UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

CENTRO DE CIENCIAS APLICADAS Y DESARROLLO TECNOLÓGICO

DISEÑO Y DESARROLLO DE UNA PLATAFORMA DE EDUCACIÓN A DISTANCIA CENTRADA EN EL USUARIO

TESIS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

YESICA YAHAIRA CARRERA TORRES

DIRECTOR DE TESIS:

DR. FERNANDO GAMBOA RODRÍGUEZ

CIUDAD UNIVERSITARIA, 2007

Agradecimientos

A Dios
Gracias por permitirme llegar a este día y cumplir este sueño que parecía inalcanzable.

A mi madre
No me bastará la vida para agradecerte todo lo que me has dado.
Tu amor, tu tiempo, tu esfuerzo, tus desvelos, gracias por el gran ejemplo, te admiro.
Gracias por nunca dejarme sola. Te amo.

A mi Abue y tía Chonita
Por ser el pilar fundamental en el logro de este sueño, por su infinito amor, por los valores y principios que me inculcaron sin los cuales no sería lo que lo hoy soy. Por su eterna lucha.

A mi hermano
En algún momento tu y yo nos perdimos, hemos cambiado, no me falles. Te quiero y estaremos siempre juntos.

A mi tío Juan y Arturo
Gracias por estar siempre a mi lado, por su inmenso cariño y por no soltar mi mano.

A Abel
Por tus desvelos, por haberte involucrado conmigo en este sueño y tomarlo como tuyo, por tu aliento para seguir adelante, por tu apoyo incondicional y sobre todo por enseñarme que en la vida no hay voz ni sentimiento que valga ante una situación que jamás se ha vivido.

A Gerardo
Qué decir, cómo excluirte, quisiera expresarte todo el sentimiento que me causan estas líneas…., lo resumo a agradecerte todo el tiempo dedicado a este proyecto y al gran impulso que me diste para que esto fuera posible.

A Fernando Gamboa
Por darme la oportunidad de aprender, por darme la mano en la realización de este proyecto, por tu tiempo, por tu paciencia y compresión, por tus enseñanzas.

A Francisco Cabiades
Por tu apoyo incondicional para concluir este proyecto, por tus consejos, por tu amistad.

A mis amigos
Que a lo largo de todo el camino me brindaron su apoyo y amistad y que aún en los momentos más difíciles estuvieron siempre conmigo, gracias Pao, César, Mauricio, Ana, Moni, Jorch, Ivette.

A mi querida Universidad Nacional Autónoma de México
Gracias por haberme permitido ser parte tuya, por darme la formación integral y las herramientas necesarias para ser una mujer competitiva, mi meta, poner muy en alto el nombre de mi institución.

Incontables son las personas que merecen ser nombradas en este proyecto como parte del proceso, puesto que contribuyeron enormemente en mi formación, nombrarlas es un trabajo de grandes proporciones. Quiero pedir perdón a aquellas a quienes no incluyo, en verdad, están en mi corazón.

Si quieres ser sabio, aprende a interrogar razonablemente, a escuchar con atención, a responder serenamente y a callar cuando no tengas nada que decir.

Johann Kaspar Lavater (1741-1801) Filósofo, poeta y teólogo suizo.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

E-Learning: educación a distancia

1. E-Learning: educación a distancia

El objetivo primordial de este proyecto es proporcionar una metodología tecnológica con el uso de una herramienta de educación a distancia, y los métodos que se usan para que la entrega de material educativo sea pertinente a los diferentes casos y necesidades que se presentan, ya sea para la industria o la educación. Para ello se ha seguido una metodología que pretende dar a conocer que la educación a distancia ya esta siendo utilizada mundialmente, por lo cual comenzaremos por explicar todo lo referente a la educación.

1.1. Educación

Entendemos la educación ("educere": extraer) como una ayuda al educando para que éste pueda extraer y desarrollar sus propias capacidades, sus potencialidades humanas y con ellas poder tomar decisiones en su propio beneficio, de manera libre y responsable.

Siguiendo una vieja definición, modificada para adaptarla a los tiempos actuales, educación se define como (educación 2001):

"Un proceso continuo, que interesándose por el desarrollo integral (físico, psíquico y social) de la persona, así como por la protección y mejora de su medio natural, le ayuda en el conocimiento, aceptación y dirección de sí misma para conseguir el desarrollo equilibrado de su personalidad y su incorporación a la vida comunitaria del adulto, facilitándole la capacidad de toma de decisiones de una manera consciente y responsable."

Por supuesto esta educación tenderá a ser más directiva cuanto menor sea la edad del educando y mucho menos, quedando en mera formación y orientación (libre de ser seguida), cuanto mayor sea su edad: La educación tiende a liberalizarse, a la autoeducación y al incremento de la libertad y consecuentemente, al incremento de la responsabilidad de la persona educada. Se educa para la competencia como ciudadanos sociales y solidarios libres y responsables.

Otra definición de educación del diccionario, según Acimed (2003) es:
"La acción o proceso de educar o ser educado."

Es importante precisar que las palabras operativas incluidas en las definiciones arriba mencionadas son acción o proceso; sin embargo, la educación algunas veces se asocia con un lugar, la escuela, y no con el proceso, siendo éste último el punto más importante de la educación a distancia y el cual no perderemos de vista a lo largo del proyecto.

1.2. Diversidad de denominaciones

La diversidad y complejidad de la Educación a Distancia se refleja incluso en la variedad de denominaciones que recibe en publicaciones especializadas, según países y según el sentido último que el autor desea reflejar: (Aratio 1998)

- Correspondence education o correspondence study (educación o estudio por correspondencia). La comunicación postal define el paradigma en el que el docente enseña escribiendo y el alumno aprende leyendo.

- Fernunterricht (instrucción a lo lejos) palabra alemana que enfatiza la separación física de profesor y alumno prácticamente sin posibilidad para la interacción.

- Fernsstudium (aprendizaje a distancia en educación superior). Mantiene el prefijo fern (distancia) que se aplicó a los estudios universitarios que emplearon esta modalidad educativa.

- Open learning (aprender abierto). Con este término se trata de diferenciar los procesos de enseñanza-aprendizaje dentro de un aula de los que ocurren fuera de las aulas, concediendo a los estudiantes mayor autonomía y autodirección en su aprender.

A estas definiciones podríamos agregar los conceptos de autonomía en la elección de objetivos, contenidos y evaluación por parte del alumno, así como la flexibilidad en el tiempo y en espacio.

- Home study (estudio en casa). Mediante este concepto se está indicando que el proceso de enseñanza-aprendizaje se produce en el propio hogar, donde se genera una serie de sentimientos agradables de privacidad y familiaridad, contrapuestos a las fastidiosas experiencias de determinados centros presenciales donde todos conocerán que estamos estudiando y sabrán de nuestras deficiencias y limitaciones.

- Angeleitetes Selbststudium (autoestudio guiado). Estudiar por sí mismos con la esporádica ayuda de un profesor.

- Zaochny (educación a distancia). Forma de enseñar-aprender para los rusos. Su etimología nos lleva a traducirla por enseñanza sin contacto ocular, sin ver al profesor.

- Study whithout leaving production (estudiar sin dejar de producir). Ofrece la posibilidad que tiene el estudiante de mantener su ritmo de producción mientras aprende.

- Guided didactic conversation (conversación didáctica guiada). El diálogo, la comunicación es imprescindible en la educación. También lo es en la enseñanza a distancia, por eso se refuerza con esta frase.

- Two-way communication in distance education (comunicación bidireccional en educación a distancia). Nunca comunicación unidireccional y soledad del estudiante con su material, siempre comunicación en doble sentido que evite ese aislamiento.

- Independent study (estudio independiente).El estudiante determina el cuánto, dónde y cómo realizar su aprendizaje.

- Industrialized form of instruction (forma industrializada de instrucción). Destaca el proceso de planificación previa, de organización, división del trabajo, el creciente uso de equipos técnicos para la producción de materiales y la necesidad de una evaluación más formalizada.

1.3. Educación a distancia

Existen una serie de circunstancias en el mundo de hoy que nos obliga a un replanteamiento radical sobre los desarrollos educativos. Pretender que la formación de los individuos se circunscriba al periodo escolar; es decir, que limite al alumno a que sea sólo estudiante en dichos periodos, es cancelar la actualización profesional y el progreso social. Las estructuras formales de educación no pueden dar respuesta a todas las necesidades de adaptación progresiva de este mundo en constante cambio y con creciente demanda de educación. En efecto, existe cada día una mayor solicitud de todo tipo de actividades no regladas de perfeccionamiento profesional, que impone el progreso de todos los sectores.

Con el desarrollo de las tecnologías de la información y la comunicación (TIC´s) se hizo posible un intenso intercambio de ideas, experiencias y conocimientos entre millones de seres humanos, y con ello se encuentra en la educación a distancia una alternativa para enfrentar los diversos problemas que han mermado el desarrollo de la educación, como lo es los costos que ésta implica.

Un aspecto relacionado con la Educación a Distancia que conviene tener en consideración, es la nueva forma de concebir la Educación a Distancia. Ya no podemos entenderla como aquella dirigida a adultos, quienes de forma individual estudian en su tiempo libre y cuyo modelo representa en esencia una forma de distribución que quiere ser un sustituto de la educación cara a cara. La nueva forma de concebir la educación a distancia viene representada por el aprendizaje abierto, que independientemente de la distancia o de si la enseñanza es presencial, ofrece al estudiante la posibilidad de tomar decisiones sobre su aprendizaje. Estas decisiones afectan todos los aspectos del aprendizaje (Aratio 1998): si se realizará o no; qué aprendizaje (selección de contenido o destreza); cómo (métodos, media, itinerario); dónde aprender (en casa, en un centro específico, en el lugar de trabajo,....); cuándo aprender (comienzo y fin, ritmo); a quién recurrir para solicitar ayuda (tutor, amigos, colegas, profesores, etc..); cómo será la valoración del aprendizaje (y la naturaleza de la retroalimentación proporcionada); aprendizajes posteriores, etc.

Es por ello que el término de “Educación a Distancia” describe todo proceso de enseñanza-aprendizaje en el que alumno y profesor están, en general, separados geográficamente, y se centra en la naturaleza especial del diseño del curso, el aprendizaje y la instrucción bajo estas circunstancias. La educación a distancia presenta dos características (Aratio 1998):

1. La comunicación entre profesores y alumnos se realiza a través de medios impresos y escritos o por medios electrónicos, o más comúnmente, mediante la combinación de estos medios.
2. La segunda característica se desprende de la primera y consiste en un nuevo enfoque de la construcción, en el que el proceso de enseñanza es descompuesto en sus partes constituyentes. Algunas de ellas, o todas, son preparadas lejos del estudiante, siéndole comunicadas mediante las tecnologías de la comunicación, con la posibilidad de interacción entre el estudiante y un instructor también a través de éstas mismas tecnologías. En la educación a distancia los cursos son diseñados para ser distribuidos a mayores audiencias y en áreas geográficas más amplias que la educación convencional.

Hoy, el acceso a la información y a la cultura no reconoce distancia ni fronteras. Los medios de aprendizaje basados en materiales impresos, de laboratorio, audio, video o informático y la emisión de mensajes educativos en sus distintas variantes (correo electrónico, teléfono, radio, televisión, videotexto, etc.), eliminan o reducen sustancialmente los obstáculos de carácter geográfico, económico, laboral, familiar o de índole similar para que el estudiante pueda acceder a la información.

Los sistemas de Educación a Distancia no sólo pretenden la acumulación de conocimientos, sino capacitar al estudiante para "aprender a aprender" y "aprender a hacer" de forma flexible, forjando su autonomía en cuanto a tiempo, estilo, ritmo y método de aprendizaje, permitiéndole la toma de conciencia de sus propias capacidades y posibilidades para su autoformación, por lo cual la característica de flexibilidad se destaca dentro de esta modalidad educativa al poderse llevar a cabo la comunicación en tiempo, forma y ritmo que podrá ser marcado por el propio alumno. En resumen, lo que la educación a distancia pretende es:

- Que el estudiante adquiera actitudes, intereses y valores que le faciliten los mecanismos precisos para regirse a sí mismo, lo que le llevará a responsabilizarse de un aprendizaje permanente.

- Convertirlo en sujeto activo de su formación y al profesor en guía y orientador, tratando de superar las deficiencias del sistema presencial tradicional.

- Posibilitar un aprendizaje que está ligado fundamentalmente a la experiencia y en contacto inmediato con la vida laboral y social.

- Fomentar el logro de una independencia de criterio, capacidad para pensar, trabajar y decidir por sí mismo y satisfacción por el esfuerzo personal.

1.4. Breve apunte histórico de la Educación a Distancia

La educación a distancia no es un concepto que haya surgido intempestivamente, sino el resultado de un proceso que inició hace muchos años con la enseñanza por correspondencia. Para conocer un poco más acerca de la historia de la Educación a Distancia se muestra la tabla 1 **(**Aratio**,** 1998 **:**28-31**)** :

Tabla 1 Historia de la Educación a Distancia. Fechas más destacadas.

HISTORIA DE LA EDUCACIÓN A DISTANCIA FECHAS MÁS DESTACADAS	
1728	El 20 de marzo aparece un anuncio en la Gaceta de Boston ofreciendo material de enseñanza y tutorías por correspondencia.
1833	Otro anuncio similar al anterior, encontrado en el número 30 del periódico sueco Lunds Weckoblad, avisaba a quienes por correspondencia cursaban <<Composición>>, que durante el mes de agosto cambiaba la dirección para los envíos postales.
1843	Se fundó la Phonographic Correspondence Society que se encargaba de corregir las tarjetas con los ejercicios de taquigrafía anteriormente aludidos.
1856	Charles Toussain y Gustav Laugenschied en Berlín, fueron patrocinados por la Sociedad de Lenguas Modernas para enseñar francés por correspondencia. Fue quizás la primera institución verdaderamente de enseñanza por correspondencia.
1873	Aparece en Boston la Sociedad para el fomento del estudio en el hogar.
1883	Inicia en Ithaca - Estado de Nueva York - su actividad la Universidad por Correspondencia.
1891	El rector de la Universidad de Chicago, W. Rainey Harper fundó un departamento de enseñanza por correspondencia. En la Universidad de Wisconsin, los profesores del Colegio de Agricultura intercambiaron cartas con estudiantes que no podían abandonar su trabajo para volver a las clases en el campus.
1891	Se crea en Francia el Centro Ecole Chez Soi.
1891	En Estados Unidos nacen las escuelas internacionales por correspondencia.
1894	El Rutinsches Fernlehrinstitut de Berlín, organizó cursos por correspondencia para la obtención del Abitur (previo al ingreso en la Universidad).
1897	En USA se funda la Escuela americana.

1903	Julio Cervera Baviera funda en Valencia, España, la Escuela Libre de Ingenieros.

1905	Las escuelas Calvert de Baltimore crean un Departamento de Formación en el Hogar que acoge a niños de escuelas primarias que estudiaban en casa bajo la tutela de sus padres.
1910	En Victoria, Australia, los profesores rurales de primaria comenzaron a recibir temas de educación secundaria por correo.
1911	En Australia, y con la intención de aminorar el problema de las enormes distancias, a través de la Universidad de Queensland, se comenzó esta experiencia.
1914	En Noruega se funda la Norst Correspondanseskole y en Alemania la Fernschule Jena.
1920	En la antigua URSS se implantó también éste sistema de correspondencia.
1922	Inicia sus actividades la New Zeland Correspondence School con la inicial atención de atender a niños aislados o con dificultades para acudir a las aulas convencionales. A partir de 1928 atienden también a alumnos de secundaria.
1938	En la ciudad de Victoria (Canadá) se celebró la Primera Conferencia Internacional sobre la Educación por Correspondencia.
1939	Nace el Centro Nacional de Enseñanza a Distancia en Francia que al principio atendió, por correspondencia, a los niños que escaparon de la guerra. Es un centro público dependiente del Ministerio francés de educación nacional.
1940	En la década de los cuarenta diversos países europeos del centro y este iniciaron esta modalidad de estudios. Ya por estos años los avances técnicos posibilitaron otras perspectivas que las de enseñanza meramente por correspondencia.
1946	La Universidad de Sudáfrica (UNISA) -convención hasta entonces- comenzó a enseñar también por correspondencia.
1947	A través de Radio Sorbonne se transmitieron clases magistrales en casi todas las materias literarias de la facultad de letras y ciencias humanas de París.
1951	La Universidad de Sudáfrica se dedica exclusivamente a impartir cursos a distancia.
1951	Nacen las Escuela Australianas del Aire que posibilitan que niños geográficamente aislados participen de la enseñanza diaria con un profesor y otros niños a través de la radio.
1960	Se funda el Berijing Televisión College en China, que se cerró como el resto de la educación postsecundaria durante la Revolución Cultural.
1962	Se inicia en España una experiencia de Bachillerato radiofónico.
1962	La Universidad de Delhi abrió un Departamento de Estudios por Correspondencia, como ensayo para atender a los estudiantes que de otro modo no podrían recibir una enseñanza universitaria
1963	Se crea en España el Centro Nacional de Enseñanza Media por Radio y Televisión que sustituyó al Bachillerato radiofónico creado el año anterior.
1963	Se inicia en Francia una enseñanza universitaria por radio en cinco Facultades de Letras (París, Bordeaux, Lille, Nancy, Strasbourg) y en la Facultad de Derecho de París, para los estudiantes de primer curso.
1963	Dos instituciones neocelandesas se unen (Victoria University of Welington y Massey Agrucyultural College) y forman la Massey University Centre for University Extramural Studies de Nueva Zelanda.
1963	El Centro Nacional de Enseñanza Media por Radio y Televisión de España, se transforma en Instituto Nacional de Enseñanza de Media a Distancia (INEMAD).
1969	Se crea la Open University Británica, institución verdaderamente pionera y señera de lo que hoy se entiende como educación superior a distancia. Inició sus cursos en 1971. A partir de esa fecha la expansión de esta modalidad ha sido inusitada.
1969	La Universidad Nacional Autónoma de México establece el Sistema de Universidad Abierta y es la primera en América Latina.
1972 y 1979	En Australia el número de instituciones a distancia pasó de 15 a 48. Sin embargo es en los países industrializados o desarrollados como Canadá, Inglaterra, Alemania, los Estados Unidos y Japón donde se le dio más valor a esta modalidad.

1979	En México la Universidad Pedagógica Nacional es la primera institución de educación pública Superior en México que cubrió el país con esta modalidad, al establecer unidades de educación a distancia en todos los estados de la República.
1989	La UNAM emite el curso "Formación de Profesores" a través de la señal de televisión vía satélite. El curso fue organizado por la Facultad de Contaduría y Administración.
1990	A partir de los noventas se inició el "Furor por el E-learning" gracias al desarrollo de Internet que al extender y facilitar el acceso a la comunicación le da a la educación a distancia una plataforma excelente para la comunicación educativa, el acceso a la información y el fortalecimiento de los procesos de aprendizaje. Impulso que se renueva permanentemente con la aparición de innovaciones tecnológicas que amplían y diversifican las posibilidades educativas.

A partir de la Open University Británica (citada en la tabla anterior) comienzan a surgir otros programas de instituciones de educación superior a distancia en todo el mundo usando medios didácticos muy semejantes. Las nuevas opciones tecnológicas aplicadas a la educación como la informática y las telecomunicaciones han contribuido al desarrollo de esta modalidad educativa hacia lo que hoy se conoce como "*educación en línea"*. Algunos ejemplos de este modelo de educación es: la Universidad de Gobernadores de Occidente y el Campus Mundial Virtual de la Universidad Estatal de Pensylvania, ambas instituciones iniciándose en verano de 1998 en los Estados Unidos y la Confederación de Instituciones de Aprendizaje Abierto de Sudáfrica (COLISA). Los recursos tecnológicos utilizados en estas instituciones (texto, vídeo, audio, fotografías digitalizadas, revistas electrónicas, bibliotecas virtuales, enciclopedias electrónicas, etc.) posibilitan, mediante la metodología adecuada, suplir e incluso superar en algunos aspectos, a la educación presencial.

No debemos perder de vista el desarrollo desigual de estos procesos, hay instituciones que siguen con la marca y prácticas del momento histórico en que nacieron y los procesos tradicionales suelen traslaparse con las nuevas propuestas. Las prácticas socio educativas suelen tener en su interior procesos simultáneos y desiguales, e incluso recurrentes y regresivos. De manera que no son extrañas las prácticas docentes más tradicionales montadas en las más avanzadas tecnologías o grandes innovaciones pedagógicas que utilizan tecnologías tradicionales. Así mismo podemos decir que los nuevos medios no llegaron a desplazar a los existentes, sino que vinieron a diversificar el menú de posibilidades, en una permanente búsqueda de las mejores opciones para la educación a distancia.

1.5. Límites de la Educación a Distancia

A través de los conceptos ya expuestos podemos detectar ciertas inconvenientes de la educación a distancia, aquí se enuncian aquellos que son sobresalientes:

- Pocas ocasiones pueden presentarse para la realización de actividades culturales, deportivas, de movilización a nivel comunitario o grupo, etc. La acción educativa presencial debe aminorar esta dificultad.

- Los objetivos del ámbito afectivo, formación y cambio de actitudes de los alumnos, así como los del área psicomotriz que no atiendan a capacidades que se expresen por escrito, se suelen lograr de manera más efectiva mediante los contactos personales.

- Por la razón anterior se hace necesaria una rigurosa planificación muy a largo plazo, con las desventajas que ello podría ocasionar, aunque con la ventaja que supone un repensar y reflexionar a muy largo plazo.

- Aunque cada vez menos, aún hay instituciones y académicos que dudan de la capacidad de los sistemas de enseñanza a distancia para producir algo más que no sea instrucción o transferencia de contenidos.
- El peligro de la homogeneidad de los materiales – todos aprenden lo mismo – por el único prototipo de paquete instruccional, obliga a elaborar materiales muy abiertos que den pie a la espontaneidad, creatividad e ideas del alumno.

- La ambición de pretender llegar a todos provoca la realidad de los masivos abandonos, deserciones o fracasos. Es cierto que generalmente se abandona más en la enseñanza a distancia que en la presencial, aunque debería distinguirse entre el abandono real y el abandono sin comenzar de aquellos que ni siquiera realizaron una sola prueba de evaluación.

- Aunque los costos corrientes son más bajos que en la enseñanza a distancia hay que considerar los altos gastos que se precisan para la inversión inicial.

- Falta de programas de apoyo técnico disponible y conveniente puede resultar en frustración por tanto de los docentes como de los estudiantes.

- Necesidad de evaluación continua para informar la toma de decisiones en cuanto a los diseños, técnicas y métodos de la educación a distancia.

- Reconocimiento de investigación de enseñanza en línea como un objeto legítimo para la promoción y la permanencia.

- Mantenimiento de calidad a la vez de obtener economías de escala.

A todos estos inconvenientes y a otros que pudiera encontrar el profesor-alumno, estaríamos en condiciones de brindar argumentos, con la debida amplitud, en función de las características, nivel e índole del curso en cuestión y de los recursos o medios materiales y humanos puestos a disposición de los alumnos de manera que todas estas limitaciones se vayan resolviendo.

1.6. Retos para el E-learning

Aunque la experiencia sea aún limitada, el uso y las aplicaciones de las tecnologías de la información y la comunicación en las aulas han progresado rápidamente en los últimos años. No cabe duda de que la amplitud de miras, la capacidad de liderazgo y una gestión comprometida son requisitos indispensables para llevar a buen término este proceso. Los resultados de diversas investigaciones ponen de manifiesto los efectos que, por lo que respecta a la motivación y el rendimiento, tienen en los jóvenes el uso de métodos de enseñanza basados en las tecnologías de la información y la comunicación. Un informe elaborado en el Reino Unido en noviembre de 2000 (Presidency 2001) destacó los buenos resultados cosechados por las escuelas que poseen instalaciones informáticas apropiadas e imparten una enseñanza tecnológica de alta calidad. Una buena gestión de la tecnología potencía la motivación, la adquisición de conocimientos y la eficacia de los alumnos en la realización de las tareas escolares.

Gracias a los nuevos métodos de aprendizaje que hacen posible los recursos en línea, los estudiantes pueden desarrollar nuevas capacidades en materia de organización de datos, evaluación de materiales originales, presentación visual y trabajo en equipo, al tiempo que se potencían otras competencias nada desdeñables, como el pensamiento crítico. Pero no todo iba a ser positivo: Internet

ofrece acceso inmediato no sólo a recursos educativos de gran valor sino también a materiales social y moralmente execrables. La capacidad de reflexionar críticamente, de evaluar la información disponible y de desechar los contenidos nocivos son competencias cruciales que todos los estudiantes deben adquirir. Se precisan igualmente medidas de protección para los estudiantes más jóvenes, así como el desarrollo de actitudes responsables (entre otras, de carácter técnico) hacia el acceso a Internet de los estudiantes de más edad.

Es preciso encontrar la forma más apropiada de dotar al personal docente con la confianza y las competencias que les permitan aprovechar las posibilidades que ofrecen las tecnologías de la información y la comunicación en la enseñanza. Es igualmente indispensable definir los conocimientos en TIC que deben poseer los profesores, así como poner a punto mecanismos que les permitan actualizar sus niveles de formación. Tanto a escala nacional como internacional pueden tomarse medidas para fomentar y compartir contenidos educativos de alta calidad y experiencias pedagógicas modélicas. Es igualmente importante intercambiar investigaciones e informaciones en relación con los métodos más adecuados para ofrecer a los estudiantes el apoyo y la motivación necesarios para ver coronados con éxito sus esfuerzos. El acceso a portales Web internacionales puede ofrecer tanto a estudiantes como a profesionales de la enseñanza un foro apropiado para compartir, en beneficio mutuo, sus puntos de vista y sus opiniones con una comunidad más amplia.

Este capítulo denota la realidad de la educación, que hoy en día es insuficiente para atender las demandas de formación. De ahí surge el sistema no formal y la educación no presencial, la educación a distancia, es por ello que esta tesis toma para su desarrollo, los avances tecnológicos que se ponen a disposición de la educación.

No es fácil encontrar una definición de educación-enseñanza a distancia más o menos consensuada, como pudimos observar existe una diversidad de denominaciones. Tampoco significa lo mismo enseñanza abierta y enseñanza a distancia para algunos autores, sin embargo estos dos términos suelen utilizarse como sinónimos, es por ello que aquí entenderemos a la educación a distancia como un sistema tecnológico de comunicación direccional, que puede ser masivo y que sustituye la interacción personal en el aula de profesor y alumno como medio preferente de la enseñanza, por la acción sistemática y conjunta de diversos recursos didácticos y el apoyo de una organización y tutoría, que propicien el aprendizaje independiente y flexible de los estudiantes.

La amplitud de horizontes y, especialmente, el diálogo internacional que facilitan las modernas tecnologías de la información y la comunicación son factores que contribuyen a promover una actitud constructiva y a disipar actitudes negativas y estereotipadas, reforzando así la responsabilidad social y contrarrestando la «alienación» de los jóvenes con respecto al proceso democrático. Una actitud abierta y el contacto con personas de otros países y otras culturas constituyen asimismo un medio eficaz para promover la tolerancia hacia lo desconocido y lo diverso. Las TIC pueden ser, en este sentido, una herramienta inestimable para combatir el racismo y la xenofobia.

1.7. Bibliografía

Acimed, R. C. d. (2003). Revista Cubana de Acimed. **2003**.

Aratio, C. L. G. (1998). La educación a distancia y la UNED. Madrid.

Asociación Mexicana en Dirección de Recursos Humanos, A. C. (19 Abril 2004). E-LEARNING EN MÉXICO. **Abril**.

Marcelo, C. and D. Puente (2002). E-learning: Teleinformación, desarrollo y evaluación de la formación. Barcelona, Ediciones Gestión 2000, S.A.

Presidency, T. S. (2001). El aprendizaje electrónico permanente. Las TIC en la escuela -Retos y Oportunidades, Riga, Letonia.

Salinas, J. (2004). EDUCACION A DISTANCIA BASADA EN SATELITES: EXPERENCIAS Y PERSPECTIVAS, Universitat de les Illes Balears. **2004**.

CAPÍTULO 2

Ingeniería de Software

2. Ingeniería de Software

La Ingeniería de Software es una disciplina o área de la informática o Ciencias de la Computación, que ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelve problemas de todo tipo.

Para poder comprender lo que es el software (y consecuentemente la ingeniería del software), es importante examinar las características del software que lo diferencian de otras cosas que se pueden construir. Cuando se construye hardware, el proceso creativo humano (análisis, diseño, construcción, prueba) se traduce finalmente en una forma física. Si construimos una nueva computadora, nuestro boceto inicial, diagramas formales de diseño y prototipo de prueba, evolucionan hacia un producto físico (tarjetas de circuitos eléctricos, fuentes de potencia, etc.).

