientos.

En otros de los casos, debido a los escasos recursos económicos, se ven en la necesidad de impartir esta importante materia, con la ayuda de un libro de texto, por lo que limita a los educandos a tomar en consideración un solo autor, y salen de la institución con la visión de 2 metodologías para la documentación y administración del proceso de desarrollo del software, por lo que en este nuevo modelo educativo para la impartición de dicha materia, comprenderá las metodologías vigentes dentro del ámbito laboral, de tal manera que el alumno comprenda y conozca con lo que se va a enfrentar una vez que termine sus estudios universitarios.

Planteamiento del problema

Antecedentes

Dentro del Centro de Estudios Universitarios se han desarrollado una serie de manuales para facilitar la tarea del docente, pero ninguno de estos está orientado a la materia de Ingeniería de Software I, por tal motivo y debido a la evidente carencia de material de lectura dentro del área de la biblioteca y el alto costo de los libros referentes a este tema, los alumnos, así como los docentes, se ven limitados en la adquisición de más metodologías para llevar a cabo un buen proyecto de software.

Este proyecto es pionero dentro de la institución, ya que aunque se han realizado manuales para la creación de productos de software en particular, y estos solo están orientados al desarrollo de los requerimientos funcionales y no a la adecuada adquisición de los conocimientos necesarios para llevar a cabo un buen proyecto de software desde sus bases fundamentales, como los es la identificación de requerimientos y su definición, así como la administración del desarrollo de todo el proyecto; este último tema tan importante, ya que el alumno únicamente aprende el proceso de desarrollo, pero no como administrar un proyecto y mucho menos darle seguimiento para mantener informado al cliente y cumplir con uno de los objetivos principales de la Ingeniería de Software, que es el de hacer tangible y palpables los avances del mismo, con fin de una mejor comunicación entre cliente y desarrollador.

Definición del objeto de estudio

La carencia principal es por la falta de un manual que guíe al catedrático para llevar a cabo un adecuado flujo dentro de la asignatura, y sobre todo, que tome en consideración el proceso del desarrollo del software. Debido a esto, el catedrático se ve en la difícil tarea de guiar a los alumnos siguiendo solamente el mapa curricular de la asignatura, pero en más de las veces, la limitante es contar con un solo libro de apoyo el cual crea una dependencia directa con ese autor, y no permitiendo a los alumnos evaluar y conocer diferentes metodologías existentes para llevar a cabo el proceso del desarrollo del software. Por tal motivo, se ve en la necesidad de crear un modelo educativo vigente y actual, del cual surja un manual que cubra los aspectos más importantes sobre la documentación del proceso, la recopilación de requerimientos, la administración de un proyecto, así como el prototipado, para cumplir con los elementos fundamentales y medulares para el desarrollo de cualquier producto de software.

Delimitación del problema

El modelo será desarrollado para cubrir las necesidades dentro del Centro de Estudios Universitarios, en la facultad de Ingeniería e Informática, dentro del Campus Loma Larga, ubicado en Loma Redonda 1509 Col. Loma Larga, en Monterrey, Nuevo León, México, teléfonos. 8342 1890 al 93; para la carrera de Ingeniero en Sistemas Computacionales, en la materia de Ingeniería de Software I, para los alumnos que cursan el octavo cuatrimestre.

Justificación

Con este modelo educativo será una herramienta, no solo para el docente, sino para cualquier alumno que requiera establecer metodologías para el desarrollo de software, ya que se pretende dotar a las bibliotecas establecidas dentro de los campus del Centro de Estudios Universitarios, para que logre su objetivo. Así mismo, los alumnos del Campus Loma Larga, contaran con todo el material que será impartido en la materia de Ingeniería de Software I, por lo que bastará con saber que tema es el que va a ser visto en clase en la semana, para poder llevar a cabo una lectura previa o en su defecto, cuando se tenga alguna falta se pueda poner al corriente sin retrasar el programa del docente.

Objetivo

Con este proyecto se pretende diseñar un modelo que cuente con una herramienta útil para guiar al docente para impartir la materia de Ingeniería de Software I, de una manera estructura y tomando en consideración las metodologías vigentes para el proceso de la documentación del desarrollo del software, desde sus fundamentos más elementales, como lo son la adecuada recopilación de requerimientos, tanto funcionales como no funcionales, así como la administración de proyecto identificando la ruta crítica y determinar los tiempos de entrega, calendarización del proyecto total y su monitoreo.

Todo esto, empleando no solamente un autor, sino recopilando las metodologías vigentes dentro del mercado laboral para que el alumno egrese con un panorama más amplio y pueda desempeñarse de una mejor manera.

Al contar con este modelo, el cual su instrumento principal es el manual, el alumno tendrá desde el primer día de clase, todo el material que será visto a lo largo del periodo de enseñanza – aprendizaje, de tal manera que tendrán las herramientas necesarias para salir adelante y de este modo el docente pueda realizar una mejor función, desarrollándose como guía dentro de clase y no solamente como transmisor de conocimientos.

Marco referencial institucional

Antecedentes del Centro de Estudios Universitarios

El profesor Antonio Coello Elizondo nació el 19 de noviembre de 1915 en la Villa de Guadalupe. Sus padres: El Profesor Antonio D. Coello Borrego y la Señora Elvia

Elizondo Zambrano. Ingresa a la normal Miguel F. Martínez en el año de 1929 y termina sus estudios en el año de 1934. Al recibirse como maestro solicita la autorización para impartir clases en las islas Marías, su petición es aceptada y al poco tiempo lo nombran director de ese reclusorio. Es ahí donde se inclina al área de la educación popular. Regresa a su tierra natal y en Linares, Nuevo León se dedica a la alfabetización de la población adulta. El 1 de septiembre de 1938 fue designado director del Centro Obrero Monterrey, al que cuatro meses después se le otorga al nombre de ilustre educador Adán Villarreal institución educativa que fue creada para ofrecer a los trabajadores la oportunidad de terminar su escuela primaria y adquirir su certificado. En septiembre de 1940 nace la Escuela secundaria Federal Nocturna para trabajadores No 10, hoy conocida por la Secretaria de educación Pública como la No 1 por ser la primera secundaria nocturna; en 1956 funda la Escuela Preparatoria Federal Nocturna y cuatro años más tarde una escuela Normal Federal Nocturna por cooperación, estas tres últimas forman lo que se conoció en Monterrey como Centro Escolar Federal. En el año de 1969 funda una escuela preparatoria diurna incorporada a la Universidad Autónoma de Nuevo León. El gobernador Lic. Eduardo Elizondo tenía como proyecto de gobierno la creación de Escuelas Universitarias Privadas, invito al ingeniero Treviño dueño del Instituto Modelo de Enseñanza; a maestros Maristas y Lasallistas que tenia él CUM y al profesor Antonio Coello Elizondo del Centro Escolar Federal a presentar sus proyectos de Universidades Privadas Estatales. El ingeniero Treviño convierte al instituto Modelo de Enseñanza en la Universidad Regiomontana (UR). Los maestros Maristas y Lasallistas hicieron la universidad de Monterrey (UDEM). El profesor Antonio Coello Elizondo funda el Centro de Estudios Universitarios (CEU). El profesor Antonio Coello Elizondo alcanza el ideal más alto de su vida al fundar la tan anhelada Universidad Privada, libre, para trabajadores, la cual inicia sus clases con 312 alumnados en la calle Hidalgo, en una casona vieja, en donde ahora se encuentra el departamento de Admisiones y en el que anteriormente se encontraba la Escuela Normal de Educadores de Laura Arce.

Institución Pionera en cuanto a la creación de escuelas y sistemas educativos para la clase trabajadora, también lo es en la aplicación en la teoría de la escuela Full Time de Edward J. Power, ya que desde julio de 1974 se trabaja con el sistema cuatrimestral, en tres periodos académicos por año de calendario, por lo que los estudios y carreras que imparte, se cursan en una tercera parte del tiempo menos que con el sistema semestral tradicional, cumpliendo así con el imperativo de nuestro fundador: El país tiene prisa histórica de preparar sus recursos humanos. El 6 de marzo de 1982 el profesor Antonio Coello Elizondo muere. Después del fallecimiento del eminente educador, la responsabilidad de continuar al frente de la Rectoría recayó en el Ingeniero y Licenciado Antonio Coello Valadez, su hijo mayor, quien de inmediato inicio una nueva etapa tendiente a consolidar una renovada imagen institucional.

En la búsqueda constante de acrecentar su prestigio institucional, elevar su excelencia en el proceso de enseñanza-aprendizaje y promover en sus egresados el servicio a la comunidad con los más altos valores éticos y morales establece objetivos y metas a corto, mediano y largo plazo. En el año 1999 surge un proyecto: Visión 2010 el cual contempla al Centro de Estudios Universitarios como una Institución educativa Superior de mayor prestigio y reconocimiento social en Nuevo León y en el país.

En el año 2003 El Centro de Estudios Universitarios es dirigido por el Lic. Jaime Loya Garza, en esta etapa la institución se enfrenta a grandes retos y su meta primordial es la de consolidarse entre las mejores opciones de educación superior en el norte del país, es aquí donde se cumple el antiguo sueño de expansión, empezando por la instauración del Centro de Estudios Universitarios, Campus Ciudad Victoria, Tamaulipas, ya en funcionamiento.

Este proyecto de expansión se extiende también a los estudios de Durango, Zacatecas y Coahuila, en donde las autoridades estatales reciben con beneplácito la oportunidad que el CEU puede brindar a la comunidad estudiantil. En esta misma etapa se concluyen las actividades necesarias para adquirir la certificación ante la Federación de Instituciones Mexicanas Particulares de Educación Superior (FIMPES), instancia que acredita a la institución entre las mejores del país.

Filosofía del Centro de Estudios Superiores

El centro de Estudios Universitarios considera que cada alumno además de procurarle un buen rendimiento académico, debe dársele desde un inicio como estudiante, una correcta directriz para su futuro desempeño profesional. Así mismo debe incentivarse su capacidad intelectual y su deseo de superación, su iniciativa, su creatividad y el liderazgo potencial. El centro de Estudios Universitarios está abierto a diversas vertientes del trabajo intelectual y busca que su Filosofía Institucional se concentre en la práctica de los siguientes postulados:

Respetar las ideas, creencias y pensamientos de cada integrante de la comunidad universitaria, en un ambiente de interacción constructiva. Siendo un lema: **IN ORDINE LIBERTAS** , la disciplina dentro del Centro de Estudios Universitarios se basa en el principio de acatar la libertad de pensar, de expresarse, de disentir y de proponer, sin coartar ni lastimar los derechos de los demás al hacerlo, buscando la armonía en la diversidad de opiniones que impulsen una vida democrática plena.

Promover los valores humanos esenciales dentro de la Cátedra y en Actividad Extracurriculares. El centro de estudios Universitarios considera de la dignidad del ser humano debe reflejarse en principios éticos, que orienten al profesionista a ser honesto, responsable, participativo, y solidario en su coexistir social. Estimula la formación integral del individuo abriendo causes institucionales a sus aptitudes culturales y deportivas; por lo tanto se responsabiliza de que no decaigan las actitudes positivas hacia el bien, pues con esto se coadyuva en la ordenación de la nación.

Propulsar la superación académica del personal docente y del alumnado, estimulando la investigación. La universidad es la institución social más obligada a contribuir al cambio positivo que mejore la calidad de vida de la población. Cada profesor y estudiante del Centro de Estudios Universitarios en los ámbitos de las ciencias exactas, biológicas, económicas administrativas y humanidades, es sensibilizado para discernir los conocimientos adquiridos, cuestionarlos y proponer innovaciones como motivación para inducir la investigación.

Inculcar una cultura para propicie en el estudiante una comprensión cabal del ser humano y sus recursos, con sentido patriótico. El Centro de Estudios Universitarios

sustente en su filosofía institucional los principios que dan origen a las acciones que le permitan ser una institución a la vanguardia de las exigencias y retos de un mundo vertiginosamente cambiante por los avances tecnológicos y mística de calidad, que le permitan afianzarse de la técnica y la ciencia, sobre una sólida base humanista y conscientes de los valores nacionales, capaces de propiciar al desarrollo de cada comunidad donde estén inmersos, enfrentando los desafíos del presente siglo.

Misión

Ofrecer servicios educativos de calidad de grado de excelencia para que sus egresados, además de sustentar los altos Valores, se desempeñen profesionalmente con alto grado de eficiencia y sentido de responsabilidad social, constituyéndose en ciudadanos dignos de confianza, que por sus cualidades y conocimientos, legítimamente puedan ser considerados futuros guías de la sociedad.

Visión

Consolidarse como una Institución de prestigio y reconocimientos social por su destacado desempeño académico y su calidad educativa.

Valores

Los valores y actitudes que el Centro de Estudios Universitarios pretende promover entre sus estudiantes, personal académico y administrativo son los siguientes:

Veracidad

Procurar siempre defender la verdad adoptado una actitud objetiva en todas las circunstancias.

Academismo

Promover el permanente mejoramiento académico y la constante separación profesional.

Lealtad

Guardar fidelidad a las normas y principios del honor, el respeto, el patriotismo y la justicia.

Orden

Mantener la ubicación de cada cosa y cada hecho en el lugar que justamente le corresponde.

Responsabilidad

Cumplir puntualmente las obligaciones contraídas y aceptar las consecuencias de las decisiones tomadas.

Entereza

Enfrentar los retos cotidianos con valor, integridad, disciplina y firmeza de ánimo.

Seguridad

Actuar con plena convicción y certeza ante un comportamiento de serenidad y confianza.

Constancia

Preservar y ser consistente en los proyectos y actividades que se realicen.

