UNIVERSIDAD NACIONAL AUTONÓMA DE MÉXICO

FACULTAD DE CIENCIAS

UNA PLANTILLA Y PRÁCTICAS PARA LA ENSEÑANAZA DE LA INGENIERÍA DE SOFTWARE

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
LICENCIADA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

DANIELA AVALOS MONROY

DIRECTORA DE TESIS:
M EN C. MARÍA GUADALUPE ELENA IBARGÜENGOITIA GONZÁLEZ

Datos del Jurado

Datos del alumno
Avalos
Monroy
Daniela
5544-0178
Universidad Nacional Autonoma de Mexico
Facultad de Ciencias
Lic. en Ciencias de la Computación

Datos del tutor
Maestra en Ciencias
Maria Guadalupe Elena
Ibarguengoitia
Gonzalez

Datos del Sinodal 1
Doctora
Hanna
Oktaba

Datos del Sinodal 2
Doctora
Amparo
Lopez
Gaona

Datos del Sinodal 3
Maestro en Ciencias
Gustavo Adolfo
Arellano
Sandoval

Datos del Sinodal 4
Matematica
Maria Concepcion Ana Luisa
Solis
Gonzalez-Cosio

Datos del trabajo escrito
Una Plantilla y Practicas para la Enseñanza de la Ingenieria de Software
84 paginas
2006

A mi mamá, que desde donde se encuentra,
siempre me apoyó en todo.
A mi familia, gracias por todo su apoyo incondicional.
A mis amigos, y compañeros.
A mis profesores, y toda la gente que contribuyó a que se hiciera realidad.
Gracias.

Índice General

Introducción

Por ser la ingeniería de software una rama casi nueva de las ciencias de la computación, no se ha logrado todavía establecer la manera mas adecuada de su enseñanza a nivel profesional. Siendo una recién egresada de la carrera de Ciencias de la Computación, y por lo tanto alumna reciente de la materia, pude darme cuenta de algunas de las deficiencias en cuestión de docencia de la materia.

Hasta el momento, todas las prácticas de las materias de la carrera Ciencias de la Computación estaban enfocadas únicamente a la enseñanza de programación, dejando a un lado el concepto fundamental de la ingeniería de software, que es el desarrollo de un plan y del seguimiento de un proceso para el desarrollo de un producto. Los alumnos no tomaban de manera seria todo el proceso de desarrollar un sistema, y todas las fases que este proceso involucra. Los alumnos tomaban a la ingeniería de software como una materia que sólo exigía la entrega de documentos que atrasaban el desarrollo de un proyecto. No se detenían a pensar que cuando se enfrenten al mundo real y se confronten con clientes tangibles que exijan proyectos que necesiten de investigación exhaustiva, del establecimiento claro de requerimientos, de un diseño previo, y ya sobre estas etapas el comienzo de la implementación, van a tener que poner en práctica sus conocimientos con relación a esta materia. Desgraciadamente por la falta de interés que mostraron en la materia, y por la manera no tan eficiente del desarrollo de la misma durante el curso, dichos conocimientos serán deficientes.

Enfrentando esta problemática y deseando tomar acciones al respecto se decidió el desarrollo de este trabajo de tesis, cuyo tema y objetivo principal es innovar con la realización prácticas para facilitar el aprendizaje y representar escenarios laborales comunes para mejorar la experiencia de los alumnos en la materia de ingeniería de software.

La presente tesis está organizada de la siguiente manera:

El primer capítulo de la tesis está enfocado a una investigación de distintos métodos de enseñanza, de la forma de establecer los objetivos de la misma y las características que estos deben tener a la hora de ser definidos, así como la definición de técnicas aplicables a la docencia, y las distintas técnicas más comunes y eficientes. Se hizo también la investigación de qué es una práctica, y por qué son necesarias en la impartición de ciertas materias. Así mismo se agregó un apartado donde se explica el beneficio del uso de prácticas para ciertas materias.

El segundo capítulo abarca la problemática que enfrenta actualmente la ingeniería de software, sobre todo a nivel de enseñanza, así como una explicación concreta de por qué es necesaria la realización de prácticas en la enseñanza de la materia y la especificación de los temas de la misma que requieren prácticas. En ese mismo capítulo se da una explicación de los procesos que utiliza la ingeniería de software para el desarrollo de

proyectos, tales como el Proceso de Software en Equipo (TSP, Team Software Process), Proceso de Software Personal (PSP,Personal Software Process), y el modelo de la madurez de la capacidad (CMM, Capability Maturity Model), y explica cómo se usan e integran dentro de esta ingeniería. Así como también una breve introducción al software que se utiliza para apoyar el desarrollo de diagramas de la ingeniería de software, tal como Racional Rose, Microsoft Visio, etc.

En el tercer capítulo se conformó una plantilla estándar para la realización de las prácticas. Este formato se realizó con la finalidad de mantener un orden tanto en presentación como en organización de las prácticas. Se estableció un primer bosquejo que estaba formado por Objetivos, Introducción, Material, Desarrollo, Preguntas de control, Resultados, Conclusiones y Bibliografía. En una segunda versión se tomó la decisión de omitir el campo de resultados, ya que la mayor parte de las veces se tomaban igual las conclusiones que resultados.

En el cuarto capítulo este trabajo se enfoca en las prácticas desarrolladas para cada etapa de la ingeniería de software. Cabe mencionar que este proyecto tiene como complemento otra tesis elaborada por Guadalupe Aguilar Morales [TC], en la cual se desarrollaron las prácticas para las etapas de Principios de TSPi, Planeación, Diseño, Implementación, Pruebas, Post Mortem, mientras que en ésta se desarrollaron prácticas para las etapas Lanzamiento, Estrategia, Requerimientos, Implementación, y Post Mortem. El listado de prácticas completas para el curso se muestra a continuación. Las prácticas que tiene (TC) se presentan en el trabajo complementario.

Principios de TSPi

- **Practica 1.** Motivación al trabajo en equipo (TC)
- **Practica 2.** Llenado de formas básicas (TC)

Lanzamiento

- **Practica 3**. Definir el equipo
- **Practica 4**. Establecer objetivos

Estrategia

- **Practica 5**. Definición de estándares.
- **Practica 6**. Estrategia de desarrollo.

Planeación

- **Practica 7.** Introducción a Microsoft Project (TC)
- **Practica 8.** Planeación del proyecto y plan de calidad (TC)

Requerimientos

- **Practica 9**. Diagramas de caso de uso con Visio.
- **Practica 10**. Detallar los casos de uso.
- **Practica 11**. Construcción del prototipo.
- **Practica 12**. Requerimientos no funcionales y plan de pruebas del sistema

Diseño

- **Practica 13.** Diseño de la arquitectura del sistema y plan para la prueba de integración. (TC)
- **Practica 14.** Diagramas de clases y de secuencia (TC)

Implementación

- **Practica 15.** Creación y conexión a bases de datos (TC)
- **Practica 16.** Paso del modelo de clases a código Java (TC)
- **Practica 17**. JSP's, Servlets y Tomcat.
- **Practica 18**. Plan de pruebas unitarias y realización de pruebas unitarias.

Pruebas

Postmortem	➢ **Practica 19.** Pruebas de integración y del sistema. (TC) ➢ **Práctica 20.** Manuales del sistema. (TC) ➢ **Practica 21**. Recolección de métricas del proceso. ➢ **Practica 22.** Evaluación de roles en el equipo. (TC)

TC – Tesis Complementaria[TC]

Es importante agradecer a la empresa Microsoft Research de EUA y en especial a Jaime Puente,
Director de Latioamerica, por la donación del software Visio que fue utilizado para la diagramación en UML(Unified Modeling Language), facilitando el trabajo de algunas de las prácticas que forman parte de esta tesis.

Capítulo 1

Técnicas de Enseñanza

1.1 Objetivos de la enseñanza.

Al momento de comenzar un curso los profesores deben tener previstos cuáles serán los objetivos del mismo, lo que se desea cumplir mediante la enseñanza de la materia. Los profesores deben enfocarse en el producto final de la enseñanza.
Sería adecuado que primero se decidiera que resultado se quiere obtener al término de la escolaridad y luego que se disponga de los medios necesarios para que todos los alumnos, con sus diferentes capacidades e inclinaciones, lleguen lo más cerca posible a esa meta.

Cualidades de los objetivos de enseñanza
Las cualidades generales más importantes que deben poseer los objetivos de enseñanza son las siguientes:

- Lógicos. Permitir el razonamiento metódico y justo sin contraindicaciones internas.
- Precisos. Abarcar todos los aspectos necesarios y evitar términos inútiles.
- Concretos. Evitar verbos y adjetivos de significación vaga.
- Factibles. Deben ser realizables según el nivel de enseñanza, lugar, tiempo y recursos disponibles.
- Evaluables. Posibilitar la comprobación de su logro.

1.1.1 Características de los objetivos de enseñanza

Las características más destacadas de los objetivos de enseñanza son las siguientes:

- Nivel de generalidad. General, particular y específico.
- Funciones pedagógicas. Educativas e instructivas.
- Nivel de asimilación. Familiarización, reproducción, aplicación y creación.
- Contenido de enseñanza. Conocimientos y habilidades.

- Nivel de profundidad. Esencia del contenido a asimilar (grados de complejidad y abstracción).
- Condiciones existentes. Recursos con que se cuenta.
- Tiempo disponible. Según la actividad docente que se realiza.
-

A continuación se detallan cada una de estas características

Nivel de generalidad
De acuerdo con el nivel de generalidad o sistematización, los objetivos de enseñanza se clasifican en generales, particulares y específicos, y ello determina el volumen del contenido
Función pedagógica
La función pedagógica se refiere a la intención del proceso dirigido a la transformación que se aspire a alcanzar en los estudiantes, y que comprende la educación y la instrucción.
La educación tiene un carácter más amplio y general, orientada hacia la formación de la personalidad.
La instrucción tiene un carácter más particular, orientada hacia la asimilación de los contenidos de las asignaturas

Nivel de asimilación
Los niveles de asimilación de los contenidos de enseñanza son los grados de dominio de los conocimientos

En el nivel de familiarización, los estudiantes son capaces de reconocer los conocimientos y habilidades presentados, aunque no los puedan reproducir.

En el nivel de reproducción, los estudiantes son capaces de repetir el conocimiento asimilado y la habilidad adquirida en iguales condiciones.

En el nivel de producción o aplicación, los estudiantes son capaces de aplicar o utilizar los conocimientos y habilidades en la solución de los problemas que se les presenten.

En el nivel de creación, los estudiantes son capaces de resolver problemas en situaciones nuevas, sin disponer de los conocimientos suficientes para ello.

Contenidos de enseñanza
Los contenidos de enseñanza son los componentes que caracterizan el proceso de enseñanza-aprendizaje y comprenden los sistemas de conocimientos y habilidades de las asignaturas

Nivel de profundidad
El nivel de profundidad se refiere al nivel de esencia de los contenidos a asimilar; que determina las generalizaciones esenciales, las características fundamentales de la materia de estudio y precisa el grado de complejidad de las habilidades y el grado de abstracción de los conocimientos que deben dominar los estudiantes en cada nivel de enseñanza y disciplina.

Condiciones existentes y tiempo disponible
Las condiciones existentes y el tiempo disponible para el cumplimiento de la tarea planteada dependen del tipo de actividad docente que se desarrolla, y en el primer caso están determinadas por los recursos materiales y personales.

1.2 ¿Qué es una técnica de enseñanza?

Es necesario que el profesor sepa como usar diferentes métodos y técnicas que ayuden a los jóvenes a aprender. El profesor debe preguntarse a sí mismo que es lo que quiere enseñar y qué podría hacer para que la audiencia capte la idea. Al enseñar a los jóvenes, se debe tratar de usar técnicas que requieran el uso de los cinco sentidos. El hecho de observar las reacciones de la audiencia permite saber si ellos están entendiendo lo que se les está diciendo. Si se observan expresiones de confusión o se nota que los jóvenes no están poniendo atención, se debe intentar explicar el tema de alguna otra manera. Lo más importante es que se conozca qué técnicas de la enseñanza hay disponibles y cómo se pueden usar. La experiencia en el uso de estas técnicas sólo viene a través de la práctica. Enseguida se presentan varias técnicas de la enseñanza que se pueden usar:

1.2.1 Presentación Oral

Usualmente el profesor se dirige al grupo usando notas preparadas con anticipación sin tener ayudas visuales o gráficas y sin dar oportunidad para que el grupo haga preguntas. Esta presentación es útil cuando se va a presentar información completamente nueva para el grupo. El método de presentación oral se puede usar solamente por un periodo corto de tiempo. Es recomendable decirle a los jóvenes antes de empezar que se va a hablar por un periodo corto de tiempo y que después ellos tendrán una oportunidad para discutir lo que escucharon.
Esta técnica se debe combinar con otras como debates en grupos pequeños o la técnica de preguntas y respuestas que permitan al grupo expresar su opinión.

1.2.2 Presentación Oral Ilustrada

Esta es parecida a la técnica anterior. En esta, el profesor apoya su presentación oral con dibujos, posters, copias de artículos publicados y otros materiales.
Los dibujos o posters no necesitan ser profesionales, solamente interesantes y claros.

1.2.3 Debate

Esta técnica da oportunidad para que cada uno exprese sus ideas. Los miembros se pueden organizar en pequeños grupos y escoger diferentes temas a discutir. El debate ayuda a que los miembros compartan experiencias, ideas e información entre sí mismos. De esta manera se contribuye a que los jóvenes participen activamente y aprendan bastante si es que desde el principio ellos contribuyen con sus ideas a la discusión o debate.
La discusión tiene que enfocarse en una dirección definida. El grupo debe saber:

- Qué temas se van a discutir
- Cómo formar equipos

- Cómo conducir la discusión
- Cuánto tiempo va a durar la discusión
- Cómo reportar el trabajo

1.2.4 Técnica de preguntas y respuestas

El uso de la técnica de preguntas y respuestas es una manera rápida y efectiva para compartir el conocimiento que el grupo tiene. El profesor puede elaborar preguntas para estimular la atención y concentración del grupo en la materia que se presenta. Una sesión de preguntas y respuestas se puede llevar a cabo de diferentes maneras:

- El profesor puede formular la pregunta y dirigirla específicamente a un miembro del grupo
- El profesor puede solicitar preguntas de los miembros del grupo para responderlas personalmente o preguntarle a algún experto en la materia
- El profesor puede solicitar preguntas del grupo y a la vez dirigirlas a otros miembros del grupo para que respondan.

1.2.5 Demostraciones

A esta técnica también se le conoce como demostración de método. Básicamente el profesor demuestra a la vez que explica a los jóvenes como hacer algo. Por ejemplo cambiar una llanta, preparar una receta o hacer un nudo de corbata.
Otra clase de demostración es la demostración de resultados. Esta es una manera efectiva de enseñar buenas prácticas. Esta técnica permite demostrar visualmente los resultados que se pueden obtener al experimentar con objetos, plantas, etc... Por ejemplo se podría demostrar que le sucede a una flor blanca cuando se le agrega colorante azul al agua. Esta técnica es un instrumento muy efectivo para la enseñanza.

1.2.6 Sesión de trabajo o ejercicio práctico

Este método puede ser usado junto con cualquiera de los otros ya mencionados, debido a que provee a los jóvenes la oportunidad de “aprender mientras van haciendo”. El método de sesión de trabajo o ejercicio práctico es básico. Bajo la guía del profesor, los jóvenes pueden probar y practicar sus nuevas experiencias de aprendizaje. Por ejemplo: después de haber demostrado a un grupo como cambiar una llanta, cada joven lo puede practicar.