El software es un elemento del sistema que es lógico, en lugar de físico. Por tanto el software tiene unas características considerablemente distintas a las del hardware (Pressman 1998) :

1. El software se desarrolla, no se fabrica en un sentido clásico. Aunque existen similitudes entre el desarrollo de software y la construcción de hardware, ambas actividades son fundamentalmente diferentes. En ambas actividades la buena calidad se adquiere mediante un buen diseño, pero la fase de construcción del hardware puede introducir problemas de calidad que no existen (o que son fácilmente corregibles) en el software. Ambas actividades dependen de las personas, pero la relación entre las personas dedicadas y el trabajo realizado es completamente diferente para el software. Ambas actividades requieren la construcción de un producto, pero los métodos son diferentes.

Los costos de software se encuentran en la ingeniería. Esto significa que los proyectos de software no se pueden gestionar como si fueran proyectos de fabricación.

2. El software no se estropea. El hardware exhibe relativamente muchos fallos al principio de su vida (estos fallos son atribuibles normalmente a defectos del diseño o de la fabricación); una vez corregidos los defectos, la tasa de fallos cae hasta un nivel estacionario donde permanece durante un cierto período de tiempo. Sin embargo, conforme pasa el tiempo, los fallos vuelven a presentarse a medida que los componentes del hardware sufren efectos acumulativos del uso, la suciedad, la vibración, los malos tratos, las temperaturas extremas y muchos otros males externos, sencillamente el hardware comienza a estropearse.

El software no es susceptible a los males del entorno que hacen el hardware se estropee. Los defectos no detectados cuando el software es entregado harán que falle el programa durante las primeras etapas de su vida. Sin embargo, una vez que se corrigen, el sofftware se estabiliza, es claro que el software no se estropea. Pero se deteriora.

Esto parece una contradicción, durante su vida el software sufre cambios (mantenimientos). Conforme se hacen los cambios es probable que se introduzcan nuevos defectos. Antes de que el software vuelva a estabilizarse, se vuelve a requerir otro cambio y eso hace que se vaya deteriorando debido a dichos cambios.

Otro aspecto de ese deterioro ilustra la diferencia entre el hardware y software. Cuando un componente de hardware se estropea, se sustituye por una nueva pieza. No hay piezas de repuesto para el software. Cada fallo en el software indica un error en el diseño o en el proceso mediente el que se tradujo el diseño a código. Por tanto, el mantenimiento del software tiene una complejidad considerablemente mayor que la del mantenimiento del hardware.

3. La mayoría del software se construye a medida, en vez de ensamblar componentes existentes. Considerando la forma en que se diseña y construye el hardware. El ingeniero de diseño construye un sencillo diseño de circuitería digital, hace algún análisis fundamental para asegurar que se realiza la función adecuada y va al catálogo de ventas de componentes digitales existentes. Cada circuito integrado tiene un número de identificación, una función definida y válida, una interfaz bien definida y un conjunto estándar de criterios de integración. Después de seleccionar cada componente, puede solicitarse la compra.

Por desgracia, los diseñadores de software no disponen de esa comodidad que se acaba de describir. Con unas pocas excepciones, no existen catálogos de componentes de software. Se puede comprar software ya desarrollado, pero sólo como una unidad completa, no como componentes que puedan reensamblarse en nuevos programas.

El software se ha convertido en el elemento clave de la evolución de los sistemas y productos informáticos. En las pasadas cuatro décadas, el software ha pasado de ser una resolución de problemas especializada y una herramienta de análisis de información, a ser una industria por sí misma. El software se compone de programas, datos y documentos. Cada uno de estos elementos componen una configuración que se crea como parte del proceso de la ingeniería del software. El intento de la ingeniería del software es proporcionar un marco de trabajo para construir software con mayor calidad.

El proceso del software ha sido el foco de atención de la última década. Un proceso de software define el enfoque que se toma cuando el software es tratado por la ingeniería; pero la tecnología del software también acompaña a las tecnologías que pueblan el proceso (métodos técnicos y herramientas automatizadas).

2.1. Proceso, métodos y herramientas

La ingeniería del software es una tecnología multicapa (Figura 1). Cualquier enfoque de ingeniería (incluida ingeniería de software) debe descansar sobre un empeño de organización de calidad. La gestión total de calidad y las filosofías similares fomentan una cultura continua de mejoras de procesos, y esta cultura la conduce últimamente al desarrollo de enfoques cada vez más robustos para la ingeniería del software. Los cimientos que son la base de la ingeniería del software están orientados hacia la calidad.

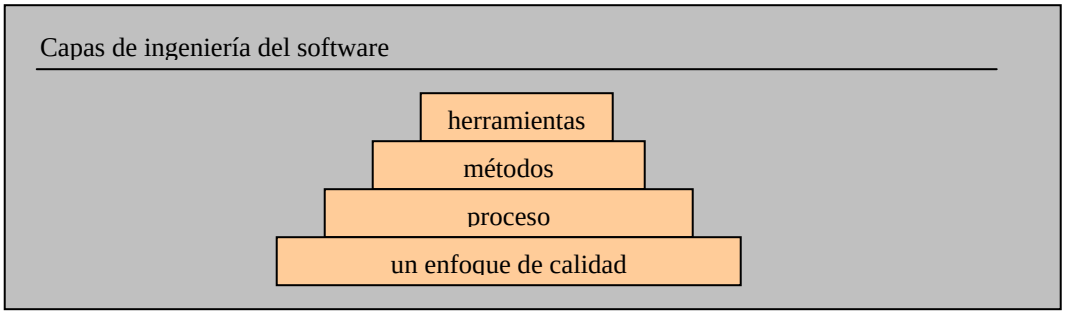

Figura 1 Capas de Ingeniería del Software (Presuman 1998)

El fundamento de la ingeniería del software es la capa de proceso. El proceso de la ingeniería del software es la unión que mantiene juntas las capas de tecnología y que permite un desarrollo racional y oportuno de la ingeniería del software. El proceso define un marco de trabajo para un conjunto de áreas clave de proceso (ACPs) que se debe establecer para la entrega efectiva de la tecnología de la ingeniería

de software. Las áreas clave del proceso forman la base del control de gestión de proyectos del software y establecen el contexto en el que se aplican los métodos técnicos, se producen resultados del trabajo (modelos, documentos, datos, informes, formularios, etc.), se establecen hitos, se asegura la calidad y el cambio se gestiona adecuadamente.

Los *métodos* de la ingeniería del software indican cómo construir técnicamente el software. Los métodos abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Los métodos de la ingeniería del software dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.

Las *herramientas* de la ingeniería del software proporcionan un soporte automático o semi-automático para el proceso y para los métodos. Cuando se integran herramientas para que la información creada por una herramienta la pueda utilizar otra, se establece un sistema de soporte para el desarrollo del software llamada ingeniería del software asistida por computadora (computer-aided software engineering CASE). CASE combina software, hardware y una base de datos de ingeniería del software (un depósito que contiene información importante sobre el análisis, el diseño, la construcción de programas y pruebas) para crear un entorno de ingeniería del software que sea análogo a CAD/CAE (computer-aided design/engineering) (diseño/ingeniería asistida por computadora) para el hardware.

2.1.1. Una visión general de la ingeniería del Software

Para resolver los problemas reales de una industria, un ingeniero de software o un equipo de ingenieros deben incorporar una estrategia de desarrollo que acompañe al proceso, métodos, capas de herramientas y fases genéricas las cuales para Pressman (Pressman 1998) se dividen en tres:

- **Fase de definición**

Se centra sobre el qué. Es decir, durante la definición, el que desarrolla el software intenta identificar qué información ha de ser procesada, qué función y rendimiento se desea, qué comportamiento del sistema, qué interfaces van a ser establecidas, qué restricciones de diseño existen, y qué criterios de validación se necesitan para definir un sistema correcto.

- **Fase de desarrollo**

Se centra en el cómo. Es decir, durante el desarrollo se intenta definir cómo se han de diseñar las estructuras de datos, cómo debe implementarse la función como una arquitectura del software, cómo han de implementarse detalles como procedimiento, cómo han de caracterizarse las interfaces, cómo ha de traducirse el diseño en un lenguaje de programación y cómo han de realizarse las diferentes pruebas. Los métodos aplicados durante la fase de desarrollo varían, aunque tres tareas específicas técnicas siempre aparecen: diseño del software, generación de código y prueba del software.

- **Fase de Mantenimiento**

Se centra en el cambio que va asociado a la corrección de errores, a las adaptaciones requeridas a medida que evoluciona el entorno de software y a cambios debidos a las mejoras producidas por los requisitos cambiantes del cliente. La fase de mantenimiento vuelve a aplicar los pasos de las fases de definición y de desarrollo, pero en el contexto del software ya existente. Durante la fase de mantenimiento se encuentran cuatro tipos de cambios:

1. **Corrección.** El mantenimiento correctivo modifica el software para corregir los defectos.

2. **Adaptación** El mantenimiento para adaptación de software produce modificaciones éste para acomodarlo a los cambios de su entorno externo.

3. **Mejora.** El mantenimiento perfectivo lleva al software más allá de sus requisitos funcionales originales.

4. **Prevención.** El software se deteriora debido al cambio, y por esto el mantenimiento preventivo también llamado reingeniería de software, se debe conducir para permitir que el software sirva para las necesidades de los usuarios finales. En esencia, el mantenimiento preventivo hace cambios en programas de computadora a fin de que se puedan corregir, adaptar y mejorar más fácilmente.

2.2. Procesos del Software

Un proceso de software se puede caracterizar como se muestra en la figura 2.2. Se establece un marco común del proceso, definiendo un pequeño número de actividades del marco de trabajo que son aplicables a todos los proyectos del software, con independencia de su tamaño o complejidad. Un conjunto de tareas –cada uno es una colección de tareas de ingeniería del software, hitos del proyecto, posibles entregas y productos de trabajo del software, y puntos de garantía de calidad- que permiten que las actividades del marco de trabajo se adapten a las características del proyecto de software y a los requisitos del equipo del proyecto. Finalmente, las actividades de protección, tales como garantía de calidad de software, gestión de configuración del software y medición, abarcan el modelo del proceso. Las actividades de protección son independientes de cualquier actividad del marco de trabajo y aparecen durante todo el proceso.

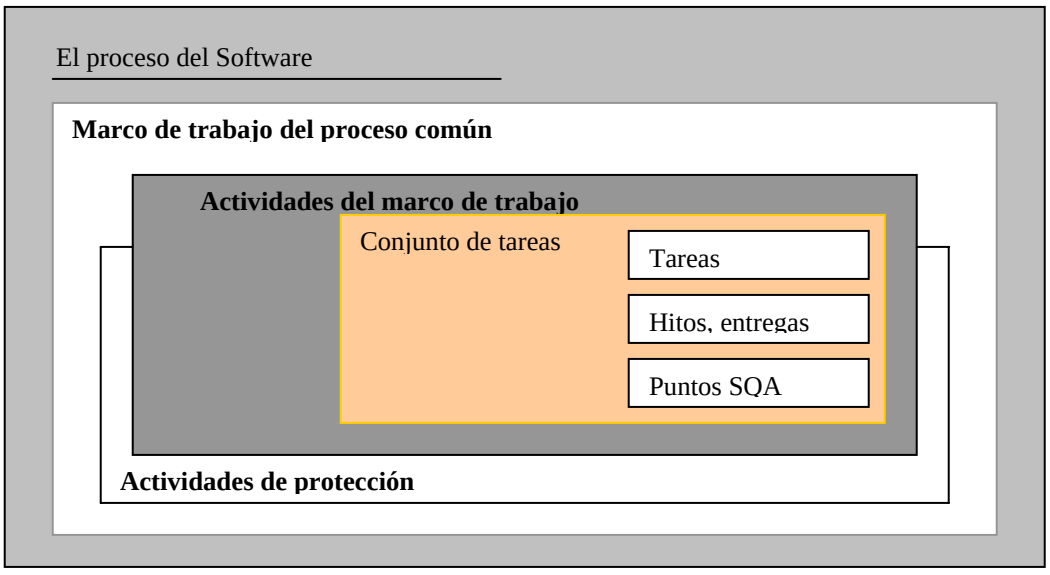

Figura 2 El Proceso del Software (Pressman 1998)

2.3. Modelos de proceso del Software

Para resolver los problemas reales de una industria, un ingeniero del software o un equipo de ingenieros deben incorporar una estrategia de desarrollo que acompañe al proceso, métodos, capas de herramientas y las fases genéricas ya mencionadas. Esta estrategia a menudo se llama *modelo de proceso o paradigma de ingeniería del software.* Se selecciona un modelo de proceso para la ingeniería del software según la naturaleza del proyecto y de la aplicación, los métodos y las herramientas a utilizarse, y los controles y entregas que se requieren.

Todo el desarrollo del software se puede caracterizar como un ciclo de resolución de problemas en el que se encuentran cuatro etapas distintas: *status quo*, definición de problemas, desarrollo técnico e integración de soluciones, como lo podemos observar en la figura 3.

Status Quo representa el estado actual de sucesos; la definición del problema identifica el problema específico a resolverse; el desarrollo técnico resuelve el problema a través de la aplicación de alguna tecnología, y la integración de soluciones ofrece los resultados esperados.

El ciclo de resolución de problemas se aplica al trabajo de ingeniería de software a diferentes niveles de resolución. Se puede utilizar en el nivel macro cuando se tiene en consideración la aplicación entera; en un nivel medio cuando se están considerando los componentes del programa, e incluso a nivel de código. Por consiguiente, se puede utilizar una representación fractal (los fractales se proponen originalmente para representaciones geométricas. Se define un patrón y se aplica recursivamente a escalas más pequeñas en sucesión; los patrones se dividen a su vez en patrones) para proporcionar una visión idealizada del proceso. Todas las etapas –status quo, definición de problemas, desarrollo técnico e integración de soluciones- coexisten simultáneamente en algun nivel de detalle.

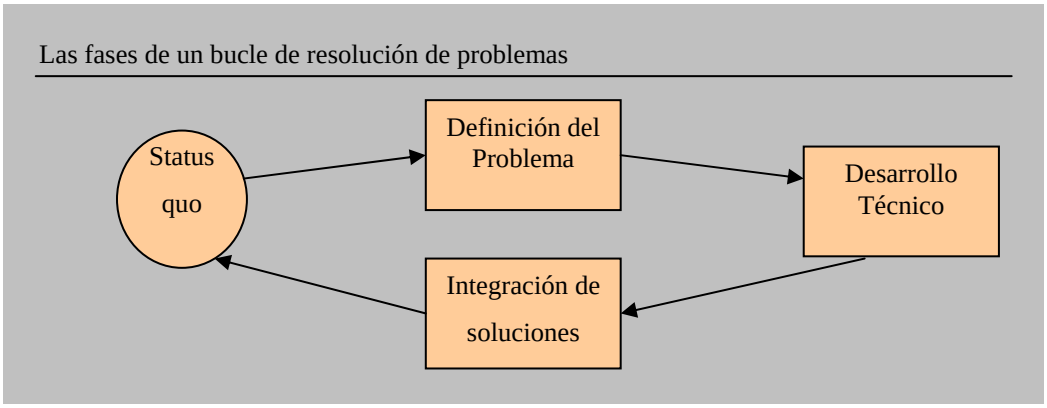

Figura 3 Las fases de un bucle de resolución de problemas (Pressman 1998)

Raccoon (Raccoon 1995) sugiere un “Modelo del caos” que describa el desarrollo del software como una extensión desde el usuario hasta el desarrollador y la tecnología. Conforme progresa el trabajo hacia un sistema completo, las etapas descritas anteriormente se aplican recursivamente a las necesidades del usuario y a la especificación técnica del desarrollo del software.

Acontinuación se presentarán los diferentes modelos de procesos para la ingeniería de software. Cada uno representa un intento de ordenar una actividad inherentemente caótica. Es importante recordar que cada uno de los modelos se han caracterizado de forma que ayuden al control y a la coordinación de un proyecto de software real.

2.4. El modelo lineal secuencial

El modelo secuencial, llamado algunas veces “Ciclo de vida básico” o “Modelo en Cascada”, sugiere un modelo sistemático y secuencial del desarrollo de software que comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento, como se muestra en la figura 4:

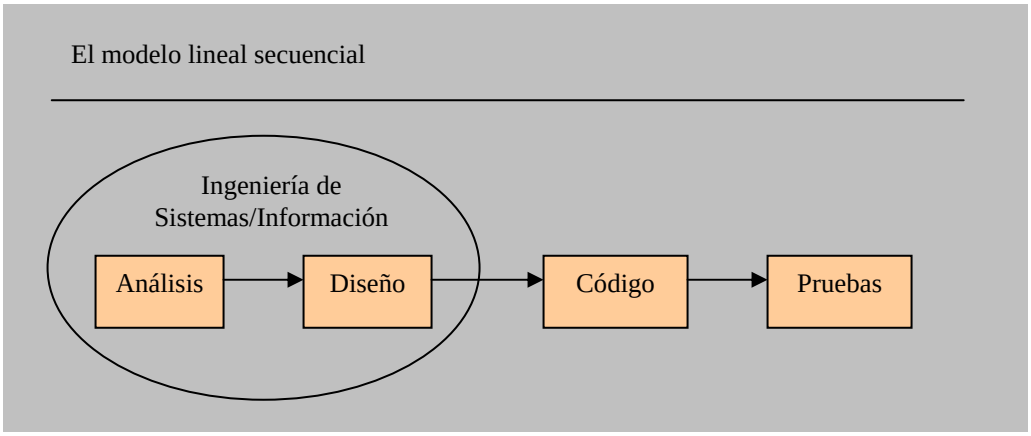

Figura 4 El modelo lineal secuencial (Pressman 1998)

El modelo lineal secuencial acompaña a las siguientes actividades:

- **Ingeniería y modelado de Sistemas/Información**

  Como el software siempre forma parte de un sistema más grande (o empresa), el trabajo comienza estableciendo requisitos de todos los elementos del sistema. Esta visión del sistema es esencial cuando el software se debe interconectar con otros elementos como hardware, personas y bases de datos. La ingeniería y el análisis de sistemas acompaña a los requisitos que se recogen en el nivel del sistema con una pequeña parte de análisis y de diseño, ya que los requisitos son los que dan forma al sistema. La ingeniería de información acompaña a los requisitos que se recogen en el nivel estratégico de empresa y en el nivel del área de negocio.

- **Análisis de los requisitos del software**

  El proceso de reunión de requisitos se intensifica y se centra especialmente en el software. Para comprender la naturaleza de el (los) programa(s) a construirse, el ingeniero (analista) del software debe comprender el dominio de información del software, así como la función requerida, comportamiento, rendimiento, e interconexión. El cliente documenta y repasa los requisitos del sistema y del software.

- **Diseño**

  El diseño del software consta de una serie de pasos, que se centra en cuatro atributos distintos de un programa: estructura de datos, arquitectura del software, representaciones de interfaz y detalle procedimental (algoritmo). El proceso de diseño traduce requisitos en una representación del software que se pueda evaluar antes de que comience la generación del código. Al igual que los requisitos, el diseño se documenta y se hace parte de la configuración del software.

- **Generación de código**

  El diseño se debe traducir en una forma legible por la máquina. El paso de la generación de código lleva a cabo esta tarea. Si se lleva a cabo el diseño de una forma detallada y formal, la generación de código se realiza mecánicamente.

- **Pruebas**

  Una vez que se ha generado código, comienzan las pruebas del programa. El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en los procesos externos funcionales, es decir, la realización de las pruebas para la detección de errores y el sentirse seguro de que la entrada definida produzca resultados reales de acuerdo con los resultados requeridos.

- **Mantenimiento**

  Indudablemente después de haber hecho entrega al cliente del software, este sufrirá cambios, ya sea porque se han encontrado errores o simplemente porque el cliente requiere mejoras funcionales o de rendimiento. El mantenimiento vuelve a aplicar cada una de las fases precedentes a un programa ya existente y no a uno nuevo.

El modelo lineal secuencial es el paradigma más antiguo y más extensamente utilizado en la ingeniería de software. Sin embargo, existen críticas acerca del paradigma que han puesto en duda su eficacia y han logrado detectar los siguientes errores:

1. Los proyectos reales raras veces siguen el modelo lineal secuencial que propone el modelo. Aunque el modelo lineal puede establecer interacción, lo hace indirectamente. Como resultado, los cambios pueden causar confusión cuando el equipo de proyecto comienza.

2. A menudo es difícil que el cliente exponga explícitamente todos los requisitos. El modelo lo requiere y tiene dificultades a la hora de manejar la incertidumbre natural al comienzo de muchos proyectos.

3. El cliente debe tener paciencia. Una versión de trabajo del (los) programa(s) no estará disponible hasta que el proyecto esté avanzado. Un error puede ser eventualmente desastroso si se detecta hasta que se revisa el programa.

4. Los responsables del desarrollo de software siempre se retrasan innecesariamente, ya que la naturaleza lineal del desarrollo lleva a estados de bloqueo en el que algunos miembros del equipo del proyecto deben esperar a otros miembros del equipo para completar tareas dependientes. En efecto, el tiempo que se pasa esperando puede sobrepasar el tiempo que se emplea en el trabajo productivo. Los estados de bloqueo tienden a ser más importantes al principio y al final de un proceso lineal secuencial.

Cada uno de estos problemas es real. Sin embargo, el paradigma del ciclo de vida lineal tiene un lugar definido e importante en el trabajo de la ingeniería de software, pues proporciona una plantilla en la que se encuentran métodos para análisis, diseño, codificación, pruebas y mantenimiento. El ciclo de vida lineal sigue siendo el modelo de proceso más utilizado. Pese a tener debilidades, es significativamente mejor que un enfoque hecho al azar para el desarrollo de software.

2.5. El modelo de Construcción de Prototipos

El paradigma de construcción de prototipos comienza con la recolección de requisitos, como se muestra en la figura 5 (Pressman 1998) :

Figura 5 Modelado de Prototipazo

El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos, y las áreas del esquema en donde es obligatoria más definición. Entonces aparece un “diseño rápido”. El diseño rápido se centra en una representación de esos aspectos de software que serán visibles para el usuario/cliente. El diseño rápido lleva a la construcción de un prototipo. El prototipo lo evalúa el cliente/usuario y lo utiliza para refinar los requisitos del software a desarrollar. La interacción entre el desarrollador y el cliente ocurre cuando el prototipo satisface las necesidades del cliente, a la vez que permite que el desarrollador comprenda mejor lo que se necesita hacer.

A los clientes y a los desarrolladores les gusta el paradigma de construcción de prototipos. A los usuarios les gusta el sistema real y a los desarrolladores les gusta construir algo inmediatamente. Sin embargo, la construcción de prototipos también puede ser problemática por las siguientes razones:

1. El cliente ve lo que parece ser una versión de trabajo del software, sin saber que con la prisa de hacerlo funcional no se ha tenido en cuenta la calidad global del software o la facilidad de mantenimiento a largo plazo. Cuando se informa de que el producto se debe construir otra vez para que se puedan mantener los altos niveles de calidad, el cliente no lo entiende y pide que se apliquen unos "pequeños ajustes" para que se pueda hacer del prototipo un producto final. De forma demasiado frecuente la gestión de desarrollo del software es muy lenta.

2. El desarrollador a menudo hace compromisos de implementación para hacer que el prototipo funcione rápidamente. Se puede utilizar un sistema operativo o lenguaje de programación inadecuado simplemente porque está disponible o porque es conocido; un algoritmo deficiente se puede implementar simplemente para demostrar la capacidad. Después de algún tiempo, el desarrollador debe familiarizarse con estas selecciones, y olvidarse de las razones por las cuales son inadecuadas. La solución menos ideal ahora es una parte integral del sistema.

Aunque pueden surgir problemas, la construcción de prototipos puede ser un paradigma efectivo de la ingeniería de software. La clave es definir las reglas del negocio al comienzo; es decir, el cliente y el desarrollador se deben poner de acuerdo en que el prototipo se construya para servir como un mecanismo de definición de requisitos. Entonces se descarta (al menos en parte), y se realiza la ingeniería de software con una visión hacia la calidad y la facilidad de mantenimiento.

2.6. El modelo DRA

El *Desarrollo Rápido de Aplicaciones (DRA)* (Rapid Aplication Development, RAD) es un modelo de proceso del desarrollo del software lineal secuencial que enfatiza un ciclo de desarrollo extremadamente corto. El modelo DRA es una adaptación a "alta velocidad" del modelo lineal secuencial en el que se logra el desarrollo rápido utilizando un enfoque de construcción basado en componentes. Si se comprenden bien los requisitos y se limita bien el ambito del proyecto, el proceso DRA permite al equipo de desarrollo crear un "sistema completamente funcional" dentro de periodos cortos de tiempo. El enfoque DRA comprende las siguientes fases (Pressman 1998):

- **Modelado de Gestión**

El flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce el proceso de gestión?, ¿qué información se genera?, ¿quién la genera?, ¿a dónde va la información? y ¿quién la procesa?, obteniendo así, modelos que:

  - define los procesos que satisfacen las necesidades de la visión bajo consideración, es decir que sólo son amigables a la vista;

  - representen el comportamiento de los procesos y los supuestos en los que se basa el comportamiento.

- **Modelado de datos**

El flujo de información, definido como parte de la fase de modelado de gestión, se refina como un conjunto de objetos de datos necesarios para apoyar a la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.

- **Modelado del proceso**

Los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir, o recuperar un objeto de datos.

- **Generación de aplicaciones**

El DRA asume la utilización de técnicas de cuarta generación (T4G), que no son más que herramientas que generan automáticamente código fuente, basándose en la especificación del técnico. El paradigma T4G se orienta hacia la posibilidad de especificar el software usando formas de lenguaje especializadas o notaciones gráficas que describan el problema que haya que resolver en términos que los entienda el cliente y la computadora.

En lugar de crear software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o a crear componentes reutilizables (cuando sea necesario). En todos los casos se utilizan herramientas automáticas para facilitar la construcción del software.

- **Pruebas y entrega**

Como el proceso DRA enfatiza la reutilización, ya se han comprobado muchos de los componentes de los programas. Esto reduce tiempo de pruebas. Sin embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfaces a fondo.

Si una aplicación de gestión puede modularse de forma que permita completarse cada una de las funciones principales en menos de tres meses, es un candidato del DRA. Cada una de las funciones pueden ser afrontadas por un equipo DRA diferente y ser integradas en un solo conjunto.

Al igual que todos los modelos de procesos, el enfoque DRA tiene inconvenientes:

1. Para proyectos grandes aunque por escalas, el DRA requiere recursos humanos suficientes como para crear el número correcto de equipos DRA.
2. DRA requiere clientes y desarrolladores comprometidos en las rápidas actividades necesarias para completar un sistema en un marco de tiempo abreviado. Si no hay compromiso, por alguna de las partes constituyentes, los proyectos DRA fracasarán.

No todos los tipos de aplicaciones son apropiados para DRA. Si un sistema no se puede modularizar adecuadamente, la construcción de los componentes necesarios para DRA, será problemático. Si está en juego el alto rendimiento, y se va a conseguir el rendimiento convirtiendo interfaces en componentes de sistemas, el enfoque DRA puede que no funcione. DRA no es adecuado cuando los riesgos técnicos son altos. Esto ocurre cuando una nueva aplicación hace uso de tecnologías nuevas, o cuando el nuevo software requiere un alto grado de interoperatividad con programas de computo ya existentes.

2.7. Modelos de Procesos Evolutivos

El desarrollo de software requiere un modelo de proceso que se haya diseñado explícitamente para acomodarse a un producto que evolucione con el tiempo.

El modelo lineal secuencial, como ya se mencionó anteriormente, se diseña para el desarrollo en línea recta. En esencia, este enfoque en cascada asume que se va a entregar un sistema completo una vez que la secuencia lineal se haya finalizado. El modelo de contrucción de prototipos, también ya mencionado, se utiliza para ayudar al cliente (o al que desarrolla) a comprender los requisitos. En general, no se diseña para entregar un sistema de producción. En ninguno de los paradigmas de ingeniería de software se tiene en cuenta la naturaleza evolutiva del software.