Excelencia

Fomentar la calidad superior y hace digno de aprecio y valor el trabajo efectuado.

Universalidad

Anteponer al interés personal las diversas expresiones del conocimiento y las múltiples corrientes del pensamiento universal.

Divisiones de estudios

División de educación media superior

- Loma larga
- América
- Guadalupe
- Hidalgo

División de ingeniería e informática

- Lic. En informática administrativa
- Ing. En sistemas computacionales
- Ing. Mecánico administrativo
- Ing. Mecánico electricista
- Ing. Industrial administrador
- Ing. Mecánico
- Ing. Electricista

- Ing. Industrial y de sistemas
- Ing. En telecomunicaciones

División de humanidades

- Lic. en ciencias jurídicas
- Lic. En comunicaciones

División de ciencias Económico-Administrativas

- Lic. En administración de empresas
- Contador público y auditor
- Lic. En relaciones industriales
- Lic. En comercio internacional
- Lic. En mercadotecnia

División de ciencias agropecuarias

- Ing. Agrónomo zootecnista
- Médico veterinario zootecnista

División de deportes

- Lic. En tecnología deportiva

División en ciencias psicopedagógicas

- Lic. En pedagogía
- Lic. En psicopedagogía
- Lic. En psicología

Diplomados

- Cursos especiales

Postgrado

- Maestría en administración
- Maestría en educación superior

Maestría en comunicación en el área de medios electrónicos

Maestría en administración educativa

Línea Curricular

La carrera de Ingeniero en Sistemas Computacionales se encuentra ubicada dentro de la línea curricular de las Ciencias Exactas.

Programa de Estudios

Ingeniero en Sistemas Computacionales

PRIMER TETRAMESTRE

Física I

Álgebra

Matemáticas I

Introducción a las ciencias computacionales

Lógica computacional I

Valores

Ingles I

SEGUNDO TETRAMESTRE

Álgebra lineal

Física II

Matemáticas II

Lógica computacional II

Programación I

Ingles II

TERCER TETRAMESTRE

Física III

Matemáticas III

Contabilidad general

Matemáticas discretas

Programación II

Ingles III

CUARTO TETRAMESTRE

Electrónica básica

Matemáticas IV

Finanzas

Estructura de datos I

Programación III

Ingles IV

Automatización y administración de oficinas I

QUINTO TETRAMESTRE

Sistemas digitales

Administración de archivos

Base de datos I

Estructura de datos II

Programación IV

Investigación de operaciones I

Automatización y administración de oficinas II

SEXTO TETRAMESTRE

Arquitectura de computadoras I

Análisis y documentación de sistemas

Base de datos II

Paquete de micros

Investigación de operaciones II

Administración de centros de cómputo

Automatización y administración de oficinas III

SEPTIMO TETRAMESTRE

Diseño de sistemas

Ingeniería económica

Telecomunicaciones I

Sistemas operativos I

Arquitectura de computadoras II

Lenguaje y autómatas

Seminario I

OCTAVO TETRAMESTRE

Ingeniería de software I

Inteligencia artificial

Telecomunicaciones II

Estadística

Administración general

Economía

Sistemas operativos II

Seminario II

NOVENO TETRAMESTRE

Ingeniería de software II

Reingeniería de sistemas

Graficas por computadoras

Simuladores matemáticos

Ética profesional

Legislación informática

Expresión oral y escrita

Seminario III

Objetivo de la Carrera

Formar profesionales en el área de sistemas computacionales capaces de analizar la problemática existente en la industria y dar una solución óptima con el diseño, implementación y mantenimiento de sistemas computacionales.

Marco teórico

Definición de modelo

"Conjunto de variables relacionadas entre sí e inter actuantes, que en bloque dinámico conducen a obtener un resultado predeterminado o a solucionar un problema" (www.businesscol.com 2007).

Definición de manual

"Libro en que se compendia lo más sustancial de una materia. Documento o cartilla que contiene las nociones básicas de un arte o ciencia y su forma correcta de aplicación. Documento que contiene información válida y clasificada sobre una determinada materia de la organización. Es un compendio, una colección de textos seleccionados y fácilmente localizables" (www.businesscol.com 2007).

Definición de proyecto

"Conjunto de actividades interrelacionadas que tienen un objetivo común, alcanzable autónomamente como unidad de acción en un período de tiempo determinado, a los que están asignados personas y medios materiales, informativos y financieros" (www.ripit.granma.inf.cu 2007).

Definición de docencia

"La docencia se inscribe dentro del campo educativo como actividad que promueve conocimientos, que sitúa al docente como factor especial, tanto con referencia a los conocimientos mismos, como con respecto a las condiciones específicas en que éstos son producidos" (Zabalza, 2006).

Computadora

"Máquina capaz de efectuar una secuencia de operaciones mediante un programa, de tal manera, que se realice un procesamiento sobre un conjunto de datos de entrada, obteniéndose otro conjunto de datos de salida. Se clasifican de acuerdo al principio de operación de Analógicas y Digitales" (Benhrouz, 2003).

Computadora analógica

Según Benhrouz (2003), aaprovechando el hecho de que diferentes fenómenos físicos se describen por relaciones matemáticas similares (Exponenciales, Logarítmicas, etc.) pueden entregar la solución muy rápidamente. Pero tienen el inconveniente que al cambiar el problema a resolver, hay que re alambrar la circuitería (cambiar el Hardware).

Computadora digital

"Están basadas en dispositivos bí estables, que sólo pueden tomar uno de dos valores posibles: '1' ó '0'. Tienen como ventaja, el poder ejecutar diferentes programas para diferentes problemas, sin tener que la necesidad de modificar físicamente la máquina" (Benhrouz, 2003).

"Primera generación: 1946-1955. La tecnología de esta generación se basaba en grandes y pesadas válvulas de vacío; las cuales se sobre calentaban, y había que cambiarlas con frecuencias. El ingreso y salida de los datos se realizaba mediante tarjetas o cintas perforadas, por lo que el procesamiento de la información era lento y secuencial" (Benhrouz, 2003).

"Segunda generación: 1956-1964. En esta generación las computadoras utilizaban transistores que eran mucho más pequeños y confiables que las válvulas de vacío. El tamaño de las computadoras se redujo considerablemente. Los datos comenzaron en cilindros y cintas magnéticas. También aparece un nuevo periférico de salida, la impresora y se desarrollan los primeros lenguajes de alto nivel: Fortran, Cobol" (Benhrouz, 2003).

"Tercera generación: 1965-1970. Esta generación se caracteriza por la utilización de chips de circuitos integrados. Un chip permite agrupar miles de transistores en una oblea de silicio apenas más grande que un transistor. De ese modo, la velocidad de procesamiento se incrementó sustancialmente, asimismo, se mejoran los sistemas de almacenamiento existentes y se desarrollaron nuevos lenguajes de programación: Pascal; Basic; logo" (Benhrouz 2003).

"Cuarta generación: 1971-2000. Durante esta generación se optimizaron los sistemas de producción de chips logrando circuitos integrados de alta escala de integración (LSI) y muy alta escala de integración (VLSI). Surgieron las PC. Y las hogareñas, con lo cual su uso se popularizó. Internet, que existía de la generación anterior, se volvió también accesible a los hogares, y todo el mundo comenzó a estar conectado con un precio bajo" (Benhrouz 2003).

Informática

Según Stair (2000), la palabra informática se creó en Francia en 1962 y es el resultado de las palabras INFORmación y autoMATICA. Es una disciplina científica, que se ocupa de obtener información automática, no se limita solamente al uso de la PC, sino que se nutre de las siguientes disciplinas: Electrónica, Lógica, Matemáticas, Teoría de la información y el Comportamiento humano.

Hardware

"Equipo utilizado para el funcionamiento de una computadora" (Stair, 2000). Stair (2000) define al hardware como los componentes materiales de un sistema informático. La función de estos componentes suele dividirse en tres categorías principales: entrada, salida y almacenamiento. Los componentes de esas categorías están conectados a través de un conjunto de cables o circuitos llamado bus con la unidad central de proceso (CPU) del ordenador, el microprocesador que controla la computadora y le proporciona capacidad de cálculo.

El software

Stair (2000) establece que el software o programas de computadoras, son las instrucciones responsables de que el hardware (la máquina) realice su tarea. Como concepto general, el software puede dividirse en varias categorías basadas en el tipo de trabajo realizado. Las dos categorías primarias de software son los sistemas operativos (software del sistema), que controlan los trabajos del ordenador o computadora, y el software de aplicación, que dirige las distintas tareas para las que se utilizan las computadoras. Por lo tanto, el software del sistema procesa tareas tan esenciales, aunque a menudo invisibles, como el mantenimiento de los archivos del disco y la administración de la pantalla, mientras que el software de aplicación lleva a cabo tareas de tratamiento de textos, gestión de bases de datos y similares. Constituyen dos categorías separadas el software de red, que permite comunicarse a grupos de usuarios, y el software de lenguaje utilizado para escribir programas.

Sistemas operativos

"Sistema operativo, software básico que controla una computadora. El sistema operativo tiene tres grandes funciones: coordina y manipula el hardware del ordenador o computadora, como la memoria, las impresoras, las unidades de disco, el teclado o el Mouse; organiza los archivos en diversos dispositivos de almacenamiento, como discos flexibles, discos duros, discos compactos o cintas magnéticas, y gestiona los errores de hardware y la pérdida de datos. Los sistemas operativos controlan diferentes procesos de la computadora. Un proceso importante es la interpretación de los comandos que

permiten al usuario comunicarse con el ordenador. Algunos intérpretes de instrucciones están basados en texto y exigen que las instrucciones sean tecleadas. Otros están basados en gráficos, y permiten al usuario comunicarse señalando y haciendo clic en un icono. Por lo general, los intérpretes basados en gráficos son más sencillos de utilizar" (Norton, 2006).

Según Norton (2006), los sistemas operativos pueden ser de tarea única o multitarea. Los sistemas operativos de tarea única, más primitivos, sólo pueden manejar un proceso en cada momento. Por ejemplo, cuando la computadora está imprimiendo un documento, no puede iniciar otro proceso ni responder a nuevas instrucciones hasta que se termine la impresión. Todos los sistemas operativos modernos son multitarea y pueden ejecutar varios procesos simultáneamente. En la mayoría de los ordenadores sólo hay una UCP; un sistema operativo multitarea crea la ilusión de que varios procesos se ejecutan simultáneamente en la UCP. El mecanismo que se emplea más a menudo para lograr esta ilusión es la multitarea por segmentación de tiempos, en la que cada proceso se ejecuta individualmente durante un periodo de tiempo determinado. Si el proceso no finaliza en el tiempo asignado, se suspende y se ejecuta otro proceso. Este intercambio de procesos se denomina conmutación de contexto. El sistema operativo se encarga de controlar el estado de los procesos suspendidos. También cuenta con un mecanismo llamado planificador que determina el siguiente proceso que debe ejecutarse. El planificador ejecuta los procesos basándose en su prioridad para minimizar el retraso percibido por el usuario. Los procesos parecen efectuarse simultáneamente por la alta velocidad del cambio de contexto.

"Los sistemas operativos pueden emplear memoria virtual para ejecutar procesos que exigen más memoria principal de la realmente disponible. Con esta técnica se emplea espacio en el disco duro para simular la memoria adicional necesaria. Sin embargo, el acceso al disco duro requiere más tiempo que el acceso a la memoria principal, por lo que el funcionamiento del ordenador resulta más lento" (Norton, 2006).

Sistemas operativos actuales

Norton (2006) define que los sistemas operativos empleados normalmente son UNIX, Mac OS, MS-DOS, OS/2 y Windows-NT. El UNIX y sus clones permiten múltiples tareas y múltiples usuarios. Su sistema de archivos proporciona un método sencillo de organizar archivos y permite la protección de archivos. Sin embargo, las instrucciones del UNIX no son intuitivas. Otros sistemas operativos multiusuario y multitarea son OS/2, desarrollado inicialmente por Microsoft Corporation e International Business Machines Corporation (IBM), y Windows-NT, desarrollado por Microsoft. El sistema operativo multitarea de las computadoras Apple se denomina Mac OS. El DOS y su sucesor, el MS-DOS, son sistemas operativos populares entre los usuarios de computadoras personales. Sólo permiten un usuario y una tarea.

Tecnologías futuras

Norton (2006) establece que los sistemas operativos siguen evolucionando. Los sistemas operativos distribuidos están diseñados para su uso en un grupo de ordenadores conectados pero independientes que comparten recursos. En un sistema operativo distribuido, un proceso puede ejecutarse en cualquier ordenador de la red (normalmente,

un ordenador inactivo en ese momento) para aumentar el rendimiento de ese proceso. En los sistemas distribuidos, todas las funciones básicas de un sistema operativo, como mantener los sistemas de archivos, garantizar un comportamiento razonable y recuperar datos en caso de fallos parciales, resultan más complejas.

Ingeniería de software

Galvis (2000) establece que el reciente aumento de aplicaciones en donde se utiliza la computadora ha sido posible debido a un hardware de bajo costo, por lo cual la demanda de software ha crecido de forma exponencial. Esto implica que son necesarias técnicas y tecnología eficientes de Ingeniería de Software para resolver los múltiples problemas que se derivan de las aplicaciones en donde se desarrollan sistemas de software. La Ingeniería de Software tiene como principal objetivo servir como base para la producción de software de calidad, lo cual se logra definiendo el proceso del software, el cual comprende las actividades involucradas en la producción del software.