1.2.7 La experimentación

Esta es una técnica exitosa para todos los jóvenes cuando está basada en el estudio individual o de grupo. Aquí vemos si las ideas funcionan.

1.2.8 Recorridos y Visitas

Esta técnica es particularmente útil cuando los proyectos que los jóvenes están llevando a cabo no se pueden transportar prácticamente a cada reunión del grupo. Aquí cabe mencionar proyectos tales como huertos familiares, jardinería, agricultura y la crianza de ganado.
También se pueden hacer arreglos para otros tipos de recorridos para visitar proyectos que se estén llevando a cabo a largo plazo o donde se puedan ver resultados de proyectos parecidos a los del grupo.

1.2.9 Exhibiciones

Este método consiste en compartir una experiencia de aprendizaje con otras personas. En una exhibición se muestra o se habla de un tema relacionado a un proyecto específico. Es una actividad que sirve para que otras personas se den cuenta de lo que los jóvenes están aprendiendo en sus proyectos.

1.3 ¿Qué es una práctica?

Las prácticas son las técnicas educativas que facilitan el aprendizaje. Con este tipo de actividades se pueden evaluar los conocimientos previos adquiridos, y el grupo puede aprender de experiencias tal vez nuevas que no podían considerarse en la teoría. Con la realización de dichos trabajos prácticos se pueden lograr un menor fracaso escolar y una mayor profundidad en los aprendizajes.
Con ellas se aumenta la eficacia de las actividades formativas que se desarrollan con los alumnos.

Aunque no todas las prácticas tendrán la misma potencialidad educativa, todas ellas se diseñarán con la finalidad de hacer un bien didáctico y pedagógico en general, de parte del profesor, que de acuerdo con las fases del acto didáctico debe considerar:

- **Momento preactivo:** Antes de comenzar la realización de una práctica, el profesor debe considerar:
    - Las características grupales e individuales de los estudiantes: conocimientos, forma de aprender e intereses.
    - La definición previa de los objetivos que se pretenden alcanzar, y la preparación adecuada de los contenidos concretos que se tratarán.
    - El diseño de una estrategia didáctica que considere la realización de actividades didácticas con metodologías de trabajo activas y coolaborativas. Dichas acciones ayudarán al aprendizaje.
    - La organización de un sistema de evaluación para conocer el progreso de los aprendizajes, que realicen los estudiantes, sus logros y sus dificultades y facilite el asesoramiento cuando sea necesario.
- **Intervención Docente:** A partir de una especificación de objetivos y metodología, se realizará un desarrollo flexible con los estudiantes adecuando esta estrategia didáctica con los conocimientos que se desean inculcar.

- **Momento Postactivo:** Después de la intervención docente, el profesor deberá llevar a cabo una reflexión del proceso realizado, analizando los resultados obtenidos y los posibles cambios que se pudieran llevar a cabo para mejorar la actividad en próximas ocasiones.

1.3.1 Factores que facilitan la realización de buenas prácticas

Las buenas prácticas se realizan considerando los siguientes factores:

- Factores relacionados con los alumnos.
    - Grado de homogeneidad de los alumnos, intereses, conocimientos, etc...

- Factores relacionados con el profesor.
    - Habilidad didáctica del profesor.
    - Conocimiento del profesor de los recursos disponibles.
    - Motivación por su trabajo.
    - Actitud investigadora e innovadora dentro del aula.
- Factores relacionados con la escuela.
    - Infraestructuras de la escuela aceptables
    - Existencia de salas de estudio especializadas
    - Biblioteca
    - Adecuada dotación de recursos educativos (libros, software, etc...)
- Factores relacionados con la administración educativa.
    - Incentivos, planes de formación, apoyo al profesor, etc...

1.3.2 Método de Resolución

El alumno debe seguir una secuencia de pasos para resolver una práctica. Los pasos son los siguientes:

- Leer el enunciado completo y las veces que sean necesarias, hasta tener en claro lo que se debe hacer. Debe ser capaz de identificar los conceptos clave del problema.
- Pensar como va a resolver el problema. Usar cualquier método que considere necesario para encontrar una solución.
- Resolver el problema. Resolverlo poco a poco asegurándose de que lo que va realizando es correcto.
- Probar los resultados. Una vez resuelto el problema se debe verificar, si esto es posible, que la solución es correcta. Esto se puede realizar mediante varias pruebas.
- Entregar. Una vez que se está seguro del resultado se puede entregar la práctica.

Las prácticas de laboratorio son útiles para que el profesor pueda valorar el nivel de asimilación de conocimientos de los alumnos, además de las habilidades que van obteniendo. Por eso se debe tomar en cuenta:

- Los aspectos que garanticen la preparación previa a la práctica.
- La realización exitosa de la misma.
- La entrega. Con investigación propia del estudiante.

En la práctica se deben tomar en cuenta los siguientes aspectos:

- Preparación teórica de los alumnos
- Desarrollo de habilidades y hábitos de desarrollo en la práctica.
- El vínculo con otras asignaturas.
- El suministro de datos suficientes para el estudio independiente de cada alumno.

Para el diseño de la práctica de laboratorio se debe tener en cuenta lo siguiente:

- Fase. Tema en que se trabaja o fase del proceso.
- Título. Se debe enumerar y titular acorde con el programa de estudios.
- Objetivo. Debe abarcar el contenido de la práctica. Qué actividades se deben realizar
- Introducción teórica. Que le explique a los alumnos en que consiste la práctica, y cuáles son sus bases.
- Equipos y Materiales. Especificar qué equipos y materiales se utilizarán para la realización de la práctica.
- Desarrollo. Aquí se debe dar una breve y compacta lista de pasos a seguir para la correcta realización de la práctica
- Preguntas de Control. Se enfoca a realizar preguntas sobre el tema a tratar.
- Conclusiones. Por último los alumnos brindan una opinión de lo que significó para ellos el desarrollo de dicha práctica además de especificar los conocimientos adquiridos en la realización de la misma.
- Bibliografía. Enlistar el material de apoyo utilizado para la realización de las prácticas. Material que los alumnos puedan consultar individualmente.

1.3.3 Beneficio del uso de prácticas

Este tipo de metodologías posibilita entre otros aspectos:

- Atención de todos los alumnos a la práctica de laboratorio y su ejecución.
- Mayor reforzamiento a los aspectos técnicos y metodológicos. Pudiendo los estudiantes aplicar la creatividad e independencia en la práctica.
- Permite al profesor con mayor facilidad hacer preguntas sobre la práctica, que abarquen los objetivos y posibilitar un mayor grado de generalización por parte de los estudiantes mostrando los mejores caminos para la creatividad y estética en el montaje, así como las normas y procedimientos establecidas.
- Es posible el tratamiento adecuado de las diferencias individuales.

- Permite el cumplimiento del rol de la crítica y la auto crítica durante el proceso.
- Activa la función de dirección del proceso docente
- Contribuye a que el estudiante se esfuerce más en su preparación teórica.
- Motiva al estudiante y desarrolla el amor hacia la asignatura y la carrera
- Proporciona mayor atención de los estudiantes hacia la realización del experimento o práctica.

Este capítulo está basado en la referencia [TX].

Capítulo 2

La ingeniería de software y su problemática

2.1 La Ingeniería de software y su problemática.

Actualmente hablamos de ingeniería de software, pero ¿la manera en la que desarrollamos software actualmente se puede denominar, con propiedad, ingeniería? La palabra Ingeniería tiene una connotación de prestigio que provoca que muchas disciplinas traten de autocalificarse como tal. Si bien existen estudios detallados sobre el concepto de Ingeniería, en general se concibe la Ingeniería como una aplicación práctica y eficiente de los conocimientos científicos. Desde este punto de vista, el sector del software se podría encontrar en una fase de producción comercial en algunas organizaciones pero seguramente resulta demasiado optimista hablar de la aplicación generalizada de una auténtica Ingeniería.

¿Se aplica actualmente la ingeniería de software?
La investigación y el desarrollo de técnicas y métodos de ingeniería de software son constantes y suelen suponer interesantes avances en la resolución de problemas de desarrollo. Sin embargo, es habitual que en la práctica diaria profesional no se incluya ninguna de las recomendaciones más elementales de esta ingeniería. De hecho, las evaluaciones de los procesos productivos de desarrollo realizadas a raíz de los modelo de procesos de software confirman que dicha tarea suele estar básicamente en estado caótico. Y no sólo en, como uno podría pensar, pequeñas organizaciones de cualquier país sino también en las empresas grandes de países “avanzados”. Estos modelos de evaluación suelen revelar la despreocupación de los responsables de las organizaciones por la mejora de la calidad de sus procesos de trabajo o de sus productos: bien sea por contar con una situación interna de empresa poco propicia, porque el ambiente del mercado no fomenta la preocupación por estos temas o bien por su poco interés real en la búsqueda de la calidad.

Parece difícil saber exactamente las causas de esta situación. No obstante, cabe la posibilidad de reflexionar sobre algunas ideas que, quizás, estén aportando su grano de arena a mantener este caos. Por una parte, las relaciones entre desarrollador y cliente en el mercado actual de desarrollo de software. La dinámica del mercado y la tradición imperante en la gestión de las ofertas de software no fomentan el uso de planificaciones bien fundadas en un análisis apropiado de los requisitos. Lamentablemente, la gestión de los requisitos y de la planificación no se realiza en las mejores condiciones por los analistas y jefes de proyecto. Por otra parte, actualmente las empresas se quejan de la carencia de profesionales calificados para cubrir sus necesidades de personal (que, en muchos casos, se ha disparado con la explosión de Internet).

Así, el informe de IDC para Microsoft [Mi] indica la falta de hasta 1,200.000 profesionales relacionados con las TIC (tecnologías de la información y de las comunicaciones). Otros informes manejados por Cisco también confirman esta tendencia. Es de esperar que las necesidades de personal no fomenten precisamente que todos los nuevos profesionales lleguen a la empresa con los conocimientos apropiados de ingeniería de software para solventar los problemas de desarrollo de aplicaciones. De hecho, habrá que contratar personas que prácticamente no cuenten con casi ningún tipo de formación informática y a quienes será difícil transmitir ciertas técnicas y recomendaciones para un buen desarrollo de software.

Las posibilidades reales de aplicación de las técnicas de Ingeniería del software quedan, pues, limitadas también por la ya preocupante carencia de profesionales específicamente formados en ellas.

Estos problemas entroncan con el debate sobre la ingeniería de software como profesión. Lo cierto es que ésta profesión difiere muchísimo de la concepción tradicional de informático relacionada con la ciencia de la computación (computer science). Es necesario que el ingeniero de software sea, ante todo, ingeniero y que sea capaz de trasladar con sentido práctico los conocimientos científicos de la informática al desarrollo y mantenimiento de software. Esto lo obliga a que aporte soluciones reales a los problemas diarios de la organización de software, lo que puede suponer agregar a los conocimientos estrictamente técnicos, habilidades y formación en aspectos de gestión, economía, legislación, etc. Evidentemente, la formación en técnicas y métodos de ingeniería de software podría ayudar de alguna forma al futuro de la misma o ¿cuándo será el software un producto de ingeniería? Lamentablemente los actuales recintos de formación informática, asisten en muy poca medida a los aspirantes a desarrolladores de software en dicha materia, y aunado esto con el hecho de que en muchas empresas se tiene la errónea idea de que el hecho de ejercer la ingeniería de software retrasa el trabajo de desarrollo, se ha dejado atrás la fuerte necesidad de establecer un proceso bien definido para este tipo de desarrollo.

Lo cierto es que una vez que un estudiante de software se enfrenta con la vida real, y con clientes reales, debe afrontar la dura prueba de realizar un producto de calidad, con tiempos previamente establecidos, y con requerimientos estrictos, tarea para la cual, generalmente no se encuentra lo suficientemente capacitado, gracias a que no tuvo la formación suficiente con respecto a la fuertemente necesaria ingeniería de software. De ahí es que crece la duda de ¿Se estará dando la suficiente y correcta formación? Es verdad que se aprende demasiado con la experiencia laboral, es cierto también que incluso por el medio difícil uno llega a aprender cosas que jamás se hubiera imaginado que existían, pero ¿Por qué no fomentar la aplicación de ingeniería de software a nivel escolar, ayudando así a la mejor formación de Científicos de la Computación a desarrollar su trabajo con calidad, eficientemente, y con tiempos de entrega certeros?

No es sencillo inculcar a los desarrolladores, la mentalidad de realizar sus trabajos aplicando una metodología de ingeniería de software, más cuando ellos ya están acostumbrados a trabajar directamente con el código de sus productos. Es difícil a nivel escolar, hacerlos ver el fuerte trabajo que tienen encima cuando sus desarrollos sean productos mucho más complejos, en los cuales no interactuará sólo un cliente, sino que habrá muchos usuarios involucrados, y muchos procedimientos enfocados. Generalmente un estudiante de cualquier carrera de desarrollo de software no logra visualizar lo complejo que puede llegar a ser un desarrollo, porque su experiencia es sólo la de productos cortos, y precisos.

Es por ello que al presentarles una materia donde tendrán que aplicar más que sus conocimientos en programación, sino que tendrán que utilizar una metodología para realizar sus trabajos, se muestran poco interesados y hasta cierto punto apáticos al respecto. Por esto es sumamente importante generar una materia interesante y práctica, para lograr así captar más la atención de los alumnos, y generar una conciencia acerca del trabajo que se está realizando.

¿En qué difiere la ingeniería de software de otros tipos de ingeniería y en qué es similar?
Una propiedad que la ingeniería de software comparte con las otras es la necesidad de una descripción exhaustiva de lo que debe producirse. El proceso llamado "captura de requerimientos" tiene por objetivo entender el problema a resolver, su entorno y las características o cualidades que deberá cumplir el producto. Por otra parte, los proyectos de software están sujetos a cambios frecuentes, durante y después del desarrollo, por lo que se debe entender que puede cambiar aún antes de construirlo.

Sin embargo, a pesar del advenimiento de nuevas tendencias, las actividades básicas requeridas para la construcción del software han permanecido estables. Estas actividades incluyen las enumeradas en la siguiente figura.

- Definición del proceso de desarrollo de software que se usará.
- Administración del proyecto de desarrollo.
- Descripción del producto de software que se desea.
- Diseño del producto.
- Implementación del producto, es decir, desarrollo o programación.
- Prueba de las partes del producto.
- Integración de las partes del producto y pruebas del producto completo.
- Mantenimiento del producto.

Fig 1. Actividades básicas de la Ingeniería de software [Br]

La calidad en la ingeniería de software podría compararse con la establecida para cualquier otro producto obtenido de otra ingeniería, pero en comparación con estos productos, el software no se "desgasta" del modo que lo hacen las aplicaciones físicas y se requieren definiciones específicas para la creación de un software eficiente e incluso que supere las expectativas del cliente. En lugar de insistir en la perfección, se insiste en los estándares de calidad. Esto significa aceptar la obligación de definir estándares de manera precisa. Se deben tener consideraciones de dichos estándares para cada etapa del proyecto.

2.2 Material de Apoyo a la Ingeniería de Software

Algunos de los temas redactados en capítulos anteriores se pueden encontrar más a fondo en las siguientes páginas, además de otros temas relacionados con la ingeniería de software.