Los modelos evolutivos son iterativos. Se caracterizan por la forma en que permiten al equipo de trabajo desarrollar versiones cada vez más completas de software (Pressman 1998) :

2.7.1. El modelo incremental

Este modelo combina elementos del modelo lineal secuencial (aplicados repetitivamente) con la filosofía interactiva de la construcción de prototipos, este modelo se ilustra en la figura 6:

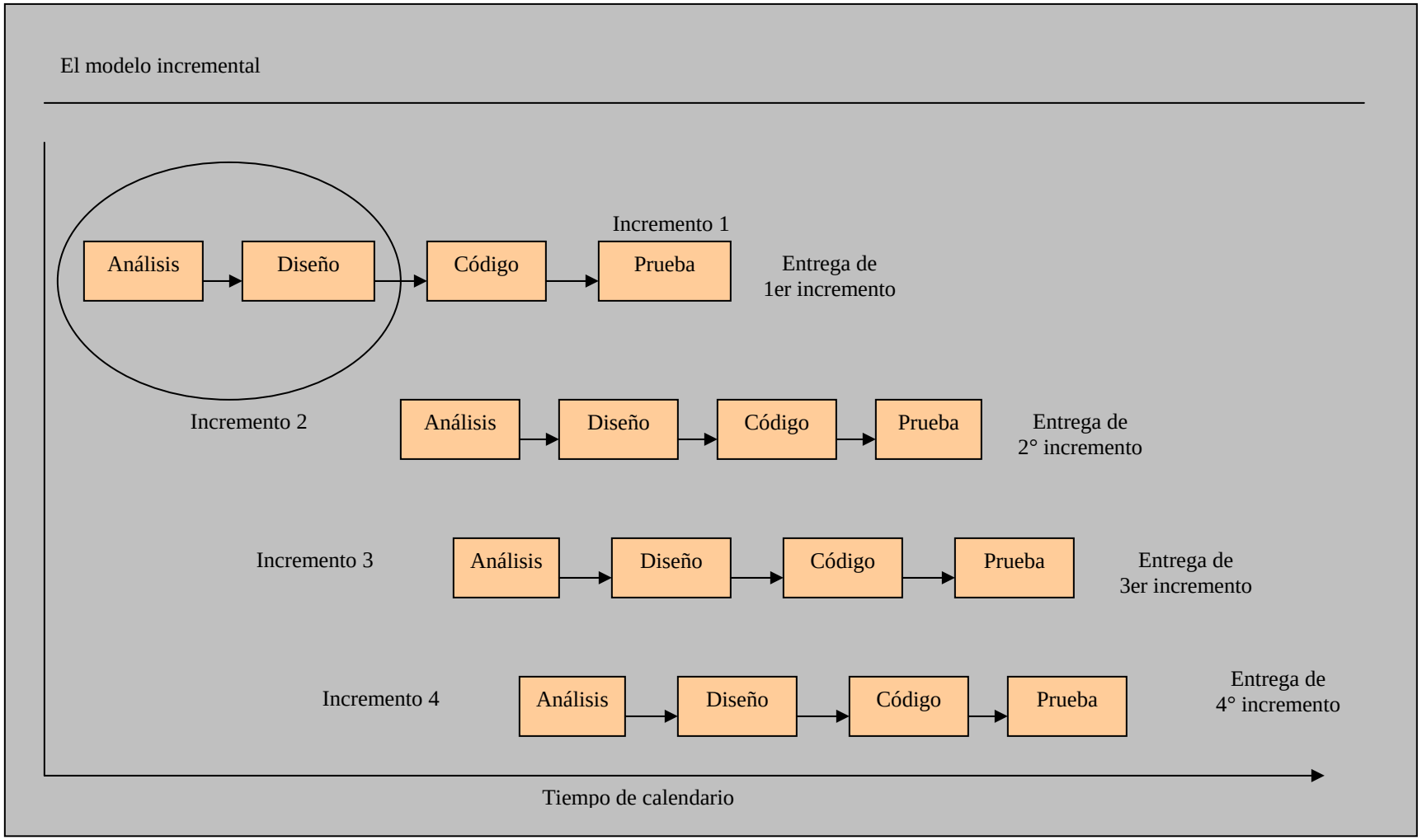

Figura 6 El modelo incremental

Este modelo combina elementos del modelo lineal secuencial (aplicados repetitivamente) con la filosofía interactiva de construcción de prototipos. Como se muestra en la figura 6, el modelo incremental aplica secuencias lineales de la misma forma en que progresa el tiempo en el calendario. Cada secuencia lineal produce un "incremento" de software.

Cuando se utiliza un modelo incremental, el primer incremento a menudo es un producto esencial (núcleo). Es decir, se afrontan requisitos básicos, pero muchas funciones suplementarias (algunas conocidas y otras no) quedan sin extraer. El cliente utiliza el producto central. Como un resultado de utilización y/o evaluación, se desarrolla un plan para el incremento siguiente. El plan afronta la modificación del producto central a fin de cumplir mejor las necesidades del cliente y la entrega de funciones, y características adicionales. Este proceso se repite siguiendo la entrega de cada incremento hasta que se elabore el producto completo.

El modelo de proceso incremental, como la construcción de prototipos y otros enfoques evolutivos, es interactivo por naturaleza, pero a diferencia de la contrucción de prototipos, se centra en la entrega de un producto operacional con cada incremento.

2.7.2. El modelo en espiral

Es un modelo de proceso evolutivo que acompaña la naturaleza interactiva de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. Se proporciona el potencial para el desarrollo rápido de versiones incrementales de software. En el modelo espiral, el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema.

El modelo en espiral se divide un número de *actividades estructurales*, también llamadas *regiones de tareas*. Generalmente, existen entre tres y seis regiones de tareas. La figura 7 representa un modelo en espiral que contiene seis regiones de tareas.

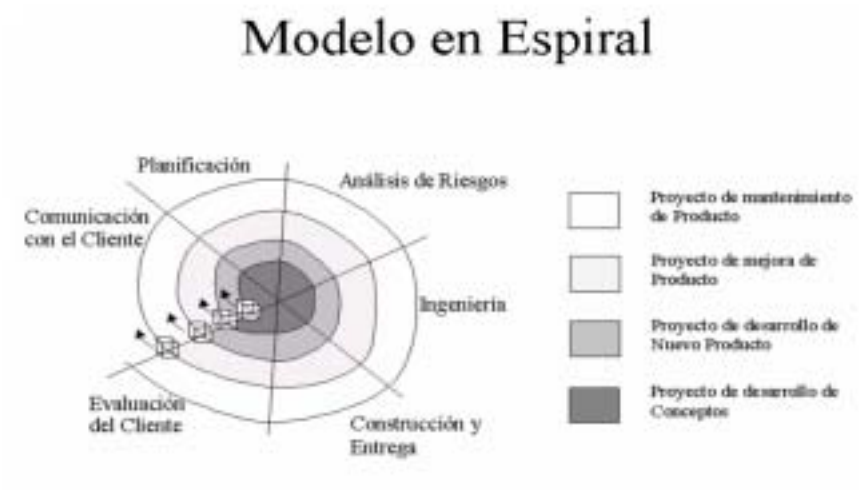

Figura 7 Modelo en Espiral (Pressman 1997)

- **Comunicación con el cliente.** Representa las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.

- **Planificación.** Representa las tareas requeridas para definir recursos, el tiempo y otras informaciones relacionadas con el proyecto.

- **Análisis de riesgos.** Representa las tareas requeridas para construir una o más representaciones de la aplicación.

- **Construcción y adaptación.** Representa las tareas requeridas para construir, probar, instalar y proporcionar soporte al usuario.

- **Evaluación del cliente.** Representa las tareas requeridas para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementada durante la etapa de instalación.

Cada una de las regiones están pobladas por una serie de tareas que se adaptan a las características del proyecto que va a emprenderse. Para proyectos pequeños, el número de tareas y su formalidad es bajo. Para proyectos mayores y más críticos, cada región contiene tareas que se definen para lograr un nivel más alto de la formalidad.

Cuando empieza este proceso evolutivo, el equipo de ingeniería del software gira alrededor de la espiral en la dirección de las agujas del reloj, comenzando por el centro. El primer circuito de la espiral produce el desarrollo de una especificación de productos; los pasos siguientes en la espiral se podrían utilizar para desarrollar un prototipo y progresivamente versiones más sofisticadas del software. Cada paso de la región de planificación produce ajustes en el plan del proyecto. El costo y la planificación se ajustan según la reacción ante la evaluación del cliente. Además, el gestor del proyecto ajusta el número planificado de iteraciones requeridas para completar el software.

A diferencia del modelo de proceso clásico que termina cuando se entrega el software, el modelo en espiral puede adaptarse y aplicarse a lo largo de la vida del software.

El modelo en espiral es un enfoque realista del desarrollo de sistemas y de software a gran escala. Como el software evoluciona, a medida que progresa el proceso, el desarrollador y el cliente comprenden y reaccionan mejor ante riesgos en cada uno de los niveles evolutivos. El modelo en espiral utiliza la construcción de prototipos como mecanismo de reducción de riesgos, pero lo que es más importante, permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto. Mantiene el enfoque sistemático de los pasos sugeridos por el ciclo de vida clásico, pero lo incorpora al marco de trabajo interactivo que refleja de forma más realista el mundo real. El modelo en espiral demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto, y si se aplica adecuadamente, debe reducir los riesgos antes de que se conviertan en problemas.

Al igual que otros paradigmas, el modelo en espiral no es la panacea. Puede resultar difícil convencer a grandes clientes (particularmente en situaciones bajo contrato) de que el enfoque evolutivo es controlable. Requiere una habilidad considerable para la evaluación del riesgo, y cuenta con esta habilidad para el éxito. Si un riesgo importante no es descubierto y gestionado, indudablemente surgirán problemas. Finalmente, el modelo en sí mismo es relativamente nuevo y no se ha utilizado tanto como los paradigmas lineales secuenciales o de construcción de prototipos.

2.7.3. El modelo de ensamblaje de componentes

El modelo de ensamblaje de componentes incorpora muchas de las características del modelo en espiral. Es evolutivo por naturaleza, y exige un enfoque interactivo para la creación del software. Sin embargo este modelo configura aplicaciones desde componentes preparados de software (algunas veces llamados “clases”), como se muestra en la figura 8:

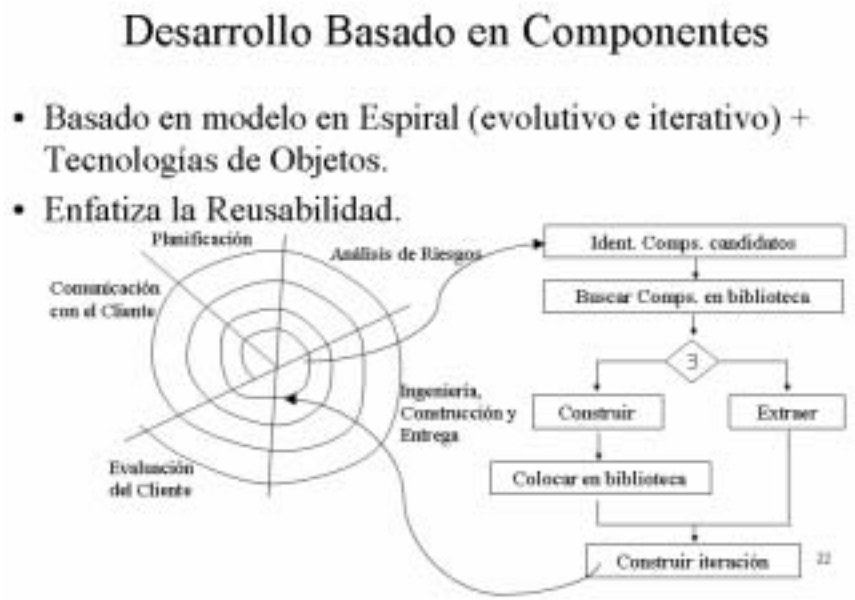

Figura 8 Desarrollo Basado en Componentes (Pressman 1997)

La actividad de la ingeniería comienza con la identificación de las clases candidatas. Esto se lleva a cabo examinando los datos que se van a manejar por parte de la aplicación y el algoritmo que se va a aplicar para conseguir el tratamiento. Los datos y los logoritmos correpondientes se empaquetan en una clase.

Las clases (llamadas componentes en la figura 8) creadas en los proyectos de ingeniería de software se almacenan en una *biblioteca de clases* o *depósito*. Una vez identificadas las clases candidatas, la biblioteca de clases se examina para determinar si estas clases ya existen. En caso de que así fuera, se extraen de la biblioteca y se vuelven a utilizar. Si una clase candidata no reside en la biblioteca, se aplican los métodos orientados a objetos. Se compone así la primera iteración de la aplicación a construirse, mediante las clases extraidas de la biblioteca y las clases nuevas construidas para cumplir las necesidades únicas de la aplicación. El flujo del proceso vuelve a la espiral y volvera a introducir por último la iteración ensambladora de componentes a través de la actividad de ingeniería.

El modelo ensamblador de componentes lleva a la reutilización del software, y la reutilización proporciona beneficios a los ingenieros de software. Según estudios de reutilización (Pressman 1998), QSM Associates, Inc informa que el ensamblaje de componentes lleva a un 70 por ciento de tiempo de ciclo de desarrollo, un 84 por ciento del costo del proyecto y un índice de productividad del 26.2 por ciento.

2.7.4. El modelo de desarrollo concurrente

El modelo de proceso concurrente se puede representar en forma de esquema como una serie de actividades técnicas importantes, tareas, y estados asociados a ellas. Este modelo define una serie de acontecimientos que dispararán transiciones de estado a estado para cada una de las actividades de la ingeniería del software.

La figura 9 muestra una representación esquemática de una actividad dentro del modelo de proceso concurrente.

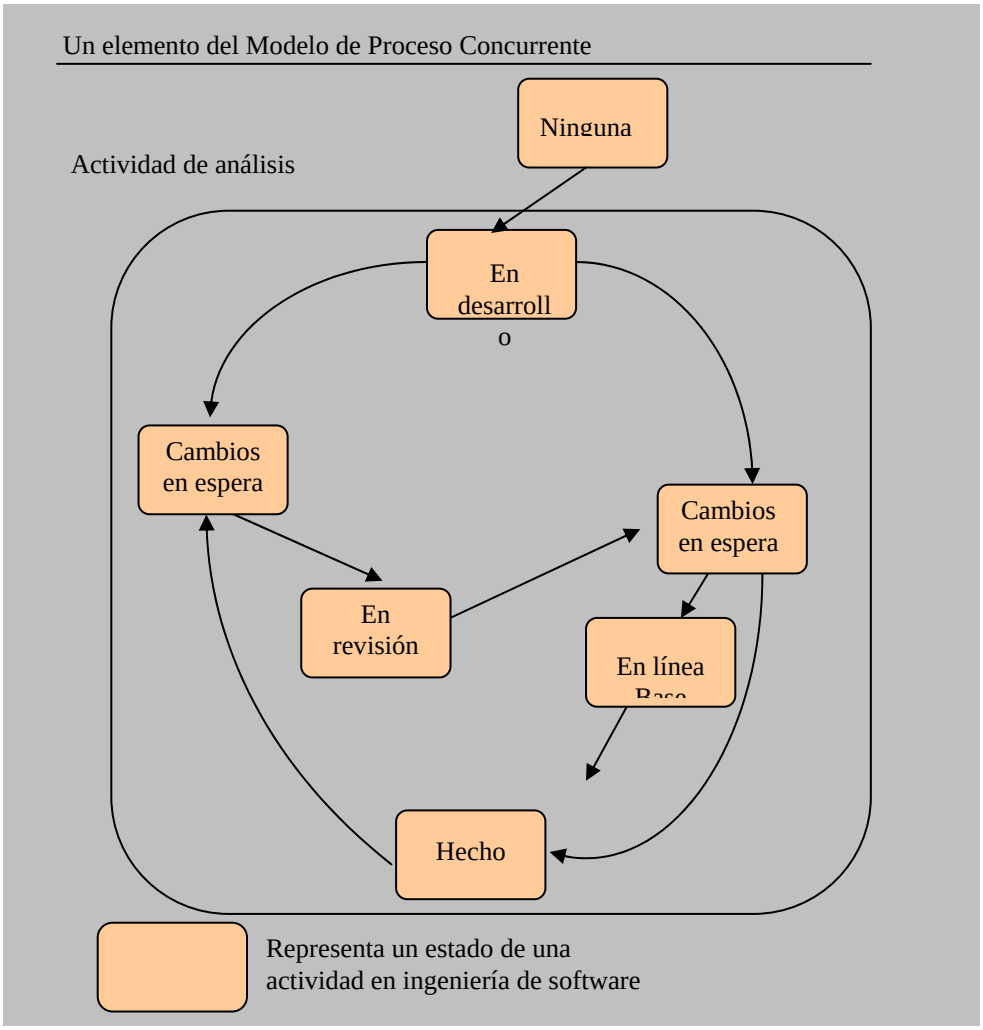

Figura 9 Un elemento del Modelo de Proceso Concurrente (Theoktisto 1999)

La actividad *análisis* se puede encontrar en uno de los estados (un estado es algún modo externamente observable del comportamiento del sistema) destacados anteriormente en cualquier momento dado. De forma similar, otras actividades se pueden representar de una forma análoga (por ejemplo, diseño o comunicación con el cliente). Todas las actividades existen concurrentemente, pero residen en estados diferentes. Por ejemplo, al principio del proyecto, la actividad *de comunicación con el cliente* (no mostrada en la figura) ha finalizado su primera interacción y existe en el estado de **cambios en espera**. La actividad de *análisis* (que existía en el estado **ninguno** mientras que comenzaba la comunicación inicial con el cliente) ahora hace una transición al estado **bajo desarrollo**. Sin embargo, si el cliente indica que se deben hacer cambios en requisitos, la actividad *análisis* cambia del estado **bajo desarrollo** al estado **cambios en espera**.

El modelo de proceso concurrente define una serie de acontecimientos que dispararán transiciones de estado a estado para cada una de las actividades de la ingeniería del software. Por ejemplo, durante las primeras etapas del diseño, no se contempla una inconsistencia del modelo de análisis. Esto generá la corrección del modelo de análisis de sucesos, que disparará la actividad de análisis del estado **hecho** al estado **cambios en espera.**

El modelo de proceso concurrente se utiliza a menudo como el paradigma de desarrollo de aplicaciones cliente/servidor. Un sistema cliente/servidor se compone de un conjunto de componentes funcionales. Cuando se aplica a cliente/servidor, el modelo de proceso concurrente define actividades en dos dimensiones: una dimensión de sistemas y una dimensión de componentes. Los aspectos del nivel de sistemas se afrontan mediante tres actividades: *diseño, ensamblaje* y *uso*. La dimensión de componentes se afronta con dos actividades: *diseño* y *realización*. La concurrencia se logra de dos formas:

1. Las actividades de sistemas y de componentes ocurren simultáneamente y pueden modelarse con un enfoque orientado a objetos.
2. Una aplicación cliente/servidor típica se implementa con muchos componentes, cada uno de los cuales se pueden diseñar y realizar concurrentemente.

En realidad, el modelo de proceso concurrente es aplicable a todo tipo de desarrollo de software y proporciona una imagen exacta del estado actual de un proyecto. En vez de confinar las actividades de ingeniería del software a una secuencia de sucesos, define una red de actividades. Todas las actividades de la red existen simultáneamente con otras. Los sucesos generados dentro de una actividad dada o en algún otro lugar en la red de actividad inicia las transiciones entre los estados de una actividad.

2.8. El modelo de métodos formales

Este modelo acompaña a un conjunto de actividades que conducen a la especificación matemática del software de computadora. Los métodos formales permiten especificar, desarrollar y verificar un sistema basado en computadora aplicando una notación rigurosa.

Cuando se utilizan métodos formales durante el desarrollo, se proporciona un macanismo para eliminar muchos de los problemas que son difíciles de superar con paradigmas de ingeniería de software. La ambigüedad, lo incompleto y la inconsistencia se descubren y se corrigen más fácilmente mediante la aplicación del análisis matemático. Cuando se utilizan métodos formales durante el diseño, estos sirven

como base para la verificación de programas y por consiguiente permiten que el ingeniero de software descubra y corrija errores que no se pudieron detectar de otra manera.

Aunque todavía no hay un enfoque establecido, los modelos de métodos formales ofrecen la promesa de un software libre de defectos. Sin embargo, se ha hablado de una gran preocupación sobre su aplicabilidad en un entorno de gestión:

1. El desarrollo de métodos formales actualmente es caro y lleva mucho tiempo.
2. Pocos responsables de desarrollo de software tienen los antecedentes necesarios para aplicar métodos formales.
3. Es difícil utilizar los modelos como un mecanismo de comunicación con clientes que no tienen onocimientos técnicos.

No obstante, es posible que el enfoque a través de métodos formales tenga más partidarios entre los desarrolladores de software que deben construir software de mucha seguridad, y entre los desarrolladores que pasan grandes penurias económicas al aparecer errores de software.

2.9. Métodos Ágiles

Debido a que el desarrollo de software no es una tarea fácil existen numerosas propuestas metodológicas que influyen en distintas dimensiones dentro de un proceso de desarrollo. Hasta hace poco el proceso de desarrollo llevaba asociado un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema tradicional para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de atención en el proceso. Sin embargo este enfoque no ha resultado ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo de software se resignan a prescindir del la buena realización de la ingeniería del software, asumiendo el riesgo que ello conlleva. En este escenario, las metodologías ágiles surgen como la posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas a proyectos pequeños, las metodologías ágiles constituyen una solución a la medida la para ese entorno, aportando una elevada simplificación que no renuncia a las prácticas esenciales de la ingeniería de software para asegurar la calidad del producto.

Las ventajas frente a otros métodos más usuales son varias, ya que se hace hincapié en una mejor gestión de los recursos humanos, y por otro, se intenta acortar las distintas fases de desarrollo, realizando muchas más pruebas a lo largo de todo el proceso. La idea de los métodos ágiles es reducir el tiempo y el costo del desarrollo de aplicaciones, buscando otras alternativas a las soluciones tradicionales.

Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo de software, con el objetivo de asegurar que el código que se obtenga satisfaga los requerimientos del usuario y reúna estándares aceptables de calidad. El trabajo de planificación es riguroso, aún cuando en la práctica muchas veces estas planificaciones no se respeten. En contraposición, las metodologías ágiles aportan nuevos métodos de trabajo que apuestan por una cantidad apropiada de procesos. Es decir, no se

desgastan con una excesiva cantidad de cuestiones administrativas (planificación, control, documentación) ni tampoco defienden la postura extremista de total falta de proceso. Ya que se tiene conciencia de que se producirán cambios, lo que se pretende es reducir el costo de rehacer el trabajo.

Como ventajas de las metodologías ágiles se puede decir que son adaptativas más que predictivas. Una metodología tradicional potencia la planificación detallada y de largo alcance de prácticamente todo el desarrollo de software (Por ejemplo El Modelo Lineal o en Cascada). En contraste, las metodologías ágiles proponen procesos que se adaptan y progresan con el cambio, llegando hasta el punto de cambiar ellos mismos.

Las metodologías ágiles están orientadas más a los desarrolladores que a los procesos de desarrollo. Intentan entonces trabajar con la naturaleza de las personas (desarrolladores y usuarios) asignadas a un proyecto, más que contra ellos, de tal forma que permiten que las actividades de desarrollo de software se conviertan en una actividad de colaboración grata e interesante.

Se presentan algunas limitaciones a la aplicación de los procesos ágiles en algunos tipos de proyectos. Los agilistas y sus críticos focalizan la discusión en torno a temas como la documentación y codificación. Por un lado se sostiene que el código es el único entregable que realmente importa, desplazando el rol del análisis y de diseño en la creación de software. Los críticos precisan que el énfasis en el código puede conducir a la pérdida de la memoria corporativa o conocimiento organizacional, porque hay poca documentación y modelos para apoyar la creación y evolución de sistemas complejos.

Los métodos ágiles se fundamentan en la socialización, por medio de la comunicación cara a cara, la colaboración, y la programación en parejas, para compartir conocimiento tácito entre los desarrolladores y usuarios.

Los métodos ágiles privilegian las personas, la comunicación y la colaboración. Lo que facilita el compartir conocimiento tácito. Fomenta también, una cultura de confianza y motivación, lo que a su vez posibilita la implementación de herramientas para compartir conocimiento explícito.

Ciertos dominios pueden ser más adecuados a un tipo de proceso, y en general, algunos aspectos del desarrollo de software se beneficiarán del enfoque agilista mientras otros obtendrán beneficios de un enfoque tradicional. Desde esta perspectiva los procesos de desarrollo de software pueden ser clasificados dentro de un amplio espectro dependiendo de su “grado de agilidad”. Lo importante es saber ubicarse debidamente dentro de él y optar por el tipo de proceso y herramientas que mejor sirvan a cada proyecto.

2.9.1. EXTREME PROGRAMING (XP)

La Programación Extrema es sin duda alguna el método ágil que primero viene a la mente, en la programación extrema se da por supuesto que es imposible prever todo antes de empezar a codificar. Es imposible capturar todos los requisitos del sistema, saber qué es todo lo que tiene que hacer y no hacer en un diseño al principio. Es bastante normal hacer un diseño, ponerse a codificar, ver que hay faltantes o errores en el diseño, empezar a codificar fuera del diseño y al final el código y el diseño, o no se parecen, o empleamos demasiado tiempo en cambiar la documentación de diseño para que se parezca al código.

En vez de tratar de luchar contra todo eso, extreme programming lo asume y busca una forma de trabajar que se adapte fácilmente a esas circunstancias. Básicamente la idea de la programación extrema consiste en trabajar estrechamente con el cliente, haciéndole mini-versiones con mucha frecuencia (cada dos semanas). En cada mini-versión se debe hacer el mínimo de código y lo más simple posible para que funcione correctamente. El diseño se hace sobre la marcha, haciendo un mini-diseño para la primera mini-versión y luego modificándolo en las siguientes mini-versiones. Además, no hay que hacer una documentación para el diseño, no hay mejor documentación que el mismo código. El código, por tanto, también se modifica continuamente de mini-versión en mini-versión, añadiéndole funcionalidad y extrayendo sus partes comunes.

XP se funda en cuatro valores (Corporation 2004): comunicación, coraje, simplicidad y feedback.

- **La Comunicación**:

  El eXtreme Programming se nutre del ancho de banda más grande que se puede obtener cuando existe algún tipo de comunicación: *la comunicación directa* entre personas.
  Es muy importante entender cuales son las ventajas de este medio. Cuando dos (o más) personas se comunican directamente pueden no solo *consumir* las palabras formuladas por la otra persona, sino que también aprecian los gestos, miradas, etc. que hace su compañero.
  Sin embargo, en una conversación mediante el correo electrónico, hay muchos factores que hacen de esta una comunicación, por así decirlo, mucho menos efectiva.

- **El Coraje:**

  El coraje es un valor muy importante dentro de la programación extrema. Un miembro de un equipo de desarrollo extremo debe de tener el coraje de exponer sus dudas, miedos, experiencias sin "embellecer" estas de ninguna de las maneras. Esto es muy importante ya que un equipo de desarrollo extremo se basa en la confianza para con sus miembros. Faltar a esta confianza es una falta más que grave.

- **La Simplicidad:**

  Dado que no se puede predecir como va a ser en el futuro, el software que se esta desarrollando; un equipo de programación extrema intenta mantener el software lo más sencillo posible. Esto quiere decir que no se va a invertir ningún esfuerzo en hacer un desarrollo que en un futuro pueda llegar a tener valor.

En el XP frases como *"...en un futuro vamos a necesitar..."* o *"Haz un sistema genérico de..."* no tienen ningún sentido ya que no aportan ningún valor en el momento.

- **El Feedback:**

  La agilidad se define (entre otras cosas) por la capacidad de respuesta ante los cambios que se van haciendo necesarios a lo largo del camino. Por este motivo uno de los valores que nos hace más ágiles es el continuo seguimiento o feedback que recibimos a la hora de desarrollar en un entorno ágil de desarrollo. Este *feedback* se toma del cliente, de los miembros del equipo, en cuestión de todo el entorno en el que se mueve un equipo de desarrollo ágil.

Las características fundamentales de Extreme Programming son (Jeffries 2001):

1. **Desarrollo iterativo e incremental**: pequeñas mejoras, unas tras otras.

2. **Pruebas unitarias continuas**, frecuentemente repetidas y automáticas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.

3. **Programación por parejas**: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.
   Frecuente interacción del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.

4. **Corrección de todos los errores** antes de añadir nueva funcionalidad. Hacer entregas frecuentes.

5. **Refactoreo del código**, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en el refactoreo no se ha introducido ningún fallo.

6. **Propiedad del código compartida**: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.

7. **Simplicidad** en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario.

Un proyecto usando programación extrema se desarrolla de la siguiente manera:

El cliente junto con el equipo de desarrollo, definen qué es lo que se quiere hacer. Para ello utilizan las "historias de usuario" (Programación 2003).

Una historia de usuario en un texto de una o dos frases en las que se dice algo que debe hacer el sistema, es más extensa que un requisito (que suele ser una frase corta) y menos que un caso de uso (que puede ser de una o dos páginas). Se evalúa para cada historia de usuario el tiempo que puede llevar, que debe ser corto, de aproximadamente una semana. Un programador puede estimar con cierta fiabilidad un trabajo que le lleve unos días, pero la estimación es menos fiable si es de un plazo superior a una semana. Si es más largo, hay que partir la historia en otras más pequeñas. Luego se ordenan en el orden en que se van a desarrollar y se establecen las mini-versiones, de forma que cada mini-versión implementa varias de las historias de usuario.