Según Lawrence (2002) todo desarrollo de software es riesgoso y difícil de controlar, pero si se lleva una metodología de por medio, lo que obtenemos son clientes satisfechos con el resultado y desarrolladores aún más satisfechos. Sin embargo, muchas veces no se toma en cuenta el utilizar una metodología adecuada, sobre todo cuando se trata de proyectos pequeños de dos o tres meses. Lo que se hace con este tipo de proyectos es separar rápidamente el aplicativo en procesos, cada proceso en funciones, y por cada función determinar un tiempo aproximado de desarrollo. Cuando los proyectos que se van a desarrollar son de mayor envergadura, ahí si generalmente toma sentido el basase en una metodología de desarrollo, y empezar a buscar cual sería la más apropiada para el caso. Lo cierto es que por lo general no se encuentra la más adecuada y se termina por hacer o diseñar una metodología propia, algo que por supuesto no está mal, siempre y cuando cumpla con el objetivo.

Sommerville (1997) establece que otras de las veces se realiza el diseño del software de manera rígida, con los requerimientos que el cliente solicitó, de tal manera que cuando el cliente en la etapa final (etapa de prueba), solicita un cambio y se hace muy difícil realizarlo, pues si lo hacemos, altera muchas cosas que no se habían previsto, y es justo éste, uno de los factores que ocasiona un atraso en el proyecto y por tanto la incomodidad del desarrollador por no cumplir con el cambio solicitado y el malestar por parte del cliente por no tomar en cuenta su pedido. Obviamente para evitar estos incidentes se debe de llegar a un acuerdo formal con el cliente, al inicio del proyecto, de tal manera que cada cambio o modificación no perjudique al desarrollo del mismo. Por lo tanto, y tomando en consideración lo anterior, los proyectos en problemas son los que se salen del presupuesto, tienen importantes retrasos, o simplemente no cumplen con las expectativas del cliente.

Metodologías para el desarrollo de software

Para resolver estas complicaciones, se han desarrollado muchas metodologías para llevar a cabo el proceso de la documentación del proyecto de software, las cuales a su vez son auxiliadas por herramientas de software para agilizar el desarrollo de los mismos.

"Metodología Feature -Driven Development (FDD): Define un proceso iterativo que consta de 5 pasos. Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software. Sus impulsores son Jeff De Luca y Peter Coad" (Presuman, 2005).

"Metodología Lean Development (LD): Definida por Bob Charetteís a partir de su experiencia en proyectos con la industria japonesa del automóvil en los años 80 y utilizada en numerosos proyectos de telecomunicaciones en Europa. En LD, los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la productividad del cliente. Su principal característica es introducir un mecanismo para implementar dichos cambios" (Poppendieck, 2003).

"Metodología Dynamic Systems Development Method (DSDM): Define el marco para desarrollar un proceso de producción de software. Nace en 1994 con el objetivo de crear una metodología RAD unificada. Sus principales características son: es un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos. Propone cinco fases: estudio viabilidad, estudio del negocio, modelado funcional, diseño y construcción, y finalmente implementación. Las tres últimas son iterativas, además de existir realimentación a todas las fases" (Pressman, 2005).

"Metodología SCRUM: Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requerimientos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración" (Abrahamsson, Salo, Ronkainen, Warsta, 2000).

"Metodología Crystal Methodologies: Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. Han sido desarrolladas por Alistair Cockburn. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros)" (Beck. 1999).

"Metodología Adaptive Software Development (ASD): Su impulsor es Jim Highsmith. Sus principales características son: iterativo, orientado a los componentes software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: especulación, colaboración y aprendizaje. En la primera de ellas se inicia el proyecto y se planifican las características del software; en la segunda desarrollan las características y finalmente en la tercera se revisa su calidad, y se entrega al cliente. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo" (Beck, 1999).

“METODOLOGÍAS ÁGILES. En febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace el término ÁGIL, aplicado al desarrollo de software. En esta reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas” (Cockbun, 2001)

“Tras esta reunión se crea ágil Alliance3, una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida es fue el Manifiesto ÁGIL, un documento que resume la filosofía ÁGIL. Según el Manifiesto se valora:

a) Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. La gente es el principal factor de éxito de un proyecto software. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.

b) Desarrollar software que funciona más que conseguir una buena documentación. La regla a seguir es no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Estos documentos deben ser cortos y centrarse en lo fundamental.

c) La colaboración con el cliente más que la negociación de un contrato. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.

d) Responder a los cambios más que seguir estrictamente un plan. La habilidad de responder a los cambios que puedan surgir a los largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

Los valores anteriores inspiran los doce principios del manifiesto. Son características que diferencian un proceso ágil de uno tradicional. Los dos primeros principios son generales y resumen gran parte del espíritu ágil. El resto tienen que ver con el proceso a seguir y con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo. Los principios son:

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.

II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.

IV. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.

V. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.

VI. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.

VII. El software que funciona es la medida principal de progreso.

VIII. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.

X. La simplicidad es esencial.

XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.

XII. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento" (Cockbun, 2001).

"Extreme Programing (XP) es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizadas para proyectos de corto plazo, corto equipo y cuyo plazo de entrega era ayer. Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre.

Kent Beck, el padre de XP, describe la filosofía de XP sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente, otras publicaciones de experiencias se han encargado de dicha tarea" (Jeffries, Anderson, Hendrickson, 2001).

Según Beck (1999), las características esenciales de XP, están organizadas en los tres apartados siguientes: historias de usuario, roles, proceso y prácticas.

a) Las Historias de Usuario: Es la técnica utilizada para especificar los requerimientos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requerimientos funcionales o no

funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

b) Roles XP: Los roles de acuerdo con la propuesta original de Beck son:

1) Programador: Escribe las pruebas unitarias y produce el código del sistema.

2) Cliente: Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

3) Encargado de pruebas (Tester): Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

4) Encargado de seguimiento (Tracker): Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.

5) Entrenador (Coach): Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

6) Consultor: Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

7) Gestor (Big boss): Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

c) Proceso XP: El ciclo de desarrollo consiste en los siguientes pasos:

1. El cliente define el valor de negocio a implementar.

2. El programador estima el esfuerzo necesario para su implementación.

3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.

4. El programador construye ese valor de negocio.

5. Vuelve al paso 1.

El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega, Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

d) Prácticas XP: La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas: Beck (2000)

1) La planificación: Hay una comunicación frecuente el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.

2) Entregas pequeñas: Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.

3) Metáfora: El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema.

4) Diseño simple: Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

5) Pruebas: La producción de código está dirigida por las pruebas unitarias. Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.

6) Refactorización: Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.

7) Programación en parejas. Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas.

8) Propiedad colectiva del código. Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

9) Integración continua: Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.

10) 40 horas por semana: Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.

11) Cliente in-situ: El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El

cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.

12) Estándares de programación: XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible." (Beck, 1999).

Microsoft Solution Framework (MSF): Según Jacobson, Booch, Rumbaugh (2000), esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. MSF tiene las siguientes características:

"Adaptable: es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.

Escalable: puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.

Flexible: es utilizada en el ambiente de desarrollo de cualquier cliente.

Tecnología Agnóstica: porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología" (Jacobson, Booch, Rumbaugh, 2000).

"La metodología RUP es un proceso que define claramente quien, cómo, cuándo y qué debe hacerse, su enfoque está basado en modelos utiliza un lenguaje bien definido para tal fin, el UML. Éste aporta herramientas como los casos de uso, que definen los requerimientos. Permite la ejecución iterativa del proyecto y del control de riesgos. Las características principales del proceso son: Guiado por los casos de uso, Centrado en la arquitectura, Guiado por los riesgos e Iterativo" (Schmuller, 2000).

Según Galvis (2000) a través de un proyecto guiado por RUP, los requerimientos funcionales son expresados en la forma de casos de uso, que guían la realización de una arquitectura ejecutable de la aplicación. Además el proceso focaliza el esfuerzo del equipo en construir los elementos críticos estructuralmente y del comportamiento (llamados Elementos Arquitecturales) antes de construir elementos menos importantes. La mitigación de los riesgos más importantes guía la definición y confirmación del alcance en las primeras etapas del ciclo de vida. Finalmente RUP particiona el ciclo de vida en iteraciones que producen versiones increméntales de los ejecutables de la aplicación.

"RUP implementa las siguientes mejores prácticas asociadas al proceso de Ingeniería de Software: Desarrollo iterativo, Manejo de los requerimientos, Uso de una arquitectura basada en componentes, Modelización visual, Verificación continua de la calidad y Manejo de los cambios. La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases el desarrollo del software. Cada Fase tiene definido un conjunto de objetivos y un punto de control especifico. A saber: Fase de

inicio, elaboración, construcción y transición. Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los Objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes" (Galvis, 2000).

Coad, Lefebvre, De Luca (1999) establecen una particularidad de esta metodología, que en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software. El Proceso Unificado es un proceso de software genérico que puede ser utilizado para una gran cantidad de tipos de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de competencia y diferentes tamaños de proyectos. Provee un enfoque disciplinado en la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su meta es asegurar la producción de software de muy alta calidad que satisfaga las necesidades de los usuarios finales, dentro de un calendario y presupuesto predecible.

Desarrollo de la alternativa

Fundamentación

La presente propuesta pretende establecer un modelo educativo el cual está fundamentado en la estructuración básica de las metodologías existentes que están vigentes dentro del mercado laboral, el que arrojará como resultado un manual que fungirá como guía curricular para la impartición de la materia de Ingeniería de Software I, pero no será solo para el docente, sino que aportará los conocimientos teóricos de vital importancia para la administración de un proyecto de desarrollo de software, como lo son la identificación y definición de requerimientos, tomando en consideración el tamaño del proyecto para poder determinar cual metodología utilizar para desarrollar el mismo.

Se encuentra sustentado en la adopción de una combinación de los paradigmas conductista y constructivista, ya que se pretende formar alumnos críticos, analíticos, reflexivos que construya su aprendizaje, más sin embargo el maestro sigue siendo el poseedor del conocimiento, el que disciplina, el que indica qué y cómo se debe de aprender, el que finalmente, conduce al alumno; De acuerdo con Piaget, "el principal objetivo de la educación es crear alumnos que sean capaces de hacer cosas nuevas, no simplemente de repetir lo que han hecho otras generaciones: hombres que sean creativos, inventivos y descubridores. El segundo objetivo de la educación es formar mentes que puedan criticar, que puedan verificar, y no acepten todo lo que se les ofrezca".

Tomando en cuenta estas consideraciones, el modelo educativo propuesto, arrojará como resultado, un compendio tomado de diferentes autores haciendo referencia a las metodologías vigentes dentro del mercado laboral actual, poniendo al docente en un papel de facilitador del conocimiento, y proporcionando al alumno las herramientas para que pueda investigar y crear a partir de estos, sus propios conocimientos, dejando abierto las opciones a seguir para el desarrollo de un producto de software, tomando en

consideración sus propias decisiones en cuanto al conocimiento construido mediante el manual para la impartición de la materia de Ingeniería de Software I.

Contiene un programa académico estructurado de tal manera que él alumno pueda dar seguimiento a la materia, considerando el orden cronológico de conocimientos necesarios para poder establecer las bases firmes al pretender desarrollar un proyecto de desarrollo de software de cualquier tipo, llevándolo a cabo con alguna de las metodologías vigentes para poder recopilar, documentar y administrar el mismo, las cuales aportarán los conocimientos necesarios para avanzar con la materia siguiente, que es la de Ingeniería de Software II, en la cual se complementa el proyecto, con la programación y desarrollo del mismo.

Importancia

La presente propuesta pretende cubrir una fuerte necesidad de contar con los procesos actuales para llevar a cabo el desarrollo de cualquier software, ya que la manera tradicional con la que se imparte esta, no proporciona las herramientas necesarias a los alumnos para poder salir al mercado laboral e integrarse dentro de los departamentos de Tecnología, como analistas de sistemas. Al implementar el manual como guía para llevar desarrollar la materia, los alumnos obtendrían grandes beneficios, porque contaran con el currículo completo desde el inicio del curso, y no solamente recibir de parte del docente, el temario de lo comprende la materia.

Instrumentos de la propuesta

El instrumento principal del modelo educativo propuesto, es el manual que servirá de guía para el desarrollo de la materia, y el contiene conceptos fundamentales sobre la administración del desarrollo de software, así como la documentación del mismo, los cuales pueden ser utilizados por el docente para indicarle a los alumnos que investiguen información al respecto y que proporcionen una definición de los mismo, pero utilizando sus propias ideas, lo cual traerá como consecuencia uno alumnos reflexivos, analíticos y sobre todo con un conocimiento construido por ellos mismos, y dirigidos por el docente, este dejando de ser considerado el que proporciona los conocimientos, y redefiniendo su papel ahora como un facilitador de conocimientos. Al obtener este cambio en el docente, el alumno cambia su papel, de ser solamente el receptor de información, y ser el constructor de sus propias definiciones del conocimiento, a partir de los contenidos temáticos del manual, el que esta particionado en 7 unidades en orden cronológico, las cuales contienen los conceptos elementales de la ingeniería de software, modelos de desarrollo, la administración de proyectos, la ingeniería de requerimientos, la definición, especificación y análisis de requerimientos, así como el proceso unificado de desarrollo y el lenguaje unificado de modelado.

Resultados

Los resultados obtenidos del presente modelo educativo, los cuales emergen de la tesis propuesta, es el desarrollo del siguiente manual, el cual se detalla a continuación:

Unidad I. Ingeniería de software

"La ingeniería de software es una disciplina que concierne a todos los aspectos de la producción de software" (Sommerville, 1997).