- La ingeniería de software. El proceso de la ingeniería de software, la vida del software. Disponible en: http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html#IngSoft
- La ingeniería de software. Introducción, inicios, objetivos de la ingeniería de software. Disponible en: http://www.ilustrados.com/publicaciones/EpypklZplZydAklOcM.php
- Problemas con la ingeniería de software, ingeniería de software y sistemas. http://dis.unal.edu.co/~fgonza/courses/2003/ingSoft1/CAP2.pdf

- Introducción a la ingeniería de software. Aplicación del modelado de procesos http://redie.uabc.mx/contenido/vol3no2/contenido-mireles.pdf
- Página principal del instituto de ingeniería de software (Software Engineering Institute) http://www.sei.cmu.edu/sei-home.html
- Introducción a la ingeniería de software, inicios, problemática del desarrollo de software de antaño. http://ciberia.ya.com/diegodjm/case/intro.html

2.3 Por qué son importantes las prácticas en la Ingeniería de Software

En el desarrollo de software comúnmente se menosprecia el valor que tiene el llevar a cabo un proceso para su creación. Esto, de cierta manera, desmerita la profesión de un ingeniero de software y de las empresas de consultoría del mismo, quienes siguiendo órdenes del jefe o el cliente, sugieren que no hay tiempo para las etapas previas a la implementación, comienzan programando.

Una forma de ver la problemática que esto conlleva es imaginar que un ingeniero civil o un arquitecto, comienzan la elaboración de alguna obra sin la realización de ningún plano o proyecto. Esta elaboración ciertamente contará con demasiadas carencias, seguramente no contará con buenos cimientos, y simplemente será una obra inutilizable por su pésimo diseño. Lo mismo sucede con la mayoría de las profesiones. Un contador no puede de un día para otro llevar la contaduría de una empresa sin hacer un análisis previo de los movimientos de la misma. Un médico no puede comenzar una cirugía sin un conocimiento previo del problema del paciente y un estudio para saber cómo realizarlo. Entonces, ¿ Por qué los ingenieros de software si cedemos ante el “chantaje”? y construimos software sin planeación ni diseño que son necesarios para la elaboración de un producto de calidad que sobresalga por la buena realización, claridad y eficiencia.

La Ingeniería de software ha evolucionado significativamente en lo que se refiere a modelos conceptuales y herramientas de trabajo, lo cual ha ayudado considerablemente a que el desarrollo y el mantenimiento de software sean actividades cada vez menos dependientes del arte de quienes llevan a la práctica un diseño elaborado. Dentro de estos aportes destacan los del paradigma de orientación a objetos y el proceso de desarrollo de software que cubre todo el ciclo de vida del software.

Es por esto que es esencialmente importante fomentar en los alumnos una conciencia acerca del trabajo previo que debe realizarse para comenzar el desarrollo de software, trabajo que se facilita con el uso de un proceso. Por ello es importante realizar prácticas de ingeniería de software que ayuden a los alumnos a familiarizarse con el trabajo en equipo y con el proceso

de desarrollo, generando en ellos una cierta experiencia para que puedan poner en práctica dichos conocimientos en un ambiente de trabajo real, disminuyendo así, los problemas de desarrollo que puedan presentarse en un futuro, por la realización de trabajo sin bases y mal estructurado.

2.4 Temas de la Ingeniería de Software que requieren prácticas.

La intención principal de la realización de estas prácticas es la de hacer más sencillo y didáctico el aprendizaje de la Ingeniería de software en los desarrolladores.

Se puede deducir que el pensamiento general de los desarrolladores de software es de practicidad. La mayoría de ellos tiene una mente analítica y prefieren trabajar directamente con la implementación del software sin hacer algún análisis previo de la situación.

Muchas veces este tipo de desarrolladores tienen que enfrentarse con la problemática de que su software no era exactamente lo que el cliente necesita, y es aquí donde se enfrentan con la difícil tarea de remodelar su software sin tener la más mínima idea de por donde comenzar, pues resulta que no le encuentran ni pies ni cabeza.

Se nota evidentemente la importancia de las prácticas de laboratorio, para que el alumno adquiera conocimiento directo de los ensayos en los que se comprueba el comportamiento real de las distintas fases de la Ingeniería de software en los diversos aspectos que pueden interesar, y se proporciona al alumno la oportunidad de conocer el proceso investigador que conduce desde la teoría hasta la preparación de normas o especificaciones aplicables a cada etapa.

Otro interés que se tiene con la elaboración de prácticas de laboratorio, es la de adquirir conocimiento de lo que surge en cada etapa, para al final integrarlo todo como parte de un resultado completo. Donde todos los acontecimientos adquiridos por etapa ayudarán a la formación sólida de un buen ingeniero, constructor de software con calidad.

Es por eso la necesidad de ofrecer prácticas de laboratorio a estudiantes de ingeniería de software, de esta manera ellos comprenderán con la práctica la importancia de un proceso para realizar un producto, además de que podrán disfrutar más de adquirir conocimientos al hacerlo integrándose con su equipo al mismo tiempo.

Los equipos de desarrollo varían tanto la secuencia como la frecuencia de estas actividades. El desarrollo de software en el mundo real, por lo general depende de una demandante lista de características así como de estrictas fechas de entrega determinadas por el mercado. Como resultado, sólo los grupos bien organizados de ingenieros con conocimientos de los métodos de Ingeniería de software son capaces de llevar a cabo estas actividades de modo apropiado. La Ingeniería de software incluye personas, proceso, proyecto y producto.

Las interacciones entre las personas involucradas en un proyecto de software tienen un efecto profundo en su éxito. Los equipos trabajan mejor cuando tienen conocimiento de lo que se debe hacer, y cuando los miembros tienen papeles específicos. Otro elemento del factor

personas se refiere a los interesados en el proyecto: involucrados que ganan o pierden algo con su resultado. Estas incluyen al cliente, el usuario final y los patrocinadores financieros.

El proceso de cascada comienza con la especificación de los requerimientos del problema, después procede el diseño, luego la implementación y, por último, a la etapa de pruebas. El mantenimiento suele incluirse en el proceso. En la práctica a menudo se usan procesos iterativos para el desarrollo de software, donde la cascada se repite varias veces, completa o en partes. Cuando se realizan de una manera disciplinada, este tipo de estilos pueden ser muy benéficos.

El proyecto muestra a los ingenieros realizando varios tipos de trabajos, según su papel o responsabilidad y después pasando los resultados a otros ingenieros que realizan sus propias responsabilidades. Un proyecto es el conjunto de actividades necesarias para producir los artefactos requeridos. Incluye contacto con el cliente escribir la documentación, desarrollar el diseño, escribir el código y probar el producto. El paradigma de la orientación a objetos puede ser muy útil para el desarrollo de un proyecto, en particular, para facilitar el cambio continuo ya que se puede usar para organizar diseños y códigos en partes (clases o paquetes) que coincidan con el problema real.

Los productos o artefactos de un esfuerzo de desarrollo de software consisten en mucho más que el código objeto y el código fuente. Por ejemplo, también incluyen documentación, resultados de las pruebas y medidas de productividad. Estos productos se llamarán artefactos. La especificación es un artefacto que define lo que el producto debe ser. Aunque esta tarea de especificación parece directa, en la práctica es difícil llevarla a cabo bien. Otros artefactos son el establecimiento de la arquitectura del software, los diseños detallados, etc. Una vez establecidos los requerimientos, se debe realizar la implementación (programación), que genera el artefacto de código. Estos últimos ayudan a los desarrolladores a escribir programas más sencillos de verificar. Todas las partes de una aplicación deben ser probadas, al igual que el todo. Las pruebas son los artefactos que incluyen los procedimientos de verificación que especifican cómo realizar las pruebas y los casos de pruebas que especifican los datos de entrada y los resultados esperados.

2.5 Módulos de procesos de Software

El proceso, son las actividades que deben realizarse en orden para obtener un producto final. En el caso de la ingeniería de software, un proceso de desarrollo se conforma por etapas, esto es, un equipo debe pasar por diferentes etapas para lograr un producto de calidad, que en este caso será un producto de software.

El PSP (Personal Software Process) [Hu] es una tecnología del Instituto de Ingeniería de Software que brinda disciplina a las prácticas individuales de ésta ingeniería, mejora la calidad de los productos, ofrece la predicción de la incrementación de costos y horarios, y la reducción del tiempo de vida del ciclo de desarrollo de software.

El PSP fue creado por Humphrey Watts S. es una manera de desarrollar y medir las habilidades de ingeniería de software de cada ingeniero. Este proceso, proporciona métodos detallados para estimar y planear, muestra a los ingenieros cómo dar seguimiento a su desempeño contra estos planes, y explica cómo los procesos definidos pueden guiar su trabajo. El PSP tiene el propósito de desarrollar hábitos de programación, en especial en cuanto a la

medición (¿Cuánto tiempo he dedicado a este código?, ¿Cuántas líneas he escrito?, ¿Cuántos defectos conocidos he generado?, etc..). El PSP supone que el ingeniero ya posee los conocimientos de lenguajes de programación.
Como se muestra en la figura 2, el PSP se divide en etapas graduales de crecimiento llamadas PSP0, PSP1, PSP2 y PSP3

PSP0 Proceso base
PSP0 acepta las prácticas de desarrollo actuales del estudiante, pero requiere que el estudiante

- Mantenga un registro del tiempo dedicado a trabajar en un proyecto
- Registre los defectos encontrados
- Registre los tipos de defectos

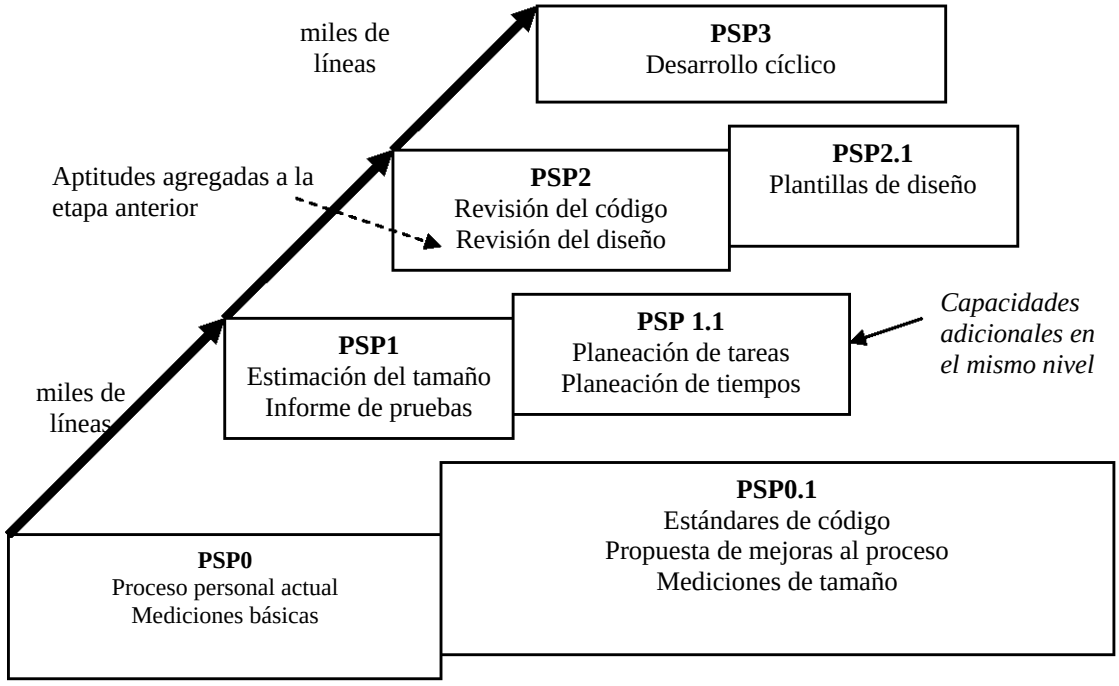


Fig. 2. Evolución del PSP (adaptado de [Hu])[Br]

Desde 1990, Watts Humphrey reportó resultados alentadores al establecer las metas de madurez y los procedimientos para el equipo de software. Llamó a este proceso Team Software Process (TSP)[Hu1]. Los objetivos de TSP se muestran en la siguiente figura .

- Formar equipos autodirigidos
  - 3 a 20 ingenieros
  - Establecer sus propias metas
  - Establecer sus propios procesos y planes
  - Rastrear el trabajo
- Mostrar a los gerentes cómo administrar equipos
  - Orientar
  - Motivar
  - Apoyar el desempeño más alto
- Acelerar la mejora del CMM
  - Hacer que CMM 5 sea “normal”
- Proporcionar guías de mejoramiento para organizaciones con alta madurez
- Facilitar la enseñanza universitaria a equipos integrados con la industria.

Fig. 3. Objetivos de TSP[Br]

El TSP enseña a los equipos de ingenieros como aplicar conceptos integrados de equipo al desarrollo de sistemas de software-intensivo. Este proceso ha sido utilizado con equipos que solo desarrollan sistemas de software y con equipos de sistemas que mezclan hardware, software y pruebas profesionales, y ha sido demostrado que reduce en gran parte el costo total del desarrollo. El TSP ha sido usado tanto para nuevos desarrollos como para mejoras y tanto para sistemas comerciales como para sistemas de tiempo real.

Antes de que los equipos de ingenieros comiencen a utilizar el TSP, deben ser entrenados en PSP.

El TSP tiene 5 objetivos principales:

1. Construir equipos autodirigidos que planean y realizan su trabajo, establecen metas y son dueños de sus procesos y planes. Estos pueden ser equipos de 3 a 20 ingenieros que solo desarrollen software, o equipos que desarrollen productos integrados.
2. Enseñar a los lideres como llevar y motivar sus equipos y como ayudarlos a mantener un máximo funcionamiento.
3. Acelerar la mejora de desarrollo de software, realizando el nivel 5 de CMM con el tipo de comportamiento normal y esperado.
4. Guiar a las empresas maduras en el mejoramiento continuo.
5. Facilita la enseñanza universitaria de habilidades de equipo en grado industrial.

El TSP fue desarrollado para ayudar a los equipos de desarrolladores a construir productos de calidad junto con la construcción de costos y horarios y para acelerar la mejora del proceso. Un equipo de software típico gasta una gran parte de tiempo y energía creativa luchando con preguntas como:

- ¿Cuales son nuestras metas?
- ¿Cuales son los roles del equipo y quien los desempeñará?
- ¿Cuales son las responsabilidades de estos roles?
- ¿Como tomará decisiones el equipo y establecerá puntos?
- ¿Que estándares y procedimientos necesita el equipo y como vamos a establecerlos?
- ¿Cuales son nuestros objetivos de calidad?
- ¿Como vamos a realizar un funcionamiento de calidad, y que debemos hacer si falla?
- ¿Que proceso debemos usar para desarrollar el proyecto?
- ¿Cual deberá ser nuestra estrategia de desarrollo?
- ¿Como debemos producir el diseño?
- ¿Como debemos integrar y probar el producto?
- ¿Como producimos nuestro plan de desarrollo?
- ¿Como podemos minimizar los tiempos de desarrollo?
- ¿Como podemos determinar el estado del proyecto?
- ¿Como podemos determinar, llevar y manejar los riesgos?
- ¿Que hacemos si nuestro plan no encaja con los objetivos del líder?
- ¿Como reportamos el estado al líder y al cliente?

TSP permite a los proyectos en equipo ahorrar tiempo y esfuerzo brindando una guía explícita en como acoplar sus objetivos. TSP agrega al PSP una capa de organización para el equipo en el proyecto.