Toda esta planificación va a ser, por supuesto, inexacta. No se puede saber todo lo que va a ser necesario ni evaluar los tiempos correctamente. La planificación deberá revisarse y modificarse continuamente a lo largo del proyecto. Las historias de usuario se modificarán, se quitarán o se añadirán nuevas sobre la marcha. Puesto que el cliente estará presente día a día durante todo el proyecto.

Para las primeras historias que se van a implementar, se define una prueba para ver si la versión cumple perfectamente con la historia. Estas pruebas deben ser automáticas, de forma que haya un programa de pruebas que ejecutemos y nos diga si la mini-versión es o no correcta.

Los programadores se ponen por parejas (dos personas en la misma computadora) para codificar esas historias. Primero codifican el programa de pruebas y ven qué falla (el código aún no está hecho!). Luego piensan cómo van a implementar la historia y hacen sus propios programas de prueba (también

automáticos). Codifican sus programas de prueba y ven que también fallan. Luego se ponen a codificar hasta que se pasen con éxito todas las pruebas. En su código hacen de la forma más sencilla posible lo mínimo imprescindible para que se pasen las pruebas automáticas.

Las parejas de programadores se intercambian con frecuencia, de forma que todos acaban trabajando con todos. El trabajo por parejas haciendo intercambios tiene las siguientes ventajas:

- Cuatro ojos ven más que dos. Al trabajar de dos en dos, el código será de mayor calidad desde el mismo momento de crearlo y tendrá menos fallas.

- Los programadores novatos aprenderán de los expertos al emparejarse con ellos.

- Si una pareja realiza un trozo de código susceptible de ser reutilizado en el proyecto, hay dos programadores que lo saben y que lo reutilizarán cuando puedan (ya que saben cómo funciona), enseñándolo a sus nuevos compañeros. De esta manera el conocimiento del código ya hecho se propaga de forma natural entre todos los programadores del equipo.

- El estilo de programación tiende a unificarse.

Cuando una nueva pareja va a realizar un nuevo código para una historia de usuario, puede que uno de ellos recuerde haber hecho un trozo de código en otro lado que podría reutilizar. Esta pareja irá a ese trozo de código y lo reutilizará, modificándolo si es necesario. Después de modificarlo, deben asegurarse que se siguen pasando las pruebas automáticas que se hicieron en su momento, así como añadir nuevas pruebas para comprobar las modificaciones que han hecho.

Si una pareja al reutilizar código ya hecho ve que es mejorable, lo mejoran, pasando las pruebas automáticas después. Si al reutilizar el código ya hecho descubren un error que las pruebas automáticas no detectan, añaden pruebas capaces de detectar el error y lo corrigen.

El código, por tanto, no es de nadie. Cualquier pareja puede tocar el código ya hecho por otras personas si lo necesita, con la condición de que después de tocarlo las pruebas automáticas sigan pasándose correctamente y que añadan sus propias pruebas automáticas para las correcciones realizadas.

Todos los días se hace una pequeña reunión a primera hora de la mañana con todo el equipo en la que se comentan problemas, código que se está realizando, historias de usuario terminadas, etc.

Cada vez que se consigue codificar y que funcione una historia de usuario, se le da al cliente para que la vea, la pruebe y añada las posibles modificaciones para las siguientes mini-versiones. Cuando se realiza una mini-versión completa (compuesta por varias de las historias de usuario), incluso se entrega al usuario final para que empiece a trabajar con ella y reportar incidencias o mejoras.

Este ciclo se va repitiendo una y otra vez hasta que el cliente se de por satisfecho y cierre el proyecto, tal como lo muestra la fgura 10. Como se ve, no se ha hecho documentación (Jeffries 2001).

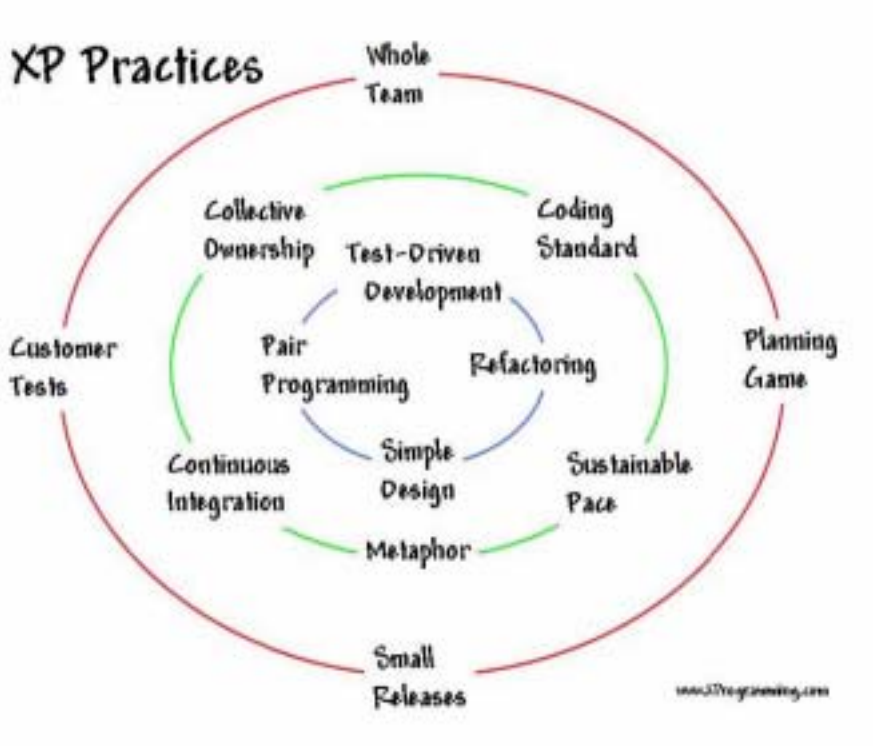

Figura 10 Extreme Programming

En algún sitio he leído que incluso la planificación inicial debe hacerse escribiendo cada historia de usuario en un tarjeta, haciendo dos montones, las que ya están hechas y las que no. Se pueden tirar las tarjetas, añadir nuevas o cambiar las que ya hay. El número de tarjetas en cada montón nos da una idea exacta de cuánto hay hecho del proyecto.

2.10. Rational Unified Process (RUP)

Un proceso es un conjunto de pasos ordenados parcialmente para alcanzar un objetivo. En la ingeniería de software, el objetivo es entregar un producto software que satisfaga las necesidades del usuario de forma eficiente y predecible.

El Rational Unified Process™ (RUP) es un proceso de ingeniería de software que mejora la productividad del equipo de trabajo y entrega las mejores prácticas del software a todos los miembros del mismo. El objetivo del Proceso Unificado de Rational es permitir la producción de un software de la mayor calidad que satisfaga las necesidades de los usuarios finales, dentro de planificaciones y presupuestos predecibles. RUP captura algunas de las mejores prácticas de desarrollo de software, de una forma que es adaptable a un amplio rango de proyectos y organizaciones. En el aspecto de la gestión, RUP proporciona un enfoque disciplinado sobre cómo asignar tareas y responsabilidades dentro de una organización de desarrollo de software.

Características del proceso

El Proceso Unificado de Rational es un proyecto iterativo. Para los sistemas simples, parece perfectamente factible definir de forma secuencial el problema completo, diseñar la solución completa, construir el software, y acontinuación, hacer pruebas con el producto final. Sin embargo, dadas la complejidad y sofisticación que demandan los sistemas actuales, este enfoque lineal al desarrollo de sistemas no es realista. Un enfoque iterativo propone una comprensión incremental a través de refinamientos sucesivos y un crecimiento incremental de una solución efectiva a través de varios ciclos. Como parte del enfoque iterativo se encuentra la flexibilidad para acomodarse a nuevos requisitos o a cambios tácticos en los objetivos del negocio. También permite que el proyecto identifique y resuelva los riesgos tempranamente.

Fases e Iteraciones

El proceso de ciclo de vida de RUP se divide en cuatro fases, una fase es el intervalo de tiempo entre dos hitos importantes del proceso durante la cual se cumple un conjunto bien definido de objetivos, se completan artefactos y se toman las decisiones sobre si pasar a la siguiente fase. Como se muestra en la figura 11 (García), el Proceso Unificado de Rational consta de las cuatro fases siguientes: Incepción, Elaboración, Construcción y Transición. Esas fases se dividen en iteraciones, cada una de las cuales produce una pieza de software demostrable. La duración de cada iteración puede extenderse desde dos semanas hasta seis meses.

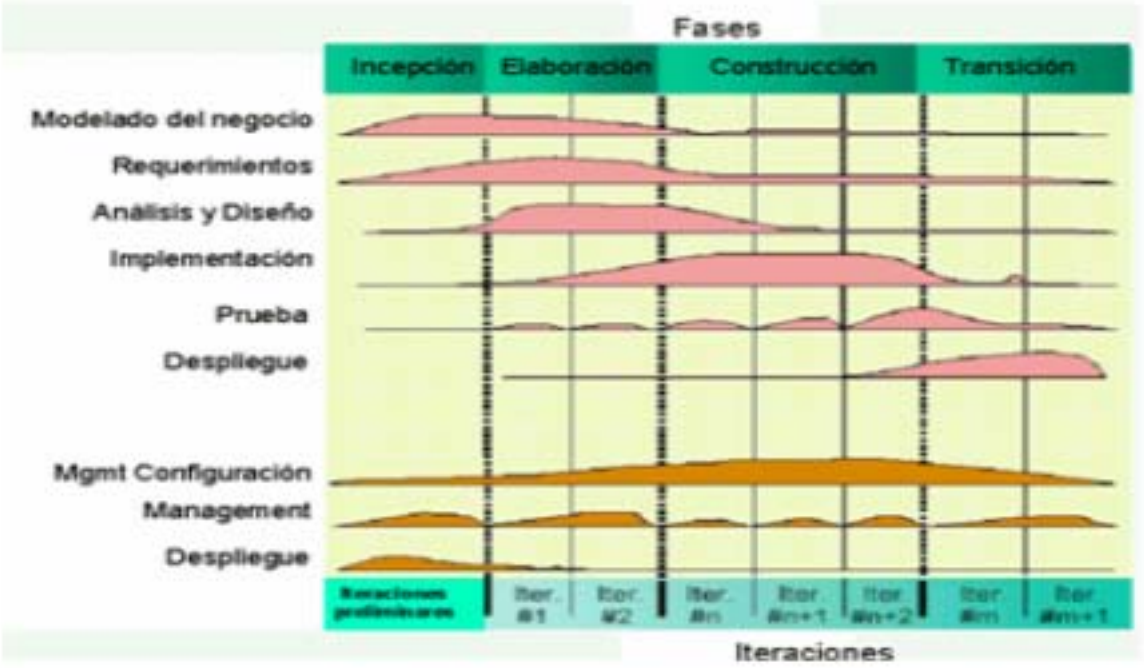

Figura 11 Proceso Unificado de Rational

Fases del Proceso Unificado:

- **Fase de Incepción**. Significa "comienzo", pero la palabra original (de origen latino y casi en desuso como sustantivo) es sugestiva y por ello la traducimos así. El objetivo de esta fase es analizar el dominio del problema, establecer una base arquitectónica sólida, desarrollar el plan del proyecto y eliminar los elementos de más alto riesgo del proyecto. Las decisiones arquitectónicas deben tomarse con una comprensión del tema global. Esto implica que se deben describir la mayoría de los requisitos del sistema. Para verificar la arquitectura, se implementa un sistema que demuestre las distintas posibilidades de la arquitectura y ejecute los casos de uso significativos.
  Al final de fase de elaboración se examinan el alcance y los objetivos del sistema, la elección de la arquitectura y la resolución de los riesgos más grandes, y se decide si se debe pasar a la construcción.

- **Fase de Elaboración.** Se analiza el dominio del problema y se define el plan del proyecto. RUP presupone que la fase de elaboración brinda una arquitectura suficientemente sólida junto con requerimientos y planes bastante estables. Se describen en detalle la infraestructura y el ambiente de desarrollo, así como el soporte de herramientas de automatización. Al cabo de esta fase, debe estar identificada la mayoría de los casos de uso y los actores, debe quedar descripta la arquitectura de software y se debe crear un prototipo de ella. Al final de la fase se realiza un análisis para determinar los riesgos y se evalúan los gastos hechos contra los originalmente planeados.

- **Fase de Construcción.** Se desarrollan, integran y verifican todos los componentes y rasgos de la aplicación. RUP considera que esta fase es un proceso de manufactura, en el que se debe poner énfasis en la administración de los recursos y el control de costos, agenda y calidad. Los resultados de esta fase se crean tan rápido como sea posible. Se debe compilar también una versión de entrega. Es la fase más prolongada de todas.

- **Fase de Transición.** Comienza cuando el producto está suficientemente maduro para ser entregado. Se corrigen los últimos errores y se agregan los rasgos pospuestos. La fase consiste en prueba beta, piloto, entrenamiento a usuarios y despacho del producto a mercadeo, distribución y ventas. Se produce también la documentación. Se llama transición porque se transfiere a las manos del usuario, pasando del entorno de desarrollo al de producción.

Iteraciones

Cada frase en el Proceso Unificado de Rational puede descomponerse en iteraciones. Una iteración es un ciclo completo de desarrollo que produce una versión (interna o externa) de un producto ejecutable, que constituye un subconjunto del producto final en desarrollo, que luego se irá incrementando de iteración en iteración hasta convertirse en el sistema final. Cada iteración pasa a través de varios flujos de trabajo del proceso, aunque con un énfasis diferente en cada uno de ellos, dependiendo de la fase en que se encuentre. Durante la iniciación, el interés se orienta hacia el análisis y el diseño. Durante la construcción, la actividad central es la implementación, y la transición se centra en el despliegue.

Ciclos de desarrollo

El paso a través de las cuatro fases principales constituye un ciclo de desarrollo, y produce una generación de software. La primera pasada a través de las cuatro fases se denomina ciclo de desarrollo inicial. A menos que acabe la vida del producto, un producto existente evolucionará a la siguiente generación repitiendo la misma secuencia de inicio, construcción y transición. Esta es la evolución del sistema, así que los ciclos de desarrollo después del ciclo inicial son los ciclos de evolución.

Flujos de trabajo del proceso

El proceso Unificado de Rational consta de nueve flujos de trabajo (Booch):

1. **Modelado del negocio.** Describe las estructura y la dinámica de la organización.

2. **Requisitos.** Describe el método basado en casos de uso para extraer los requisitos.

3. **Análisis y diseño.** Describe las diferentes vistas arquitectónicas.

4. **Implementación.** Tiene en cuenta el desarrollo de software, la prueba de unidades y la integración.

5. **Pruebas.** Describe los casos de pruebas, los procedimientos y las métricas para evaluación de defectos.

6. **Despliegue.** Cubre la configuración del sistema entregable.

7. **Gestión de configuraciones.** Controla los cambios y mantiene la integridad de los artefactos de un proyecto.

8. **Gestión del proyecto.** Describe varias estrategias del trabajo en un proceso iterativo.

9. **Entorno.** Cubre la infraestructura necesaria para desarrollar un sistema.

Dentro de cada flujo de trabajo del proceso hay un conjunto de artefactos y actividades relacionados. Un artefacto es algún documento, informa o ejecutable que se produce, se manipula o se consume. Una actividad describe las tareas (pasos de concepción, realización y revisión) que llevan a cabo los trabajadores para crear o modificar los artefactos, junto con las técnicas y guías para ejecutar las tareas, incluyendo quizá el uso de herramientas para ayudar a automatizar algunas de ellas.

Artefactos

Cada actividad del Proceso Unificado de Rational lleva algunos artefactos asociados, bien sean requeridos como entradas, bien sean generados como salidas. Algunos artefactos se utilizan como entradas directas en las actividades siguientes, se mantienen como recursos de referencias en el proyecto, o se generan en algún formato específico, en forma de entregable definidos en el contrato.

Modelos

Los modelos son el tipo de artefacto más importante en el Porceso Unificado de Rational. Un modelo es una simplificación de la realidad, creada para comprender mejor el sistema que se está creando. En el Proceso Unificado de Rational, hay nueve modelos (Booch) que en conjunto cubren todas las decisiones importantes implicadas en la visualización, especificación, construcción y documentación de un sistema con gran cantidad de software.

1. **Modelo del negocio.** Establece una abstracción de la organización.

2. **Modelo del dominio.** Establece el contexto del sistema.

3. **Modelo de casos de uso.** Establece los requisitos funcionales del sistema.

4. **Modelo de análisis (opcional).** Establece un diseño de las ideas.

5. **Modelo de diseño.** Establece el vocabulario del problema y su solución.

6. **Modelo del proceso (opcional).** Establece los mecanismos de concurrencia y sincronización del sistema.

7. **Modelo de despliegue.** Establece la topología hardware sobre la cuál se ejecutará el sistema.

8. **Modelo de implementación.** Establece las partes que se utilizarán para ensamblar y hacer disponible el sistema físico.

9. **Modelo de pruebas.** Establece las formas de validar y verificar el sistema.

Una vista es una proyección de un modelo. En el Proceso Unificado de Rational, la arquitectura de un sistema se captura en forma de cinco vistas que interactúan entre sí: la vista de diseño, las vista de procesos, la vista de despliegue, la vista de implementación y la vista de casos de uso.

Otros artefactos

Los artefactos del Proceso Unificado de Rational se clasifican en artefactos de gestión y artefactos técnicos. Los artefactos técnicos del Proceso Unificado de Rational pueden dividirse en cuatro conjuntos principales (Booch) :

1. **Conjunto de requisitos.** Describe qué debe hacer el sistema.

Este conjunto agrupa toda la información que describe lo que debe hacer el sistema. Esto puede comprender un modelo de casos de uso, un modelo de requisitos no funcionales, un modelo del dominio, un modelo de análisis y otras formas de expresión de las necesidades del usuario, incluyendo pero no limitándose a maquetas, prototipos de la interfaz, restricciones legales, etc.

2. **Conjunto de diseño.** Describe cómo se va a construir el sistema.

Este conjunto agrupa información que describe cómo se va a construir el sistema y captura las decisiones acerca de cómo se va realizar, teniendo en cuenta las restricciones de tiempo, presupuesto, aplicaciones existentes, reutilización, objetivos de calidad y demás consideraciones. Esto puede implicar un modelo de diseño, un modelo de pruebas y otras formas de expresión de la naturaleza del sistema, incluyendo, pero no limitándose, a prototipos y arquitecturas ejecutables.

3. **Conjunto de implementación.** Describe el ensamblado de los componentes software.

Este conjunto agrupa toda la información acerca de los elementos software que comprende el sistema, incluyendo, pero no limitándose, a código fuente en varios lenguajes de programación, archivos de configuración, archivos de datos, componentes software, etc., junto con la información que describe cómo ensamblar el equipo.

4. **Conjunto de despliegue.** Proporciona todos los datos para la configuración entregable.

Este conjunto agrupa toda la información acerca de la forma en que se empaqueta actualmente el software, se distribuye, se instala y se ejecuta en el entorno destino.

Durante el desarrollo de este trabajo se hará uso de UML ya que es un lenguaje estándar para escribir planos de software. UML se puede utilizar para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

UML es apropiado para modelar desde sistemas de información en empresas hasta aplicaciones distribuidas basadas en Web, e incluso para sistemas empotrados de tiempo real muy exigentes. Es un lenguaje muy expresivo, que cubre todas las vistas necesarias para desarrollar y luego desplegar el sistema. Aunque es expresivo, UML no es difícil de aprender ni de utilizar.

UML es sólo un lenguaje y por tanto es tan sólo una parte de un método de desarrollo de software. UML es independiente del proceso, aunque para utilizarlo óptimamente se debería usar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

Un lenguaje nos proporciona un vocabulario y las reglas para combinar palabras de ese vocabulario con el objetivo de posibilitar la comunicación. Un lenguaje de modelado es el lenguaje cuyo vocabulario y las reglas se centran en la representación conceptual y física de un sistema. Un lenguaje de modelado como UML es por tanto un lenguaje estándar para los planos del software.

El modelado proporciona una comprensión de un sistema. Nunca es suficiente un único modelo. Más bien, para comprender cualquier cosa, a menudo se necesitan múltiples modelos conectados entre sí, excepto en los sistemas más triviales. Para sistemas con gran cantidad de software, se requiere un lenguaje que cubra las diferentes vistas de la arquitectura de un sistema mientras evoluciona a través del ciclo de vida del desarrollo del software.

El vocabulario y las reglas de un lenguaje como UML indican cómo crear y leer modelos bien formados, pero no dicen qué modelos se deben crear ni cuándo se deberían crear. Esta es una tarea del proceso de desarrollo de software. Un proceso bien definido emplea para crearlos y gestionarlos, y cómo usar esos artefactos para medir y controlar el proyecto de forma global.

Para muchos desarrolladores, la distancia entre pensar en una implementación y transformarla en código es casi nula. Lo piensas, lo codificas. De hecho, algunas cosas se modelan mejor directamente en código. El texto es un medio maravilloso para escribir expresiones y algoritmos de forma concisa y casi directa.

En tales casos, el desarrollador todavía está haciendo algo de modelado, si bien lo hace de forma completamente mental. Incluso puede bloquear algunas ideas sobre papel. Sin embargo, esto plantea algunos problemas. Primero, la comunicación de esos modelos conceptuales a otros está sujeta a errores a menos que cualquier persona implicada hable el mismo lenguaje. Normalmente, los proyectos y las organizaciones desarrollan su propio lenguaje, y es difícil comprender lo que está pasando para alguien

nuevo o ajeno al grupo. Segundo, hay algunas cuestiones sobre un sistema de software que se no se pueden entender a menos que se construyan modelos que trasciendan el lenguaje de programación textual. Por ejemplo, el significado de una jerarquía de clases puede inferirse, pero no capturarse completamente, estudiando el código fuente del sistema. Tercero, si el desarrollador que escribió el código no dejó documentación escrita sobre los modelos que había en su cabeza, esa información se perderá para siempre o, como mucho, será sólo parcialmente reproducible a partir de la implementación, una vez que el desarrollador se haya marchado.

Al escribir modelos en UML se afronta el tercer problema: un modelo explícito facilita la comunicación.

Algunas cosas se modelan mejor textualmente; otras se modelan mejor de forma gráfica. En realidad, en todos los sistemas interesantes hay estructuras que trascienden lo que puede ser representado en un lenguaje de programación. UML es uno de estos lenguajes gráficos. Así afrontamos el segundo problema mencionado anteriormente.

UML es algo más que un simple montón de símbolos gráficos. Más bien, detrás de cada símbolo en la notación UML hay una semántica bien definida. De esta manera, un desarrollador puede escribir un modelo en UML, y otro desarrollador interpretar ese modelo sin ambigüedad. Así se afronta el primer problema mencionado anteriormente.

En este contexto se requiere especificar, es decir construir modelos precisos, no ambiguos y completos. En particular, UML cubre la especificación de todas las decisiones de análisis, diseño e implementación que deben realizarse al desarrollar y desplegar un sistema con gran cantidad de software.

UML no es un lenguaje de programación visual, pero sus modelos pueden conectarse de forma directa a una gran variedad de lenguajes de programación. Esto significa que es posible establecer correspondencias desde un modelo UML a un lenguaje de programación como Java, C++ o Visual Basic, o incluso a tablas en una base de datos relacional o al almacenamiento persistente en una base de datos orientada a objetos. Las cosas que se expresan mejor gráficamente también se representan gráficamente en UML, mientras que las cosas que se expresan mejor textualmente se plasman con el lenguaje de programación.

Esta correspondencia permite ingeniería directa: la generación de código a partir de un modelo UML en un lenguaje de programación. Lo contrario también es posible: se puede reconstruir un modelo en UML a partir de una implementación. La ingeniería inversa requiere, por lo tanto, herramientas que la soporten e intervención humana. La combinación de estas dos vías de generación de código y de ingeniería inversa produce una ingeniería “de ida y vuelta”, entendiendo por esto la posibilidad de trabajar en una vista gráfica o textual, mientras las herramientas mantienen la consistencia entre dos vistas.

Además de esta correspondencia directa, UML es lo suficientemente expresivo y no ambiguo como para permitir la ejecución directa de modelos, la simulación de sistemas y la instrumentación de sistemas en ejecución.

Un equipo de trabajo que trabaje bien produce toda clase de artefactos además de código ejecutable. Estos artefactos incluyen (aunque no se limita a):

- Requisitos
- Arquitectura
- Diseño

- Código fuente
- Planificación de proyectos
- Pruebas
- Prototipos
- Versiones

Dependiendo de la cultura de desarrollo, algunos de estos artefactos se tratan más o menos formalmente que otros. Tales artefactos no son sólo los entregables de un proyecto, también son críticos en el control, la medición y comunicación que requiere un sistema durante su desarrollo y después de su despliegue.

UML cubre la documentación de la arquitectura de un sistema y todos sus detalles. UML también proporciona un lenguaje para expresar requisitos y pruebas. Finalmente, UML proporciona un lenguaje para modelar las actividades de planificación de proyectos y gestión de versiones.

2.11. Bibliografia

Raccoon, L. B. S. (1995). The Chaos Model and the Chaos Life Cycle.

Pressman, R. S. (1998). INGENIERÍA DEL SOFTWARE. México.

Theoktisto, V. (1999). Modelos de Producción de Software.

Pressman, R. S. (1997). El producto y el proceso.

Corporation, M. (2004). Métodos Heterodoxos en Desarrollo de Software. **2004**.

Programación, P. d. (2003). Programación Extrema. **2003**.

Jeffries, R. (2001). What is Extreme Programming **2001**.

García, F. O. El Proceso Unificado de Ingeniería de Software.

Booch, R. y. J. El lenguaje Unificado del Modelado, Addison Wesley.

CAPÍTULO 3

Usabilidad y Diseño Centrado en el Usuario

3. Usabilidad y Diseño Centrado en el Usuario

La interacción del ser humano con las máquinas se ha convertido en una disciplina que tiene que ver con el diseño, la evaluación y la implementación de sistemas de cómputo interactivos para ser utilizados por los humanos, estudiando los fenómenos principales que los rodean, incluyendo psicológicos y sociológicos, además de los aspectos de diseño gráfico y ergonómicos.

Cada vez más los sistemas de cómputo son utilizados por usuarios que no son expertos en Computación y al colocarse frente a una pantalla para operar un sistema de cómputo deben ver y sentir "familiar" lo que está ante sus ojos, de tal manera que no necesiten de leer un "Manual del Usuario" extenso, sólo para realizar una operación simple como cortar y pegar un objeto que aparece en la pantalla, o poder navegar dentro de un sitio web sin problemas.

Podemos crear un magnífico sistema de computación, que tenga los mejores algoritmos para su funcionamiento, con las técnicas de programación más modernas y los lenguajes de programación más sofisticados o poderosos; si el usuario no comprende los elementos que se le presentan, no será un buen promotor del sistema, sino todo lo contrario. Y el futuro del sistema será incierto, por muy bueno que sea "por detrás".

También es necesario tener en cuenta las características de personas con capacidades diferentes que también serán potenciales usuarios de los sistemas de cómputo. Y no se diga cuando las máquinas no son precisamente un teclado, una pantalla y un ratón de una computadora tradicional, sino otro tipo de dispositivo interactivo, como una pantalla táctil ("touch-screen"), un cajero automático, un aparato doméstico computarizado, un reconocedor de voz o huellas digitales, entre otras muchas formas de comunicarse con las máquinas.

El objetivo principal a la hora de construir software es facilitar a los usuarios la consecución de objetivos concretos de manera sencilla. Esto es, que el sistema sea fácil de aprender y de recordar en su manejo, sea útil, es decir, contenga las funciones que los usuarios necesiten en su trabajo, y que su manejo sea fácil y agradable. En definitiva el objetivo básico es conseguir un sistema con un alto grado de "usabilidad".

3.1. Ciclo de vida de la Ingeniería de la Usabilidad

Podemos definir la usabilidad como Floria (2004):

"La medida en la cual un producto puede ser usado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso especificado".

- Por **efectividad** se entenderá la precisión y la plenitud con las que los usuarios alcanzan los objetivos especificados. A esta idea van asociadas la facilidad de aprendizaje (en la medida en que este sea lo más amplio y profundo posible), la tasa de errores del sistema y la facilidad del sistema para ser recordado (que no se olviden las funcionalidades ni sus procedimientos).

- Por **eficiencia** se entenderán los recursos empleados en relación con la precisión y la plenitud con que los usuarios alcanzan los objetivos especificados. A esta idea van asociadas la facilidad de aprendizaje (en tanto que se supone un costo en tiempo;

igualmente, si se requiere un acceso continuo a los mecanismos de ayuda del sistema), la tasa de errores del sistema y la facilidad del sistema para ser recordado (una asimilación inapropiada puede traducirse en errores de usuario).

- Por **satisfacciones** se entenderá la ausencia de incomodidad y la actitud positiva en el uso del producto. Se trata, pues, de un factor subjetivo.