Según Sommerville (1997) los ingenieros de software adoptan un enfoque sistemático para llevar a cabo su trabajo y utilizan las herramientas y técnicas necesarias para resolver un problema planteado, de acuerdo con las restricciones de desarrollo y recursos disponibles. La diferencia entre la Ingeniería de Software y la Computación, es que a esta le concierne la teoría y fundamentos de cualquier sistema de cómputo, ya sea de hardware o software; y la Ingeniería de Software se ocupa solo del desarrollo de sistemas o productos de software.

Retos de la ingeniería de software.

- Mantener y tratar con sistemas legados así como interactuar con una mayor diversidad de sistemas con mayores demandas de cómputo, y menores tiempos de entrega.
- Los Sistemas Legados, ya que son sistemas antiguos que deben ser mantenidos y mejorados.
- La Heterogeneidad, porque en los sistemas se incluye una mezcla de software y hardware.
- Los tiempos de Entrega, ya que existe una presión incremental por una entrega a tiempo de los productos de software.
- La Formalidad, porque existe una gran demanda de que exista formalidad en el proceso del desarrollo del software." (Sommerville, 1997).

Productos de software

Según Weitzenfield (2005), se tienen 2 tipos de productos de software, que son:

a) PRODUCTOS GENÉRICOS: Son productos de software desarrollados para cualquier negocio (como el ASPEL o el VITAL).

b) PRODUCTOS HECHOS A LA MEDIDA: Son productos de software diseñados para un proceso específico.

Las características más importantes de los productos de software son: Que sea Mantenible, Confiable, Eficiente y que tenga una utilización adecuada.

Proceso de software

Según Weitzenfield (2005), el proceso del Software es un conjunto estructurado de actividades, requeridas para desarrollar un software, y lo pasos que se deben de llevar a cabo dentro del proceso genérico del software son:

- Especificación
- Diseño
- Manufactura
- Prueba
- Instalación
- Mantenimiento

Las características más importantes de este proceso son: que debe ser Entendible, visible, soportable, aceptable, confiable, robusto, mantenible y funcionar con rapidez.

Unidad II. Modelos de desarrollo de software

Bass (2001) establece que los modelos de desarrollo de software son una representación gráfica del proceso del software, y se cuenta con los siguientes tipos de modelos Genéricos:

- Modelo de Cascada
- De Desarrollo Evolutivo
- Prototipado
- Transformación Formal
- Desarrollo basado en la Reutilización

"El modelado es un conjunto de componentes y relaciones entre esos componentes. Esto se puede ilustrar gráficamente en un modelo arquitectónico del sistema, el cual proporciona al lector un panorama de la organización del sistema" (Bass, 2001).

"Modelo de Cascada.

Su proceso es Separar en distintas fases de especificación y desarrollo y las fases en que se subdivide son:

- Análisis de requerimientos y definición.
- Diseño del sistema y del software.
- Implementación y prueba de unidades
- Integración y prueba del sistema.
- Operación y mantenimiento.

La dificultad en esta modelo reside, en realizar cambios entre las etapas.

Modelo de desarrollo evolutivo

Este enfoque entrelaza las actividades de especificación, desarrollo y evolución. Un primer sistema se desarrolla rápidamente a partir de especificaciones abstractas. Entonces, se refina con la ayuda que satisfaga sus necesidades. Las actividades o etapas de este modelo son: Exploratorio (Especificación entendida) y Desechable (Especificación pobremente extendida o no entendido completamente).

Prototipado

El prototipado de software es la animación y demostración de los requerimientos del sistema. Los usos de los prototipos del sistema son:

- El uso principal, es que le ayuda a los clientes y los desarrolladores para entender los requerimientos del sistema
- El prototipo puede ser usado para entrenamiento antes de que el sistema final sea entregado.
- El prototipo puede ser utilizado para pruebas

Modelo de transformación formal

Este enfoque se basa en la producción de una especificación matemática formal del sistema y en la transformación de esta especificación, utilizando métodos matemáticos para construir un programa. La especificación, el diseño y la validación se llevan a cabo en forma matemática.

Modelo de desarrollo basado en la reutilización

Este enfoque se basa en la existencia de un número significante de componentes reutilizables. El proceso de desarrollo del sistema se enfoca en integrar estos componentes en el sistema más que en desarrollarlos desde cero. Toma en cuenta el software existente, para integrarlos en el desarrollo de los nuevos sistemas mediante el proceso de software. Toma en cuenta el software existente para integrarlo en el desarrollo de los nuevos sistemas, mediante el proceso de software" (Bass, 2001).

Administración de proyectos de software.

En base a lo propuesto por Frame (2005) sobre los proyectos dentro de las organizaciones, podemos definir a la administración del proyecto es organizar, planear y calenda rizar proyectos de software, ya que toma en cuenta las actividades que permiten asegurar que el software se lleva a cabo a tiempo y de acuerdo a la planificación. Así como de acuerdo a los requerimientos del software.

"Actividades de la Administración

a) Escritura de la propuesta.

b) Estimación del costo del proyecto.

c) Planeación del proyecto y planificación (de tiempos).

d) Monitorización del proyecto y revisiones.

e) Selección del personal y evaluación.

f) Escritura de reportes y presentaciones.

Planeación del Proyecto

La Planeación del Proyecto es el conjunto de actividades necesarias para desarrollar el proyecto, ya que es la actividad que más consume tiempo.

Existe una actividad continua desde el concepto inicial del proyecto hasta que este es liberado. Los planes deben de ser revisados regularmente a medida que está disponible nueva información.

Estructura del plan del proyecto

1.- Introducción.

2.- Organización del proyecto.

3.- Análisis de riesgos.

4.- Requerimientos de software y hardware.

5.- Repartición del trabajo.

6.- Planificación del trabajo.

7.- Monitorización y mecanismos de reporte.

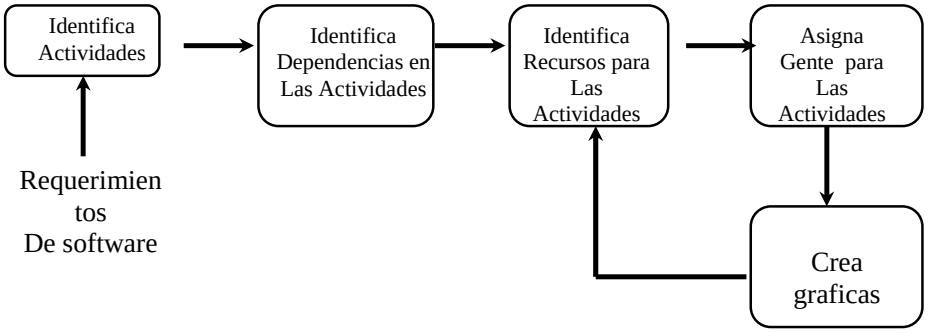
Tipos de planes del proyecto

- Plan de Desarrollo: Describe la metodología a utilizar en el desarrollo del proyecto.
- Plan de Calidad: Describe los procedimientos de calidad, y los estándares a utilizar en el proyecto.
- Plan de Validación: Describe el enfoque los recursos y la planificación utilizada por la validación.
- Plan de Mantenimiento: Predice los requerimientos de mantenimiento del sistema, los costes de mantenimiento y el esfuerzo.
- Plan de Desarrollo Personal: Describe como se adquirirán y desarrollarán los conocimientos y habilidades del personal" (Lawrence, 2002).

Planificación del Proyecto

Según Lawrence (2002), consiste en distribuir el proyecto en tareas y estimar el tiempo y los recursos requeridos para completar cada tarea, Organizándolas de forma concurrente para hacer mejor uso de la fuerza laboral y minimizar las dependencias entre las mismas, para evitar retrasos debidos a que una tarea espere a la terminación de otra.

Gráfica del proceso de planificación del proyecto



Gráficas de barras y redes de actividades.

Frame (2005) establece que se utilizan notaciones gráficas para ilustrar la planificación del proyecto y se muestra la partición del proyecto en tareas. Las tareas no deben ser muy pequeñas y deben de tener una duración de una semana o dos.

Las gráficas de actividades muestran las dependencias entre tareas y la ruta crítica, además de mostrar la planificación contra el tiempo del calendario de actividades.

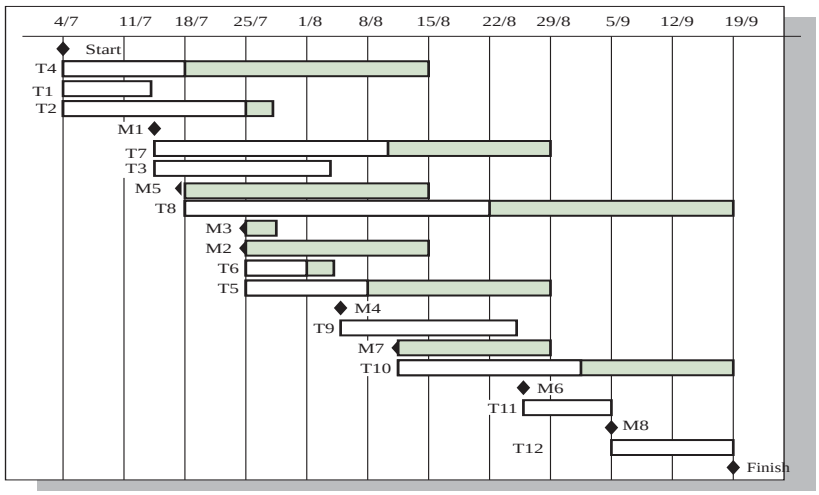
Organización de actividades

Las actividades en un proyecto deben ser organizadas para producir resultados tangibles para que la administración pueda juzgar el progreso, en las cuales se determina la duración de las tareas y sus dependencias.

Tabla de Tareas y Dependencias.

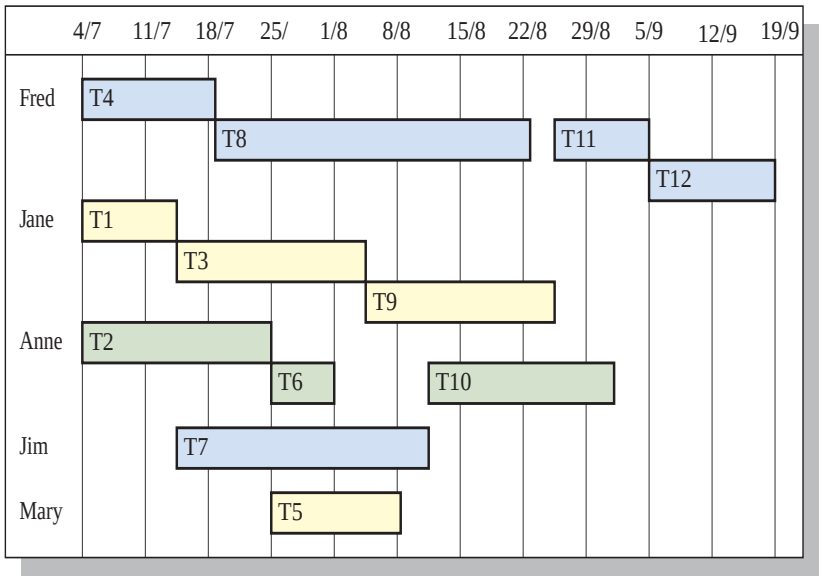
Tareas	duración (Días)	dependencias
T1	8	
T2	15	
T3	15	T1
T4	10	
T5	10	
T2,T4		
T6	5	
T1,T2		
T7	20	T1
T8	25	T4
T9	15	

Gráficas de actividades



Grafica 1

Gráfica de Alojamiento de personal



Grafica 2

Unidad IV. Ingeniería de requerimientos

"Es el proceso de establecer los servicios que el cliente requiere de un sistema y los limites bajo los cuales opera y se desarrolla. Los Requerimientos pueden ser Funcionales o No-Funcionales. Los Requerimientos funcionales describen servicios o funciones que deberá de realizar el sistema para cumplir con su objetivo. Los

Requerimientos No funcionales definen propiedades del sistema y restricciones, por ejemplo: Confiabilidad, tiempos de respuesta y requerimientos de almacenamiento. Las restricciones pueden ser capacidades de dispositivos de E/S, representaciones del sistema, etc. Los requerimientos no-funcionales pueden ser más críticos que los requerimientos funcionales. Si estos no se cumplen, el sistema no es útil" (Luque, Gómez, 1999).

Requerimiento

"Un requerimiento es un rango de instrucciones abstractas de alto nivel de un servicio o de un sistema, limitado a detallar una especificación funcional matemática.

Los Requerimientos pueden ser la base para una declaración de un contrato, por lo tanto, deber estar abierto a interpretación y debe ser definido en detalle.

Proceso de Ingeniería de Requerimientos

- Estudio de Factibilidad
- Definición de Requerimientos
- Análisis de Requerimientos
- Especificación de Requerimientos

Documento de Requerimientos

Es la declaración oficial de lo que es requerido para que el sistema sea desarrollado, incluye la definición y especificación de requerimientos. No es un documento de diseño, sino que es un conjunto de lo que es el sistema y como lo hará.

Estructura del Documento de Requerimientos

- Introducción: Describe la necesidad de crear el sistema y cuáles son sus objetivos.
- Glosario: Define los términos técnicos usados.
- Modelos del Sistema: Define los modelos que muestran los componentes del sistema y las relaciones entre ellos.
- Definición de Requerimientos Funcionales: Define los servicios que serán proporcionados.
- Definición de Requerimientos No-funcionales: Definir las limitantes del sistema y el proceso de desarrollo.
- Evolución del Sistema: Definir las suposiciones fundamentales en las cuales el sistema se basa y se anticipan los cambios.
- Especificación de Requerimientos: Especificación detallada de los requerimientos funcionales del sistema.
- Apéndices: Descripción de la plataforma de Hardware del Sistema.
- Índice." (Luke, Gómez, 1999)

Validación de Requerimientos

Según Luke, Gómez (1999), es la demostración de los requerimientos que definen el sistema, ya que es lo que el cliente realmente quiere. Los costos de errores en los requerimientos son altos, por lo cual, la validación es muy importante**; y e**l Prototipado es una técnica importante de la validación de requerimientos.