Watts Humphrey desarrolló PSP (Proceso de Software Personal) y TSP(Proceso de Software en Equipo) en el Instituto de Ingeniería de Software(SEI) a mediados de los noventas, el SEI también es fuertemente reconocido por ser el que originó el concepto de CMM (Modelo de la Madurez de la Capacidad)[Hu2].

El CMM para software (también conocido como SW-CMM) ha sido un modelo para juzgar el proceso de software que ha llevado una organización desde muchos años atrás hasta ahora. Este modelo ayuda a las organizaciones a identificar la llave requerida para incrementar la madurez de estos procesos.

El SW-CMM se convierte en el factor estándar para determinar y mejorar los procesos de software. El SW-CMM, otros CMM's y ahora el CMMI, junto con el Instituto de Ingeniería de Software y la comunidad de Mejora del Proceso establecieron un medio efectivo de

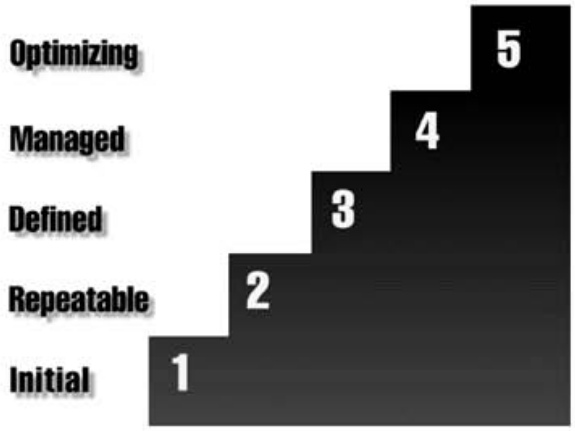


Fig. 4. Niveles de CMM

modelar, definiendo y midiendo la madurez del proceso usado durante el desarrollo por las organizaciones y manteniendo sistemas de software-intenso.

2.6 Material de Apoyo a TSP (Team Software Process)

Se puede tener contacto directo con el Software Engineering Institute (SEI) de la siguiente manera:

- ✓ Mandar un correo a tsp@sei.cmu.edu
- ✓ Visitando la página web http://www.sei.cmu.edu
- ✓ O la pagina mas directa para tsp http://www.sei.cmu.edu/tsp/courses.html

Algunas de las páginas encontradas como apoyo para TSP son:

- ➢ Construyendo equipos de alto rendimiento, usando Team Software ProcessSM (TSPSM) y Personal Software ProcessSM (PSPSM) http://www.sei.cmu.edu/tsp/

- Introducción y definición del término de Team Software Process
  http://www.softwaretechnews.com/stn3-4/teamspi.html
- Enseñando Team SoftwareProcess y Team Software Process. Breve introducción a los dos términos.
  http://oce.spsu.edu/cseet2002/Anthony-Lattanze.pdf

Y también están disponibles los libros que se incluyen en la bibliografía de este trabajo de tesis, [Hu] y [Br]

Capítulo 3

Formato de las prácticas

3.1. Estructura del curso de ingeniería de software

El curso de ingeniería de software que se imparte en el séptimo semestre de la carrera de Ciencias de la Computación en la Facultad de Ciencias, UNAM desde el 2000, se enfoca a impartir a los alumnos los conocimientos del proceso de software, principalmente el TSP, para desarrollo de software en equipo.

En el curso se les pide a los alumnos formen equipos de trabajo, y se les muestra los diferentes roles establecidos por el TSP para que puedan ellos escoger el rol que mas les convenga.

Una vez establecidos los equipos de trabajo y los roles de cada integrante del equipo, se comienza el desarrollo de software recorriendo las fases establecidas por TSP. Estas fases son Lanzamiento, Estrategia, Planeación, Requerimientos, Diseño, Implementación, y Post Mortem.

El proceso TSP propone dividir el trabajo en ciclos. Por cuestiones de tiempo y de docencia, en el curso de ingeniería de software, el proceso se divide en dos ciclos, que constan de todas las fases cada uno. En el primero los alumnos comienzan a familiarizarse con el proceso de software, es por eso que siempre se da mas tiempo para realizarlo, mientras que en el segundo, ya cuentan con cierta experiencia que les ayuda a realizar el proceso de manera mas rápida y eficiente.

TSP proporciona ciertas formas que ayudan a los ingenieros de software a realizar el trabajo de manera mas ordenada, y a tener mediciones de tiempos y tareas realizadas por cada uno de ellos. Estas fueron adaptadas para los estudiantes mexicanos de licenciatura y de maestría. Estas formas pueden consultarse en la página actual del curso de ingeniería de software y en el apéndice A.

La estrategia de TSPi consiste en aplicar un proceso cíclico para desarrollar el producto. En cada ciclo se deciden cuales funcionalidades desarrollar. En el primero de ellos se diseña, implementa y evalúa una primera versión del producto, en el segundo ciclo se incrementan las funcionalidades del producto para generar una segunda versión.

3.2 Formato de las prácticas.

Para la realización de las prácticas se conformó primero un formato estándar para las mismas. Este se realizó con la finalidad de mantener un orden tanto en presentación como en organización de las prácticas. Se diseñó un primer formato que estaba formado por Objetivos, Introducción, Material, Desarrollo, Preguntas de control, Resultados, Conclusiones, y Bibliografía. En una segunda versión de esta plantilla se tomó la decisión de omitir el campo de Resultados, ya que la mayor parte de las veces se tomaban igual las conclusiones que los resultados.

En el apartado de **objetivos** se establecen los objetivos principales que tiene la realización de la práctica, en este apartado se especifica cual es la finalidad de que los alumnos realicen dicha práctica.

En el apartado de **introducción**, como su nombre lo indica, se da una breve introducción del tema que se tocará en la práctica, es decir, aquí se explica de manera detallada el tema y todos los puntos que este involucra, de este modo se refuerzan los conocimientos de los alumnos, acerca del tema, y se amplían aun mas con detalles que talvez no conocían.

En el **material** se especifica el software o formas que necesitaran para el desarrollo de la práctica.

En el **desarrollo** se especifica detalladamente cuales son las actividades que deberán realizar, el desarrollo se da en forma de una lista donde se especifican las actividades.

En las **conclusiones**, los alumnos explican de manera breve los resultados que obtuvieron al realizar la práctica, y los conocimientos que adquirieron con la realización de la misma.

Por último con las **preguntas de control** se busca tener un control sobre los conocimientos adquiridos del alumno durante la realización de la práctica, además de que se busca una retroalimentación de parte del alumno ya que muchas de las preguntas van enfocadas a tomar en cuenta sus opiniones con respecto a que modificarían ellos en el proceso de ingeniería de software para que este sea mas eficiente.

Y con **la bibliografía** se le da al alumno el medio para realizar una investigación mas detallada acerca del tema de la práctica, dándole así las herramientas para que sus conocimientos crezcan y pueda interesarse más por la materia.

3.3 Plantilla de las prácticas.

PRÁCTICA # N

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Etapa de la práctica.**

INGENIERÍA DE SOFTWARE

TÍTULO DE LA PRÁCTICA.

OBJETIVOS :

- Lista de los objetivos de la práctica.

INTRODUCCIÓN :

Información básica y concreta de lo que trata la práctica.

MATERIAL:

- Software o material requerido para la realización de la práctica.

DESARROLLO:

- Lista de las actividades que deben realizar los alumnos en la práctica.

PREGUNTAS DE CONTROL:
1. Preguntas realizadas para saber los conocimientos adquiridos por el alumno.
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- Libros páginas Web, o tutoriales utilizados para la realización de las prácticas.

Es importante recordar que esta tesis tiene un trabajo de complementario que fue realizado por mi compañera Guadalupe Aguilar Morales.

Las prácticas que serán presentadas en este trabajo de tesis, serán las correspondientes a las fases de **Lanzamiento, Estrategia, Requerimientos,** la segunda parte de la etapa de **Implementación,** y la primer parte de **Postmortem.**

Mientras que las prácticas correspondientes a las fases de **Principios de TSPi**, **Planeación**, **Diseño**, primer parte de **Implementación**, **Pruebas**, y segunda parte de **Postmortem** se realizaron en la tesis complementaria.

Es necesario aclarar que estas prácticas se realizaron de manera intercalada, ya que este trabajo de tesis fue realizado en paralelo a la impartición de la materia con la profesora Ma. Guadalupe Ibargüengoitia, y se consideró necesario la repartición de trabajo de manera balanceada.

Capítulo 4

Prácticas para la fase de Lanzamiento

4.1. Prácticas para la fase de Lanzamiento

Durante la fase de lanzamiento se definen los objetivos del equipo y como establecerlos; se concluye con una descripción de los pasos del proceso de lanzamiento y los guiones correspondientes.

Los objetivos son un paso esencial en la formación del equipo. Se definen al inicio de cada proyecto porque establecen el marco para la estrategia y el plan proporcionando la base para cualquier producto que el equipo generara.

Práctica 3

Los objetivos de la Práctica 3, es que los alumnos comiencen a formar sus equipos de trabajo, que le den un nombre y un logotipo a su equipo, que establezcan los días de reuniones, y en una palabra, que comiencen a establecer como se realizará el trabajo en equipo.

Práctica 4

La Práctica 4, tiene como objetivo principal lograr que los alumnos comiencen a conocer bien el proyecto a desarrollar. Es por esto que en esta práctica se establecerán los objetivos a cumplir para el primer ciclo tanto del equipo como personales y del proyecto.

PRÁCTICA # 3

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Lanzamiento.**

INGENIERÍA DE SOFTWARE

DEFINIR EL EQUIPO.

OBJETIVOS :

- Formar el equipo de trabajo.
- Ponerle nombre.
- Hacer un logo.
- Fijar día y horario de reuniones semanales.
- Asignar roles a cada uno de los integrantes del equipo

INTRODUCCIÓN :

Es importante manejar un orden en la formación de equipos, y más si se trata de un equipo de trabajo. Todos conocemos, por ejemplo los equipos deportivos, donde cada uno de los integrantes juega un papel específico como miembro del equipo. En esta práctica asignaremos roles a cada uno de los integrantes del equipo, y estableceremos algunas de las reglas básicas para que el equipo funcione de manera correcta.

MATERIAL:

- Tabla 3.1. Estándar de roles y sus responsabilidades.

DESARROLLO:

- Reunirse en equipos.
- Ponerle un nombre al equipo.
- Hacer el logo para el equipo.
- Definir días y horas para las reuniones semanales del equipo.
- De acuerdo a la Tabla 3.1 de Estándares de roles establecer los roles de cada uno de los integrantes del equipo.

Tabla 3.1 Estándar de Roles y sus Responsabilidades

Responsabilidad	**LE**	**AD**	**AP**	**ACP**	**AA**
Construye y mantiene un equipo efectivo	X				
Resuelve asuntos entre los integrantes del equipo	X				
Realiza seguimiento y reporte del progreso del equipo	X				
Dirige cada reunión	X				
Se reúne con el instructor	X				
Realiza mantenimiento a la carpeta del proyecto.	X				
Ayuda a asignar las tareas en el equipo	X				
Dirige todo el trabajo de desarrollo		X			
Dirige la planeación del equipo y el seguimiento del progreso.			X		
Dirige el seguimiento y planeación de calidad				X	
Proporciona apoyo al proceso del equipo				X	
Dirige cada inspección				X	

Mantiene los estándares y glosarios				X	
Realiza el reporte de cada reunión				X	
Alerta al equipo en los problemas de calidad				X	
Obtiene las herramientas y apoyo necesarios					X
Maneja la administración de configuración					X
Dirige la mesa de control de configuración					X
Dirige el reuso del equipo.					X
Realiza el seguimiento de asuntos y riesgos.					X
Mantiene el glosario del sistema.					X
Desarrolla el producto.	X	X	X	X	X
Realiza planes personales	X	X	X	X	X
Realiza seguimiento en el trabajo personal	X	X	X	X	X
Genera productos de calidad	X	X	X	X	X
Sigue prácticas personales disciplinadas	X	X	X	X	X

INTEGRANTE DEL EQUIPO	**ROL A DESARROLLAR**
Escribe aquí	

PREGUNTAS DE CONTROL:			
1. ¿Por qué son necesarias las reuniones del equipo?			
Escribe tus respuestas aquí.			
2.- Menciona ventajas y desventajas de reuniones en persona contra reuniones virtuales.			
Reuniones en persona:		**Reuniones Virtuales:**	
Ventajas	**Desventajas**	**Ventajas**	**Desventajas**
Respuestas			
3. ¿Es importante establecer roles en un equipo de trabajo? ¿Por qué?			
Escribe tus respuestas aquí.			

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- Watts S. Humphrey. Introduction to the Team Software Process. Addison-Wesley. 1997
- Braude. Ingeniería de Software. Una perspectiva orientada a objetos. AlfaOmega. 2003

PRÁCTICA # 4

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Lanzamiento.**

INGENIERÍA DE SOFTWARE

ESTABLECER OBJETIVOS.

OBJETIVOS :

- Describir el proceso de Lanzamiento del equipo.
- Establecer los objetivos del proyecto.

INTRODUCCIÓN :

En la fase de Lanzamiento se definen los objetivos del equipo y cómo establecerlos. Los equipos necesitan establecer sus relaciones de trabajo, y sobre todo estar de acuerdo en los objetivos. Primero establecer los objetivos del proyecto, en base a las especificaciones dadas, tratando de apegarse lo mas posible a la perspectiva del cliente. Una vez establecidos estos objetivos, establecer los objetivos del equipo y personales.

MATERIAL:

- ¨Pasos para establecer objetivos¨ que se encuentran más adelante en esta misma práctica.

DESARROLLO:

- Sigue los ¨Pasos para establecer objetivos¨ que se anexan a esta práctica.
- Establecer los objetivos del equipo para el primer ciclo.
- Establecer los objetivos personales para el primer ciclo.
- Identificar los objetivos del producto según el enunciado dado en el curso.

Pasos para establecer objetivos

1. Definir los objetivos del proyecto
2. Proporcionar una copia de los objetivos a los integrantes del equipo.
3. Proporcionar una copia de los objetivos al Administrador de Apoyo para que los anexe en la carpeta del proyecto.

OBJETIVOS DEL EQUIPO PARA EL PRIMER CICLO

OBJETIVOS PERSONALES PARA EL PRIMER CICLO

OBJETIVOS DEL PRODUCTO

PREGUNTAS DE CONTROL:
1.- ¿Cómo se establecen los objetivos del producto?
Escribe tus respuestas aquí.
2.- ¿Qué es lo que se debe considerar para determinar los objetivos del equipo?
Escribe tus respuestas aquí.
3.-¿Crees que sea importante establecer objetivos al inicio de un desarrollo de software? ¿Por qué?
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- Watts S. Humphrey. Introduction to the Team Software Process. Addison-Wesley. 1997
- Braude. Ingeniería de Software. Una perspectiva orientada a objetos. AlfaOmega. 2003

Capítulo 5

Prácticas para la fase de Estrategia

5.1. Prácticas para la fase de Estrategia

Es necesario crear una estrategia para desarrollar un trabajo, se debe crear un diseño conceptual del producto, y hacer estimaciones preliminares sobre el tamaño del producto y el tiempo de desarrollo.

La estrategia en TSPi consiste en aplicar un proceso cíclico para desarrollar un producto. En cada ciclo se deciden cuáles funcionalidades desarrollar. En el primero de ellos se diseña, implementa y evalúa una primera versión del producto, en el segundo, se incrementan las funcionalidades de cada uno, por último, deberán estimar el tamaño y tiempo requerido para el desarrollo

Práctica 5
El objetivo de esta práctica es que los alumnos establezcan estándares de documentación para su trabajo en equipo, facilitando con esto la comunicación entre los integrantes del equipo y agregando seriedad a su entrega de documentos.