La usabilidad, hace referencia, a la rapidez y facilidad con que las personas llevan a cabo sus propias tareas a través del uso del producto objeto de interés, idea que descansa en cuatro puntos según Nielsen (1993):

- **Una aproximación al usuario:** Usabilidad significa enfocarse en los usuarios. Para desarrollar un producto usable, se tienen que conocer, entender y trabajar con las personas que representan a los usuarios actuales o potenciales del producto.

- **Un amplio conocimiento del contexto de uso:** Las personas utilizan los productos para incrementar su propia productividad. Un producto se considera fácil de aprender y usar en términos del tiempo que toma el usuario para llevar a cabo su objetivo, el número de pasos que tiene que realizar para ello, y el éxito que tiene en predecir la acción apropiada para llevar acabo. Para desarrollar productos usables hay que entender los objetivos del usuario, hay que conocer los trabajos y tareas del usuario que el producto automatiza, modifica o embellece.

- **El producto ha de satisfacer las necesidades del usuario:** Los usuarios son gente ocupada intentando llevar a cabo una tarea. Se va a relacionar usabilidad con productividad y calidad. El hardware y software son las herramientas que ayudan a la gente ocupada a realizar su trabajo y a disfrutar de su ocio.

- **Son los usuarios, y no los diseñadores y los desarrolladores, los que determinan cuando un producto es fácil de usar.**

La usabilidad debe ser considerada en todo momento, desde el mismo comienzo del desarrollo hasta la implantación del sistema, producto o servicio al público.

Antes de iniciar el proyecto es esencial tener una idea acerca de las características de los usuarios y de los aspectos del producto de mayor interés y necesidad. Teniendo en cuenta estas consideraciones de forma temprana se ahorra tiempo y dinero dado que la posterior implementación de nuevos aspectos o nuevas interfaces de usuario implican un enorme esfuerzo adicional. Incluso una vez que el producto está en el mercado se debería preguntar a los usuarios acerca de sus necesidades y actitud respecto al mismo.

Para asegurar que el software tenga un alto nivel de usabilidad, existe una serie de recomendaciones para verificar puntos críticos en interfaces de usuario, a estas recomendaciones las denominamos heurísticas.

3.2. Heurísticas

Las evaluaciones heurísticas, no son más que "revisiones expertas" de la interfaz de un producto digital. Es la auditoría de sitios web por profesionales de la usabilidad, es decir sin usuarios, aplicando convenciones reconocidas de diseño de interfaz.

Estas revisiones deben ser realizadas por expertos en usabilidad cuya experiencia les permite identificar:

- Las mejores prácticas en diseño de interfaz para un contexto determinado.
- La gravedad real de los problemas detectados y cómo corregirlos.

Siguiendo en principio de que cuatro ojos ven más que dos, Nielsen (1994), realizó varios estudios que muestran las ventajas de contar con varios revisores para identificar posibles problemas. El número de revisores recomendado es de 3 a 5 expertos.

Las revisiones heurísticas se realizan adoptando la perspectiva de un usuario tipo para una interfaz. Es importante saber que cada sector suele tener sus normas o convenciones que se reflejan en el interfaz de sus sitios y aplicaciones y en la forma de trabajo de sus usuarios.

Antes de lanzarse a una crítica errónea, es necesario observar estas convenciones, sin confundirlas con errores o malas prácticas.

La revisión heurística es rápida y se puede realizar en momentos diferentes del avance de un proyecto (Loaiza 2001):

- **En fases iniciales de un proyecto:** nos permite trabajar con interfaces aún no implementadas, testando prototipos y buscando aquellos puntos que pueden ser mejorados.

- **Durante el desarrollo:** se realizan revisiones para localizar y corregir a bajo costo errores y fallas.

- **Sitios en funcionamiento:** las revisiones heurísticas son muy utilizadas como "carta de presentación" a la hora de vender servicios profesionales de consultoría de usabilidad.

Primero hemos de identificar qué objetivos se persiguen a través de interfaz e identificar las tareas principales que soporta: hacer la compra, reservar un viaje, solicitar un certificado de retenciones, hacer una transferencia, etc.

Una revisión heurística se puede acometer en dos capas (Loaiza 2001) :

- **Evaluación de alto nivel**, examinando el aspecto y comportamiento del interfaz desde un punto de vista de tareas y objetivos, procesos y pasos.

- **Evaluación en detalle:** centrada en aspectos concretos del interfaz. Pantalla por pantalla, analizando en detalle la interfaz atendiendo a puntos como el carácter autoexplicativo de la información, ubicación de la misma, controles, textos, accesos a sistema de ayuda, etc.

De estas revisiones, obtenemos un listado de problemas de usabilidad que debemos evaluar y documentar.

Una vez más, señalo que la usabilidad se aplica para usuarios concretos en contextos determinados. Para cuestiones de detalle, es importante apoyarnos en:

- **Nuestra experiencia:** el usuario suele ser muy condescendiente y hay que identificar los problemas por su verdadera gravedad.

- **Las prácticas y convenciones del sector** al que pertenece el interfaz objeto de revisión: usuarios, forma de trabajar, etc. Si hemos trabajado en varios proyectos del mismo sector observaremos que hay fallas y "tolerancias" comunes.

- **Las limitaciones de tecnología**, por ejemplo, la ausencia de integración de sistemas deriva en graves fallas de usabilidad en intranets

3.3. Principios de Usabilidad

La heurística trata de aplicar normas conversacionales a la interacción entre una persona y un sistema: trata de hacer que ambos se entiendan y trabajen juntos, para ello existen principios de usabilidad.

3.3.1. Principios de Jakob Nielsen

Jakob Nielsen es uno de los personajes más reconocidos del diseño de interfaces de sistemas interactivos.

Basándose en su experiencia profesional y en el trabajo de antecesores acuñó una serie de principios reconocidos a aplicar en el diseño de interfaces, denominadas heurísticas.

Nielsen ha promocionado tan bien estos principios que prácticamente se ha apropiado de ellos (Nielsen 2004):

1. Visibilidad del estado del sistema.
2. Encaje entre el sistema y el mundo real.
3. Libertad y control por parte del usuario.
4. Consistencia y estándares.
5. Prevención de errores.
6. Reconocimiento antes que recuerdo.
7. Flexibilidad y eficiencia en el uso.
8. Diseño estético y minimalista.
9. Ayuda a los usuarios a reconocer, diagnosticar y recuperarse de los errores.
10. Ayuda y documentación.

3.3.2. Principios de usabilidad de Larry Constantine

Uno de los pioneros en el diseño de interacción moderna, Larry Constantine, identificó en 1994 varios principios a aplicar en el desarrollo de un interfaz (Constantine & Lockwood 2003):

1. **Estructura:** organiza con significado.
2. **Simplicidad:** haz fáciles las tareas comunes.
3. **Visibilidad:** muestra toda aquella información necesaria para una tarea.
4. **Retroalimentación:** reduce la necesidad a los usuarios de recordar.
5. **Tolerancia:** permite, cancelar, deshacer, volver.
6. **Reutilización:** reduce la necesidad de los usuarios de recordar.

3.3.3. Keith Instone

Instone es otro experto en materia de usabilidad y diseño de interfaz de usuario. Durante años mantuvo su Usableweb, una colección de enlaces clasificados en materia de interfaz de usuario, desde el concepto hasta la tecnología pasando por el diseño.

Actualmente, trabaja en IBM como Arquitecto de Información dentro del equipo de Estrategia de Experiencia de Usuario.

En su informe técnico Usability Engineering on the Web, enumeró una serie de principios básicos a tener en cuenta al diseñar un interfaz (Instone):

1. Diálogo simple y natural.
2. Habla el lenguaje del usuario.
3. Minimiza la carga de memoria del usuario.
4. Consistencia.
5. Retroalimentación.
6. Salidas claramente marcadas.
7. Atajos.
8. Buenos mensajes de error.
9. Prevención de errores.
10. Ayuda y documentación.

3.3.4. Bruce "Tog" Tognazzini

Reconocido profesional compañero de Jakob Nielsen en NGroup, antes pasó por Sun y por Apple. Ha sido autor de numerosos artículos y libros acerca de diseño de interfaz de usuario.

Ask Tog, muestra sus Principios básicos para el diseño de interfaz (Tog):

1. Anticipación.
2. Autonomía.
3. Ceguera al color.
4. Consistencia.
5. Configuraciones por defecto.
6. Eficiencia del usuario.
7. Interfaces explorables.
8. Ley de Fitts.
9. Objetos de interfaz humanos.
10. Reducción de tiempos de latencia.
11. Aprendizaje.
12. Uso de metáforas.
13. Protección del trabajo del usuario.
14. Legibilidad.
15. Seguir el estado.
16. Navegación visible.

3.3.5. Ben Schneiderman

Reconocido autor, ha extraído en sus Eight Golden Rules of Interface Design, principios básicos (Genise 2002) :

1. Lucha por la consistencia.
2. Crea atajos para los usuarios frecuentes.
3. Ofrece feedback.
4. Diseña el diálogo para mostrar trabajo pendiente.
5. Ofrece una gestión sencilla de los errores.
6. Permite una fácil recuperación de acciones.
7. Soporta el control por el usuario.
8. Reduce la carga de memoria reciente en el usuario.

3.3.6. Deborah Mayhew

La Doctora Mayhew, extrajo en 1992 una serie de principios para el diseño de sistemas centrados en el usuario.

Su obra más destacada es "The Usability Engineering Lifecycle", un libro que muestra cómo incorporar la usabilidad al ciclo de vida de un producto. Leyéndolo, podemos darnos cuenta de lo lejos que nos encontramos aún de llegar al desarrollo idóneo en el mundo web.

Entre sus principios se encuentran (Mayhew 1999):

1. Compatibilidad de usuario.
2. Compatibilidad de producto.
3. Compatibilidad de tareas.
4. Compatibilidad de procesos.
5. Consistencia.
6. Familiaridad.
7. Simplicidad.
8. Manipulación directa.
9. Control.
10. Flexibilidad.
11. Sensibilidad y feedback.
12. Tecnología invisible.
13. Robustez.
14. Protección.
15. Facilidad de aprendizaje y facilidad de uso.

Concluyendo acerca del diseño de interfaces no ser que se este diseñando un videojuego o un producto experimental, debemos recordar:

1. **Diseña conversaciones:** Un interfaz es simplemente un punto de diálogo y entendimiento entre un usuario y un sistema. Asegúrate de conocer bien a los interlocutores y apóyate en normas de cortesía.

2. **Feedback:** La interacción es un diálogo. Informa al usuario de qué hacer en cada momento y qué sucede con cada una de sus acciones.

3. **Sencillez:** no seas barroco al diseñar el aspecto de un interfaz. Un interfaz sencillo será más rápido y fácil de usar.

4. **Consistencia y reutilización:** No reinventes la rueda. El usuario ya llega aprendido por su uso de otros sitios, aplicaciones y sistemas operativos, no le hagas "estudiar" otra vez. Sigue las convenciones establecidas por las plataformas y las buenas prácticas del diseño de interfaz.

5. **Naturalidad:** impresiona por la simpleza. No utilices terminologías técnicas ni pedantes.

6. **Errores:** evítalos y en caso de que se produzcan permite una gestión sencilla de los mismos mediante explicaciones claras indicando qué hacer. Evita a toda costa culpar al usuario de cualquier error o utilizar tecnicismos al describirlos.

7. **Oculta la tecnología:** el usuario ni sabe ni tiene porqué saber tecnología para usar un sitio. No conoce qué es flash o un applet. Haz que las cosas funcionen de manera limpia y fluida.

8. **No hagas trabajar al usuario:** espérale con las acciones más habituales en cada pantalla, tanto si es novato como experto.

9. **Interfaz autoexplicativo:** una pantalla debe explicar por si misma al usuario dónde se encuentra, qué puede hacer, qué no debe hacer.

10. **Cuando todo falla, ofrece apoyo:** en la web esto es especialmente cierto. Hasta que el usuario no se bloquea no acude a sistemas de ayuda o de apoyo. Desarrolla sistemas de ayuda fácilmente accesibles y contextualizados a la tarea que esté desarrollando el usuario. En tareas críticas, ofrece vías de contacto.

3.4. Diseño centrado en el usuario

La idea del diseño centrado en el usuario (UCD) propone que los diseñadores comprendan el contexto de uso: esto significa un profundo entendimiento del usuario, del entorno en el que se desarrolla el trabajo y las tareas del usuario. Además se contemplan los aspectos de mantenimiento del producto o sistema, asistencia al usuario y documentación. En contraste, la expresión “diseño centrado en el uso", sugiere que el diseñador sólo necesita concentrarse en las tareas de usuario, lo que parece que hace entender al usuario y al contexto como algo menos importante.

En el estándar ISO 13407 Usability Net(2003), de título “*Human centred design processes for interactive systems" (Procesos de diseño centrado en el hombre para sistemas interactivos)*, se hace uso del término “centrado en el usuario”. Sin embargo hay quien indica que en la práctica, se plantean muy pocas diferencias entra ambas aproximaciones, y entiende el término “centrado en el uso” como una cuestión de marketing.

El diseño centrado en el uso no involucra al usuario en el proceso de diseño, mientras que el diseño centrado en el usuario apunta a una presencia activa directa de este en el proceso de desarrollo del mismo.

Otro significado, ciertamente más interesante y casi opuesto al anteriormente, de la expresión “centrado en el uso” es el de los modos y posibilidades de uso. Así, en el estudio de una interacción particular producto-usuario se requerirán de dos fuentes de datos básicas: las precedentes de los estudios ergonómicos (antropometría, funciones fisiológicas,...) y las de las actividades de los individuos en su uso del producto como gran condicionante de la funcionalidad del mismo. Lo verdaderamente importante es, el modo en el que el usuario interactúa con un producto y, en particular, con un prototipo, durante los diversos experimentos en el proceso de desarrollo.

Se podrían proponer cuatro principios básicos en la aproximación al Diseño Centrado en el Usuario:

1. La implicación activa de los usuarios y un claro entendimiento de los requerimientos y tareas de usuario.
2. Un reparto apropiado de funciones entre los usuarios y la tecnología.
3. La iteración de las soluciones de diseño.
4. Un diseño multidisciplinar.
    - Usuario final.
    - Comprador, cliente, representante del usuario.
    - Especialista en el ámbito del sistema, analista de negocios.
    - Analista de sistemas, ingeniero de sistemas, programador.
    - Especialistas de mercado, comerciales.
    - Diseñador de interfaz de usuario, diseñador visual.
    - Experto en ergonomía y factores humanos, especialista en la interacción hombre-máquina.
    - Escritores técnicos, personal para entrenamiento, asistencia y mantenimiento.

Se diferencia el diseño participativo del diseño centrado en el usuario en el que mientras éste considera al usuario tan sólo al comienzo y al final del proceso de diseño, aquel lo integra por completo en el proceso de desarrollo. La distinción es importante porque las gestiones de los respectivos procesos de desarrollo son completamente diferentes y especialmente más complejas en el caso del diseño participativo.

3.5. Principios del Diseño Centrado en el Usuario

El diseño, sea cual sea el objeto del mismo, tiene que basarse en el usuario, y el usuario puede ser cualquier individuo. Es claro ver que los principios del Diseño Centrado en el Usuario no son más que una reformulación de los principios más elementales de la Ergonomía Clásica (Tabla 2) y de aquellos se derivan, en general, las guías de accesibilidad:

a) El control de la situación debe estar en manos del usuario:

- Ha de ser el usuario quien inicie las acciones y controle las tareas.
- El usuario ha de tener la oportunidad de personalizar la interfaz.
- El sistema debe ser lo más interactivo posible, facilitando el cambio y gestión de sus modos.

b) Es preciso un planteamiento directo:

- El usuario ha de comprobar cómo sus acciones afectan a la salida del sistema.
- La accesibilidad de la información y de las opciones van a reducir la carga mental de trabajo del usuario.
- Las metáforas familiares proporcionan una interfaz intuitiva.
- Se asocia un significado con un objeto mejor que un comando, siempre y cuando la asociación resulte apropiada.

Tabla 1 Decálogo de la Ergonomía (Floria 2004)

Decálogo de la Ergonomía:

1. El trabajo, los productos y los servicios deben adaptarse al usuario, respetando sus limitaciones fisiológicas, psicológicas y sociales.
2. Siempre hay que tener en cuenta las diferencias poblacionales, la mejora para los extremos acostumbra a repercutir positivamente para toda la población.
3. El bienestar en el trabajo y el ocio no es fácilmente definible, es el punto de encuentro entre los requerimientos del producto y los deseos del usuario.
4. El usuario es creador; por lo tanto, hay que facilitar su creatividad mediante:
    - una interacción armónica con el entorno
    - una mejora en la seguridad en trabajo y ocio
    - una disminución de la carga física y nerviosa
    - la creación de puestos de trabajo con elevado contenido, reduciendo la infracarga.
5. El buen funcionamiento del sistema se basa en buenas condiciones de trabajo y uso.
6. El bienestar en el trabajo y el ocio no es un lujo: es una necesidad.
7. El contenido del trabajo es también una parte de las condiciones del trabajo y repercute sobre la salud del usuario.
8. La participación de los usuarios en la organización del trabajo mejora el rendimiento de los sistemas productivos.
9. El usuario es el factor más importante en el diseño, montaje, funcionamiento, mantenimiento, uso y reciclaje de cualquier producto o servicio.
10. El análisis exhaustivo en la fase del proyecto de las capacidades fisiológicas, psíquicas y sociales de los usuarios repercutirá en una interacción armónica del sistema Hombre-Máquina que incidirá positivamente en la seguridad, la productividad, la calidad y el buen uso.

La consistencia es parte indispensable en el diseño:

- Se ha de facilitar la aplicación de los conocimientos adquiridos de forma previa al desarrollo de nuevas tareas, lo que a su vez se va a traducir en un aprendizaje rápido.
- Consistencia y estabilidad se van a traducir en facilidad de uso.
- Ha de darse la consistencia dentro de un producto (el mismo comando desarrollaría funciones que el usuario interpreta como similares), en un entorno (se efectúa una adopción de convenciones para todo el conjunto), con las metáforas (si un comportamiento particular es más característico de un objeto diferente que el que su metáfora implica, el usuario puede tener dificultad en asociar comportamiento y objeto).

Hay que posibilitar la recuperación de errores:

- El diseño minimiza los riesgos y las consecuencias adversas de las acciones accidentales o involuntarias.
- Hay que posibilitar el descubrimiento interactivo y el aprendizaje ensayo - error.
- Hay que posibilitar la reversibilidad y la recuperabilidad de las acciones.
- Hay que contemplar los potenciales errores de los usuarios.

Retroalimentación apropiada por el sistema:

- Es precisa una respuesta apropiada a las acciones del usuario por parte del sistema.
- Tal respuesta ha de ser inevitablemente de complejidad variable y ha de darse en un tiempo apropiado.
- El estado de un sistema (esperando entrada, comprobando, transfiriendo datos,...) debería estar siempre disponible para el usuario.

No se puede descuidar la estética:

- Determinados atributos visuales o auditivos concentran la atención del usuario en la tarea que está desarrollando.
- Es preciso proporcionar un entorno agradable que contribuya al entendimiento por parte del usuario de la información presentada.

El diseño debe caracterizarse por su simplicidad:

- La interfaz ha de ser simple (que no simplista), fácil de aprender y usar, con funcionalidades accesibles y bien definidas.
- El uso del diseño ha de ser fácil de entender, independientemente de la experiencia, conocimiento, capacidades lingüísticas o nivel de concentración del usuario.
- Hay que controlar la información que se explicita, que se ha de reducir al mínimo necesario.
- El diseño ha de comunicar la información necesaria al usuario de forma efectiva, independientemente de las condiciones ambientales o de las capacidades sensoriales del mismo.

Es fundamental seguir una rigurosa metodología de diseño:

- Una actitud centrada en el usuario, en etapas iniciales y durante el diseño, así como una rigurosa metodología que contemple los principios que se tratan.

El equipo de diseño debe ser equilibrado:

- Se han de cubrir todos los aspectos: desarrollo, expresión, representación, factores humanos, usabilidad.
- El trabajo en equipo ha de caracterizarse por la posibilidad de una comunicación e interacción rápida y efectiva.

Se distinguen cuatro partes en el proceso de diseño:

- Definición clara de los objetivos, entendiendo a los usuarios y contemplando factores como la edad, la experiencia, las limitaciones físicas, las necesidades más especiales, el entorno de trabajo, las influencias sociales y culturales. Hay que definir el marco de trabajo conceptual para presentar el producto en cuestión con el conocimiento y la experiencia de trabajo sobre el objetivo; a partir de ahí, procede una documentación apropiada a este estado.

- Comunicación del diseño mediante el prototipado y establecimiento de un flujo de tareas. Se puede tratar de incluir más aspectos y comprobar la reacción a los mismos de los usuarios objetivo o tratar de centrarse en los detalles de dichos aspectos, en su funcionalidad.

- Mediante el test, en el proceso de diseño, la participación del usuario proporciona la inestimable ayuda de determinar en qué medida el producto se está ajustando a las necesidades y a las expectativas creadas. No se trata tanto de evaluar la eficiencia de las tareas y los posibles errores en el diseño, sino de conocer las percepciones del usuario, su satisfacción, sus preguntas, sus problemas, etc.

- Después del test va a ser preciso el rediseño en mayor o menor medida, tras el cual inevitablemente, es preciso de nuevo el test, volviendo así a iniciar el ciclo.

Son indispensables las consideraciones de usabilidad en el proceso de diseño:

- En todas las etapas del proceso de diseño, se aplicarán las técnicas de evaluación de la usabilidad que se estimen más apropiadas.

Hay que entender al usuario:

- Las diferencias en los modos de aprendizaje reflejan múltiples variantes que se manifiestan continuamente desde ligeras preferencias hasta profundas necesidades. Así, es preciso acomodar esta diversidad mediante representaciones alternativas de la información clave. A partir de diferentes preferencias y necesidades (originadas por el propósito de la actividad de trabajo o aprendizaje y, por supuesto, de la naturaleza de los propios usuarios) se puede seleccionar el medio de representación más apropiado o conseguir la información a través de una amplia gama de medios de representación.

- De la misma forma que ningún modo de representación se puede ajustar a todos los usuarios, ningún modo de expresión lo hará tampoco. La forma habitual de expresión ha sido texto impreso, pero otras opciones artísticas, fotográficas, musicales, el vídeo, la animación, etc. resultan una exitosa forma de comunicar ideas para ciertos individuos. Es preciso asumir esta diversidad ofreciendo múltiples opciones para la expresión y el control. Las preferencias y necesidades particulares siempre encontrarán, así, medios, apoyos y opciones que permitan al usuario mostrar su conocimiento de la forma que les resulte más efectiva.

- No cabe la menor duda de que para abordar una tarea, sea el conocimiento y uso de un determinado producto en el contexto que nos ocupa, son precisas unas dosis adecuadas de confianza, entusiasmo e intencionalidad. La misma tarea que influye en el carácter competitivo y en la confianza de un usuario de forma positiva, puede llevar al aburrimiento y a la frustración en otros. La motivación puede venir porque la materia en cuestión resulta fascinante, constituye un reto, el proceso de aprendizaje resulta satisfactorio, la circunstancia de la novedad resulta muy atractiva, las posibilidades de mejorar en el desarrollo de la tarea son enormes por las características de los elementos involucrados, se puede establecer un paralelismo con la vida real. Así, las estrategias de aprendizaje deben soportar diferentes niveles de capacidad, preferencias e intereses, proporcionando opciones flexibles.

Hay que realizar renuncias en el diseño:

- Cada aspecto adicional que se incluye en el sistema está afectando potencialmente a la complejidad, estabilidad, mantenimiento, capacidad de acción, costos de apoyo, etc., es decir, afecta significativamente el proyecto.

- Siempre habrá consideraciones de marketing que afectan a la forma del producto y que pueden condicionar, en un determinado momento, un rediseño a mayor o menor escala.

3.6. Procesos para el desarrollo centrado en el humano

Dentro de la perspectiva genérica que se ha decidido adoptar en el enfoque de la materia se puede entrar en una dimensión más práctica. Los principios de Diseño Centrado en el Usuario que se acaban de citar constituyen ya por sí mismos una herramienta interesante y la vigilancia de su cumplimiento puede aportar importantes beneficios a un diseño en curso. Sin embargo, tales consideraciones han de estar necesariamente enmarcadas en una filosofía más amplia, en un conjunto de procesos relacionados entre sí, fundamentalmente por la información que generan el modo en el que la gestionan. Asimilando la terminología de la norma ISO 13407 (diseño centrado en el usuario para sistemas iterativos) vamos a hablar de los procesos que corresponden a un desarrollo centrado en el hombre, tal y como muestra en la tabla 3.

Tabla 2 Procesos para el desarrollo centrado en el hombre (DCH)

DCH 1	Asegurar el contenido DCH en la estrategia de sistemas.
DCH 2	Planificar y dirigir el proceso DCH.
DCH 3	Especificar los requerimientos organizacionales y de los implicados.
DCH 4	Entender y especificar el contexto de uso.
DCH 5	Generar soluciones de diseño.
DCH 6	Evaluar los diseños contra los requerimientos.
DCH 7	Presentar y operar el sistema.

Procesos para el desarrollo centrado en el hombre (DCH) (Usability Maturity Model: Human Centredness Scale; Annex 4; J. Earthy; Versión 1.2 27/12/1998)

Tabla 3 Seis niveles de capacidad en la realización de los procesos

Nivel 0	-**Incompleto** (incapaz de desarrollar proceso).
Nivel 1	-**Realizado** (algunos individuos desarrollan procesos).
Nivel 2	-**Dirigido** (los requerimientos en calidad, tiempo y recursos para cada proceso son conocidos y controlados).
Nivel 3	-**Establecido** (el proceso se desarrolla tal y como se ha especificado por la organización y los recursos están definidos).
Nivel 4	-**Predecible** (la realización del proceso entra dentro de los límites de calidad y recursos que se predijeron).
Nivel 5	-**Optimización** (la organización puede ajustar de forma fiable el proceso a los requerimientos particulares).

Seis niveles de capacidad en la realización de los procesos (Usability Maturity Model: Human Centredness Scale; Annex 4; J. Earthy; Versión 1.2 27/12/1998)

De la misma forma que se habló de niveles de madurez respecto de la usabilidad con anterioridad, se puede hablar ahora, desde un punto de vista más práctico, de la capacidad de las organizaciones para llevar a cabo estos procesos, tal y como indica la tabla anterior. Los procesos en cuestión van a ser analizados seguidamente.

3.6.1. Asegurar el contenido DCH en la estrategia de sistemas (DCH 1)

El objetivo de este proceso es establecer y mantener un enfoque en los aspectos relacionados con usuarios y demás implicados en cada parte de la organización que trata con los mercados, conceptos, desarrollo y asistencia al sistema.

Como resultado de una implementación exitosa de este proceso:

- El departamento de marketing tendrá en cuenta cuestiones sobre usabilidad, ergonomía y otros aspectos sociales y tecnológicos.
- Los sistemas se dirigirán a satisfacer las necesidades y expectativas de los usuarios.
- La planificación original tendrá en cuenta los requerimientos de los implicados y de organización para el establecimiento de la estrategia de sistemas.
- Los sistemas admitirán más positivamente los cambios en los usuarios (necesidades, tareas, contexto,...).
- La empresa será más receptiva a los cambios en sus usuarios.
- Los sistemas difícilmente serán rechazados en el mercado.

Tabla 4 Asegurar el contenido DCH en la estrategia de sistemas

Entrada	Salida
Estrategia de la compañía Encuestas de mercado Pronósticos tecnológicos Pronósticos expertos Metodologías de estrategia DCH	Visión del sistema o producto Especificación original Demandas sociales y sociotécnicas de los grupos objetivo Predicción de los contextos de uso Valoraciones sobre el estado del mercado Análisis de tendencias Proceso contable del sistema Estrategia de sistemas centrada en el hombre

Asegurar el contenido DCH en la estrategia de sistemas (Usability Maturity Model: Human Centredness Scale; Annex 2, Tabla 1; J. Earthy; Versión 1.2 27/12/1998)

3.6.2. Planificar y dirigir el proceso DCH (DCH 2)

El objetivo de este proceso es especificar el modo en el que las actividades centradas en el hombre encajan en el ciclo de vida del sistema completo y en la empresa.