Unidad V. Definición y especificación de requerimientos

Definición de Requerimientos

Según Kendall & Kendall (2001), son descripciones orientadas al cliente, de las funciones del sistema y de las restricciones en su operación, y debe especificar el comportamiento externo del sistema de forma que los requerimientos no sean definidos usando un modelo computacional y deben incluir los requerimientos funcionales y no-funcionales. Para su escritura, se debe utilizar el lenguaje natural, aunque su puede encontrar un problema, ya que se basa en la especificación dada por los que lo escriben, por lo que es demasiado flexible y sujeta a distintas interpretaciones

Especificación de Requerimientos

Kendall & Kendall (2001) estable que son descripciones detalladas y precisas de la funcionalidad del sistema y de sus restricciones. Pretende comunicar lo que los desarrolladores del sistema requieren y servir de base como contrato para el desarrollo del sistema.

La especificación añade detalles a la definición de los requerimientos, por lo que debe ser consistente con estos.

Racionalidad en los requerimientos

Kendall & Kendall. (2001) define la importancia de proveer racionalidad en los requerimientos, ya que esto ayuda al desarrollador a entender el dominio de la aplicación y él por que los requerimientos se encuentran en su forma actual.

Rastreo de requerimientos

Kendall & Kendall (2001) especifica que el rastreo de requerimientos, significa que los requerimientos relacionados deben estar ligados de alguna manera. El rastreo es una propiedad de la especificación de los requerimientos que refleja las facilidades en encontrar requerimientos relacionados.

"Técnicas de rastreo

- Asignar un número único a todos los requerimientos
- Hace una referencia cruzada de los requerimientos relacionados utilizando este número único

- Produzca una matriz de referencias cruzadas para cada documento de requerimientos mostrando los requerimientos relacionados" (Kendall & Kendall, 2001).

Verificabilidad de los requerimientos

Kendall & Kendall (2001) define que los requerimientos deben estar escritos de forma que pueden ser objetivamente verificados, ya que el problema principal es el uso de términos "vagos" tales como "los errores deben ser minimizados".

Separación de Requerimientos

Según Kendall & Kendall (2001), los requerimientos funcionales y no-funcionales deben en principio, ser distinguidos en la especificación de requerimientos. Este es un requisito difícil de llevar a cabo ya que los requerimientos pueden estar expresados como requerimientos del sistema en vez de cómo restricciones en funciones individuales. Es difícil a veces decidir si un requerimiento es funcional o no-funcional

Unidad VI. Análisis de Requerimientos

Lawrence (2002) hace referencia que en algunas ocasiones es definida como extracción ó exploración de los requerimientos, esto involucra trabajo técnico de grupo con los clientes para averiguar el dominio de la aplicación, los servicios que el sistema debe proporcionar y las restricciones operacionales propias del sistema. Debe involucrar a los usuarios finales, administradores, ingenieros de mantenimiento.

El análisis de requerimientos puede involucrar tres actividades estructurales las cuales resultan en los diferentes modelos siguientes:

- Particionamiento: Identifica las relaciones estructurales entre las entidades
- Abstracción: Identifica las generalidades entre las entidades
- Proyección: Identifica diferentes modos de ver un problema

Deben emplearse métodos estructurados en el análisis de requerimientos. Éstos deben incluir un modelo del proceso, notaciones de modelado del sistema, reglas y apuntes para el modelado del sistema y reportes estándar

Actividades del proceso

- Comprensión del dominio
- Colección de requerimientos
- Clasificación
- Solución de conflictos
- Priorización
- Validación de requerimientos" (Lawrence, 2002).

Unidad VII. El Proceso Unificado de Desarrollo de Software (RUP)

Bass (2001) establece que el Proceso Unificado es un proceso de software genérico que puede ser utilizado para una gran cantidad de tipos de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de competencia y diferentes tamaños de proyectos.

Provee un enfoque disciplinado en la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su meta es asegurar la producción de software de muy alta calidad que satisfaga las necesidades de los usuarios finales, dentro de un calendario y presupuesto predecible.

Según Bass (2001), El Proceso Unificado tiene dos dimensiones

La primera dimensión representa el aspecto dinámico del proceso conforme se va desarrollando, se expresa en términos de fases, iteraciones e hitos.

La segunda dimensión representa el aspecto estático del proceso: cómo es descrito en términos de componentes del proceso, disciplinas, actividades, flujos de trabajo, artefactos y roles.

El Proceso Unificado se basa en componentes, lo que significa que el sistema en construcción está hecho de componentes de software interconectados por medio de interfaces bien definidas.

Según Bass (2001), el Proceso Unificado usa el Lenguaje de Modelado Unificado (UML) en la preparación de todos los planos del sistema. De hecho, UML es una parte integral del Proceso Unificado, fueron desarrollados a la par.

Los aspectos distintivos del Proceso Unificado están capturados en tres conceptos clave: dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental. Esto es lo que hace único al Proceso Unificado.

El Proceso Unificado es dirigido por casos de uso

Booch, Jacobson, Rumbaugh (1998) establecen que un sistema de software se crea para servir a sus usuarios. Por lo tanto, para construir un sistema exitoso se debe conocer qué es lo que quieren y necesitan los usuarios prospectos.

El término usuario se refiere no solamente a los usuarios humanos, sino a otros sistemas. En este contexto, el término usuario representa algo o alguien que interactúa con el sistema por desarrollar.

Booch, Jacobson, Rumbaugh (1998) establece que un caso de uso es una pieza en la funcionalidad del sistema que le da al usuario un resultado de valor. Los casos de uso capturan los requerimientos funcionales. Todos los casos de uso juntos constituyen el modelo de casos de uso el cual describe la funcionalidad completa del sistema. Este modelo reemplaza la tradicional especificación funcional del sistema. Una especificación funcional tradicional se concentra en responder la pregunta: ¿Qué se

supone que el sistema debe hacer? La estrategia de casos de uso puede ser definida agregando tres palabras al final de la pregunta: ¿por cada usuario? Estas tres palabras tienen una implicación importante, nos fuerzan a pensar en términos del valor a los usuarios y no solamente en términos de las funciones que sería bueno que tuviera. Sin embargo, los casos de uso no son solamente una herramienta para especificar los requerimientos del sistema, también dirigen su diseño, implementación y pruebas, esto es, dirigen el proceso de desarrollo. Aún y cuando los casos de uso dirigen el proceso, no son elegidos de manera aislada. Son desarrollados a la par con la arquitectura del sistema, esto es, los casos de uso dirigen la arquitectura del sistema y la arquitectura del sistema influencia la elección de los casos de uso. Por lo tanto, la arquitectura del sistema y los casos de uso maduran conforme avanza el ciclo de vida.

El Proceso Unificado está centrado en la arquitectura

Jacobson, Booch, Rumbaugh (2000) establecen que el papel del arquitecto de sistemas es similar en naturaleza al papel que el arquitecto desempeña en la construcción de edificios. El edificio se mira desde diferentes puntos de vista: estructura, servicios, plomería, electricidad, etc. Esto le permite al constructor ver una radiografía completa antes de empezar a construir. Similarmente, la arquitectura en un sistema de software es descrita como diferentes vistas del sistema que está siendo construido.

Según Jacobson, Booch, Rumbaugh (2000), el concepto de arquitectura de software involucra los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura surge de las necesidades de la empresa, tal y como las interpretan los usuarios y otros stakeholders, y tal y como están reflejadas en los casos de uso. Sin embargo, también está influenciada por muchos otros factores, tales como la plataforma de software en la que se ejecutará, la disponibilidad de componentes reutilizables, consideraciones de instalación, sistemas legados, requerimientos no funcionales (ej. desempeño, confiabilidad). La arquitectura es la vista del diseño completo con las características más importantes hechas más visibles y dejando los detalles de lado. Ya que lo importante depende en parte del criterio, el cual a su vez viene con la experiencia, el valor de la arquitectura depende del personal asignado a esta tarea. Sin embargo, el proceso ayuda al arquitecto a enfocarse en las metas correctas, tales como claridad, flexibilidad en los cambios futuros y rehusó.

¿Cómo se relacionan los casos de uso con la arquitectura? Cada producto tiene función y forma. Uno sólo de los dos no es suficiente. Estas dos fuerzas deben estar balanceadas para obtener un producto exitoso. En este caso función corresponde a los casos de uso y forma a la arquitectura. Existe la necesidad de intercalar entre casos de uso y arquitectura. Es un problema del “huevo y la gallina”. Por una parte, los casos de uso deben, cuando son realizados, acomodarse en la arquitectura. Por otra parte, la arquitectura debe proveer espacio para la realización de todos los casos de uso, hoy y en el futuro. En la realidad, ambos arquitectura y casos de uso deben evolucionar en paralelo.

El Proceso Unificado es Iterativo e Incremental

Según Schmuller (2000), desarrollar un producto de software comercial es una tarea enorme que puede continuar por varios meses o años. Es práctico dividir el trabajo en

pequeños pedazos o mini-proyectos. Cada mini-proyecto es una iteración que finaliza en un incremento. Las iteraciones se refieren a pasos en el flujo de trabajo, los incrementos se refieren a crecimiento en el producto. Para ser más efectivo, las iteraciones deben estar controladas, esto es, deben ser seleccionadas y llevadas a cabo de una manera planeada.

Los desarrolladores basan su selección de qué van a implementar en una iteración en dos factores. Primero, la iteración trata con un grupo de casos de uso que en conjunto extienden la usabilidad del producto. Segundo, la iteración trata con los riesgos más importantes. Las iteraciones sucesivas construyen los artefactos del desarrollo a partir del estado en el que fueron dejados en la iteración anterior.

En cada iteración, los desarrolladores identifican y especifican los casos de uso relevantes, crean el diseño usando la arquitectura como guía, implementan el diseño en componentes y verifican que los componentes satisfacen los casos de uso. Si una iteración cumple sus metas – y usualmente lo hace – el desarrollo continúa con la siguiente iteración. Cuando la iteración no cumple con sus metas, los desarrolladores deben revisar sus decisiones previas y probar un nuevo enfoque.

Unidad VIII. UML (Lenguaje Unificado de Modelado)

"UML, por sus siglas en inglés, Unified Modeling Language, es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; aún cuando todavía no es un estándar oficial, está apoyado en gran manera por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Es importante remarcar que UML es un "lenguaje" para especificar y no un método o un proceso, se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir -es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para soportar una metodología de desarrollo de software, tal como el RUP, pero no especifica en sí mismo qué metodología o proceso usar" (Schmuller, 2000).

Modelos

Según Schmuller (2000), un modelo representa a un sistema software desde una perspectiva específica. Al igual que la planta y el alzado de una figura en dibujo técnico nos muestran la misma figura vista desde distintos ángulos, cada modelo nos permite fijarnos en un aspecto distinto del sistema.

"Los modelos de UML que se tratan en esta parte son los siguientes:

- Diagrama de Estructura Estática
- Diagrama de Casos de Uso.

- Diagrama de Secuencia.
- Diagrama de Colaboración.
- Diagrama de Estados.” (Schmuller, 2000)

“Elementos Comunes a Todos los Diagramas

Notas: Una nota sirve para añadir cualquier tipo de comentario a un diagrama o a un elemento de un diagrama. Es un modo de indicar información en un formato libre, cuando la notación del diagrama en cuestión no nos permite expresar dicha información de manera adecuada. Una nota se representa como un rectángulo con una esquina doblada con texto en su interior. Puede aparecer en un diagrama tanto sola, como unidad a un elemento por medio de una línea discontinua. Puede contener restricciones, comentarios, el cuerpo de un procedimiento, etc.

Figura 1. Notas

Dependencias: La relación de dependencia entre dos elementos de un diagrama significa que un cambio del elemento destino puede implicar un cambio en el elemento origen (por tanto, si cambia el elemento destino habría que revisar el elemento origen). Una dependencia se representa por medio de una línea de trazo discontinuo entre los dos elementos con una flecha en su extremo. El elemento dependiente es el origen de la flecha y el elemento del que depende es el destino (junto a él está la flecha).

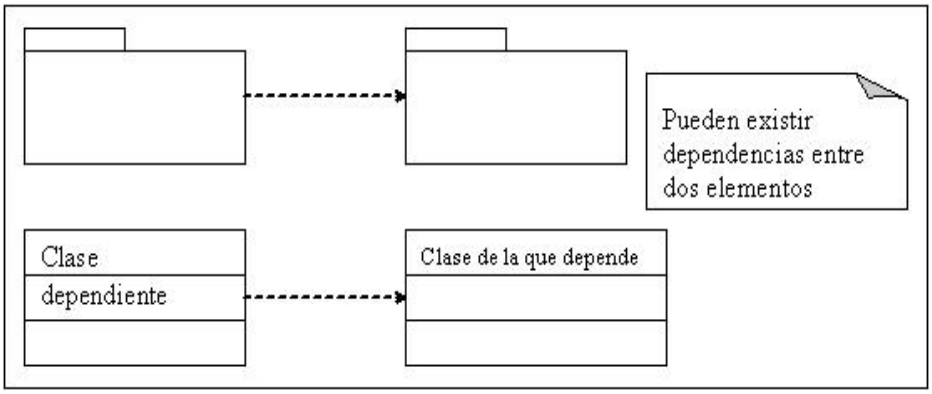


Figura 2. Dependencias.