Práctica 6
El objetivo de esta práctica es crear una estrategia para el inicio del proyecto, esto es establecer la división de actividades por ciclos, el orden adecuado de su realización, así como identificar los riesgos que se pueden presentar durante el desarrollo del proyecto y que podrían poner en riesgo los tiempos de entrega.

La identificación de riesgos consiste en escribir todas las inquietudes o preocupaciones de quienes están relacionados con el proyecto, después de presionar continuamente a los integrantes del equipo a pensar en más inquietudes. La figura 5 ennumera los factores de riesgo más comunes en los proyectos que Keil [Ke] identificaron al realizar estudios en Estados Unidos, Hong Kong y Finlandia

1. Falta de compromiso de la alta administración
2. Falla al obtener el compromiso del usuario
3. Error al entender los requerimientos
4. Participación inadecuada del usuario
5. Falla al manejar las expectativas del usuario final
6. Cambio de alcance y/o de objetivos
7. Falta de conocimientos o aptitudes requeridas del personal

Fig5. Fuentes de riesgo en orden de importancia [Br]

PRÁCTICA # 5

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Estrategia.**

INGENIERÍA DE SOFTWARE

DEFINICIÓN DE ESTÁNDARES.

OBJETIVOS :

- Definir los estándares de documentación.

INTRODUCCIÓN :

Cuando se trabaja en equipo en el desarrollo de software, es importante mantener un estándar en la documentación, ya que para la entrega de un producto de calidad es necesario mantener un orden. El desarrollo de esta práctica les ayudará a establecer sus estándares de documentación así como a mantener dicho estándar para entregar productos de calidad, también ayudará a la comunicación del equipo y a establecer una integración uniforme de la documentación por todos los miembros del equipo.

MATERIAL:

- Ejemplo de un documento estándar utilizado en otro curso

DESARROLLO:

- Todos definirán el estándar para la documentación.
- Establecerán un estándar para nombrar cada uno de los documentos y especificarán el por qué de dicho nombre.
- Definirán dónde se almacenará toda la documentación haciéndola así accesible para todos los miembros del equipo.

ESTÁNDAR PARA LA DOCUMENTACIÓN:

Datos del Encabezado:	**Escribe aquí:**
Formatos y tamaños de letra para el encabezado:	
Formatos y tamaños de letra para títulos:	
Formatos y tamaños de letra para cuerpo:	
Formatos y tamaños de letra para pie de página.	

ESTÁNDAR PARA LOS NOMBRES DE LOS DOCUMENTOS:

A continuación escribe el estándar para nombrar los documentos y explícalo.
Escribe aquí

Definir dónde se almacenará toda la documentación.
Escribe aquí

PREGUNTAS DE CONTROL:
1.- ¿Cuál crees que sea la importancia de definir estándares de documentación?
Escribe tus respuestas aquí.
2.- ¿Qué debe tener un estándar de documentación?
Escribe tus respuestas aquí.
3.- Si no se estableciera un estándar de documentación ¿Cómo resolverías la entrega de documentos?
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- Watts S. Humphrey. Introduction to the Team Software Process. Addison-Wesley. 1997
- Braude. Ingeniería de Software. Una perspectiva orientada a objetos. AlfaOmega. 2003

PRÁCTICA # 6

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Estrategia.**

INGENIERÍA DE SOFTWARE

ESTRATEGIA DE DESARROLLO.

OBJETIVOS :

- Crear y documentar una estrategia para el inicio del proyecto y la división de actividades por ciclo
- Identificar riesgos en un proyecto
- Identificar el software que se necesitará para el proyecto

INTRODUCCIÓN :

La estrategia de TSPi consiste en aplicar un proceso cíclico para desarrollar el producto. En cada ciclo se deciden cuales funcionalidades desarrollar. En el primero de ellos se diseña, implementa y evalúa una primera versión del producto; en el segundo ciclo se incrementan las funcionalidades del producto para generar una segunda versión.

La estrategia de desarrollo especifica el orden en el que las funcionalidades del producto son definidas, diseñadas, implementadas y evaluadas.

- Cómo el producto se incrementará en ciclos futuros
- Cómo dividir el trabajo de desarrollo entre todos los integrantes.

La estrategia de desarrollo se produce al inicio del proceso para guiar la estimación en tamaños y la planeación de recursos.

- Si la estrategia cambia durante la planeación, se deberán actualizar los requerimientos o el desarrollo.

Las estimaciones preliminares en tamaño y tiempo

- Crear el plan de trabajo para cada ciclo de desarrollo
- Proporcionar las bases para asignar el trabajo entre todos los integrantes

El Administrador de Desarrollo debe dirigir al equipo en la definición de los **criterios de estrategia**, esto es, en la definición de todos los puntos mencionados anteriormente.

MATERIAL:

- Forma ESTRAT.
- Forma de Registro de Riesgos (RR).
- Ejemplo de necesidades de un sistema.

Forma ESTRAT

Nombre Equipo _____ Fecha Ciclo _____

Referencia	Funcionalidades	Ciclo	
		1	2
Ref dada x el equipo	Funcionalidad del proyecto a desarrollar.	✓	
Totales			

Forma Registro de Riesgos RR

Equipo _____ Fecha: _____
Ciclo:

Fecha	Fecha en la que puede presentarse el riesgo
Riesgo o asunto	Riesgo que puede presentarse
Descripción	Descripción del riesgo
Gravedad B,M, A	Baja /Media /Alta
Asignado a	Quien podría solucionarlo
Fecha de resolución	Para que fecha.

Para llenar la forma con la estrategia, nos podemos ayudar de la siguiente técnica.

Ejemplo:

La Gráfica de Dependencias de Necesidades ayuda a seleccionar estrategia que cumpla con los criterios establecidos.

1. *El producto del primer ciclo debe de proporcionar un subconjunto ejecutable de funcionalidades mínimas del producto final.* La gráfica permite revisar y seleccionar las funcionalidades mínimas del primer criterio de la fase de Estrategia, mediante la selección de los nodos terminales (no necesariamente todos) y una o mas trayectorias, de poca longitud, que se dirijan a esos nodos. Los nodos incluidos dentro de las trayectorias representan necesidades a satisfacer en el primer ciclo.

2. *El producto del primer ciclo debe proporcionar una base fácilmente escalable.* A partir de los nodos seleccionados como parte de las funcionalidades mínimas, es posible ir añadiendo otros nodos dependientes durante los siguientes ciclos. De esta forma, el esfuerzo de desarrollo (costo y trabajo) por cada ciclo se aprovecha e incrementa de manera mas eficiente

3. *El diseño de producto tiene una estructura modular que permite trabajo independiente de los miembros del equipo.* La gráfica sugiere una probable división de trabajo. Observando aquellos nodos que tienen más de un vértice de entrada (necesidades cuya satisfacción habilita la satisfacción de dos o más necesidades), sus nodos descendientes representan potencialidad de desarrollo independiente dentro de la estructura del producto, lo que los hace candidatos a una primera aproximación de componentes del diseño conceptual.
4. Posibles gráficas de dependencias para el ejemplo:

“Necesidades de un sistema procesador de encuestas.”

Objetivo: Desarrollar un sistema que apoye al usuario a realizar encuestas a una población dada sobre algunos temas de interés.

N1. El sistema pedirá al encuestado algunos datos personales y le ofrecerá distintos temas de encuesta a contestar.

N2. El usuario (solicitante del sistema) podrá definir el tema de la encuesta, los datos personales de los encuestados que le interesan, y las preguntas de la encuesta, así como las posibles opciones de respuesta.

N3. El sistema podrá efectuar algunas estadísticas directas sobre un tema de la encuesta, como cuántas personas eligieron cada opción de cada pregunta y qué porcentaje son de la población total.

N4. El sistema permitirá imprimir las estadísticas de las encuestas por tema.

N5. El sistema permitirá imprimir una encuesta de un tema sin llenar.

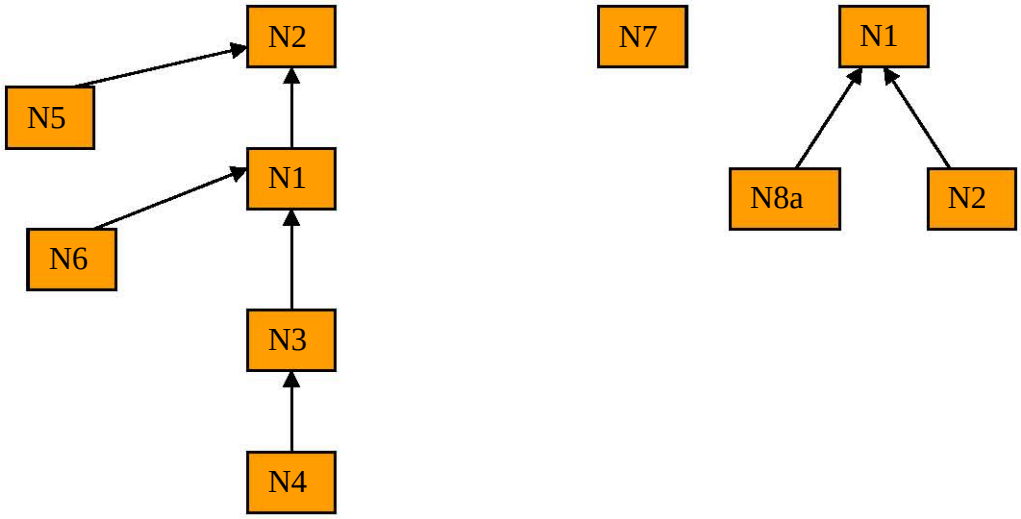
N6. El sistema permitirá imprimir cada encuesta contestada de un tema.

N7. Se programará en Java y se usarán tablas para guardar los datos que sean persistentes.

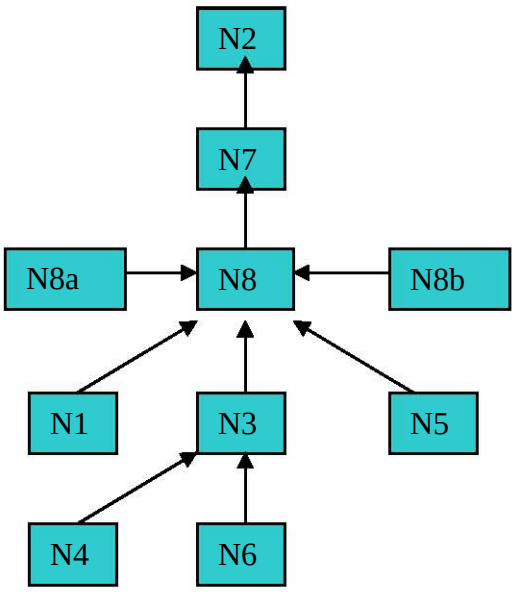
N8. La interfaz del sistema podrá ser (a) gráfica (usando GUI) o tipo (b) texto.

Ejercicio: generar la gráfica de dependencias.

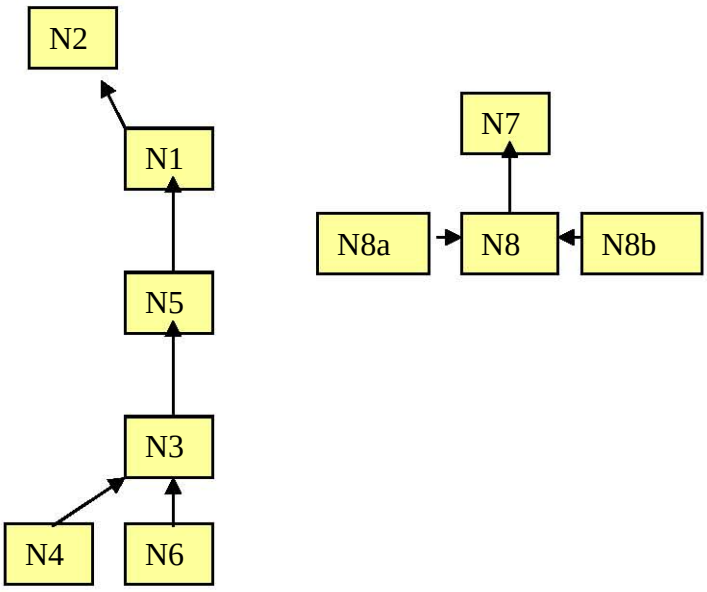
GRÁFICA 1



GRÁFICA 2



GRÁFICA 3



DESARROLLO:

- Siguiendo el ejemplo anexo a esta práctica, todos los integrantes del equipo crearán una lista de necesidades del proyecto y sus funcionalidades así como sus dependencias.
- El Administrador de Desarrollo dirigirá al equipo para componer diferentes estrategias. Entre todos los integrantes del equipo decidirán cuál será la más conveniente.
- El Administrador de Desarrollo coordinará la asignación de funcionalidades del producto en cada ciclo.
- El Administrador de Calidad documentará la estrategia elegida en la forma ESTRAT.
- Todos los integrantes del equipo determinarán riesgos y la gravedad de los mismos, así como quién se encargará de resolverlos y en cuánto tiempo máximo. Estos se deberán documentar en la forma RR.
- Todos los integrantes del equipo crearán una lista de software necesario para el desarrollo del proyecto. Especificarán con cuáles cuentan y cuáles necesitan.
- El Administrador de Apoyo especificará y facilitará las herramientas de soporte necesarias.

PREGUNTAS DE CONTROL:
1.- Da 2 ventajas y 2 desventajas de la creación de una Estrategia al inicio de un proyecto

de software:
Escribe tus respuestas aquí.
2.- ¿Por qué planear la identificación de riesgos y el plan de contención al desarrollar la estrategia del proyecto?
Escribe tus respuestas aquí.
3.- Describe en qué casos la tarea de identificar y prever riesgos en un proyecto no se justifique
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- Watts S. Humphrey. Introduction to the Team Software Process. Addison-Wesley. 1997
- Braude. Ingeniería de Software. Una perspectiva orientada a objetos. AlfaOmega. 2003

Capítulo 6

Prácticas para la fase de Requerimientos

6.1 Prácticas para la fase de Requerimientos

En esta fase el equipo genera la especificación de requerimientos de software. En tal documento se hace una descripción clara de lo que será el producto, deberá incluir el criterio preciso para evaluarlo cuando esté terminado y asegurar que las funcionalidades sean las correctas. También proporciona retroalimentación al cliente acerca de lo que se pretende construir.

Se debe saber exactamente lo que el producto debe hacer antes de construirlo. En el inicio de esta fase se realiza la definición de necesidades por parte del cliente, si existieran necesidades que no son claras se debe redactar un documento con preguntas que hagan referencia a todas aquellas dudas para obtener una respuesta posteriormente. Después de haberlas aclarado, de nuevo se escriben los requerimientos en lenguaje coloquial y se revisan con los usuarios para verificar si realmente es lo que desean.

Es bien dicho que un beneficio importante del análisis de requerimientos es la comprensión y acuerdo de la aplicación que se construirá [Br1], es por ello la importancia que tiene para la ingeniería de software el análisis de requerimientos.

Con frecuencia los requerimientos se expresan de manera natural como una interacción entre la aplicación y una agencia externa a ella, como el usuario. El caso de uso, concepto creado por Jacobson [Ja] es una forma muy útil de mostrar estas interacciones, es por eso que en las siguientes prácticas se pone una atención especial en el desarrollo de los mismos.