Se ha de desarrollar un plan para especificar el modo en el que las actividades centradas en el hombre se ajustan en el proceso global de desarrollo del sistema. El plan debería identificar:

a. Las actividades del proceso de diseño centrado en el hombre: entender e identificar el contexto de uso, especificar los requerimientos organizacionales y de usuario, generando prototipos y evaluando los diseños de acuerdo con los criterios de usuario.

b. Procedimientos para la integración de estas actividades con otras vinculadas al desarrollo del sistema; esto es, análisis, diseño y test.

c. Los individuos y el responsable de la organización u organizaciones para las actividades de diseño centrado en el hombre así como el espectro de capacidades y puntos de vista que proporcionan.

d. Procedimientos efectivos para el establecimiento de retroalimentación y comunicación sobre las actividades de diseño centrado en el hombre en la medida en que afecta a otras actividades de diseño, así como los métodos para documentar estas actividades.

e. Referencias apropiadas para actividades centradas en el hombre integradas en el diseño global y el proceso de desarrollo.

f. Espacios de tiempo apropiados que permitan que la retroalimentación, así como los posibles cambios de diseño, sea incorporada en la planificación del proyecto.

Para dirigir el proceso de diseño iterativo, será preciso registrar los resultados de las actividades:

- Hacer uso del conocimiento existente para generar propuestas con entradas de carácter multidisciplinar.
- Concretar las soluciones de diseño a través de simulaciones, modelos y maquetas.
- Presentar las soluciones de diseño a los usuarios y permitirles la realización de tareas reales o simuladas.
- Modificar el diseño en respuesta a la retroalimentación del usuario hasta que se alcancen las metas de diseño centrado en el hombre.

Estos registros pueden ser completamente documentales o pueden incluir el artefacto de diseño, por ejemplo, algún prototipo de hardware o software. Incluye:

a. Las fuentes del conocimiento existente y los estándares utilizados, con una indicación sobre cómo han sido incorporados (o porqué no han sido considerados, caso de que sea apropiado).

b. Los pasos adoptados para asegurar que el prototipo cubre los requerimientos clave y opera según una práctica correcta.

c. La naturaleza de los problemas identificados y los cambios procedentes en el diseño.

Como resultado de una implementación exitosa de este proceso, por otra parte:

- La planificación del proyecto permitirá la iteración y la incorporación de la retroalimentación del usuario.
- Se asignarán los recursos para una comunicación efectiva entre los participantes del equipo de diseño (multidisciplinar).
- Se solucionaran los conflictos potenciales entre el concepto *centrado en el hombre.*
- Los procesos centrados en el hombre se incorporarán a los sistemas, procedimientos y estándares de calidad.
- El concepto *centrado en el hombre* será apoyado y promovido dentro de la organización.

3.6.3. Especificar los requerimientos organizacionales y de los implicados (DCH 3)

El objetivo de este proceso es establecer los requerimientos de la organización y demás partes interesadas en el sistema. Este proceso toma en consideración las necesidades, competencias y entornos de trabajo de cada implicado en el sistema de forma relevante.

Como resultado de una implementación exitosa de este proceso, se definirá:

a. Realización requerida del nuevo sistema frente a objetivos operacionales y financieros.

b. Requerimientos legislativos o estatutarios relevantes, incluyendo seguridad y salud.

c. Cooperación y comunicación entre usuarios y otras partes relevantes.

d. Los trabajos del usuario (incluyendo el reparto de tareas, el bienestar de los usuarios y su motivación).

e. La realización de tareas.

f. El trabajo de diseño y su organización.

g. La dirección y gestión de modificaciones, incluyendo entrenamiento y personal implicado.

h. Viabilidad de la operación y del mantenimiento.

i. Diseño de la interfaz hombre - máquina y diseño del puesto de trabajo.

La especificación de los requerimientos organizacionales y de usuario debería, además:

a. Identificar el rango de usuarios relevantes y demás personal implicado en el proceso de diseño.

b. Proporcionar una situación clara en cuanto a las metas del diseño centrado en el hombre.

c. Establecer prioridades apropiadas para los diferentes requerimientos.

d. Proporcionar un conjunto de criterios susceptibles de ser medidos contra los que pueda ser testado el diseño en curso.

e. Confirmación por los usuarios o aquellos que representen sus intereses en el proceso.

f. Incluir cualquier requerimiento legislativo o estatutario.

g. Estar debidamente documentados.

Tabla 5 Especificar los requerimientos organizacionales y de los implicados

Entrada	Salida
Perspectiva del proyecto Representantes del usuario Instrucciones de trabajo Legislación Estándares industriales Estrategia de sistemas Contexto de uso Sistemas competidores	Rango y relevancia de los usuarios y demás personal en el diseño Evaluación de riesgos Establecimiento de las metas de diseño centrado en el hombre Especificación de los requerimientos de usuarios y demás implicados Especificación de los requerimientos de organización Prioridades para los diferentes requerimientos Metas de usabilidad específicas y susceptibles de ser medidas Referencias contra las que testar el diseño Lista de requerimientos estatutarios o legislativos Fuentes de las que derivan los requerimientos de organización y de usuario

Especificar los requerimientos organizacionales y de los implicados (Usability Maturity Model: Human Centredness Scale; Annex 2, Tabla 3; J. Earthy; Versión 1.2 27/12/1998)

3.6.4 Entender y especificar el contexto de uso (DCH 4)

El objetivo de este proceso es identificar, clarificar y registrar las características de los implicados, sus tareas y el entorno físico y organizacional en el que el sistema operará.

La descripción del contexto de uso debería:

a. Especificar el rango de los usuarios, tareas y entornos pretendidos en un detalle suficiente para asistir la actividad de diseño.

b. Derivar de fuentes apropiadas.

c. Ser confirmado por los usuarios o, caso de no estar disponibles, por quienes representan sus intereses del proceso.

d. Estar adecuadamente documentada.

e. Ser puesta a disposición del equipo de diseño en intervalos de tiempo apropiados y en formatos aptos para las actividades de diseño.

Como resultado de una implementación exitosa de este proceso, se conseguirá:

- La definición de las características de los usuarios objetivo.
- La definición de las tareas que los usuarios han de realizar.
- La definición de la organización y el entorno en el que los sistemas han de ser utilizados.
- La disponibilidad del contexto de uso para su utilización de todos los puntos relevantes del ciclo de vida.

Tabla 6 Entender y especificar el contexto de uso

Entrada	Salida
Requerimientos del sistema Especificación de los requerimientos de usuario y demás implicados Especificación de los requerimientos de organización Perspectiva del proyecto Representantes de los usuarios Instrucciones de trabajo Tiempo y formato de la información sobre el contexto de uso que se proporcionará al equipo de desarrollo	Especificación del rango pretendido de usuarios, tareas y entornos Información sobre los implicados Información sobre el usuario Información sobre la tarea Análisis organizacional Fuentes de las que deriva la información sobre el contexto de uso

Entender y especificar el contexto de uso (Usability Maturity Model: Human Centredness Scale; Annex 2, Tabla 4; J. Earthy; Versión 1.2 27/12/1998)

3.6.5 Generar soluciones de diseño (DCH 5)

El objetivo de este proceso es el de generar potenciales soluciones de diseño a partir de la experiencia, los conocimientos de los participantes y los resultados del análisis del contexto de uso.

Como resultado de una implementación exitosa de este proceso:

- Se considerará en el diseño el sistema completo con todos sus partes integrantes
- Todos los componentes del sistema tienen en cuenta las características y necesidades del usuario.
- Las características y las necesidades del usuario serán tenidas en cuenta en el diseño del sistema.
- Se integrará en el sistema el conocimiento existente en disciplinas como ingeniería de sistemas, ergonomía, psicología cognitiva y otras de relevancia.
- La comunicación entre los implicados mejorará porque las decisiones en materia de diseño serán más explícitas.
- El equipo de desarrollo tendrá la posibilidad de explorar diversos conceptos de diseño antes de optar por uno.
- Se incorporará en las etapas tempranas del proceso de desarrollo la retroalimentación procedente de usuarios finales y demás implicados.
- Será posible evaluar varias iteraciones de un diseño para diseños alternativos.
- Se diseñará la interfaz entre el usuario y el sistema.
- Se desarrollará el entrenamiento y la asistencia al usuario.

Tabla 7 Generar soluciones de diseño

Entrada	Salida
Requerimientos del sistema Requerimientos de usuario y demás implicados Requerimientos de organización Contexto de uso Metas de usabilidad susceptibles de ser medidas Requerimientos ergonómicos Estándares y guías Pautas de estilo Experiencia previa Retroalimentación procedente de evaluaciones	Fuentes del conocimiento existente y de los estándares utilizados, con una indicación sobre cómo han sido incorporados (o el motivo por el que no han sido seguidos, en caso de que resulte apropiado) Especificación de la interacción del usuario Detalle del diálogo, aspecto y sensaciones Disposición y otras cuestiones sobre interfaces de usuario Simulaciones de la especificación Prototipado del sistema o de alguna o algunas de sus partes Modelo de tareas Asignación de funciones Diseño de la operación del sistema Evidencia de la revisión conforme a los resultados de las evaluaciones Planificación del entrenamiento para usuarios y personal encargado de su soporte y mantenimiento Definición de la asistencia al usuario en relación al sistema Lista de estándares y modo en el que se han aplicado Justificación de las desviaciones de los estándares para llegar a requerimientos específicos Informe sobre el modo en el que los conflictos entre los requerimientos del diseño y los conocimientos previos fueron solventados en el diseño Mecanismos de retroalimentación y utilización de resultados en otras actividades de diseño Pasos seguidos para asegurar que el prototipo o prototipos cubrieron los requerimientos clave y operaron de forma apropiada

Generar soluciones de diseño (Usability Maturity Model: Human Centredness Scale; Annex 2, Tabla 5; J. Earthy; Versión 1.2 27/12/1998)

La utilización de simulaciones, modelos y maquetas y cualquier otra forma de prototipo en general permite a los diseñadores una comunicación mucho más efectiva con los usuarios y reduce la necesidad y el costo del rediseño que puede ser preciso cundo los productos han de ser revisados con posterioridad, en algunos casos tras su lanzamiento y comercialización entre clientes reales.

Los beneficios son los siguientes:

a. Hacer las decisiones de diseño más explícitas (lo que capacita a los miembros del equipo a comunicarse entre ellos desde etapas tempranas en el proceso de diseño).

b. Permite a los diseñadores explorar diversos conceptos de diseño antes de optar por uno en particular.

c. Posibilita la incorporación de la retroalimentación del usuario en las etapas tempranas del proceso de desarrollo.

d. Posibilita la evaluación de diversas iteraciones de un diseño y de diseños alternativos.

e. Mejora la calidad y el carácter de conjunto de la especificación funcional del diseño.

3.6.6 Evaluar los diseños contra los requerimientos (DCH 6)

El objetivo de este proceso es el de reunir retroalimentación sobre el diseño en desarrollo de los usuarios finales y otras fuentes representativas.

Como resultado de una implementación exitosa de este proceso:

- La retroalimentación proporcionada contribuye a la mejora del proceso.
- Se evaluará si los objetivos sobre la organización y los implicados han sido logrados o no.
- Se monitorizará el uso a largo plazo del sistema.

Caso de que el objetivo de la evaluación sea identificar mejoras en el sistema (evaluación formativa), una implementación exitosa del proceso reflejará:

- Los problemas potenciales y la perspectiva en cuanto a las mejoras en: tecnología, elementos de asistencia, entorno físico y organizacional y entrenamiento.
- La opción de diseño que mejor encaja en los requerimientos funcionales, organizacionales, y de los implicados.
- Retroalimentación y requerimientos adicionales de los usuarios.

Caso de que el objetivo de la evaluación sea el de comprobar si los objetivos han sido logrados (evaluación informativa), una implementación exitosa del proceso reflejará:

- El modo en que el sistema logra las metas organizacionales.
- Que un diseño particular cumple los requerimientos centrados en el hombre.
- Una conformidad con los requerimientos nacionales, internacionales y/o estatutarios.

La evaluación, es un paso esencial en el diseño centrado en el hombre y debería tener lugar en todas las etapas del ciclo de vida del diseño. La evaluación puede ser utilizada para:

a. Proporcionar la retroalimentación que puede ser utilizada para mejorar el diseño.
b. Evaluar si los objetivos organizacionales y de usuario han sido conseguidos.
c. Monitorización del uso a largo plazo del producto o sistema.

En las etapas tempranas del proceso de diseño, el énfasis radica en la obtención de retroalimentación que puede ser utilizada para guiar el diseño, mientras que, más tarde, cuando se ha conseguido un prototipo más completo es posible medir si los objetivos organizacionales y de usuario han sido alcanzados.

En las etapas tempranas del proceso de desarrollo y el proceso de diseño, los cambios son relativamente baratos. Cuanto más haya progresado el proceso y cuanto más completamente está definido, más cara será la introducción de cambios. Así pues, es importante comenzar la evaluación lo más pronto posible.

Un plan de evaluación:

Debería desarrollarse un plan de evaluación que identifique los aspectos relevantes sobre:

a. Las metas del diseño centrado en el hombre.

b. El responsable de la evaluación.

c. Las partes del sistema a ser evaluadas y el modo en el que han de ser evaluadas, por ejemplo el uso de los escenarios de test, maquetas, prototipos.

d. El modo en el que va a ser realizada la evaluación y los procedimientos para desarrollar los tests.

e. Recursos requeridos para la evaluación y el análisis de los resultados y el acceso a los usuarios (caso que sea necesario).

f. Establecimiento de horarios sobre las actividades de evaluación y su relación con la planificación del proyecto.

g. Retroalimentación y uso de los resultados en otras actividades de diseño.

Las técnicas de evaluación varían en el grado de formalidad, rigor e implicación del usuario, dependiendo del entorno en el que la evaluación es conducida. La elección viene determinada por limitaciones financieras y temporales, la etapa en el ciclo de desarrollo y la naturaleza del sistema bajo desarrollo.

Interesa proporcionar retroalimentación para el diseño:

Las evaluaciones deberían tener lugar en todas las etapas en el ciclo de vida del sistema para influir en el sistema a generar. Las metas particulares de la evaluación deberían reflejar uno o más de los siguientes objetivos:

a. Evaluar en qué medida el sistema alcanza los objetivos organizacionales.

b. Diagnosticar problemas potenciales e identificar las necesidades para mejoras en la interfaz, el material de asistencia, el entorno del lugar de trabajo o los objetivos de entrenamiento.

c. Seleccionar la opción de diseño que mejor se ajusta a los requerimientos funcionales y de usuario.

d. Captar la retroalimentación y los requerimientos adicionales de usuario.

Información de los resultados:

Con el propósito de dirigir y gestionar el progreso en el diseño iterativo, los resultados de las evaluaciones deberían ser registrados de una forma sistemática.

Conviene aportar evidencia apropiada de que:

a. Un número apropiado de usuarios tomaron partes en el test y que estos usuarios fueron representativos de aquellos identificados en el contexto de uso.

b. Se testaron los objetivos centrados en el hombre más destacados.

c. Se utilizaron métodos básicos para el test y la reunión de datos.

d. Se dio un tratamiento apropiado de los resultados de test.

e. Las condiciones del test fueron apropiadas.

Se consideran tres tipos de información para la evaluación que pueden resultar de utilidad durante el proceso de diseño, dependiendo de si el propósito de la evaluación es proporcionar retroalimentación para el diseño, testar contra estándares específicos o proporcionar la evidencia de la consecución de las metas centradas en el hombre, por ejemplo, en términos de usabilidad o salud y seguridad de usuario.

El informe de la retroalimentación hacia el diseño:

- Debería tener lugar en un momento apropiado en el proceso de desarrollo.
- Debería basarse en fuentes apropiadas de evaluación (usuarios, revisiones de diseño,...).
- Debería proporcionar retroalimentación de diseño en un formato que asista a las decisiones de diseño.
- Debería resultar en cambios visibles en el sistema, siempre que sea apropiado.

El informe de los tests del diseño contra estándares específicos:

- Debería identificar los estándares relevantes y proporcionar razones para su uso.
- Debería proporcionar evidencia de que la evaluación fue conducida por una persona competente haciendo uso de procedimientos apropiados.
- Debería proporcionar evidencia de que un conjunto suficiente de partes del sistema fueron testadas para proporcionar resultados significativos para el sistema como un conjunto.
- Debería informar sobre el modo en el que las no conformidades son tratadas en el diseño.
- Debería justificar cualquier desviación de los estándares aplicables.

El informe del test de usuario:

- Debería definir el contexto de uso utilizado para la evaluación.
- Debería proporcionar información sobre los requerimientos organizacionales y de usuario.
- Debería describir el producto testado y su estado (prototipo en evolución).
- Debería describir las medidas realizadas y los usuarios y métodos utilizados.
- Debería contener los resultados con un análisis estadístico relevante.
- Debería indicar la decisión de conformidad en relación con los requerimientos.

3.6.7 Presentar y operar el sistema (DCH 7)

El objetivo de este proceso es el de establecer las cuestiones hombre - sistema concernientes a la asistencia e implementación del sistema.

Como resultado de una implementación exitosa de este proceso:

- Se considerará en el proyecto las necesidades de todos los implicados en el sistema.
- Se especificará la dirección y gestión de cambios, incluyendo las responsabilidades de usuarios y desarrolladores.
- Se dirigen los requerimientos de usuarios finales, encargados de mantenimiento y otros implicados.
- Hay consenso en cuanto a los procedimientos sobre seguridad e higiene.
- Se consideran las variantes del sistema.
- Se reúnen las reacciones de usuario y se informa de los cambios resultantes en el sistema a los implicados.

La usabilidad nos ofrece muchas ventajas, las cuales nos permiten la implementación de un software más funcional desde que inicia su desarrollo, hasta que concluye, además de que permite hacer pruebas desde el principio, las cuales impiden un retrabajo que finalmente afecta los recursos y retraza la fecha de entrega del producto, de tal manera que se puede alcanzar un alto grado de usabilidad adaptando el proceso de desarrollo a los principios del Diseño Centrado en el Usuario.

En el caso de software de gestión, los usuarios reales son la referencia constante para el diseño de la interfaz. En todas las etapas del ciclo de desarrollo se recurre a ellos para determinar perfiles, establecer el lenguaje, diseñar procesos, y evaluar el diseño de la aplicación. Este enfoque está orientado a evitar que la interfaz de usuario refleje el criterio de interacción de los analistas y programadores, sino que, por el contrario, se adapte a la forma de interactuar de los usuarios.

Los beneficios que reporta son los siguientes:

- Una reducción de tiempos y costos de desarrollo, porque su implementación asegura que el software se ajuste a las necesidades del usuario desde la primera versión, evitando los altos costos de rediseño.

- Reducción de costos de entrenamiento, soporte de usuario, tareas de mantenimiento por errores del usuario.

Por consiguiente la usabilidad es una disciplina en constante evolución, íntimamente ligada a las disciplinas que la utilizan. No puede encontrarse una teoría formalizada de la usabilidad, ni existe un acuerdo completo sobre las técnicas a utilizar y los parámetros a considerar en las mismas. Las principales tendencias de investigación y desarrollo en usabilidad de sedes web dependen, en el momento actual, del resultado de estudios y observaciones llevadas a cabo sobre grupos de usuarios, cuyos resultados han sido trasladados como reglas de aplicación al campo del diseño de interfaces para el web. La mayor parte de trabajos y escritos sobre usabilidad muestran la adopción de una técnicas de análisis, sobre un grupo de usuarios (bien sea un grupo de enfoque, un grupo de observación o la evaluación heurística por expertos), y la obtención de conclusiones en un contexto específico.

La usabilidad no puede, por sí misma, ser el núcleo del diseño, planificación y desarrollo de productos de información para Internet, que en el momento actual se identifican con las sedes web. Es más adecuado plantear un enfoque de diseño fundamentado en las experiencias y necesidades del usuario, complementado por la actividad en arquitectura de información, hasta conformar lo que se ha dado en llamar “diseño de información”. Ejemplo de esta aproximación es la aparición de monografías sobre el diseño centrado en el usuario (*user centered*), en los dos últimos años, como resultado de una adopción más profunda y reposada de los principios de la HCI y del diseño de interacción hombre-máquina en el entorno del web.

La planificación, selección y ejecución de técnicas de análisis y aseguramiento de la usabilidad depende sobremanera del contexto en el que se lleven a cabo, del tipo de producto y de los recursos disponibles en cada caso. Los costos asociados a las mismas suponen que la mayor parte de los proyectos de tamaño medio y pequeño utilicen evaluación heurística de expertos. Las principales referencias de diseño de información para el web coinciden en señalar la necesidad de llevar a cabo estudios de usabilidad del producto durante la fase de diseño y producción, pero adecuando las técnicas de las mismas a las características del proyecto en sí.

La importancia de la usabilidad para el desarrollo de este trabajo, como calidad que los usuarios de un sitio web perciben en su uso, puede ser un factor para mejorar su competitividad. La mayor satisfacción de los usuarios mejora la imagen del sitio web y la efectividad y eficiencia en su uso contribuye decisivamente a su éxito.

La usabilidad de un sitio web se puede definir como la calidad del mismo según la perciben los usuarios que lo visitan. Los aspectos que intervienen son variados e incluyen principalmente la satisfacción del usuario, la facilidad para aprender y recordar su organización y funcionalidad, la efectividad para el usuario y su eficiencia.

Otro elemento importante de la usabilidad es la probabilidad de que el usuario cometa errores al realizar las tareas para las que el sitio ha sido diseñado. Por ejemplo, encontrar unos datos que se buscaban, completar una operación de comercio electrónico o descargarse unas imágenes en un formato especial. La usabilidad es como la salud personal; se percibe sobre todo cuando falta. Cuando está presente el usuario no es consciente de ella, pues la interacción con el sitio web se realiza de forma fácil, rápida e intuitiva.

Un sitio web usable se puede aprender mejor y su aprendizaje perdura más en la memoria. La usabilidad reduce los errores cometidos por los usuarios y lleva a que estos realicen las tareas deseadas de manera más eficiente y efectiva, aumentando así su satisfacción y mejorando su experiencia global con el sitio.

La usabilidad no es el único factor importante en el éxito de un sitio web. Obviamente su contenido y los servicios disponibles, así como su popularidad en Internet, contribuyen también a su éxito. Sin embargo, ante dos sitios web que ofrezcan productos o servicios similares los usuarios optarán por aquel sea más fácil de aprender, eficiente en su uso, efectivo en los resultados y satisfactorio en la experiencia.

Por último, aunque no por ello menos importante, un sitio usable es también más accesible con lo que llega a una audiencia potencialmente mayor.

3.7. Bibliografía

Nielsen, D. J. (2004). Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier. **2004**

Loaiza, C. R. (2004). Evaluación heurística y de accesibilidad. **2004**.

Constantine & Lockwood, L. (2003). Larry Constantine. **2003**

Instone, K. (2005) Diseño Web Centrado en el Usuario: Usabilidad y Arquitectura de la Información. **2005**

Villa, L.(2003). Alzado: técnología al servicio de las personas. Usabilidad sin usuarios: heurística. Madrid, España.

Genise, P. (2002). Shneiderman's "Eight Golden Rules of Interface Design". **2002**

Mayhew, D. J. (1999). “The Usability Engineering Lifecycle.” **1999**.

Nielsen, D. J. (2003). “Componentes de la Usabilidad”. **2003**.

Floria, Alejandro(2004). Usabilidad y diseño centrado en el usuario. Universidad Zaragoza, Madrid, España.

Usability Net (2003).International standards for HCI and usability. WAMMI projects. E.U. (**2003**).

CAPÍTULO 4

Tecnologías para Internet

4. Tecnologías para Internet

Hay literalmente cientos de tecnologías a disposición de los equipos de desarrollo de hoy en día y, aprovechándolas adecuadamente, permiten desarrollar sitios dinámicos, comprensibles y exitosos. En este capítulo hablaremos de las tres tecnologías más importantes vigentes para Internet.

4.1. PHP ('Personal Home Page)

4.1.1. Historia

PHP es un lenguaje creado por una comunidad de informáticos, desarrollado originalmente en el año 1994 por Rasmus Lerdorf como un CGI escrito en C que permitía la interpretación de un número limitado de comandos. El sistema fue denominado Personal Home Page Tools y adquirió relativo éxito gracias a que otras personas pidieron a Rasmus que les permitiera utilizar sus programas en sus propias páginas. Dada la aceptación del primer PHP, su creador diseñó un sistema para procesar formularios al que le dio el nombre de FI (Form Interpreter). El conjunto de estas dos herramientas sería la primera versión compacta del lenguaje: PHP/FI.

La siguiente gran contribución al lenguaje se realizó a mediados del 97 cuando se volvió a programar el analizador sintáctico, se incluyeron nuevas funcionalidades como el soporte a nuevos protocolos de Internet y el soporte a la gran mayoría de las bases de datos comerciales. Todas estas mejoras sentaron las bases de PHP versión 3. Actualmente PHP se encuentra en su versión 4, que utiliza el motor Zend, desarrollado con mayor meditación para cubrir las necesidades actuales y solucionar algunos inconvenientes de la anterior versión. Algunas mejoras de esta nueva versión son su rapidez -gracias a que primero se compila y luego se ejecuta, mientras que antes se ejecutaba mientras se interpretaba el código-, su mayor independencia del servidor web -creando versiones de PHP nativas para más plataformas- y un API más elaborado y con más funciones.

A continuación se mostrará una gráfica del número de dominios y direcciones IP que utilizan PHP figura 12.

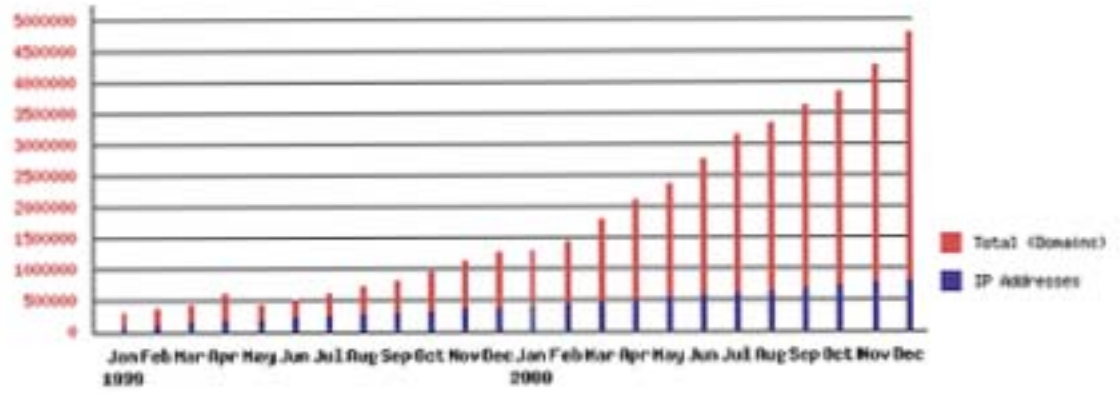

Figura 1 Estadística de Netcraft (Alvarez 2004)

En el último año, el número de servidores que utilizan PHP se ha disparado, logrando situarse cerca de los 5 millones de sitios y 800.000 direcciones IP, lo que ha convertido a PHP en una tecnología definitivamente popular. Esto es debido, entre otras razones, a que PHP es el complemento ideal para que el tándem Linux-Apache sea compatible con la programación del lado del servidor de sitios web. Gracias a la aceptación que ha logrado, y los grandes esfuerzos realizados por una creciente comunidad de colaboradores para implementarlo de la manera más óptima, podemos asegurar que el lenguaje se convertirá en un estándar que compartirá los éxitos augurados al conjunto de sistemas desarrollados en código abierto.

4.1.2. Tareas principales del PHP

Poco a poco PHP se ha convertido en un lenguaje de propósito general. Si bien en un principio fue diseñado para realizar poco más que un contador y un libro de visitas, ha experimentado en poco tiempo una verdadera revolución y, a partir de sus nuevas funciones, en estos momentos es capaz de realizar una multitud de tareas útiles para el desarrollo del web:

- **Funciones de correo electrónico**

Podemos con una facilidad asombrosa enviar un e-mail a una persona o lista parametrizando toda una serie de aspectos tales como el e-mail de procedencia, asunto, persona a responder, etc.

Otras funciones menos frecuentes pero de indudable utilidad para gestionar correos electrónicos son incluidas en su librería.

- **Gestión de bases de datos**

Resulta difícil concebir un sitio actual, potente y rico en contenido que no es gestionado por una base de datos. El lenguaje PHP ofrece interfaces para el acceso a la mayoría de las bases de datos comerciales y por ODBC a todas las bases de datos posibles en sistemas Microsoft, a partir de las cuales podremos editar el contenido de nuestro sitio con absoluta sencillez.

- **Gestión de archivos**

Crear, borrar, mover, modificar o cualquier tipo de operación más o menos razonable que se nos pueda ocurrir, puede ser realizada a partir de una amplia librería de funciones para la gestión de archivos por PHP. También podemos transferir archivos por FTP a partir de sentencias en nuestro código, protocolo para el cual PHP ha previsto también gran cantidad de funciones.

- **Tratamiento de imágenes**

Evidentemente resulta mucho más sencillo utilizar Photoshop para el tratamiento de imágenes pero...¿Y si tenemos que tratar miles de imágenes enviadas por los usuarios de nuestra páginas?. La verdad es que puede resultar muy tedioso uniformar en tamaño y formato miles de imágenes recibidas día tras día. Todo esto puede ser también automatizado eficazmente mediante PHP.