Diagramas de Estructura Estática: Los Diagramas de Estructura Estática de UML se van a utilizar para representar tanto Modelos Conceptuales, como Diagramas de Clases de Diseño. Ambos usos son distintos conceptualmente, mientras los primeros modelan elementos del dominio los segundos presentan los elementos de la solución software. Ambos tipos de diagramas comparten una parte de la notación para los elementos que los forman (clases y objetos) y las relaciones que existen entre los mismos

(asociaciones). Sin embargo, hay otros elementos de notación que serán exclusivos de uno u otro tipo de diagrama.

Clases: Una clase se representa mediante una caja subdividida en tres partes: En la superior se muestra el nombre de la clase, en la media los atributos y en la inferior las operaciones. Una clase puede representarse de forma esquemática, con los atributos y operaciones suprimidos, siendo entonces tan solo un rectángulo con el nombre de la clase. Como ejemplo, se muestra la siguiente figura, en donde se ve cómo una misma clase puede representarse a distinto nivel de detalle según interese, y según la fase en la que se esté.

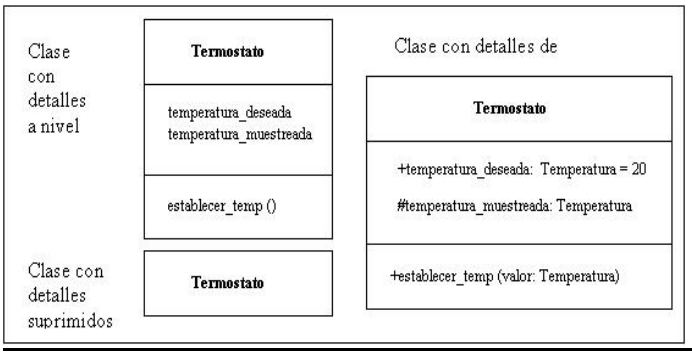


Figura 3. Clases.

Objetos: Un objeto se representa de la misma forma que una clase. En el compartimiento superior aparecen el nombre del objeto junto con el nombre de la clase subrayados, según la siguiente sintaxis: nombre_del_objeto: nombre_de_la_clase Puede representarse un objeto sin un nombre específico, entonces sólo aparece el nombre de la clase.

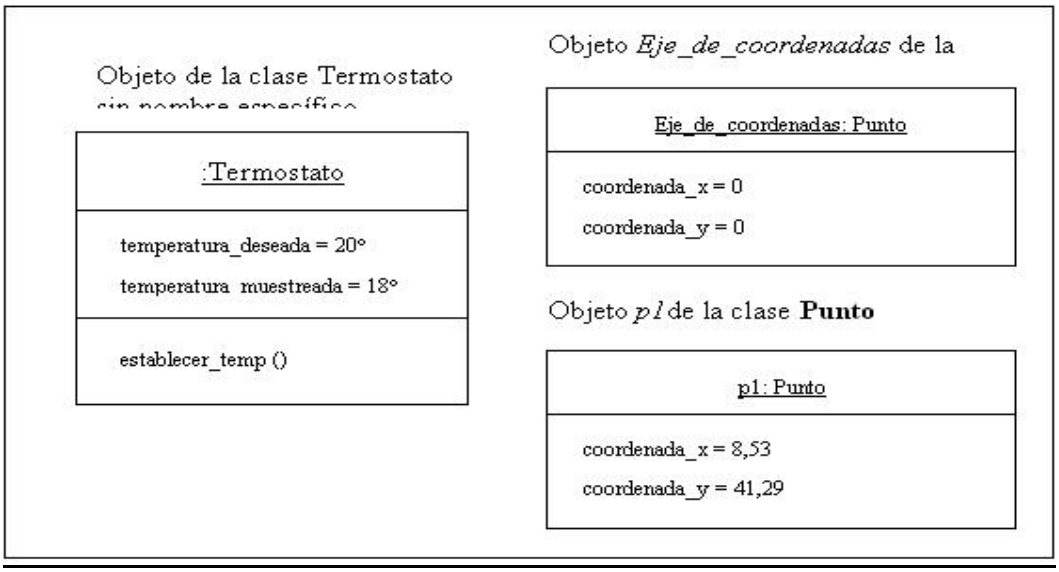


Figura 4. Objeto.

Asociaciones: **Las** asociaciones entre dos clases se representan mediante una línea que las une. La línea puede tener una serie de elementos gráficos que expresan características particulares de la asociación. A continuación se verán los más importantes de entre dichos elementos gráficos.
Nombre de la Asociación y Dirección: El nombre de la asociación es opcional y se muestra como un texto que está próximo a la línea. Se puede añadir un pequeño triángulo negro sólido que indique la dirección en la cual leer el nombre de la asociación. En el ejemplo de la Figura siguiente, se puede leer la asociación como "Director manda sobre Empleado".

Figura 5. Asociaciones.

Los nombres de las asociaciones normalmente se incluyen en los modelos para aumentar la legibilidad. Sin embargo, en ocasiones pueden hacer demasiado abundante la información que se presenta, con el consiguiente riesgo de saturación. En ese caso se puede suprimir el nombre de las asociaciones consideradas como suficientemente conocidas. En las asociaciones de tipo agregación y de herencia no se suele poner el nombre.

Multiplicidad: La multiplicidad es una restricción que se pone a una asociación, que limita el número de instancias de una clase que pueden tener esa asociación con una instancia de la otra clase. Puede expresarse de las siguientes formas:

Con un número fijo: 1.

Con un intervalo de valores: 2..5.

Con un rango en el cual uno de los extremos es un asterisco. Significa que es un intervalo abierto. Por ejemplo, 2..* significa 2 o más.

Con una combinación de elementos como los anteriores separados por comas: 1, 3..5, 7, 15..*.

Con un asterisco: * . En este caso indica que puede tomar cualquier valor (cero o más).

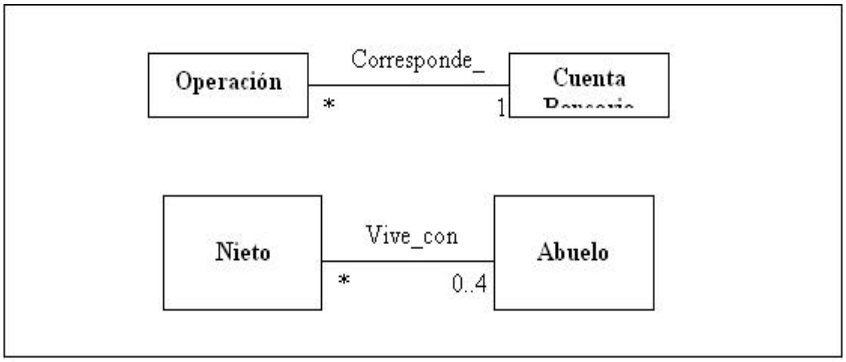


Figura 6. Multiplicidad.

Roles: Para indicar el papel que juega una clase en una asociación se puede especificar un nombre de rol. Se representa en el extremo de la asociación junto a la clase que desempeña dicho rol.

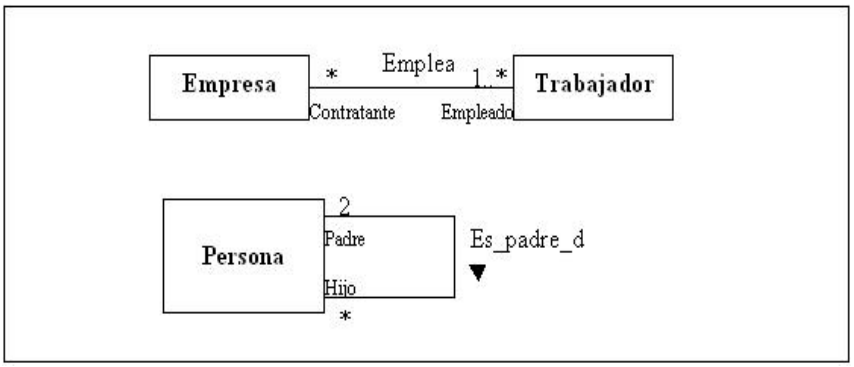


Figura 7. Roles.

Agregación: **El** símbolo de agregación es un diamante colocado en el extremo en el que está la clase que representa el "todo".

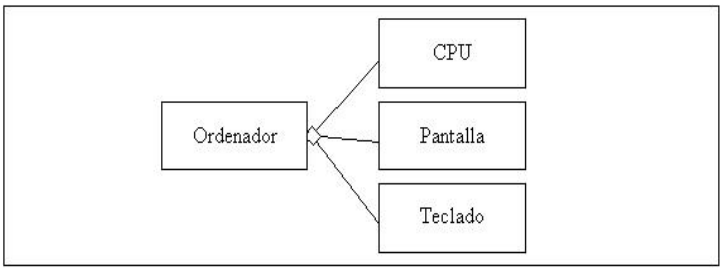


Figura 8. Agregación.

Clases Asociación: Cuando una asociación tiene propiedades propias se representa como una clase unida a la línea de la asociación por medio de una línea a trazos. Tanto la línea como el rectángulo de clase representan el mismo elemento conceptual: la

asociación. Por tanto ambos tienen el mismo nombre, el de la asociación. Cuando la clase asociación sólo tiene atributos el nombre suele ponerse sobre la línea. Por el contrario, cuando la clase asociación tiene alguna operación o asociación propia, entonces se pone el nombre en la clase asociación y se puede quitar de la línea.

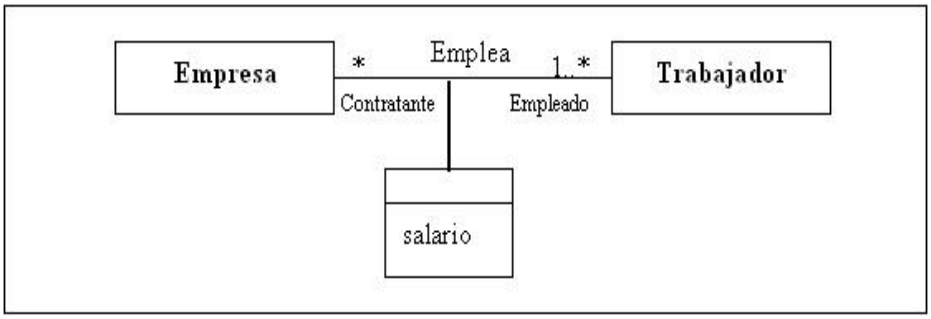


Figura 9. Clases Asociación.

Asociaciones N-Arias: En el caso de una asociación en la que participan más de dos clases, las clases se unen con una línea a un diamante central. Si se muestra multiplicidad en un rol, representa el número potencial de duplas de instancias en la asociación cuando el resto de los N-1 valores están fijos. En la siguiente figura se ha impuesto la restricción de que un jugador no puede jugar en dos equipos distintos a lo largo de una temporada, porque la multiplicidad de "Equipo" es 1 en la asociación ternaria.

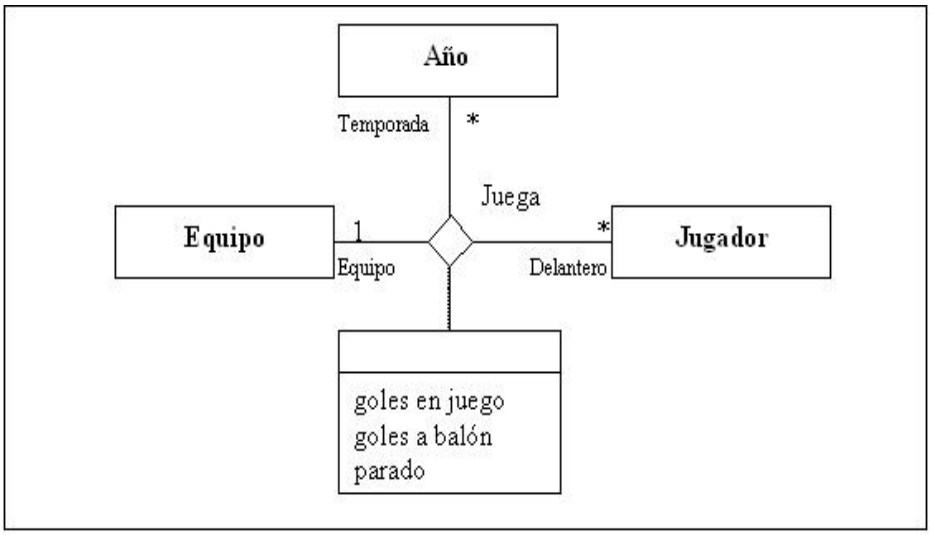


Figura 11. Asociaciones N-Arias.

Navegabilidad: En un extremo de una asociación se puede indicar la navegabilidad mediante una flecha. Significa que es posible "navegar" desde el objeto de la clase origen hasta el objeto de la clase destino. Se trata de un concepto de diseño, que indica que un objeto de la clase origen conoce al (los) objeto(s) de la clase destino, y por tanto puede llamar a alguna de sus operaciones.

Herencia: La relación de herencia se representa mediante un triángulo en el extremo de la relación que corresponde a la clase más general o clase “padre”.

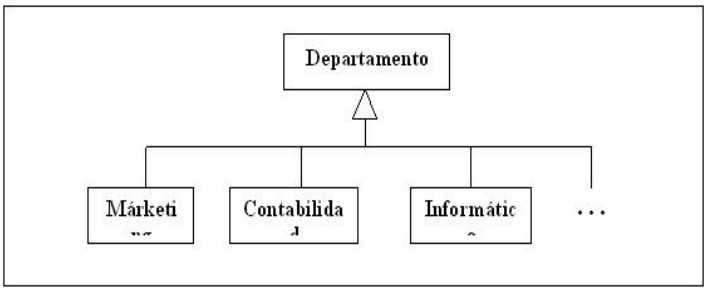


Figura 12. Herencia.