Práctica 9
El objetivo de la Práctica 9 es conocer a fondo el software Microsoft Visio para el desarrollo de diagramas de UML (Unified Modeling Language), para la fase de requerimientos del proyecto, el diseño de los mismos ayudarán a los integrantes del equipo y al cliente a comprender cuáles serán las funcionalidades del sistema.

Práctica 10

Los casos de uso se pueden expresar a diferentes niveles de generalidad. El proceso unificado de desarrollo de software (USDP, [Ja1]) recomienda usar casos de uso detallados para especificar una fracción grande de requerimientos.

El objetivo principal de la Práctica 10 es definir de una manera detallada cada uno de los casos de uso del proyecto, esto es, dar una lista de pasos a seguir, especificando qué acciones ejecutará el sistema y qué acciones deberá ejecutar el usuario para que se realice un caso de uso.

Práctica 11

El objetivo de la Práctica 11 es realizar las interfaces prototipos del sistema. De este modo el usuario y el diseñador tendrán una idea en común de cómo se verá el sistema una vez que quede terminado. Este desarrollo de prototipos es sumamente importante porque no se ha desarrollado nada aún y todavía hay tiempo de hacer las correcciones necesarias para que el sistema quede al gusto del cliente.

Práctica 12

El objetivo de la Práctica 12 es el de encontrar los requerimientos no funcionales del sistema. Los requerimientos no funcionales del sistema, son aquellas características no funcionales que solicita el cliente, y que si no son considerados, pueden afectar en la funcionalidad de la aplicación y el plan de prueba del sistema.

PRÁCTICA # 9

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Requerimientos.**

INGENIERÍA DE SOFTWARE

DIAGRAMAS DE CASOS DE USO CON VISIO.

OBJETIVOS :

- Conocer Microsoft Visio para realizar diagramas de UML
- Realizar diagramas de casos de uso

INTRODUCCIÓN :

El lenguaje de modelado unificado (UML) fue desarrollado como una manera de estandarizar la descripción de los diseños de software, en particular los orientados a objetos.

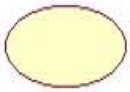
El diagrama de casos de uso representa la forma como un Cliente (Actor) operará con el sistema, además aclara la funcionalidad del sistema (operaciones o casos de uso). Para comenzar el diseño de los casos de uso se dará una breve definición de todos sus componentes.

Un diagrama de casos de uso consta de los siguientes elementos:

Actor:

Un actor es un usuario que interacciona con el sistema para obtener un valor o servicio. Es importante destacar que un actor no necesariamente representa a una persona en particular también puede ser otro sistema.

Caso de Uso:



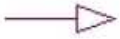
Es una funcionalidad o tarea específica que se realiza tras una orden de algún actor.

Relaciones:

Asociación:

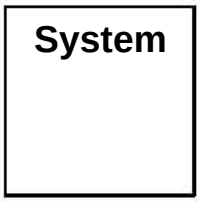
Es el tipo de relación más básica que indica la invocación de un actor a una funcionalidad. Dicha relación se denota con una flecha.

Generalización:



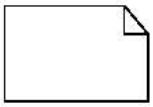
Se usa para actores, especializa el comportamiento del actor.

System Boundary:



Añade el límite del alcance del sistema en el diagrama de casos de uso.

Notas:



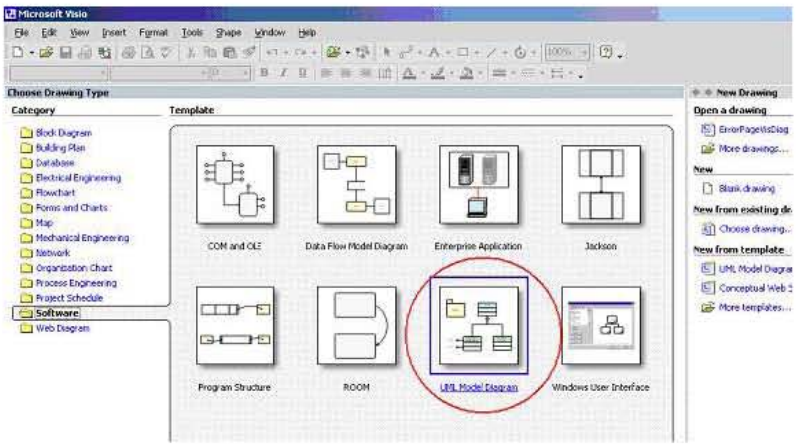
Se ponen las notas que se deseen agregar como texto dentro del diagrama de casos de uso.

MATERIAL:

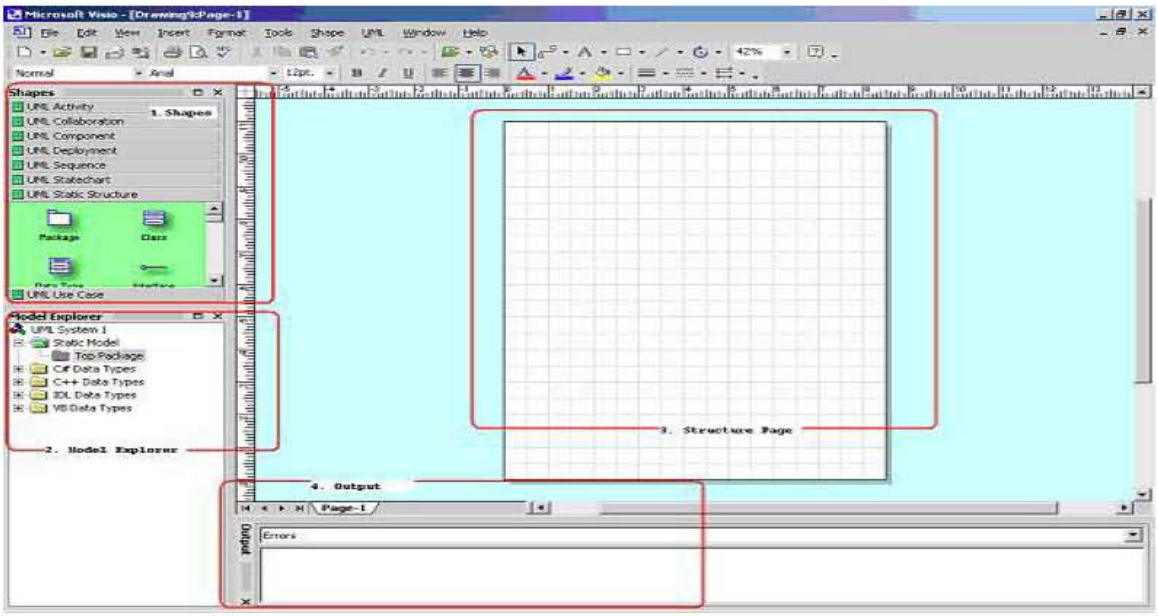
- Software Microsoft Visio

DESARROLLO:

- Entrar al software Microsoft Visio. En la pantalla de entrada seleccionarás del lado izquierdo la carpeta software, y de la pantalla central la opción UML Model Diagram, como se muestra en la siguiente figura.



- A continuación aparecerá una pantalla como la que se te muestra a continuación, que será tu área de trabajo. Del lado izquierdo, aparecen los componentes del diagrama de casos de uso. Para crear un diagrama de casos de uso basta con seleccionar y arrastrar dichos componentes al área de trabajo.



- De acuerdo con los conocimientos adquiridos hasta el momento, realizarás distintos diagramas de casos de uso para el siguiente ejemplo.

Supón que se construirá el software de una librería electrónica en la cual se venden distintos productos, ya sea libros, cd´s de música o dvd´s de películas en la que se pueden realizar las siguientes actividades:

El administrador de la librería podrá:

- ✓ Hacer consultas para verificar la existencia de algún producto.
- ✓ Dar de alta algún producto a la base de datos. Esta actividad la podrá realizar únicamente proporcionando antes una clave de seguridad.
- ✓ Dar de baja algún producto de la base de datos. Esta actividad la podrá realizar únicamente proporcionando antes una clave de seguridad.
- ✓ Sacar inventario de los artículos existentes
- ✓ Emitir reportes de faltantes. Esta actividad la podrá realizar únicamente proporcionando una clave de seguridad.

El vendedor de la librería podrá:

- ✓ Únicamente consultar la existencia de algún producto
- ✓ Sacar inventario de los artículos existentes

- Realiza un diagrama de casos de uso muy general para las actividades que puede realizar el administrador en la librería electrónica.
- Diseña otro diagrama general para las actividades que puede realizar el vendedor.
- Establece un último diagrama donde muestres todas las acciones que se pueden realizar en la librería electrónica, tanto el vendedor como el administrador.

Si crees que exista algún caso particular de relación de generalización en los actores de dicho software, márcalo en un diagrama de casos de uso extra.

PREGUNTAS DE CONTROL:
1.- ¿Por qué crees que sea importante el uso de diagramas de casos de uso para la definición de Requerimientos en el proceso de realizar un software?
Escribe tus respuestas aquí.
2.- ¿Es sencillo para ti interpretar un diagrama de casos de uso o no? Justifica cualquier respuesta
Escribe tus respuestas aquí.
3.- Da 2 ventajas y 2 desventajas de usar diagramas de casos de uso como parte de la Ingeniería de Software.
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- http://www.msdnaa.net/Resources/Display.aspx?ResID=1949

PRÁCTICA # 10

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Requerimientos**

INGENIERÍA DE SOFTWARE

ESPECIFICACIÓN DE CASOS DE USO.

OBJETIVOS :

- Definir de manera detallada los casos de uso del proyecto

INTRODUCCIÓN :

El gran reto que se enfrenta en la fase de requerimientos, es expresar con claridad lo que los clientes desean y necesitan.

Con frecuencia los requerimientos se expresan de manera natural como una interacción entre la aplicación y un agente externo a ella, que es el actor. El detalle del caso de uso consiste en una interacción típica entre un actor y una aplicación.

MATERIAL:

- Paquete Visio para el desarrollo de Diagramas de Casos de Uso.
- Ejemplo de especificación de Casos de Uso.

EJEMPLO DE ESPECIFICACIÓN DETALLADA DE CASOS DE USO.

Para el software de una librería electrónica de la práctica anterior, detallar el caso de uso Baja de artículo por el administrador.

Caso de Uso : Baja de artículos

Actor: Administrador

Descripción: El administrador da de baja un artículo, esto querrá decir, que ya no tendrá disponible ese artículo personal para prestarlo.

Precondiciones:

- El administrador ha dado su clave personal

Flujo:

ACTOR		SISTEMA		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Selecciona la opción de dar de baja un artículo	2	Muestra una lista en la pantalla con todos los artículos registrados	E1
3	Selecciona el artículo a eliminar			E1
4	Da clic al botón Eliminar	5	El sistema elimina el artículo del sistema.	

Excepciones:

Id	Nombre	Acción
E1	Cancelación de transacción, cuando el propietario da clic en Cancelar	El sistema se posiciona nuevamente la pantalla principal

Poscondiciones:

- El artículo quedó eliminado del sistema.

DESARROLLO:

- Crear el diagrama general de casos de uso del sistema con sus actores.
- Especificar cada uno de los casos de uso del diagrama general, explicando paso por paso el flujo del mismo.
- El detalle del caso de uso debe incluir el nombre del caso de uso, el diagrama, el actor, la acción, y la lista de acciones que genera el sistema y que debe generar el usuario para que se lleve a cabo el caso de uso.
- Identificar la lista de excepciones que podrían existir en cada caso de uso.

PREGUNTAS DE CONTROL:
1.- ¿Para qué es necesario detallar los casos de uso?
Escribe tus respuestas aquí.
2.- Si no tuvieras el conocimiento de los diagramas de casos de uso. ¿Qué herramientas

utilizarías para mostrarle al cliente tu perspectiva de sus necesidades, es decir de lo que desea que haga el sistema?
Escribe tus respuestas aquí.
3.- ¿Por qué es necesario detallar cada caso de uso en la fase de Requerimientos?
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- Watts S. Humphrey. Introduction to the Team Software Process. Addison-Wesley. 1997

PRÁCTICA # 11

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Requerimientos.**

INGENIERÍA DE SOFTWARE

CONSTRUIR PROTOTIPO.

OBJETIVOS :

- Realizar el prototipo de la interfaz de usuario, para tener una idea inicial de cómo se verá el sistema

INTRODUCCIÓN :

El diseño de la interfaz de usuario se incluye en la etapa de "Diseño" del desarrollo de software, pero también puede considerarse parte de la etapa de requerimientos.
En general, los clientes entienden qué hará una aplicación al visualizar la interfaz de usuario, por lo que una buena manera de ayudar a describir la aplicación es desarrollar el diseño preliminar de la interfaz. De manera que se presente al cliente las pantallas que tendrá el sistema.

Galitz [Ga] proporciona 8 pasos básicos para desarrollar interfaces de usuario.

PASOS PARA DESARROLLAR INTERFACES DE USUARIO

1. Conocer al usuario.
2. Aplicar los principios del buen diseño de pantallas. (Evitar el exceso de interfaces innecesarias, mantener una pantalla de inicio y una opción en cada una de las demás para poder regresar al inicio, resaltar las opciones importantes, etc.)
3. Seleccionar el tipo adecuado de ventanas.
4. Desarrollar los menús del sistema según el diagrama general de casos de uso. Para cada caso de uso, crear las ventanas que permitan cubrir sus funcionalidades.
5. Organizar y distribuir la pantalla.
6. Elegir los colores adecuados.
7. Crear íconos significativos.
8. Crear ventanas y mensajes para los errores y situaciones excepcionales descubiertas en los detalles de los casos de uso.

MATERIAL:

- Tutorial de HTML.

DESARROLLO:

- Como el proyecto que vamos a realizar para este curso debe ser un sistema en la Web, se propone como lenguaje de programación, únicamente para las interfaces de usuario, el HTML. Para este lenguaje existen varias herramientas interactivas que facilitan su programación. Tienen la libertad de usar la herramienta que gusten, o si desean programar directamente con código, pueden hacerlo.
- De acuerdo con los pasos anteriores para la realización de interfaces de usuario, realizarás las interfaces que conformarán tu sistema de acuerdo con los casos de uso de tu sistema. Es decir en tus interfaces o pantallas se deben incluir todos los casos de uso hechos anteriormente.

PREGUNTAS DE CONTROL:
1.- ¿Por qué es bueno realizar el prototipo de las interfaces de usuario en la fase de Requerimientos?
Escribe tus respuestas aquí.
2.- ¿Por qué se hacen prototipos de pantallas de usuario para el cliente?
Escribe tus respuestas aquí.
3.- ¿Propones alguna otra manera de darle a entender al cliente cómo se verá el sistema que él usará?
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- Braude. Ingeniería de Software. Una perspectiva orientada a objetos.Alfaomega. 2003
- [Ga] Galitz, W. The Essential Guide to User Interface Design: An Introduction to GUI Principles and Techniques, Nueva York: John Wiley & Sons, 1996

PRÁCTICA # 12

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Requerimientos.**

INGENIERÍA DE SOFTWARE

REQUERIMIENTOS NO FUNCIONALES Y PLAN DE PRUEBAS

OBJETIVOS :

- Conocer el concepto de requerimientos no funcionales y por qué es importante tomarlos en cuenta en la construcción de requerimientos.
- Encontrar los requerimientos no funcionales del sistema
- Desarrollar el plan de pruebas para la realización de las mismas

INTRODUCCIÓN :

El software en general, tiene algunas peticiones que no sólo tienen que ver con el funcionamiento. A las características no funcionales que solicita el cliente se les llama requerimientos no funcionales y pueden ser:

Requerimientos de Desempeño

Estos especifican las restricciones de tiempo que debe observar la aplicación. Los requerimientos de desempeño son una parte crítica de las aplicaciones en donde las acciones deben terminar dentro de limites de tiempo, especificados.