También puede parecer útil el crear botones dinámicos, es decir, botones en los que utilizamos el mismo diseño y sólo cambiamos el texto. Podremos por ejemplo crear un botón haciendo una única llamada a una función en la que introducimos el estilo del botón y el texto a introducir obteniendo automáticamente el botón deseado. A partir de la librería de funciones gráficas podemos hacer esto y mucho más.

- **Propósito general**

Muchas otras funciones pensadas para Internet (tratamiento de cookies, accesos restringidos, comercio electrónico...) o para propósito general (funciones matemáticas, explotación de cadenas, de fechas, corrección ortográfica, compresión de archivos...) son realizadas por este lenguaje. A esta inmensa librería cabe ahora añadir todas las funciones personales que uno va creando por necesidades propias y que luego son reutilizadas en otros sitios y todas aquellas intercambiadas u obtenidas en foros o sitios especializados.

Como puede verse, las posibilidades que se nos presentan son sorprendentemente vastas.

4.2. JAVA

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems. Sun describe al lenguaje Java de la siguiente manera: Simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutral, portable, de alto rendimiento, multitarea y dinámico. Sun admite totalmente que lo dicho anteriormente es una cadena de halagos por parte suya, pero el hecho es que todo ello describe al lenguaje Java. Java permite hacer cosas excitantes con las páginas Web que antes no eran posibles. De manera que en este momento la gran interactividad que proporciona Java marca la diferencia.

4.2.1. Historia

El lenguaje de programación Java fue desarrollado por Sun Microsystems en 1991. Nace como parte de un proyecto de investigación para desarrollar software para comunicación entre aparatos electrónicos de consumo como videos, televisores, equipos de música, etc. Durante la fase de investigación surgió el problema que dificultaba enormemente el proyecto iniciado: cada aparato tenía un microprocesador diferente y muy poco espacio de memoria; esto provocó un cambio en el rumbo de la investigación que desembocó en la idea de escribir un nuevo lenguaje de programación independiente del dispositivo que fue bautizado inicialmente como Oak.

La explosión de Internet en 1994, gracias al navegador gráfico Mosaic para la Word Wide Web (WWW), no pasó desapercibida para el grupo investigador de Sun. Se dieron cuenta de que los logros alcanzados en su proyecto de investigación eran perfectamente aplicables a Internet. Comparativamente, Internet era como un gran conjunto de aparatos electrónicos de consumo, cada uno con un procesador diferente. Y es cierto; básicamente, Internet es una gran red mundial que conecta múltiples computadoras con diferentes sistemas operativos y diferentes arquitecturas de microprocesadores, pero todas tienen en común un navegador que utilizan para comunicarse entre sí. Esta idea hizo que el grupo investigador abandonara el proyecto de desarrollar un lenguaje que permitiera la comunicación entre aparatos electrónicos de consumo y dirigiera sus investigaciones hacia el desarrollo de un lenguaje que permitiera crear aplicaciones que se ejecutaran en cualquier computadora de Internet con el único soporte de un navegador.

A partir de aquí ya todo es conocido. Se empezó a hablar de Java y de sus aplicaciones, conocidas como applets. Un applet es un programa escrito en Java que se ejecuta en el contexto de una página Web en cualquier computadora, independientemente de su sistema operativo y de la arquitectura de su procesador. Para ejecutar un applet sólo se necesita un navegador que soporte la máquina virtual de Java como, por ejemplo, Microsoft Internet Explorer o Netscape. Utilizando un navegador de éstos, se puede descargar la página Web que contiene el applet y ejecutarlo. Precisamente en este campo, es donde Java como lenguaje de programación no tiene competidores. No obstante con Java se puede programar cualquier cosa, razón por la que también puede ser considerado como un lenguaje de propósito general.

4.2.2. Características

- **Lenguaje simple**

Resulta relativamente sencillo escribir applets interesantes desde el principio. Todos aquellos familiarizados con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características, como los punteros. Debido a su semejanza con C y C++, y dado que la mayoría de los desarrolladores los conoce aunque sea de forma elemental, resulta muy fácil aprender Java. Los programadores experimentados en C++ pueden migrar a Java y ser productivos en relativamente poco tiempo.

- **Orientado a objetos**

Java fue diseñado como un lenguaje orientado a objetos desde el principio. Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos. La tendencia del futuro, a la que Java se suma, apunta hacia la programación orientada a objetos, especialmente en entornos cada vez más complejos y basados en red.

- **Distribuido**

Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

- **Interpretado y compilado a la vez**

Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador.
Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (run-time).

- **Robusto**

Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria.

- **Seguro**

Las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de ciertos virus. Java no posee una semántica específica para modificar la pila de programa, la memoria libre o utilizar objetos y métodos de un programa sin los privilegios del kernel del sistema operativo. Además, para evitar modificaciones por parte de los crackers de la red, implementa un método ultraseguro de autentificación por clave pública. El

Cargador de Clases puede verificar una firma digital antes de realizar una instancia de un objeto. Por tanto, ningún objeto se crea y almacena en memoria, sin que se validen los privilegios de acceso. Es decir, la seguridad se integra en el momento de compilación, con el nivel de detalle y de privilegio que sea necesario.

Dada, pues, la concepción del lenguaje, y si todos los elementos se mantienen dentro del estándar marcado por Sun, no hay peligro. Java imposibilita, también, abrir ningún fichero de la máquina local (siempre que se realizan operaciones con archivos, éstas trabajan sobre el disco duro de la máquina de donde partió el applet), no permite ejecutar ninguna aplicación nativa de una plataforma e impide que se utilicen otras computadoras como puente, es decir, nadie puede utilizar nuestra máquina para hacer peticiones o realizar operaciones con otra. Además, los intérpretes que incorporan los navegadores Web son aún más restrictivos. Bajo estas condiciones, se puede considerar que Java es un lenguaje seguro y que los applets están libres de virus.

Respecto a la seguridad del código fuente, no ya del lenguaje, JDK proporciona un desensamblador de byte-code, que permite que cualquier programa pueda ser convertido a código fuente, lo que para el programador significa una vulnerabilidad total a su código. Utilizando *javap* no se obtiene el código fuente original, pero sí desmonta el programa mostrando el algoritmo que se utiliza, que es lo realmente interesante. La protección de los programadores ante esto es utilizar llamadas a programas nativos, externos (incluso en C o C++) de forma que no sea descompilable todo el código; aunque así se pierda portabilidad. Esta es otra de las cuestiones que Java tiene pendientes.

- **Indiferente a la arquitectura**

Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a WindPows Nt, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. Para acomodar requisitos de ejecución, el compilador de Java genera bytecodes: un formato intermedio indiferente a la arquitectura diseñado para transportar el código eficientemente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java.

- **Portable**

La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.
Estas dos últimas características se conocen como la *Máquina Virtual Java* (JVM).

- **Multihebra**

Hoy en día ya se ven como terriblemente limitadas las aplicaciones que sólo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (*multithreading*) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.

- **Dinámico**

El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

- **Produce applets**

Java puede ser usado para crear dos tipos de programas: aplicaciones independientes y applets.

Las aplicaciones independientes se comportan como cualquier otro programa escrito en cualquier lenguaje, como por ejemplo el navegador de Web HotJava, escrito íntegramente en Java.

Por su parte, las applets son pequeños programas que aparecen embebidos en las páginas Web, como aparecen los gráficos o el texto, pero con la capacidad de ejecutar acciones muy complejas, como animar imágenes, establecer conexiones de red, presentar menús y cuadros de diálogo para luego emprender acciones, etc.

- **Rendimiento**

Para los casos en que la velocidad del intérprete Java no resulte suficiente, se dispone ya de mecanismos como los compiladores JIT(Just In Time), que se encargan de traducir (a medida que va siendo necesario) los bytecodes Java a instrucciones de código nativo de la máquina en dónde se ejecuta el programa. Otros mecanismos, como compiladores incrementales y sistemas dedicados para tiempo real, se encuentran también en el mercado.

4.2.3. Plataformas de Java

Sun ha agrupado sus tecnologías Java en tres ediciones:

Java 2 Platform, Standard Edition (J2SE): Constituye la base de la tecnología Java para el desarrollo de aplicaciones de propósito general.

Java 2 Platform, Enterprise Edition (J2EE): Simplifica el desarrollo de aplicaciones empresariales mediante la utilización de diversos componentes (EJB´s, Servlets, JSP´s, etc.), proporcionando un conjunto completo de servicios para dichos componentes y manejando automáticamente muchos detalles del comportamiento de la aplicación.

Java 2 Platform. Micro Edition (J2ME): Dirigida específicamente al mercado de consumo, que cubre un rango de productos portátiles pequeños como, smart cards, teléfonos móviles, PDA´s, etc.

4.2.4. Aspectos técnicos de Java

- **La Máquina Virtual Java (MVJ)**

La Máquina Virtual Java es el núcleo del lenguaje de programación Java. De hecho, es imposible ejecutar un programa Java sin ejecutar alguna implantación de la MVJ. En la MVJ se encuentra el motor que en realidad ejecuta el programa Java y es la clave de muchas de las características principales de Java, como la portabilidad, la eficiencia y la seguridad.

Siempre que se corre un programa Java, las instrucciones que lo componen no son ejecutadas directamente por el hardware sobre el que subyace, sino que son pasadas a un elemento de software intermedio, que es el encargado de que las instrucciones sean ejecutadas por el hardware. Es decir, el código Java no se ejecuta directamente sobre un procesador físico, sino sobre un procesador virtual Java, precisamente el software intermedio del que habíamos hablado anteriormente.

La representación de los códigos de instrucción Java (*bytecode*) es simbólica, en el sentido de que los desplazamientos e índices dentro de los métodos no son constantes, sino que son cadenas de caracteres o nombres simbólicos. Estos nombres son resueltos la primera vez que se ejecuta el método, es decir, el nombre simbólico se busca dentro del archivo de clase (*.class*) y se determina el valor numérico del desplazamiento. Este valor es guardado para aumentar la velocidad de futuros accesos. Gracias a esto, es posible introducir un nuevo método o sobreescribir uno existente en tiempo de ejecución, sin afectar o romper la estructura del código.

En la figura 13 (Méndez and Carballeira 2000) se observa la capa de software que implementa a la máquina virtual Java. Esta capa de software oculta los detalles inherentes a la plataforma, a las aplicaciones Java que se ejecuten sobre ella. Debido a que la plataforma Java fue diseñada pensando en que se implementaría sobre una amplia gama de sistemas operativos y de procesadores, se incluyeron dos capas de software para aumentar su portabilidad. La primera dependiente de la plataforma es llamada *adaptador*, mientras que la segunda, que es independiente de la plataforma, se le llama *interfaz de portabilidad*. De esta manera, la única parte que se tiene que escribir para una plataforma nueva, es el adaptador. El sistema operativo proporciona los servicios de manejo de ventanas, red, sistema de archivos, etcétera.

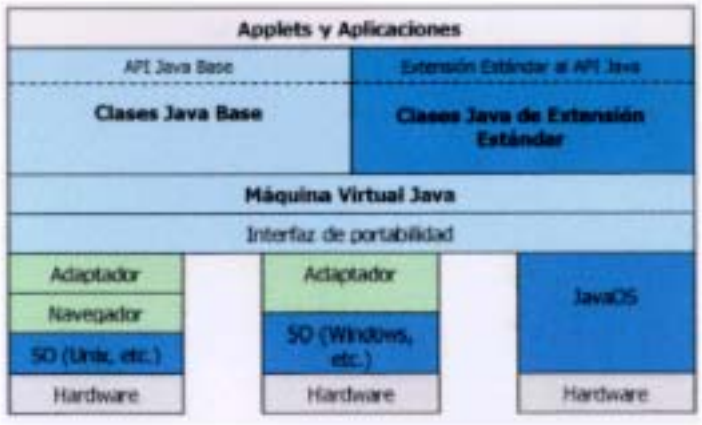

Figura 2 La Máquina Virtual Implementada para una variedad de plataformas

- **La plataforma Java (Sistema en tiempo de ejecución)**

Sun utiliza el término "Máquina Virtual Java", para referirse a la especificación abstracta de una máquina de software para ejecutar programas Java. La especificación de esta máquina virtual, define elementos como el formato de los archivos de clases de Java (*.class*), así como la semántica de cada una de las instrucciones que componen el conjunto de instrucciones de la máquina virtual. A las implantaciones de esta especificación se les conocen como "Sistemas en Tiempo de Ejecución Java". En la figura 13 (Méndez and Carballeira 2000) se puede observar los componentes típicos de un sistema de tiempo de ejecución. Ejemplos de Sistemas de tiempo de ejecución son el Navegador de Nestcape, el Explorador de Microsoft y el programa Java (incluido en el JDK). Un sistema de tiempo de ejecución incluye típicamente:

  - **Motor de ejecución.** El procesador virtual que se encarga de ejecutar el código (bytecode), generado por algún compilador de Java o por algún ensamblador del procesador virtual Java.

- **Manejador de memoria**. Encargado de obtener memoria para las nuevas instancias de objetos, arreglos, etcétera, y realizar tareas de recolección de basura.

- **Manejador de errores y excepciones.** Encargado de generar, lanzar y atrapar excepciones.

- **Soporte de métodos nativos**. Encargado de llamar métodos de C++ o funciones de C, desde métodos Java y viceversa.

- **Interfaz multihilos.** Encargada de proporcionar el soporte para hilos y monitores.

- **Cargador de clases.** Su función es cargar dinámicamente las clases Java a partir de los archivos de clase (.class).

- **Administrador de seguridad**. Se encarga de asegurar que las clases cargadas sean seguras, así como controlar el acceso a los recursos del sistema.

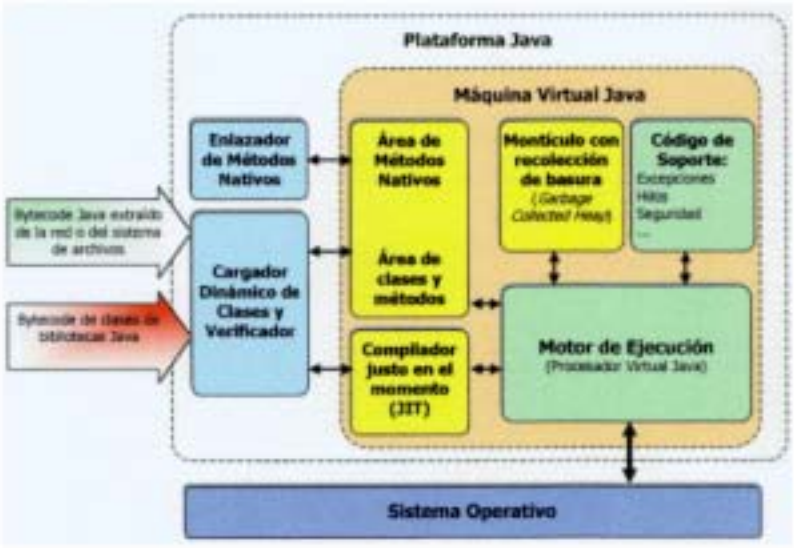

Figura 3 Arquitectura del Sistema de Tiempo de Ejecución Java

Adicionalmente, existe un conjunto de clases Java estándar, fuertemente ligadas a la implantación de cada MVJ en particular. Ejemplos de esto los tenemos en las clases encargadas de funciones, como los accesos a los recursos de la red, manejar el sistema de ventanas, los hilos y el sistema de archivos local. Todos estos elementos en conjunto actúan como una interfaz de alto nivel, para acceder a los recursos del sistema operativo. Es esta interfaz la clave de la portabilidad de los programas Java, debido a que independientemente del hardware o sistema operativo sobre el que se esté trabajando, la máquina virtual Java oculta todas estas diferencias.

A continuación describiremos con mayor detalle cada uno de estos elementos.

- **Motor de Ejecución**

Es la entidad de hardware o software, que ejecuta las instrucciones contenidas en los códigos de operación (*bytecodes*) que implementan los métodos Java. En las versiones iniciales de Sun, el motor de ejecución consistía de un intérprete de códigos de operación.

En las versiones más avanzadas de nuestros días, se utiliza la tecnología de "generación de código justo en el momento" (*Just-in-Time code generation*), en donde las instrucciones que implementan a los métodos, se convierten en código nativo que se ejecuta directamente en la máquina sobre la que se subyace. El código nativo se genera únicamente la primera vez que se ejecuta el código de operación Java, por lo que se logra un aumento considerable en el rendimiento de los programas.

- **El Conjunto de Instrucciones del Procesador Virtual**

Muchas de las instrucciones del procesador virtual Java, son muy similares a las que se pueden encontrar para los procesadores comunes y corrientes, como los Intel, es decir, incluyen los grupos de instrucciones típicos como los aritméticos, los de control de flujo, de acceso a memoria, a la pila, etcétera.

Una de las características más significativas del conjunto de instrucciones del procesador virtual Java, es que están basadas en la pila y utilizan "posiciones de memoria" numeradas, en lugar de registros. Esto es hasta cierto punto lógico, debido a que la máquina virtual está pensada para correr sobre sistemas con procesadores sustancialmente diferentes.

- **El Verificador de Java**

Como hemos mencionado anteriormente, una de las principales razones para utilizar una máquina virtual, es agregar elementos de seguridad a nuestro sistema, por lo que si un intérprete falla o se comporta de manera aleatoria, debido a código mal formado, es un problema muy serio. La solución trivial a este problema sería incluir código encargado de capturar errores y verificar que el código sea correcto. Es evidente que la principal desventaja de esta solución, es que volveremos a caer en un sistema sumamente seguro, pero altamente ineficiente.

Cuando estaban diseñando el conjunto de instrucciones para la máquina virtual de Java, tenían dos metas en mente. La primera era que el conjunto de instrucciones fuera similar a las instrucciones que se pueden encontrar en los procesadores reales. La segunda era construir un conjunto de instrucciones que fuera fácilmente verificable. En Java, justo después de que se obtiene una clase del sistema de archivos o de Internet, la máquina virtual puede ejecutar un verificador que se encargue precisamente de constatar que la estructura del archivo de clase es correcta. El verificador se asegura que el archivo tenga el número mágico (0xCAFEBABE) y que todos los registros que contiene el archivo tengan la longitud correcta, pero aún más importante, comprueba que todos los códigos de operación sean seguros de ejecutar.

- **Administrador de Memoria**

Java utiliza un modelo de memoria conocido como "administración automática del almacenamiento" (*automatic storage management*), en el que el sistema en tiempo de ejecución de Java mantiene un seguimiento de los objetos. En el momento que no están siendo referenciados por alguien, automáticamente se libera la memoria asociada con ellos.

Existen muchas maneras de implementar recolectores de basura, entre ellas tenemos:

- ***Contabilizar referencias*.** La máquina virtual Java asocia un contador a cada instancia de un objeto, donde se refleja el número de referencias hacia él. Cuando este contador es 0, la memoria asociada al objeto es susceptible de ser liberada.

- ***Marcar e intercambiar (Mark-and-Sweep)*.** Consiste en almacenar los objetos en un montículo (heap) de un tamaño considerable y marcar periódicamente (generalmente mediante un bit en un campo que se utiliza para este fin) los objetos que no tengan ninguna referencia hacia ellos. Adicionalmente existe un montón alterno, donde los objetos que no han sido marcados, son movidos periódicamente. Una vez en el montículo alterno, el recolector de basura se encarga de actualizar las referencias de los objetos a sus nuevas localidades. De esta manera se genera un nuevo montículo, que contiene únicamente objetos que están siendo utilizados.

- **Administrador de Errores y Excepciones.** Las excepciones son la manera como Java indica que ha ocurrido algo extraño durante la ejecución de un programa Java. Comúnmente las excepciones son generadas y lanzadas por el sistema, cuando uno de estos eventos ocurre. De la misma manera, los métodos tienen la capacidad de lanzar excepciones, utilizando la instrucción de la MVJ, *athrow*.

  Cuando se genera una excepción, el sistema de tiempo de ejecución de Java, y en particular el manejador (*handler*) de errores y excepciones, busca un manejador para esa excepción, comenzando por el método que la originó y después hacia abajo en la pila de llamadas. Cuando se encuentra un manejador, éste atrapa la excepción y se ejecuta el código asociado con dicho manejador. Lo que ocurre después depende del código del manejador, pero en general, puede suceder que:
  - Se utiliza un goto para continuar con la ejecución del método original
  - Se utiliza un return para salir del método
  - Se utiliza athrow para lanzar otra excepción

  En el caso que no se encuentre un manejador para alguna excepción previamente lanzada, se ejecuta el manejador del sistema, cuya acción típica es imprimir un mensaje de error y terminar la ejecución del programa.

- **Soporte para Métodos Nativos**

Las clases en Java pueden contener métodos que no estén implementados por códigos de operación (*bytecode*) Java, sino por algún otro lenguaje compilado en código nativo y almacenado en bibliotecas de enlace dinámico, como las DLL de Windows o las bibliotecas compartidas SO de Solaris.

El sistema de tiempo de ejecución incluye el código necesario para cargar dinámicamente y ejecutar el código nativo que implementa estos métodos. Una vez que se enlaza el módulo que contiene el código que implementa dicho método, el procesador virtual atrapa las llamadas a éste y se encarga de invocarlo. Este proceso incluye la modificación de los argumentos de la llamada, para adecuarlos al formato que requiere el código nativo, así como transferirle el control de la ejecución. Cuando el código nativo termina, el módulo de soporte para métodos nativos se encarga de recuperar los resultados y de adecuarlos al formato de la máquina virtual Java.

De manera análoga, el módulo de soporte para código nativo se encarga de canalizar una llamada a un método escrito en Java, hecha desde un procedimiento o método nativo.

- **Interfaz de Hilos**

Java es un lenguaje que permite la ejecución concurrente de varios hilos de ejecución, es decir, el sistema de tiempo de ejecución de Java tiene la posibilidad de crear más de un procesador virtual Java, donde ejecutar diferentes flujos de instrucciones, cada uno con su propia pila y su propio estado local. Los procesadores virtuales pueden ser simulados por software o implementados mediante llamadas al sistema operativo sobre el cual subyace.

- **Cargador de Clases**

Los programas Java están completamente estructurados en clases. Por lo tanto, una función muy importante del sistema en tiempo de ejecución, es cargar, enlazar e inicializar clases dinámicamente, de forma que sea posible instalar componentes de software en tiempo de ejecución. El proceso de cargado de las clases se realiza sobre demanda, hasta el último momento posible.

Los cargadores especializados por los programadores, pueden definir la localización remota de donde se cargarán las clases o asignar atributos de seguridad apropiados para sus aplicaciones particulares. Finalmente, se puede usar a los cargadores para proporcionar espacios de nombres separados a diferentes componentes de una aplicación.

- **Arquitectura de Seguridad en Java**

Java utiliza una serie de mecanismos de seguridad, con el fin de dificultar la escritura de programas maliciosos que pudiesen afectar la integridad de las aplicaciones y los datos de los usuarios. Cada sistema en tiempo de ejecución Java tiene la capacidad de definir sus propias políticas de seguridad, mediante la implantación de un "administrador de seguridad" (*security manager*), cuya función es proteger al sistema de tiempo de ejecución, definiendo el ámbito de cada programa Java en cuanto a las capacidades de acceder a ciertos recursos, etcétera.

El modelo de seguridad original proporcionado por la plataforma Java, es conocido como la "caja de arena" (Méndez and Carballeira 2000) (*sandbox*), que consiste en proporcionar un ambiente de ejecución muy restrictivo para código no confiable que haya sido obtenido de la red. Como se muestra en la figura 15 (Méndez and Carballeira 2000) , la esencia del modelo de la caja de arena, es que el código obtenido del sistema de archivo local es por naturaleza confiable. Se le permite el acceso a los recursos del sistema, como el mismo sistema de archivos o los puertos de comunicación. Mientras, el código obtenido de la red se considera no confiable. Por lo tanto, tiene acceso únicamente a los recursos que se encuentran accesibles desde la caja de arena.

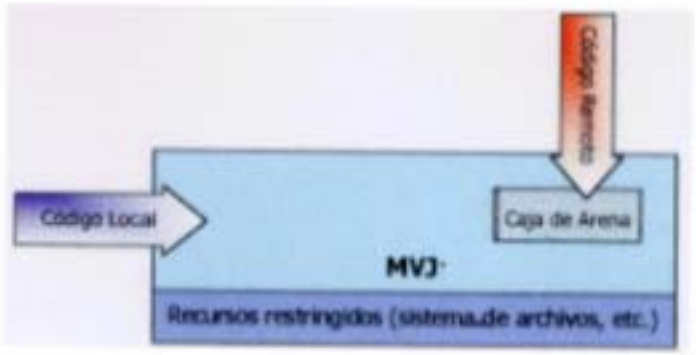

Figura 4 Modelo de seguridad del JDK 1.0.

Como hemos mencionado, la máquina implementa otros mecanismos de seguridad, desde el nivel de lenguaje de programación, como la verificación estricta de tipos de datos, manejo automático de la memoria, recolección automática de basura, verificación de los límites de las cadenas y arreglos, etcétera. Todo con el fin de obtener, de una manera relativamente fácil, código "seguro".

En segunda instancia, los compiladores y los verificadores de código intentan asegurar que sólo se ejecuten códigos de ejecución (*bytecodes*) Java, con la estructura correcta y no maliciosos. Asimismo, analizamos cómo con el cargador de clases se pueden definir espacios de nombres locales, lo que ayuda a garantizar que un *applet* no confiable pueda interferir con el funcionamiento de otros programas.

Finalmente, el acceso a los recursos importantes del sistema, es administrado entre el sistema de tiempo de ejecución y el administrador de seguridad (*Security Manager*). De esta manera, es posible para las aplicaciones determinar si una operación es insegura o contraviene las políticas de seguridad, antes de ejecutarla. Como se muestra en la figura 16 (Méndez and Carballeira 2000).

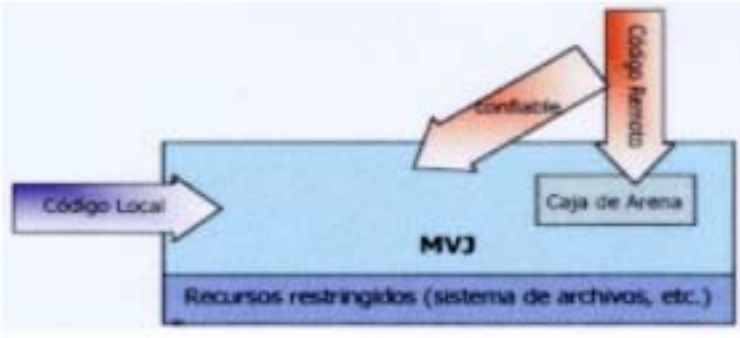

Figura 5 Modelo de seguridad JDK 1.1

El JDK 1.1 introduce el concepto de "*applet* firmado" (*signed applet*), en el que los *applets* que poseen una firma digital correcta, son considerados como confiables. Por lo tanto, reciben los mismos privilegios que el código obtenido del sistema de archivos. Los *applets* firmados, junto con la firma, se envían en un archivo de formato JAR (*Java Archive*). En este modelo de applets sin firma, continúan corriendo en la caja de arena. En la figura 16 se puede observar el modelo de seguridad del JDK 1.1.

Finalmente, como se muestra en la figura 17 (Méndez and Carballeira 2000), en la arquitectura de la plataforma de seguridad de Java 2 se introdujeron diferentes niveles de restricción, se eliminó la idea de que el código proveniente del sistema de archivo local siempre es confiable, etcétera.

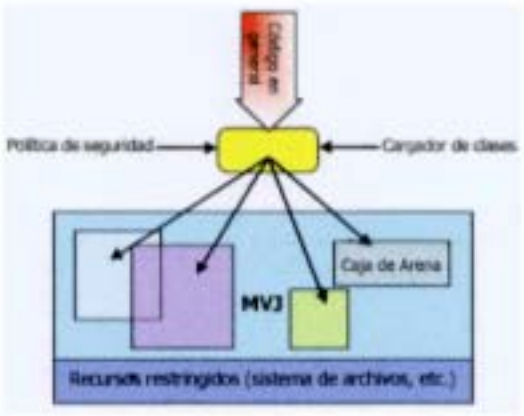

Figura 6 Modelo de seguridad de Java 2

4.3. J2EE

4.3.1. Introducción a la Tecnología y Conceptos J2EE

Java 2 Enterprise Edition (J2EE) es una arquitectura multicapa para implementar aplicaciones de tipo empresarial y aplicaciones basadas en la Web. Esta tecnología soporta una gran variedad de tipos de aplicaciones desde aplicaciones Web de gran escala a pequeñas aplicaciones cliente-servidor. El objetivo principal de la tecnología J2EE es crear un simple modelo de desarrollo para aplicaciones empresariales utilizando componentes basados en el modelo de aplicación. En este modelo dichos componentes utilizan servicios proporcionados por el contenedor, que de otro modo tendrían que estar incorporados en el código de la aplicación. Observa que esto podría no ser lo ideal para todos los escenarios: por ejemplo, una pequeña aplicación se cubriría mejor utilizando una solución de la tecnología Java de peso ligero (por ejemplo Servlets, JSPs, etc.).