Si se tiene una relación de herencia con varias clases subordinadas, pero en un diagrama concreto no se quieren poner todas, esto se representa mediante puntos suspensivos. En el ejemplo de la figura anterior, sólo aparecen en el diagrama 3 tipos de departamentos, pero con los puntos suspensivos se indica que en el modelo completo (el formado por todos los diagramas) la clase “Departamento” tiene subclases adicionales, como podrían ser “Recursos Humanos” y “Producción”.

Elementos Derivados: Un elemento derivado es aquel cuyo valor se puede calcular a partir de otros elementos presentes en el modelo, pero que se incluye en el modelo por motivos de claridad o como decisión de diseño. Se representa con una barra “/” precediendo al nombre del elemento derivado” (Booch, Jacobson, Rumbaugh. 1998).

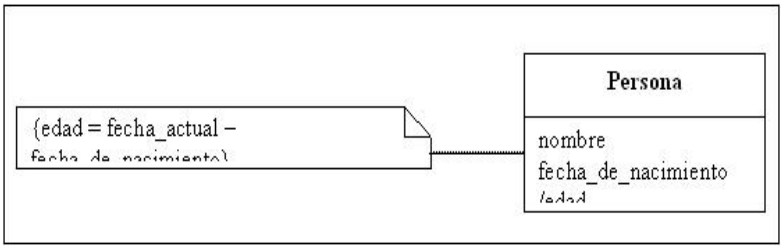


Figura 13. Elementos Derivados.

Diagrama de Casos de Uso.

Según Coad, Lefebvre, De Luca (1999), un Diagrama de Casos de Uso muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa. En el diagrama de casos de uso se representa también el sistema como una caja rectangular con el nombre en su interior. Los casos de uso están en el interior de la caja del sistema, y los actores fuera, y cada actor está unido a los casos de uso en los que participa mediante una línea. En la siguiente figura se muestra un ejemplo de Diagrama de Casos de Uso para un

cajero automático.

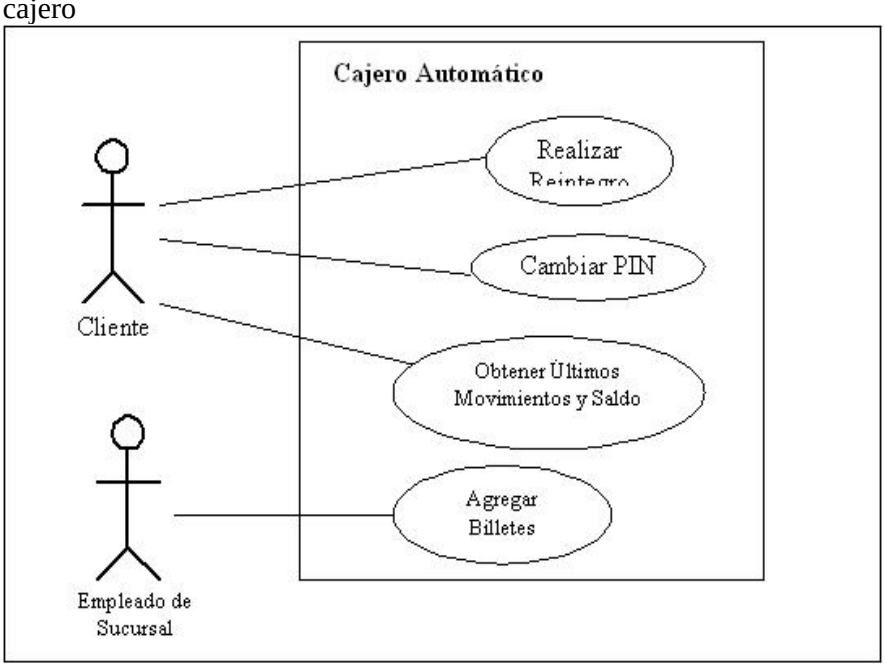


Figura 14. Diagrama de Casos de Uso.

"Elementos: Los elementos que pueden aparecer en un Diagrama de Casos de Uso son: actores, casos de uso y relaciones entre casos de uso.

Actores: Un actor es algo con comportamiento, como una persona (identificada por un rol), un sistema informatizado u organización, y que realiza algún tipo de interacción con el sistema. Se representa mediante una figura humana dibujada con palotes. Esta representación sirve tanto para actores que son personas como para otro tipo de actores.

Casos de Uso: Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el Diagrama de Casos de Uso mediante una elipse con el nombre del caso de uso en su interior. El nombre del caso de uso debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.

Relaciones entre Casos de Uso: Un caso de uso, en principio, debería describir una tarea que tiene un sentido completo para el usuario. Sin embargo, hay ocasiones en las que es útil describir una interacción con un alcance menor como caso de uso. La razón para utilizar estos casos de uso no completos en algunos casos, es mejorar la comunicación en el equipo de desarrollo, el manejo de la documentación de casos de uso. Para el caso de que queramos utilizar estos casos de uso más pequeños, las relaciones entre estos y los casos de uso ordinarios pueden ser de los siguientes tres tipos: • Incluye (<>): Un caso de uso base incorpora explícitamente a otro caso de uso en un lugar especificado en dicho caso base. Se suele utilizar para encapsular un comportamiento parcial común a varios casos de uso. En la Figura 15 se muestra cómo el caso de uso Realizar Reintegro puede incluir el comportamiento del caso de uso Autorización.

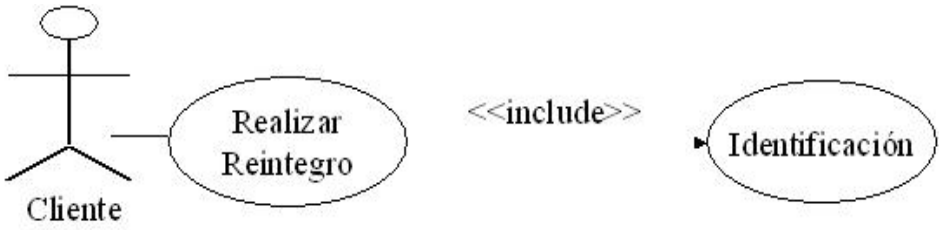


Figura 15. Relación entre Casos de Uso.

En la figura anterior- Ejemplo de Relación <> • Extiende (<>): Cuando un caso de uso base tiene ciertos puntos (puntos de extensión) en los cuales, dependiendo de ciertos criterios, se va a realizar una interacción adicional. El caso de uso que extiende describe un comportamiento opcional del sistema (a diferencia de la relación incluye que se da siempre que se realiza la interacción descrita) En la Figura siguiente se muestra como el caso de uso Comprar Producto permite explícitamente extensiones en el siguiente punto de extensión: info regalo. La interacción correspondiente a establecer los detalles sobre un producto que se envía como regalo están descritos en el caso de uso Detalles Regalo.

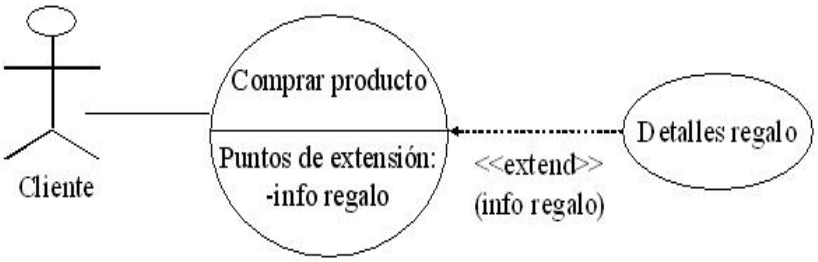


Figura 16. Caso de Uso Comprar Producto.

Ambos tipos de relación se representan como una dependencia etiquetada con el estereotipo correspondiente (<> o <>), de tal forma que la flecha indique el sentido en el que debe leerse la etiqueta. Junto a la etiqueta <> se puede detallar el/los puntos de extensión del caso de uso base en los que se aplica la extensión. • Generalización ( ): Cuando un caso de uso definido de forma abstracta se particulariza por medio de otro caso de uso más específico. Se representa por una línea continua entre los dos casos de uso, con el triángulo que simboliza generalización en UML (usado también para denotar la herencia entre clases) pegado al extremo del caso de uso más general. Al igual que en la herencia entre clases, el caso de uso hijo hereda las asociaciones y características del caso de uso padre. El caso de uso padre se trata de un caso de uso abstracto, que no está definido completamente. Este tipo de relación se utiliza mucho menos que las dos anteriores" (Coad, Lefebvre, De Luca. 1999).

Diagramas de Interacción

Costal, Rivera, Teniente, (2003) establecen que en los diagramas de interacción se muestra un patrón de interacción entre objetos. Hay dos tipos de diagrama de interacción, ambos basados en la misma información, pero cada uno enfatizando un aspecto particular: Diagramas de Secuencia y Diagramas de Colaboración.

Diagrama de Secuencia

Según Costal, Rivera, Teniente, (2003), un diagrama de Secuencia muestra una interacción ordenada según la secuencia temporal de eventos. En particular, muestra los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo. El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos. El tiempo fluye de arriba abajo. Se pueden colocar etiquetas (como restricciones de tiempo, descripciones de acciones, etc.) bien en el margen izquierdo o bien junto a las transiciones o activaciones a las que se refieren.

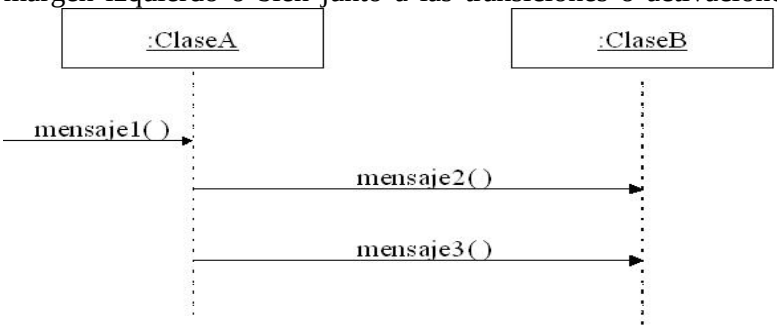


Figura 17. Diagrama de Secuencia.

Diagrama de Colaboración.

Schmuller (2000) establece que un Diagrama de Colaboración muestra una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos (en cuanto a la interacción se refiere). A diferencia de los Diagramas de Secuencia, los Diagramas de Colaboración muestran las relaciones entre los roles de los objetos. La secuencia de los mensajes y los flujos de ejecución concurrentes deben determinarse explícitamente mediante números de secuencia.

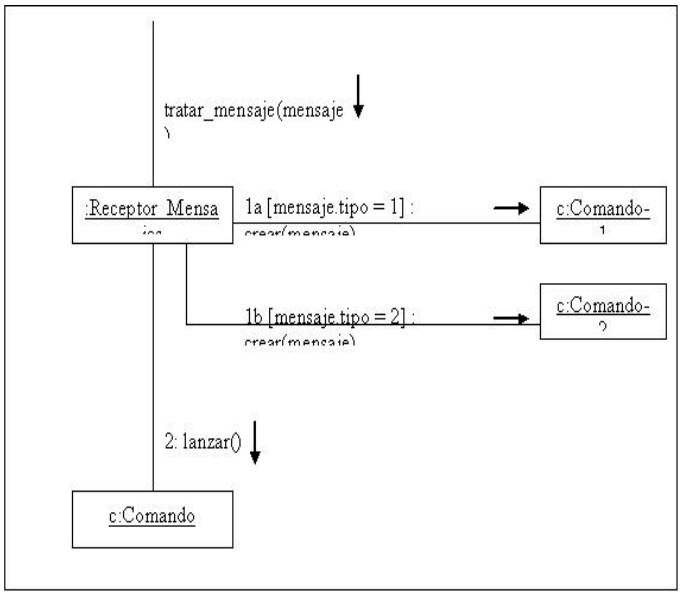


Figura 18. Diagrama de Colaboración.

En cuanto a la representación, Schmuller (2000) define que un Diagrama de Colaboración muestra a una serie de objetos con los enlaces entre los mismos, y con los mensajes que se intercambian dichos objetos. Los mensajes son flechas que van junto al enlace por el que "circulan", y con el nombre del mensaje y los parámetros (si los tiene) entre paréntesis. Cada mensaje lleva un número de secuencia que denota cuál es el mensaje que le precede, excepto el mensaje que inicia el diagrama, que no lleva número de secuencia. Se pueden indicar alternativas con condiciones entre corchetes (por ejemplo 3 [condición_de_test] : nombre_de_método()), tal y como aparece en el ejemplo de la figura anterior. También se puede mostrar el anidamiento de mensajes con números de secuencia como 2.1, que significa que el mensaje con número de secuencia 2 no acaba de ejecutarse hasta que no se han ejecutado todos los 2. x .

Diagramas de Estado

Según Schmuller (2000), un Diagrama de Estados muestra la secuencia de estados por los que pasa bien un caso de uso, bien un objeto a lo largo de su vida, o bien todo el sistema. En él se indican qué eventos hacen que se pase de un estado a otro y cuáles son las respuestas y acciones que genera.

En cuanto a la representación, un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos dirigidos son transiciones etiquetadas con los nombres de los eventos.

Un estado se representa como una caja redondeada con el nombre del estado en su interior. Una transición se representa como una flecha desde el estado origen al estado destino.

La caja de un estado puede tener 1 o 2 compartimentos. En el primer compartimiento aparece el nombre del estado. El segundo compartimento es opcional, y en él pueden aparecer acciones de entrada, de salida y acciones internas. Una acción de entrada aparece en la forma entrada/acción _ asociada donde una acción asociada es el nombre de la acción que se realiza al entrar en ese estado. Cada vez que se entra al estado por medio de una transición la acción de entrada se ejecuta.