Confiabilidad y Disponibilidad

Estos especifican la confiabilidad en términos cuantificados. Este tipo de requerimiento reconoce que es poco probable que las aplicaciones sean perfectas, por lo que circunscribe su grado de imperfección. Por ejemplo:

"La aplicación de radar del aeropuerto (ARA) debe experimentar no mas de dos fallas de nivel uno, por mes"

La disponibilidad, tiene una estrecha relación con la confiabilidad, cuantifica el grado en el que la aplicación debe estar disponible para los usuarios. Por ejemplo:

"ARA debe estar disponible a nivel uno o dos en la computadora primaria o en la de respaldo en todo momento. ARA podrá estar no disponible en una de estas computadoras a nivel uno o dos, no mas de 2% del tiempo en cualquier periodo de 30 días"

Manejo de Errores

Esta categoría de requerimientos explica cómo debe responder la aplicación a los errores en su entorno. Por ejemplo. ¿Qué debe hacer la aplicación si recibe un mensaje que no está en el formato que se acordó?

Requerimientos de Interfaz

Los requerimientos de interfaz describen el formato con el que la aplicación se comunica con su entorno.

Por ejemplo, se puede solicitar que corra en un navegador de red, o de modo texto.

Restricciones de diseño

Las restricciones de diseño o implantación describen los limites o condiciones para diseñar o implementar la aplicación. Estos requerimientos no pretenden sustituir el proceso de diseño, solo especifican las condiciones que el cliente impone al proyecto, el entorno u otras circunstancias, e incluyen la exactitud, por ejemplo. "Los cálculos de daños de la Automobile Impact Facility (AEF) deben tener una exactitud de un centímetro."

El Plan de Pruebas del sistema

Este plan debe definir la estrategia de pruebas del sistema y casos de prueba por cada requerimiento. Aún no se tiene el sistema construido, pero éste ayuda a definir los requerimientos del sistema y a que el cliente detecte si falta que se considere algún otro requerimiento y si están bien definidos.

Contenido del Plan de Pruebas del Sistema

1. Para cada caso de uso se indica qué tipo de entradas se podrían dar y qué se espera que el sistema responda.
2. Se definen entradas válidas al sistema y casos erróneos y lo que se espera del sistema.
3. Se puede hacer una tabla para cada funcionalidad o caso de uso del sistema.

Caso de uso:

Caso de uso	Entradas	Resultados esperados

MATERIAL:

- Casos de uso del sistema.

DESARROLLO:

- De acuerdo con las definiciones dadas anteriormente, hacer la lista de los requerimientos no funcionales necesario para el proyecto que se esta desarrollando y especificarlos.

- De acuerdo a la explicación dada de lo que es un plan de pruebas y los puntos que debe incluir, realizar un plan de pruebas adecuado para el proyecto, considerando todo los caso de uso.

PREGUNTAS DE CONTROL:
1.- Da una breve explicación de lo que significan para ti los requerimientos no funcionales.
Escribe tus respuestas aquí.
2.- ¿Por qué se hace el plan de pruebas del sistema si aún no hay sistema?
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- Braude. Ingeniería de Software. Una perspectiva orientada a objetos. AlfaOmega. 2003

Capítulo 7

Prácticas para la fase de Implementación

7.1 Prácticas para la fase de Implementación

Antes de iniciar la implementación se revisa que se tenga el diseño de alto nivel completo. Con frecuencia este diseño, cuando se trata de grandes sistemas requiere de varias etapas. Primero, se subdivide el sistema en subsistemas, componentes o módulos, y el diseño detallado del nivel lógico del sistema.

Práctica 17

La siguiente práctica tiene la finalidad de impartir a los alumnos los conocimientos necesarios para la implementación del proyecto. Para el proyecto en el cual se basó el desarrollo de las prácticas, se tomó la decisión de utilizar JSP´s, porque es una metodología que nos ayuda a desarrollar aplicaciones web con manejo de bases de datos. Los JSP´s son una herramienta que nos ayudará a unir una página de html, con código de java que a su vez se comunicará con una base de datos en postgresss (Especificaciones en la tesis de Guadalupe Aguilar Morales). Por ello se les otorgan las bases de los conceptos de JSP´s y Tomcat, así como bibliografía por si necesitaran buscar más información al respecto.

Práctica 18

Con la Práctica 18 se desea que los alumnos adquieran los conocimientos suficientes para realizar las pruebas unitarias del sistema. Es por ello que primero se les da una introducción de las metas de las pruebas y del significado de las pruebas unitarias.

PRÁCTICA # 17

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Implementación.**

INGENIERÍA DE SOFTWARE

DESARROLLO WEB JSP´S SERVLETS Y TOMCAT

OBJETIVOS :

- Conocer la funcionalidad de los JSP´s y las ventajas que tiene sobre otros sistemas del mismo tipo.
- Conocer la diferencia entre JSP´s y Servlets
- Tener conocimientos acerca del servidor web Tomcat, el cual utilizaremos para el desarrollo de nuestro proyecto.
- Contar con los conocimientos suficientes para desarrollar las páginas Web, JSP´s que se utilizarán en el proyecto.
- Poder utilizar de manera personal el servido web Tomcat.

INTRODUCCIÓN:

Java Server Pages (JSP)

Una JSP es código de HTML o XML en el que se insertan bloques especiales de código (también llamados scripts). Esta implementación es ejecutada en el servidor y los resultados son páginas dinámicas que son enviadas al navegador del usuario. A pesar de que las JSP son fáciles de construir, tienen a su disposición todo el poder de la orientación a objetos de Java y la de Java server API. Las JSP hacen uso de JavaBeans, que son clases que tienen constructores sin argumentos (requeridos en JSP) y los métodos públicos GET y SET, para cada uno de sus atributos.

Una JSP es convertida primero en un archivo de código fuente de Java, el que luego es compilado en un "servlet class". Esto ocurre cuando la página es requerida, así que se convierte en un pequeño problema de rendimiento. Pero las páginas JSP son compiladas en Servlets Java y cargadas en memoria la primera vez que se las llama, y son ejecutadas para todas las llamadas siguientes. Esto le da a las páginas JSP la ventaja de la velocidad y escalabilidad.

¿Qué son los Servlets Java?

Los Servlets son la respuesta de la tecnología Java a la programación CGI. Son programas que se ejecutan en un servidor y construyen páginas Web. Construir estas páginas al vuelo es útil (y comúnmente usado) por un número de razones:

- La página Web está basada en datos enviados por el usuario. Por ejemplo, las páginas de resultados de los motores de búsqueda se generan de esta forma, y los programas que procesan pedidos desde sitios de comercio electrónico también.

- Los datos cambian frecuentemente. Por ejemplo, un informe sobre el tiempo o páginas de cabeceras de noticias podrían construir la página dinámicamente, quizás devolviendo una página previamente construida y luego actualizándola.

- Las páginas Web que usan información desde bases de datos corporativas u otras fuentes. Por ejemplo, usaríamos esto para hacer una página Web en una tienda on-line que liste los precios actuales y el número de artículos en stock.

JSP vs. Servlets

En una página se pueden incluir iteraciones y condicionales que interactúan con el HTML. Esto es uno de los factores importantes a la hora de diseñar JSP, el guardar la mayor cantidad de este código fuera de la página, como sería en un bean, o en otra clase que luego sea referenciada por la página. Las secciones de código de JSP pueden volverse inmanejables y difíciles de depurar, especialmente porque el compilador está trabajando en el ".java" que se ha generado y no directamente en la JSP. En una página con funcionalidad simple del lado del servidor, por ejemplo alternando la salida basada en algunos parámetros del pedido, un JSP puede ser mucho más simple y rápido de construir que un servlet.

Una ventaja de las JSP´s es que puede ser entendido por diseñadores Web que no son programadores de Java. Siempre que sean cuidadosos, pueden cargar la JSP en una herramienta de diseño para cambiar el HTML. Esta opción puede ser preferible a alterar el archivo fuente del servlet que tiene múltiples llamadas a imprimir en pantalla o funciones que se agregan a objetos en un buffer.

Entorno de Software

Para ejecutar las páginas JSP, necesitamos un servidor web con un contenedor Web que cumpla con las especificaciones de JSP y de Servlet. El contenedor Web se ejecuta en el servidor Web y maneja la ejecución de todas las páginas JSP y de los servlets que se ejecutan en ese servidor Web. Tomcat es un servidor web para las especificaciones Java Servlet y JSP.

Ejemplo Sencillo

Imprimir en pantalla la fecha y la hora por medio de un JSP.

Ejemplo 1: date.jsp

```
<HTML>
<HEAD>
<TITLE>JSP Example</TITLE>
</HEAD>
<BODY BGCOLOR="ffffcc">
<CENTER>
<H2>Date and Time</H2>
<%
```

```
java.util.Date fecha = new java.util.Date();
out.println("Hoy es: "+fecha);
%>
</CENTER>
</BODY>
</HTML>
```

Este ejemplo contiene HTML tradicional y algún código Java. La etiqueta <% identifica el inicio de un scriptlet, y la etiqueta %> identifica el final de un scriptlet. Cuando un navegador solicite la página date.jsp veremos algo similar a la Figura 1.

Figura 1. Petición de fecha date.jsp

En el ejemplo date.jsp se usa todo el nombre de la clase Date incluyendo el nombre del paquete, lo que podría llegar a ser tedioso. Si queremos crear un ejemplar de la clase Date usando simplemente: Date today = new Date(); sin tener que especificar el path completo de la clase, usamos la directiva page de esta forma:

Ejemplo 2 :date2.jsp

```
<%@page import="java.util.*" %>
<HTML>
<HEAD>
<TITLE>JSP Example</TITLE>
</HEAD>
<BODY BGCOLOR="ffffcc">
<CENTER>
<H2>Date and Time</H2>
```

```
<%
java.util.Date fecha = new java.util.Date();
out.println("Hoy es: "+fecha);
%>
</CENTER>
</BODY>
</HTML>
```

Todavía hay otra forma de hacer lo mismo usando la etiqueta <%= escrbiendo:

Ejemplo 3:date3.jsp

```
<%@page import="java.util.*" %>
<HTML>
<HEAD>
<TITLE>JSP Example</TITLE>
</HEAD>
<BODY BGCOLOR="#ffffcc">
<CENTER>
<H2>Date and Time</H2>
Hoy es: <%= new Date() %>
</CENTER>
</BODY>
</HTML>
```

Como podemos ver, se puede conseguir el mismo resultado usando diferentes etiquetas y técnicas. Hay varios elementos de script JSP. Hay algunas reglas convencionales que nos ayudarán a usar más efectivamente los elementos de Script JSP.

Elemento	Función	Ejemplo
<% ... %>	Manejar declaraciones, expresiones, o cualquier otro tipo de código válido.	
directiva page como en <%@page ... %>	Definir el lenguaje de escript. También puede usarse para especificar sentencias import.	<%@page language="java" import="java.util.*" %>
<%! .... %>	Declarar variables o métodos.	<%! int x = 10; double y = 2.0; %>
<%= ... %>	Definir una expresión y forzar el resultado a un String.	<%= a+b %> o <%= new java.util.Date() %>.
directiva include como en <%@ include ... %>	Insertar el contenido de otro fichero en el fichero JSP principal.	<%@include file="copyright.html" %>

Con la información dada anteriormente nos damos cuenta de cómo podemos realizar una página JSP conectando un código html, con código java. A continuación se darán una serie de pasos sencillos a seguir para realizarlo.

- Primero se debe desarrollar la página, generando así todo el código HTML.
- Se debe generar el código java que servirá de conexión entre nuestra interfaz y la base de datos, es decir, las clases desarrolladas en java deben tener como atributos, todos los datos que se mostrarán en la interfaz. De este modo, cuando se agregue algún dato en la interfaz, tendremos su código java correspondiente que nos ayudará a conectar la información con la base de datos.
- Por último utilizamos las directivas de JSP para ligar los datos de la página HTML con la clase en java desarrollada para dicha interfaz.
- Es recomendable, para facilidad de implementación, se desarrolle una clase en java por cada interfaz en HTML, para que siempre se mantenga una relación de uno a uno, y se eviten confusiones.

- Usando la directiva <%@import="clase.java" %>de JSP importamos en la página HTML la clase java que le corresponde (o se importan las clases java que sean necesarias)
- Y por último con las directivas <%! .... %> se definen variables o métodos que nos ayudarán a relacionar la página con el código HTML.
- Para relacionar el código java con la base de datos, es necesario consultar la Práctica de postgress.

MATERIAL:

- Java, HTML, JSP´s

DESARROLLO:

- Unir las páginas creadas como prototipos al código Java por medio de JSP´s.
- Terminar la conexión con la base de datos, utilizando JSP´s.

PREGUNTAS DE CONTROL:
1.- Define que son las JSP´s
Escribe tus respuestas aquí.
2.- Elemento con el que se añade código Java en una página HTML
Escribe tus respuestas aquí.
3.- ¿Qué es un servidor web?
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- http://glosario.panamacom.com/?id_c=16
- http://programacion.com/java/tutorial/jspyxml/1/

PRÁCTICA # 18

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Implementación.**

INGENIERÍA DE SOFTWARE

PLAN DE PRUEBAS UNITARIAS Y REALIZACIÓN DE PRUEBAS UNITARIAS

OBJETIVOS :

- Conocer los distintos tipos de pruebas que se pueden realizar a un software.
- Contar con los conocimientos suficientes para la realización del plan de pruebas de unidades.

INTRODUCCIÓN :

Metas de las Pruebas.

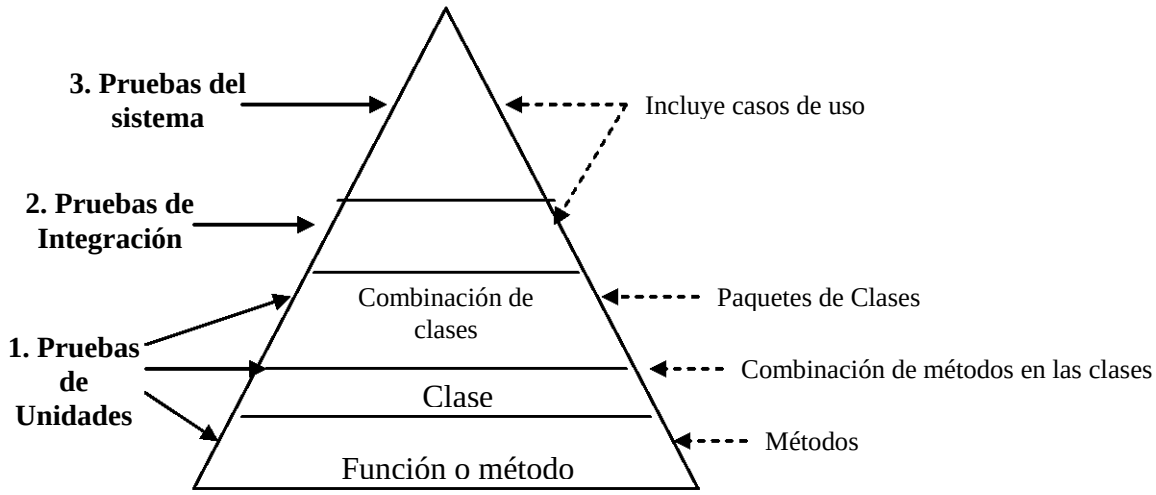
No se puede probar una aplicación para cada posibilidad, porque el número de maneras en que puede ejecutarse un programa de computadora no trivial es ilimitado. Así, cuando realizamos pruebas no podemos demostrar que una aplicación no tiene defectos. La finalidad de realizar pruebas en una aplicación es mostrar la presencia de defectos.