Una acción de salida aparece en la forma salida/acción _ asociada. Cada vez que se sale del estado por una transición de salida la acción de salida se ejecuta. Una acción interna es una acción que se ejecuta cuando se recibe un determinado evento en ese estado, pero que no causa una transición a otro estado. Se indica en la forma nombre_de_evento/acción _ asociada.

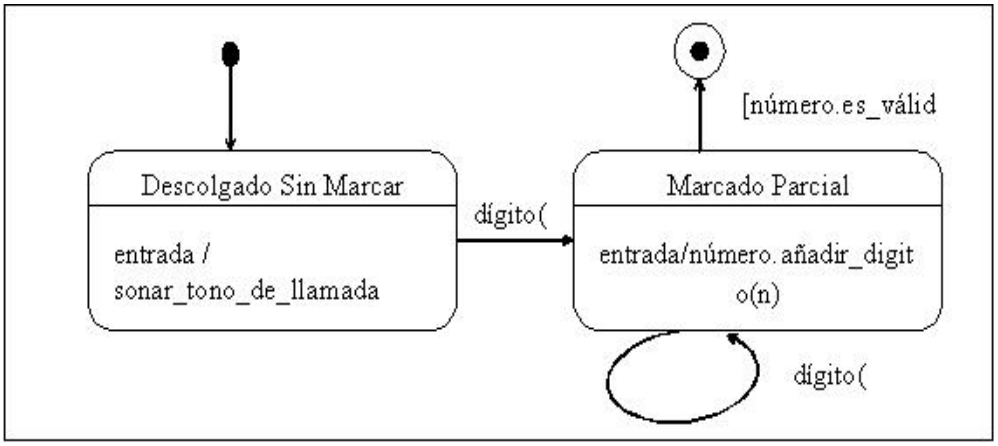


Figura 19. Diagrama de Estado.

Según Schmuller (2000), un diagrama de estados puede representar ciclos continuos o bien una vida finita, en la que hay un estado inicial de creación y un estado final de destrucción (finalización del caso de uso o destrucción del objeto). El estado inicial se muestra como un círculo sólido y el estado final como un círculo sólido rodeado de otro círculo. En realidad, los estados inicial y final son pseudos estados, pues un objeto no puede "estar" en esos estados, pero nos sirven para saber cuáles son las transiciones iníciales y finales.

Conclusión

Aunque no existe alguna metodología general para llevar a cabo el desarrollo de un producto de software, en la actualidad es vital importancia que los egresados de la carrera de Ingeniería en Sistemas Computacionales, conozcan las existentes y sobre todo las que se encuentran vigentes dentro de un entorno de las Tecnologías de las Información, ya que cuando sean introducidos al mercado laboral, las exigencias principales son las de desarrollar aplicaciones, ya sea dentro de una empresa, o en una consultoría de software; por estos motivos, estas organizaciones requieren que todo egresado de la carrera Ingeniero En Sistemas Computacionales, cuenten con estos fundamentos, para reducir la inversiones que tienen que hacer en el talento humano, y empleando esta en desarrollar otro tipo de habilidades que solo son impartidas por empresas especializadas, debido a que involucran una certificación.

La presente propuesta didáctica, la cual consta de un modelo educativo, en cual, emerge un manual para la impartición de la cátedra de la materia de Ingeniería de Software I, se desarrollo pensando en proporcionar facilidades a los docentes, ya que al contar con una guía estructurada dentro del instrumento generado, le permite seguir su trabajo de una manera más accesible y le da el tiempo suficiente para llevar a cabo otro tipo de tareas, porque se vuelve más orientado a guiar a los alumnos, y no a la preparación y documentación de una clase, desde la búsqueda de los temas a impartir, hasta conseguir instrumentos de medición y evaluación.

Adicional a esto, y como principal beneficiario del instrumento desarrollado, también se puso en consideración que los alumnos al contar con este manual desde los inicios de

clase, podrán identificar los temas que serán llevados a cabo en la misma, y podrán realizar lecturas previas; y al mismo tiempo, cuando fuese necesario ausentarse de alguna clase, contarán con el material para poder aplicar un examen, ya que podrán dar seguimiento al tema visto el día en que se ausentaron, y cuestionar sobre las dudas surgidas, en la siguiente clase con el docente, el cual deja su antigua función de transmisor de conocimiento, para llegar a ser un facilitador del mismo, el cual los guiará en la construcción de sus propios conceptos, con una lógica válida y sobre todo, vigente dentro de un mercado laboral, que día a día crece en cuanto a la competencia dentro de este mismo.

Recomendaciones

Para llevar a cabo un desarrollo optimo dentro del proceso de enseñanza-aprendizaje, es indispensable, complementar la presente propuesta didáctica, con la investigación de los alumnos, considerando algunos de los siguientes conceptos como temas de investigación: Ruta crítica, administración de proyectos, prototipado, gráficas de actividades, herramientas CASE, modelado de software y algunos otros que ya vienen definidos dentro del manual, por lo que el docente tendrá que validar que sean cuando menos dos definiciones de diferentes autores, para que construyan su propia definición y que se vean alcanzados los objetivos del presente modelo.

Este proceso de construcción de conocimientos, ayudaran al alumno a una mayor comprensión de los temas, y adquirirá una lógica propia para, administrar y desarrollar un producto de software, cubriendo los objetivos generales de la Materia de Ingeniería de Software I. dentro de la carrera de Ingeniero en Sistemas Computacionales. Los beneficios que obtendrá el alumno, es contar con un mayor acervo cultural definido y comprendido por el mismo, orientado por el docente para validar que estos sean acordes a la realidad dentro del marco institucional de las organizaciones, debido a las demandas de contar con personal capacitado desde que egresan de la carrera, evitando con esto una inversión superior por parte de las empresas para la capacitación de los nuevos analistas de sistemas.

Es recomendable, solicitar a los alumnos algún control de lectura para que sea entregado antes de comenzar a exponer cada uno de los temas, para asegurar la lectura previa. Adicional a esto, es de vital importancia, que sea llevado a cabo un proyecto final, en donde los alumnos aplicarán todos sus conocimientos adquiridos a lo largo del curso. El docente deberá proporcionar la organización y guía dentro del proceso. Este puede ser elaborado por equipos de trabajo, formados de 3 a 4 alumnos cada uno. En el cual el docente debe de tratar de incentivar a que ellos mismo formen estos grupos, para qué empiecen a desarrollar el concepto de trabajo en equipo, y en caso de requerirlo, el docente deberá tomar la batuta cuando estos no cumplan el objetivo.

Una vez definidos los equipos de trabajo, asignarles un módulo o proceso estándar dentro de las organizaciones (contabilidad, administración de inventarios, cuentas por cobrar, compras, etc.) para que sea desarrollado un documento de requerimientos y en base a este, cubriendo todo el proceso, desde la identificación de requerimientos, definición de los mismos, hasta llegar a estructurar dentro un modelo, la arquitectura del producto, para ser utilizado en el desarrollo de un prototipo con las características funcionales del producto final, según el módulo asignado.

Referencias

Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. 2002 Agile software development methods Review and analysis. VTT Publications.

Bass, L. 2001. Component-Based Software Engineering. Putting the Pieces Together, capítulo Software Architecture Design Principles.

Bastide, R. Palanque, P. 1999. Formal Specification and Prototyping of CORBA Systems. En ECOOP'99, número 1628 de LNCS

Batini, C. 1994. Diseño Conceptual de Bases de Datos. Ediciones Díaz de Santos.

Beck, K. 1999 Embracing Change with eXtreme Programming Computer , Pearson Education .

Beck, K. 2000. Extreme Programming Explained. Embrace Change. Pearson Education. Traducido al Español como: Una explicación de la programación extrema. Aceptar el cambio, Addison Wesley.

Benhrouz, A. 2003 Introducción a la Ciencia de la Computación.

Bernus, P. Nemes, L. and Williams, T. 1996. Arquitectures for Enterprise Integration. Chapman & Hall.

Booch, G. Jacobson, I. and Rumbaugh, J.1998. The Unified Modeling Language User Guide. Addison-Wesley.

Coad P., Lefebvre E., De Luca J. 1999. Java Modeling In Color With UML: Enterprise Components and Process. Prentice Hall 1999.

Cockbun, A. 2001. Agile Software Development. Addison-Wesley.

Costal, D. Rivera, M. Teniente, E. 2003. Especificación de Sistemas de Software en UML. Edicions UPC.

Fernández, A. 2006. Desarrollo de sistemas de información: Una metodología basada en el modelado. Edicions UPC

Fisher, A. 1993. Tecnología CASE: Herramientas de Desarrollo de Software. Anaya Multimedia.

Frame, J. 2005. La Dirección de Proyectos en las organizaciones. Ediciones Granica S. A.

Frame, J. 2005. La Nueva Dirección de Proyectos. Ediciones Granica S. A.

Gamma, Erich. 2003. Patrones de Diseño: Elementos de Software orientado a objetos Reutilizable. Pearson Educación.

Galvis, A., 2000 Ingeniería de software educativo. 2da. reimpresión. Universidad de Los Andes.

Harmon, P. King, D. 1998. Sistemas Expertos: Aplicaciones de la Inteligencia Artificial. Ediciones Díaz de Santos.

Highsmith J., Orr K. 2000. Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Dorset House.

http://www.businesscol.com/productos/glosarios/administrativo/glossary.php?word=MANUAL

http://www.businesscol.com/productos/glosarios/administrativo/glossary.php?word=MODELO

Iribarne L., Troya, J. M., y Vallecillo, A. 2002. Selecting Software Components with Multiple Interfaces. En 28th Euromicro Conference, páginas 27–33, Dortmund, Germany. IEEE Computer Society Press.

Jacobson, Booch, Rumbaugh. 2000 "El proceso Unificado de Desarrollo Software". Adison Wesley. Segunda Edición. Madrid.

Jeffries, R., Anderson, A., Hendrickson, C. 2001.Extreme Programming Installed.. Addison-Wesley.

Kendall & Kendall. 2001. Análisis y Diseño de Sistemas; 3ª Edición; Pearson Educación.

Kemmerling, K Van Bon, J. 2004. Gestión de servicios TI: An Introduction. Van Haren Publishing

Lawrence, S. 2002 Ingeniería de Software Teoría y Práctica, editorial Prentice Hall

Lim, S. Juster, N. and Pennington, A.1997. Enterprise modelling and integration: a taxonomy of seven key aspects, Computers in Industry.

Luque, I. Gómez, M. 1999. Ingeniería del Software: Fundamentos para el Desarrollo. Universidad de Cordoba.

Luzadder, J. 1993. Fundamentals of engineering drawing :with an introduction to interactive computer graphics for design and production. Prentice-Hall

Mortier, G. 2005. Técnicas de Programación: Guía fundamental de Desarrollo de Software. MP Ediciones.

Muñóz, D. Romero, S. 2006. Introducción a la Ingeniería: Un Enfonque Indistrial. Thomson Learning Ibero.

Norton, P. 2006 Introducción a la Computación traducido por Claudia Fuentes, Héctor Esqueda, Editorial McGraw-Hill

Parra, E. 1998. Tecnologías de la Información en el Control de Gestión. Ediciones Díaz de Santos.

Pérez, L. 2001. Las Tecnologías de la Información en la Nueva Economía. Ediciones Díaz de Santos.

Piattini, M. García, F. 2002. Calidad en el Desarrollo y Mantenimiento del Software. Editorial Madrid.

Poppendieck M., Poppendieck T. 2003. Lean Software Development: An Agile Toolkit for Software Development Managers. Addison Wesley.
Pressman, R. 2005 Software Engineering: A Practitioner's Approach Editorial McGraw-Hill Professional.
Ramírez, R. 2005. Gestión del desarrollo de sistemas de telecomunicación e informáticos. Thomson Learning Ibero.
Schmuller, J. 2000 "Aprendiendo UML en 24 horas". Prentice Hall. Primera Edición. México.
Schwaber K., Beedle M., Martin R.C. 2001 .Agile Software Development with SCRUM.. Prentice Hall.
Sommerville, I., Davidsson P. 1997 Software Engineering, 5th. Edition.
Spewak, S.1993. Enterprise Architecture Planning, Developing a Blueprint for Data, Applications, and Technology. John Wiley & Sons.
Stair, R. 2000 Principios de Sistemas de Información, traducido por Julio CoroPando, Jorge L Blanco, Enrique Mercado, Editorial Thomson Learning Ibero.
Turban, A. 2001, Decision Support Systems and Intelligent Systems 6a ed., Editorial Pearson - Prentice Hall.
Valvano, J. 2003. Introducción a los Sistemas de Microcomputadores Empotrados. Editorial Thomson Learning Ibero.
www.ripit.granma.inf.cu/PerfecEmp/Paginas/Glosario.asp
Weitzenfield. A. 2005. Ingenieria de software orientada a objetos con Java e Internet. Thomson Learning Ibero.
Zabalza, M.A. 2006 Competencias Docentes del Profesorado Universitario Calidad y Desarrollo Profesional.

***Acerca de los autores**

El Profesor Luciano Rayas Moreno es egresado del Centro de Estudios Universitarios. El presente trabajo de investigación fue presentado para obtener el título de Maestría en Educación Superior. daena@spentamexico.org

El Dr. José Luis Abreu Quintero es Profesor e Investigador de Spenta University y de la Facultad de Administración y Contaduría Pública de la U.A.N.L. Monterrey, NL. abreu@spentamexico.org. Tel. 52-81-8355-5567