A menudo, hacer pruebas se malinterpreta como “probar para asegurar que es correcto”. Sin embargo, el propósito no es demostrar que una aplicación es satisfactoria, sino determinar con firmeza en qué parte no lo es. Las pruebas de código deben realizarlas personas diferentes a las que lo desarrollaron, ya que cuando un individuo prueba su propio código, tiende a ocultar justo los defectos que debe descubrir.

Las “pruebas de unidades” son el primer tipo de prueba que se aplica. El siguiente nivel consiste en las pruebas de integración.

Significado de “Pruebas de Unidades”

Como se muestra en la siguiente figura, en general, las funciones o métodos son las partes más pequeñas a las que se aplican las pruebas de unidades. La siguiente unidad en tamaño es la clase, en el caso de orientación a objetos. Algunas veces, las combinaciones de clases se consideran “unidades” para los propósitos de las pruebas. **Fig. 1.** Pruebas. Visión global



Los bloques de construcción de software deben ser completamente confiables, y ésta es la meta de las pruebas de unidades.

Tipos de Pruebas

Pruebas de caja negra y caja blanca.

Cuando el interés es saber si una aplicación proporciona la salida adecuada para la entrada dada, es una **prueba de caja negra**, porque no se pone atención al interior de la "caja" (la aplicación). Las pruebas de caja negra pueden ser suficientes si se puede asegurar que agotan todas las combinaciones de entrada.

La meta de las **pruebas de caja blanca** es mostrar las líneas de falla más probables de la aplicación. Para realizar pruebas de caja blanca, primero se desglosa el diseño de la aplicación en busca de trayectorias de control y datos. Después se diseñan datos de pruebas que recorran todas o algunas de estas trayectorias.

Plan de Pruebas de Unidades

La meta es planear y detectar cuantos errores sea posible al nivel más serio posible con los recursos disponibles. Los pasos para el plan de pruebas de unidades se enumeran en la figura 2.

Una manera de ..... planear las pruebas de unidades.

1. Decida la filosofía de las pruebas de unidades
   - ¿Qué unidades deben probarse?
   - ¿Quién probará dichas unidades?
2. Decida qué documentar.
   - La documentación consiste en los datos de prueba, los datos de entrada, el código que ejecuta las pruebas y los datos de salida esperados y obtenidos.
3. Determine el grado de las pruebas de unidades (de antemano)
   - Asigne prioridades para que las pruebas importantes se hagan
   - Debe definirse el alcance de las pruebas con cuidado
4. Decida cómo y dónde obtener los datos para las pruebas
5. Estime los recursos requeridos
6. Registre tiempo, cuente defectos, tipo y fuente

Fig. 2. Una manera de planear las pruebas de unidades

MATERIAL:

- Código de las clases.

DESARROLLO:

- Definir el plan de pruebas unitarias, incluyendo el tipo de pruebas a realizar, (caja negra y/o blanca) con la especificación de las pruebas, y la documentación necesaria.
- Realizar las pruebas específicas en el plan de pruebas unitarias. (Todo lo anterior se realizará en equipo)

PREGUNTAS DE CONTROL:
1.- ¿Cuál es el objetivo de las pruebas unitarias?
Escribe tus respuestas aquí.
2.- ¿Cuáles son los distintos tipos de pruebas unitarias?
Escribe tus respuestas aquí.
3.- Explica de manera informal los datos que debe incluir el Plan de Pruebas unitarias para su proyecto.
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- Watts S. Humphrey. Introduction to the Team Software Process. Addison-Wesley. 1997
- Braude. Ingeniería de Software. Una perspectiva orientada a objetos. AlfaOmega. 2003

Capítulo 8

Práctica para la fase Postmortem

8.1 Práctica para la fase Postmortem

El Postmortem es el último paso en el proceso de TSPi. Aquí se revisa que el equipo de trabajo haya terminado todas las tareas y registrado todos los datos requeridos. Proporciona una manera estructurada de aprendizaje y mejora, porque se evalúa el desempeño personal y del equipo. Cada ciclo de desarrollo de TSPi finaliza con un postmortem. La otra práctica de esta fase se encuentra en la tesis complementaria de Guadalupe Aguilar Morales.

Práctica 21
La finalidad principal de la práctica es lograr que los alumnos puedan determinar, por medio de métricas, el desempeño de cada uno de los integrantes del equipo. Este tipo de medidas los puede ayudar a otorgar roles y repartir actividades de una manera más conveniente para futuros ciclos, por ello es sumamente importante que este tipo de evaluaciones se haga de la manera mas justa posible.

PRÁCTICA # 21

NOMBRE DEL ALUMNO:

INSTRUCTOR:

FECHA:

FASE: **Post Mortem.**

INGENIERÍA DE SOFTWARE

RECOLECCIÓN DE MÉTRICAS DEL PROCESO.

OBJETIVOS :

- Determinar por medio de métricas el desempeño tanto de los integrantes, como del trabajo en equipo.
- Obtener evaluaciones realizadas de manera objetiva con respecto al trabajo realizado en el ciclo y proponer mejoras para el siguiente ciclo.

INTRODUCCIÓN :

En la fase de Postmortem se revisa que el equipo de trabajo haya terminado todas las tareas y registrado todos los datos requeridos. Esta fase proporciona una manera estructurada de aprendizaje y mejora, porque se evalúa el desempeño personal y del equipo. Cada ciclo de desarrollo de TSPi finaliza con un postmortem.
Para el desarrollo de esta fase es necesario que el equipo haya terminado y evaluado el producto, y que los ingenieros hayan reunido todos los datos y completado todas las formas requeridas.

Revisión de Calidad

Se compara el desempeño personal y del equipo de acuerdo al plan de calidad. Se inicia con un análisis sobre los datos referentes a los defectos y se evalúan todos los aspectos relacionados en la generación de un producto de calidad. Si el producto tuvo uno o mas defectos en las pruebas del sistema, se deberá evaluar en que parte el proceso tuvo deficiencias para realizar mejoras en el futuro. Se puede guiar mediante las siguientes preguntas:

- ✓ ¿Cuál fue el resultado de comparar el desempeño actual con el planeado?
- ✓ ¿Qué se puede aprender a partir de esta experiencia?
- ✓ ¿Dónde se pueden aplicar mejoras y cuáles son las razones de ello?

Reporte del ciclo

Se describe lo que se produjo, el proceso utilizado y los roles desempeñados, lo que sí se pudo realizar y lo que no fue conveniente. Se deberá describir el desempeño que cada integrante tuvo respecto al rol que desempeñó, de acuerdo a los indicados por TSPi, así como el rol correspondiente al área de desarrollo. El reporte debe ser breve y objetivo.
Cuántos defectos /KLC
KLC/tiempo de desarrollo
Tiempo por fase/ tiempo total.

Reporte de roles

En este reporte se debe hacer la evaluación sobre el desempeño del equipo de acuerdo a la perspectiva de TSPi que le corresponda a cada rol.

El líder de equipo analizará el desempeño del equipo desde la perspectiva de liderazgo, además tomará en cuenta la manera en la que se establecieron las reuniones de equipo y mencionará cómo poder mejorar esa responsabilidad en el futuro.

El Administrador de Desarrollo debe comparar el contenido del producto con los requerimientos y evaluar la efectividad de la estrategia de desarrollo.

El Administrador de Planeación hará una comparación entre el desarrollo real del equipo y lo que se planeó.

El Administrador de Calidad comparará los datos actuales de calidad con los objetivos de calidad establecidos para describir el desempeño del equipo.

El Administrador de Apoyo describe las facilidades de soporte, el informe del estado de la configuración con el estado de los cambios.

DESARROLLO:

- Realizar la revisión de calidad.
- Generar el reporte del Ciclo.
- Generar el reporte de los roles.

PREGUNTAS DE CONTROL:
1.- ¿Para qué sirve y por qué es importante la fase de Postmortem?
Escribe tus respuestas aquí.
2.- ¿Cuáles son los puntos importantes que deben evaluarse en un proceso de software?
Escribe tus respuestas aquí.

CONCLUSIONES Brinda una opinión de lo que significó para tí el desarrollo de la práctica, además de especificar los conocimientos adquiridos en la realización de la misma.
Escribe tus respuestas aquí.

BIBLIOGRAFÍA :

- Watts S. Humphrey. Introduction to the Team Software Process. Addison-Wesley. 1997

Conclusiones

De acuerdo al objetivo propuesto para este trabajo que era desarrollar las prácticas para la materia de ingeniería de software, a fin de mejorar el conocimiento adquirido por los alumnos en dicha materia, y para incrementar su interés hacia la misma, pudimos observar que por motivo de las prácticas desarrolladas, el trabajo de los alumnos se distinguió del trabajo realizado en semestres anteriores, dado que entregaban documentación de mejor calidad, y se mostraron más interesados en el desarrollo de la materia.

Se definió una plantilla para las prácticas, estableciendo los datos, el contenido, y el orden en que debían aparecer, incorporando puntos como la introducción, los objetivos de la práctica, las preguntas de control, los resultados obtenidos e incluso la bibliografía. Todos estos puntos se incorporaron con la finalidad de dar a los alumnos más información acerca de la finalidad y funcionalidad de las prácticas que realizarían.

Fue importante también definir cuáles serían los temas de las prácticas y qué conformaría cada una de ellas. Se decidió realizar prácticas por cada una de las fases del TSPi, y realizar dos prácticas por semana, de este modo los alumnos estarían con los conocimientos más frescos para cuando tuvieran que entregar su documentación al profesor de la materia. De este modo se generaron en promedio dos prácticas por fase, abarcando así las fases de Principios de TSPi, Planeación, Diseño, Pruebas cuyas prácticas se desarrollaron en la tesis complementaria a esta que fue realizada por Guadalupe Aguilar Morales y las fases de Lanzamiento, Estrategia, Requerimientos, que se desarrollaron en esta tesis, incluyendo también las fases de Implementación, y Post Mortem cuyas prácticas se dividieron en las dos tesis, formando así un total de 22 prácticas, con la finalidad de que todas se desarrollen durante el semestre de la materia de ingeniería de software.

Fue posible verificar el funcionamiento de las prácticas ya que se utilizaron dentro de la materia como material de laboratorio, obteniendo resultados esperados, esto es, se observó una mejora en la entrega de documentos por parte de los alumnos, los cuales tomaban con más interés el desarrollo de la materia y lograban comprender más la funcionalidad de la documentación, tenían ayuda para la realización de diagramas, así como para la definición de requerimientos.

Ahora se pudo notar que los alumnos toman mas en serio la materia, tienen más apoyo y bases para generar documentación, y se esfuerzan un poco mas en realizar trabajos de calidad. Pudieron comprender mejor la finalidad y funcionamiento de la ingeniería de software, y mejoraron su desempeño dentro de la materia, y comprendieron que el hecho de generar tanta documentación tiene una finalidad muy importante, que es la de mejorar la comunicación con el cliente, y reducir el surgimiento de errores a la mitad del desarrollo, o lo que es peor al terminar un desarrollo.

Comprendieron que gracias a la documentación sugerida por TSPi, se pueden evitar muchos contratiempos de malentendidos entre cliente y desarrolladores, e incluso entre los mismos integrantes de un equipo de desarrollo. Pudieron darse cuenta que establecer estándares de documentación, y crear documentación para el desarrollo de un software ayuda a que todos los integrantes de un equipo de software trabajen bajo la misma idea, reduciendo así la posibilidad de confusiones, o de que cada integrante trabaje bajo su propia idea, sin tomar en cuenta las decisiones de los demás.

De este modo se puede asegurar que los alumnos lograron adquirir mayor conocimiento y que serán capaces de enfrentarse a situaciones reales con la seguridad de realizar un buen trabajo.

Bibliografía

[Br] Braude. Ingeniería de software. Una perspectiva orientada a objetos. Alfaomega. 2003

[Br1] Brackett, J. ftp://ftp.sei.cmu.edu/pub/education/cm19.ps 1990

[Ga] Galitz, W. The Essential Guide to User Interface Design. An Introduction to GUI Principles and Techniques. Nueva York John Wiley & Sons. 1996

[Hu] Humphrey, Watts S. Introduction to the Personal Software Process (SEI Series in Software Engineering) Addison-Wesley. 1996

[Hu1] Humphrey, Watts S. Introduction to the Team Software Process. (SEI Series in Software Engineering) Reading MA Addison-Wesley. 1996

[Hu2] http:// www.sei.cmu.edu/publications/articles/sources/practice.preach/index.html

[IS1] La ingeniería de software. El proceso de la ingeniería de software, la vida del software. Disponible en: http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html#IngSoft

[IS2] La ingeniería de software. Introducción, inicios, objetivos de la ingeniería de software. Disponible en: http://www.ilustrados.com/publicaciones/EpypklZplZydAklOcM.php

[IS3] Problemas con la ingeniería de software, ingeniería de software y sistemas. http://dis.unal.edu.co/~fgonza/courses/2003/ingSoft1/CAP2.pdf

[IS4] Introducción a la ingeniería de software. Aplicación del modelado de procesos http://redie.uabc.mx/contenido/vol3no2/contenido-mireles.pdf

[IS5] Página principal del instituto de ingeniería de software (Software Engineering Institute) http://www.sei.cmu.edu/sei-home.html

[IS6] Introducción a la ingeniería de software, inicios, problemática del desarrollo de software de antaño. http://ciberia.ya.com/diegodjm/case/intro.html

[Ja] Jacobson, Ivar. Object Oriented software Engineering: A use case driven approach. Addison Wesley Object Technology series. Addison Wesley 1994

[Ja1] Jacobson, Ivar. Rumbaugh, James y Booch, Grady. The Unified Software Development Process. Addison Wesley Object Technology Series. Addison Wesley 1999

[Ke] Keil, M. Cule, P. Lyytinen, K., y Schmidt, R. A framework for identifying Software Project Risks. Communications of the ACM, vol 41, num 11. 1998

[Mi] A. Milroy y P. Rajah Europe´s Growing IT skills shortage, Special Report compiled for Microsoft, IDC. 2000

[TC] Aguilar Morales, Guadalupe. Proyecto Prácticas para la Ingeniería de Software. 2004

[TSP1] Construyendo equipos de alto rendimiento, usando Team Software ProcessSM (TSPSM) y Personal Software ProcessSM (PSPSM) http://www.sei.cmu.edu/tsp/

[TSP2] Introducción y definición del término de Team Software Process http://www.softwaretechnews.com/stn3-4/teamspi.html

[TSP3] Enseñando Personal SoftwareProcess y Team Software Process. Breve introducción a los dos términos. http://oce.spsu.edu/cseet2002/Anthony-Lattanze.pdf

[TX] Texas 4-H Leaders Handbook Topics. Published by Celina G. Wille, Ph. D. Extension 4-H & Youth Development Specialist. 2401 East Highway 83. Weslaco, Texas 210/968-5581. Técnicas de enseñanza. http://texas4-h.tamu.edu/publications/4H35200.pdf