4.3.2. Componentes J2EE

Las aplicaciones J2EE están compuestas de diferentes componentes. Un componente J2EE es una unidad de software funcional auto-contenido que se ensambla dentro de una aplicación J2EE con sus clases de ayuda y ficheros y que se comunica con otros componentes de la aplicación. La especificación J2EE define los siguientes componentes J2EE (McCallum 2004):

- Las Aplicaciones Clientes y los Applets son componentes que se ejecutan en el lado del cliente.
- Los componentes **Java Servlet** la tecnología **JavaServer Pages** son componentes Web que se ejecutan en el lado del servidor.
- Los **Enterprise JavaBeans** (beans enterprise) son componentes de negocio que se ejecutan en el servidor de aplicación.

En la figura 18 (McCallum 2004) se muestran los componentes de J2EE

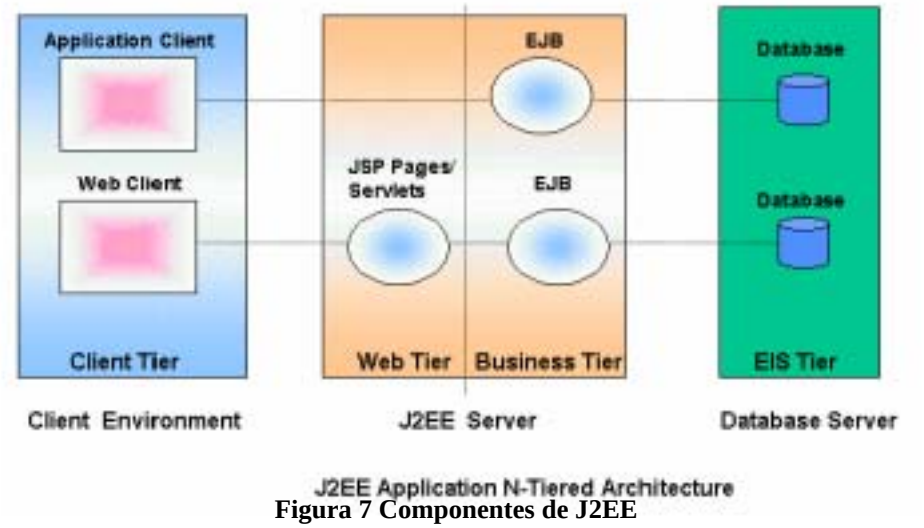

Figura 7 Componentes de J2EE

Además de estos componentes principales, J2EE incluye servicios estándar y tecnologías de soporte como:

- **Java Database Connectivity** (JDBC) tecnología que proporciona acceso a sistemas de bases de datos relacionales.
- **Java Transaction API (JTA)** o **Java Transaction Service (JTS)** proporciona soporte para transacciones a los componentes J2EE.
- **Java Messaging Service (JMS)** para comunicación asíncrona entre componentes J2EE.
- **Java Naming** y **Directory Interface (JNDI)** proporcionan accesos a nombres y directorios.

4.3.3. Clientes J2EE

Normalmente hay dos tipos de clientes J2EE: clientes Web y aplicaciones cliente como vimos en la figura anterior.

Un cliente Web consta de dos partes, páginas Web dinámicas que contienen distintos tipos de lenguajes de marcas (HTML, XML, y otros), que son generados por los componentes Web que se ejecutan en la capa Web, y un navegador Web, que dibuja las páginas recibidas desde el servidor. Otra categoría de clientes web son los conocidos como clientes *thin* (pequeños). Este tipo de pequeños clientes normalmente no hacen cosas como consultas a bases de datos o ejecutar complejas reglas de negocio. Cuando se utilizan clientes pequeños, las operaciones de peso pesado como estás las manejan los beans enterprise que se ejecutan en el servidor J2EE donde pueden tratar con la seguridad, los servicios y el rendimiento de las tecnologías del lado del servidor J2EE.

Una página Web recibida desde la capa del cliente puede incluir un applet embebido. Un appelt es una pequeña aplicación cliente escrita en Java que se ejecuta en la máquina virtual Java instalada en el navegador Web. Sin embargo, los sistemas cliente necesitarán el Plug-in Java y posiblemente un fichero de política de seguridad para poder ejecutar con éxito los applets en el navegador Web. Normalmente los componentes Web son el API preferido para crear programas clientes Web porque no necesitan plug-ins ni ficheros de política de seguridad en los sistemas clientes. Además esto permite un diseño más claro y modular de la aplicación porque proporciona un significado a la separación de la lógica de la aplicación del diseño de la página Web.

Una aplicación cliente se ejecuta sobre una máquina cliente y proporciona una forma para que los usuarios puedan manejar tareas que requieren un interface de usuario más rico que el que puede proporcionar un lenguaje de marcas. Normalmente tienen un interface gráfico de usuario (GUI) creado con los APIs **Swing** o **Abstract Window Toolkit (AWT)**. Las aplicaciones cliente acceden directamente a los beans enterprise que se ejecutan en la capa de negocio. Pero si se necesita un cliente Web pueden abrir una conexión HTTP para establecer comunicación con un servlet que se ejecute en la capa Web.

4.3.4. Componentes Web

Los componentes Web de J2EE pueden ser servlets o páginas JSP. Los servlets son clases Java que procesan dinámicamente las peticiones y construyen las respuestas. Las páginas JSP son documentos basados-en-texto que se ejecutan como servlets pero permiten una aproximación más natural para crear contenido estático. Las páginas HTML y los applets se juntan con los componentes Web durante el ensamble de la aplicación, pero la especificación J2EE no los considera como componentes J2EE. De forma similar, las clases de utilidades del lado del servidor también se unen a los componentes Web como páginas HTML, pero tampoco se consideran como componentes J2EE. En la figura 19 podemos ver la comunicación entre componentes Web (McCallum 2004):

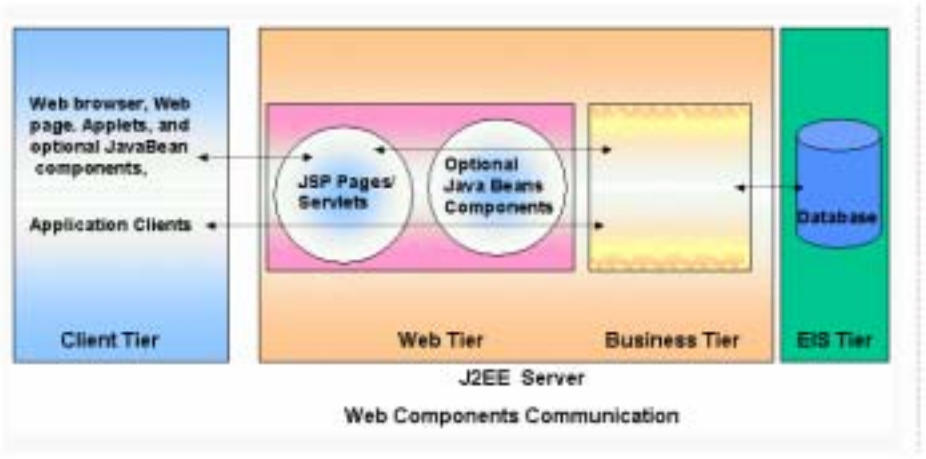

Figura 8 Comunicación entre componentes Web

La capa Web podría incluir componentes JavaBeans para manejar la entrar del usuario y enviar esta entrada a los beans enterprise que se ejecutan en la capa de negocio para su procesamiento según hemos visto en la figura anterior.

4.3.5. Componentes de Negocio

El código de negocio, que es la lógica que resuelve o cumple las necesidades de un negocio particular, como la banca, la venta, o la financiación, se maneja mediante beans enterprise que se ejecutan en la capa de negocio.

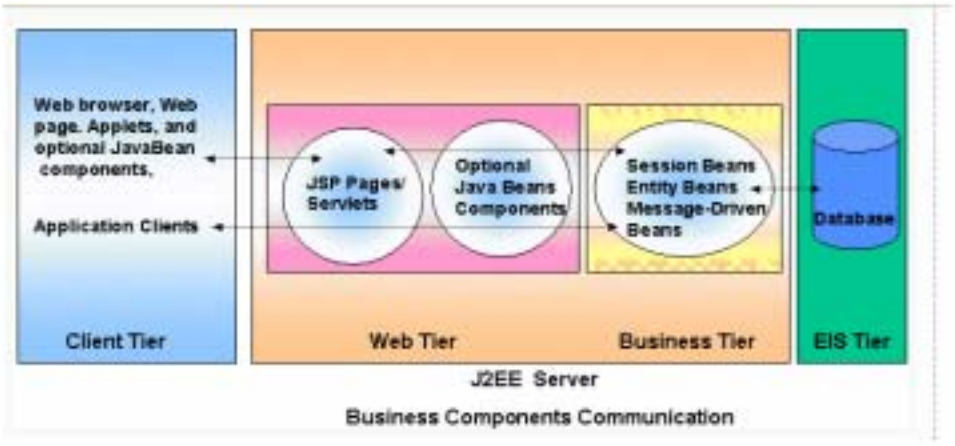

Figura 9 Comunicación entre los componentes de negocio

La figura 20 (McCallum 2004) muestra la comunicación entre los componentes de negocio, donde un bean enterprise recibe datos de los programas clientes, los procesa (si es necesario), y los envía a la capa del sistema de información empresarial para su almacenamiento. Un bean enteprise también recupera datos desde el almacenamiento, los procesa (si es necesario), y los envía de vuelta al programa cliente.

4.3.6. La Capa del Sistema de Información Empresarial

La capa del sistema de información empresarial maneja el software del sistema de información empresarial e incluye la infraestructura del sistema así como los planings de recursos (ERP), procesamiento de transacciones a mainframes, sistemas de bases de datos y otros sistemas de información legales (personalizados). Los componentes de aplicaciones J2EE podrían necesitar acceder a estos sistemas de información empresariales para conectividad con sus bases de datos.

4.3.7. Contenedores J2EE:

Los contenedores J2EE proporcionan acceso a los servicios subyancentes del entorno del Servidor J2EE mediante contenedores para diferentes tipos de componentes. Tradicionalmente, los desarrolladores de aplicaciones tenían que escribir código para el manejo de transacciones, manejo del estado, multi-threads, almacenamiento de recursos, etc. Ahora el contenedor J2EE proporciona estos servicios permitiendo que te puedas concentrar en resolver los problemas de negocio. El proceso de ensamble implica especificar las configuraciones del servidor, que incluye servicios como seguridad y control de transacciones, etc.

El servidor J2EE proporciona contenedores para **Enterprise Java Beans** (EJB) y para componentes **Web**. El contenedor EJB maneja la ejecución de los beans enterprise de las aplicaciones J2EE, mientras que el contenedor Web maneja la ejecución de las páginas JSP y los componentes servlets de la aplicación J2EE. Otros contenedores distintos a estos son el contenedor de aplicaciones clientes y el contenedor de applets, que no son parte del servidor J2EE porque residen en la máquina del cliente, como se muestra en la figura 21 (McCallum 2004):

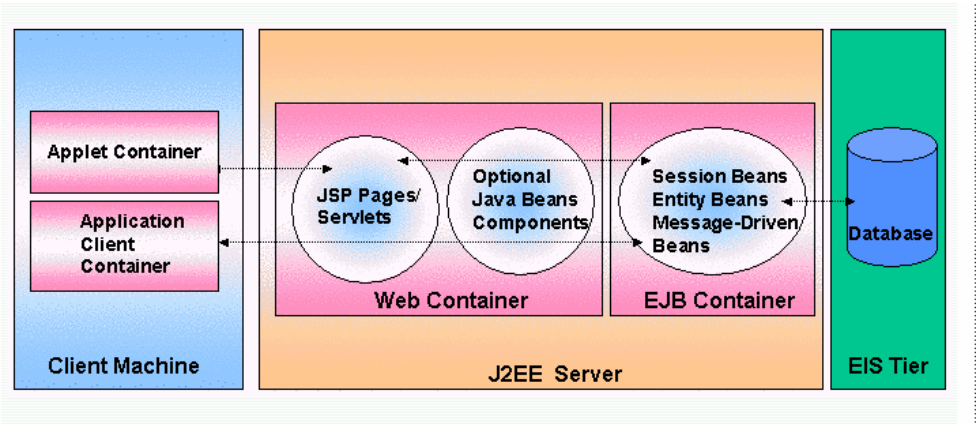

Figura 10 Contenedor de aplicaciones clientes y contenedor de applets

Un contenedor de aplicaciones cliente maneja la ejecución de los componentes de la aplicación cliente mientras que un contenedor de Applets maneja la ejecución de los applets. Normalmente están en el JRE (Java Runtime Environment) y el navegador Web compatible con Java, respectivamente.

4.3.8. Empaquetado

Para poder desplegar una aplicación J2EE, después de desarrollar los diferentes componentes, se empaqueta en ficheros de archivo especiales que contienen los ficheros de las clases relevantes y los descriptores de despliegue XML. Estos descriptores de despliegue contienen información específica de capa componente empaquetado y son un mecanismo para configurar el comportamiento de la aplicación en el momento del ensamble o del despliegue. Estos se empaquetan en diferentes tipos de archivos según los distintos componentes.

4.3.9. La Arquitectura Distribuida en J2EE

Todas las aplicaciones J2EE implementan una arquitectura distribuida. En ésta un objeto está asociado con un nombre, donde los nombres los proporciona un servicio de nombres, notificando a distintos componentes y resolviendo las referencias de clientes para estos componentes de servicio como se muestra en la figura 22 (McCallum 2004):

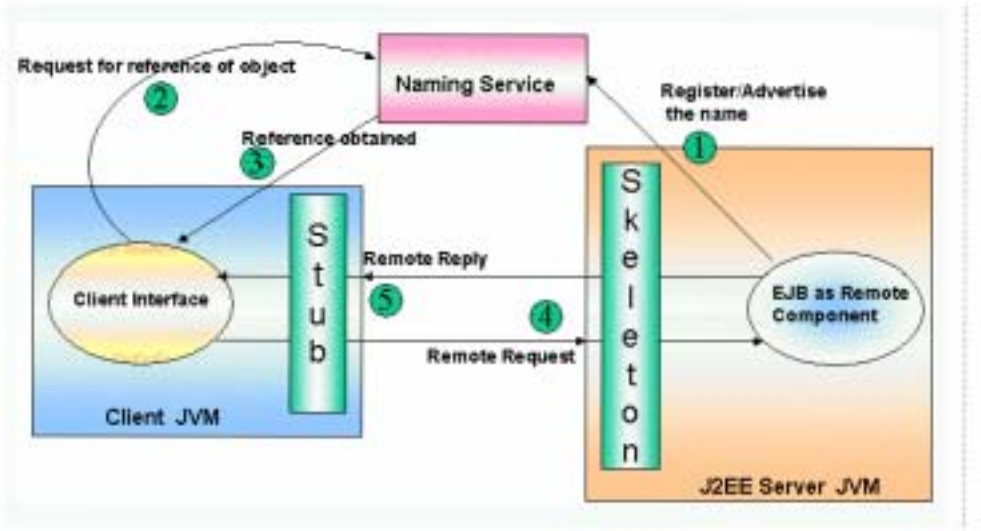

Figura 11 Arquitectura Distribuida

Como resultado de esto, las referencias de objetos se obtienen buscando un objeto por su nombre notificado, una vez encontrado, se obtiene la referencia, y se llevan a cabo las operaciones necesarias sobre ese objeto utilizando los servicios del host. Un objeto remoto notifica su disponibilidad en el servicio de nombres utilizando un nombre lógico y el servicio de nombres lo traduce a la localización física del objeto en el entorno J2EE. Una vez que la petición del cliente obtiene una referencia a un componente remoto, puede enviarle peticiones. El sistema de ejecución maneja la comunicación distribuida entre objetos remotos, lo que incluye la serialización y des-serialización de parámetros.

4.3.10. La Arquitectura Java Naming Directory Interface (JNDI)

J2EE utiliza el API JNDI para acceder genéricamente a servicios de nombrado y directorio utilizando la tecnología Java. El API JNDI reside entre la aplicación y el servicio de nombres y hace que el servicio de nombres subyacente sea transparente para los componentes de la aplicación como se muestra en la figura 23 (McCallum 2004):

Figura 12 Arquitectura Java Naming Directory Interface

Un cliente puede buscar referencias a componentes EJB u otros recursos en un servicio de nombres como el mencionado arriba. El código del cliente no se modifica, sin importar el servicio de nombres que se esté utilizando o en qué tecnología esté basada, y esto no crea ninguna diferencia en el modo en que los clientes localizan los objetos remotos mediante el API JNDI.

4.4. J2ME

Java 2 Micro Edition (J2ME) es una versión de Java Estándar (J2SE) para dispositivos con capacidades limitadas: PDAs, teléfonos móviles, electrodomésticos inteligentes, etc. Se compone de una selección de máquina virtual (máquina virtual Java reducida - KVM), configuración (para los terminales móviles el llamado Mobile Limited-device Configuration - CLDC), perfiles específicos para los diferentes dispositivos (para terminales móviles la llamada Mobile Information Device Profile - MIDP) y otros paquetes opcionales.

El software empresarial tiene unas características propias marcadas: está pensado no para ser ejecutado en un equipo sino para ejecutarse sobre una red de computadoras de manera distribuida de manera remota. De hecho, el sistema se monta sobre varias unidades o aplicaciones. En muchos casos, además, el software empresarial requiere que sea capaz de integrar datos provenientes de entornos heterogéneos. Para este entorno, para sus exigencias y características, Sun ha diseñado J2EE, Java 2 Enterprise Edition.

Sun separó J2SE de J2EE porque este último exigía una características muy pesadas o especializadas de I/O, trabajo en red , etc. Por tanto, por razones de eficiencia separó ambos productos. Hoy J2EE es un superconjunto de J2SE pues contiene toda la funcionalidad de este y más características.

Sun ha separado J2ME, Java 2 Micro Edition por las mismas razones. Los dispositivos inalámbricos tienen menos potencia y mucha menor capacidad gráfica que las PC de escritorio. Por ello, J2ME representa una versión simplificada de J2SE pensada para dispositivos con estas limitaciones.

El conjunto de J2ME, J2SE y J2EE le llamamos tecnología Java 2.

Dicho esto, es importante señalar que J2ME tiene la característica de tener una parte de su API fija, es decir, aplicable a todos los dispositivos inalámbricos y una parte que es específica para ciertos dispositivos; como ejemplo claro se me ocurre la API específica de Palm y la de móviles, que evidentemente son distintas como lo veremos a continuación (González 2002).

4.4.1. J2ME y WAP

WAP es Wireless Application Protocol o protocolo de aplicación inalámbrico. WAP permite a dispositivos inalámbricos soportar un navegador web simplificado. Para comunicaciones WAP debe estar adaptado a esta tecnología el cliente, el servidor y un gateway intermedio debe existir. El gateway WAP es el responsable de convertir las peticiones WAP y peticiones web habituales y viceversa. Las páginas que se transmiten a través de WAP no son archivos HTML sino que son WML. Si la web habitual soporta javascript, WML cuenta con un lenguaje de script simplificado a partir de javascript que se llama WMLscript.

WAP es una tecnología que está funcionando para móviles adaptados. WAP es sencillamente un protocolo para navegar la web en dispositivos móviles. Por tanto ambas tecnologías coexistirán sin problemas.

4.4.2. J2ME y SMS

SMS es la tecnología que permite hacer algo que vemos todos los días: mandar mensajes cortos entre dispositivos móviles, así cómo recibir otro tipo de mensajes. Por tanto, J2ME y SMS son cosas lo suficientemente diferentes como para no tener que competir. Salvo casos de aplicaciones de chat o de mensajería con J2ME, es muy lateral la competencia de J2ME sobre SMS.

4.4.3. J2ME y Bluetooth

La filosofía de Bluetooth es habilitar la comunicación en rangos relativamente cortos entre dispositivos. En la práctica sirve para quitarnos de encima los cables que conectan las computadoras sustituyendo estos por una conexión de radio. Esto da más comodidad y libertad en el uso de la computadora. Bluetooth no representa por tanto ninguna relación directa con J2ME.

4.5. .NET

4.5.1. Historia

En el caso de .NET Framework, los orígenes hay que buscarlos –antes de nada– en un cambio político en la empresa MICROSOFT. Allá por verano de 1996, Bill Gates, decide dejar la parte administrativa de la empresa en manos de Steve Ballmer, por entonces vicepresidente. Gates, pasaría desde ese momento a la jefatura del Departamento de Arquitectura de Software. Son momentos en los que J2EE atraviesa por un período dulce, de ascensión meteórica y rápida captación de seguidores entre ambientes diversos, especialmente universitarios.

La reacción es inmediata: hay que crear un equivalente a lo que es la plataforma de Sun, pero mejorado, si es posible. Y hay que hacerlo sin prisa, pero sin pausa. El primer movimiento fue espectacular: Anders Hejlsberg, autor del lenguaje Turbo-Pascal y arquitecto principal del entorno de desarrollo por excelencia de Borland (Delphi), es el primer objetivo. Además, Microsoft va a dejar de proponer estándares de facto propios para acogerse a estándares "de iure" (**iure** viene del latín y significa "por ley". Un estándar de iure, son estándares formales y legales acordados por algún organismo internacional de estandarización autorizado), e iniciar con ello la penetración en el mundo universitario, hasta ese momento, un tanto abandonado. Hejlsberg (y algunos más de su equipo en Borland), pasa a formar parte inicialmente al equipo de desarrollo que iba a integrar Java en Visual Studio (esto, en 1997), pero –según sus propias palabras (Villar 2004)- "aquello no era natural, y además, las ideas ya estaban formadas. Había gente que había pensado en un entorno de ejecución común, dentro de Microsoft, y decidimos ponernos manos a la obra".

Además comienza la construcción (junto a Scott Wiltamuth, Peter Kukol y Peter Golde) de un nuevo lenguaje: algo que aúne lo mejor de los mundos del C++ y Java. Algo que aprenda de los errores, asuma las novedades, mejore lo existente y aporte nuevas funcionalidades: en principio lo llamaron J++, a falta de un nombre mejor. Al menos, así se reconocía en un famoso e-mail que salió a relucir posteriormente, durante la disputa entre Sun y Microsoft, donde se añadía que el propósito del nuevo lenguaje era "conseguir un cuidadoso maridaje entre C++ y Java, que produjera un lenguaje "limpio", desprovisto de cierta parafernalia incómoda y de los inconvenientes establecidos por tendencias excesivamente puristas....". Además el nuevo lenguaje nace con vocación de estándar: se enviará su especificación completa a ECMA (la entidad de estandarización que gestionó entre otras, la normalización de Javascript), y más adelante, se seguirán dichos pasos hacia ISO (recientemente, dicho proceso ha concluido, con la estandarización oficial ISO tanto del lenguaje C# -que así se ha llamado finalmente- como de la infraestructura común de todos los lenguajes de .NET, la llamada Common Language Infrastructure).

Para poder participar activamente en la creación de los estándares, Microsoft recluta también a uno de los mayores expertos mundiales en lenguajes de marcas: el francés Jean Paoli, quién por esas fechas, pasa a formar parte del grupo de trabajo encargado de la creación de un estándar para el transporte y almacenamiento de la información basado en texto plano: lo que ahora conocemos como XML. Paoli, formó parte del grupo de tres diseñadores principales del lenguaje, que representaban así a las dos grandes ramas de la industria informática y al entorno académico: Paoli por parte del "consorcio" Microsoft, Intel. Tim Bray (representando a Netscape, Sun, Oracle, y demás) y Sperberg-McQueen en ese momento en la Universidad de Chicago. Como co-firmantes del documento figuraban absolutamente todos los que significaban algo en la informática en aquellos días: consenso garantizado.

Su trabajo concluye a finales del 1998, y en ese momento ya se trabaja a tope en la construcción de .NET: se discute sobre la gestión de memoria, sobre la integración de lenguajes y sobre el soporte de los "viejos" monstruos, como Visual Basic, y las consecuencias que su integración en .NET tendrían para el colectivo de desarrolladores. Así, XML se convertiría –en palabras del propio Ballmer– en "el aglutinante que permite toda esta nueva funcionalidad de .NET, así como la creación y utilización sencilla de servicios Web". La importancia de los servicios Web ha conducido de hecho a Microsoft a una nueva arquitectura llamada SOA (*Services Oriented Architecture*) que está comúnmente reconocida como la forma más eficiente y económica para integrar información.

4.5.2. Características

La nueva tecnología de Microsoft ofrece soluciones a los problemas de programación actuales, como son la administración de código o la programación para Internet. Para aprovechar al máximo las características de .Net es necesario entender la arquitectura básica en la que esta implementada esta tecnología y así beneficiarse de todas las características que ofrece esta nueva plataforma.

El Framework de .Net es una infraestructura sobre la que se reúne todo un conjunto de lenguajes y servicios que simplifican enormemente el desarrollo de aplicaciones. Mediante esta herramienta se ofrece un entorno de ejecución altamente distribuido, que permite crear aplicaciones robustas y escalables. Los principales componentes de este entorno son:

- Lenguajes de compilación
- Biblioteca de clases de .Net
- CLR (Common Language Runtime)

En la figura 24 podemos ver la arquitectura de .Net Framework (Villar 2004):

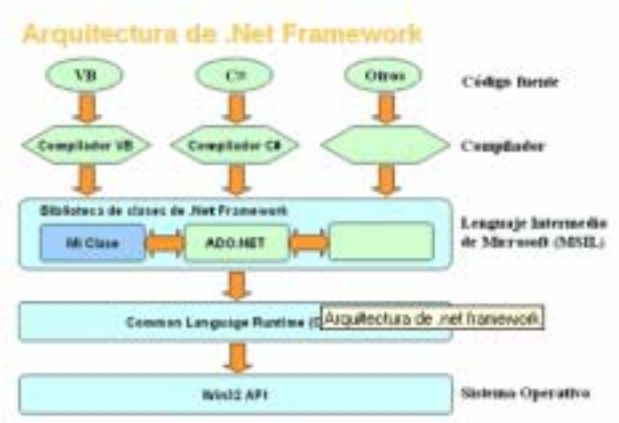

Figura 13 Arquitectura .Net Framework

Actualmente, el Framework de .Net es una plataforma no incluida en los diferentes sistemas operativos distribuidos por Microsoft, por lo que es necesaria su instalación previa a la ejecución de programas creados mediante .Net. El Framework se puede descargar gratuitamente desde la web oficial de Microsoft.

.Net Framework soporta múltiples lenguajes de programación y aunque cada lenguaje tiene sus características propias, es posible desarrollar cualquier tipo de aplicación con cualquiera de estos lenguajes. Existen más de 30 lenguajes adaptados a .Net, desde los más conocidos como C# (C Sharp), Visual Basic o C++ hasta otros lenguajes menos conocidos como Perl o Cobol.

4.5.3. Common Language Runtime (CLR)

El CLR es el verdadero núcleo del Framework de .Net, ya que es el entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes, ampliando el conjunto de servicios que ofrece el sistema operativo estándar Win32.

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .Net en un mismo código, denominado código intermedio (MSIL, Microsoft Intermediate Lenguaje). Para generar dicho código el compilador se basa en el Common Language Specification (CLS) que determina las reglas necesarias para crear código MSIL compatible con el CLR.

De esta forma, indistintamente de la herramienta de desarrollo utilizada y del lenguaje elegido, el código generado es siempre el mismo, ya que el MSIL es el único lenguaje que entiende directamente el CLR. Este código es transparente al desarrollo de la aplicación ya que lo genera automáticamente el compilador.

Sin embargo, el código generado en MSIL no es código máquina y por tanto no puede ejecutarse directamente. Se necesita un segundo paso en el que una herramienta denominada compilador JIT (Just-In-Time) genera el código máquina real que se ejecuta en la plataforma que tenga la computadora.

De esta forma se consigue con .Net cierta independencia de la plataforma, ya que cada plataforma puede tener su compilador JIT y crear su propio código máquina a partir del código MSIL.

La compilación JIT la realiza el CLR a medida que se invocan los métodos en el programa y, el código ejecutable obtenido, se almacena en la memoria caché de la computadora, siendo recompilado sólo cuando se produce algún cambio en el código fuente.

La figura 25 nos permite apreciar al CLR (Villar 2004):

Figura 14 Common Language Runtime

4.5.4. Biblioteca de clases de .Net

Cuando se está programando una aplicación muchas veces se necesitan realizar acciones como manipulación de archivos, acceso a datos, conocer el estado del sistema, implementar seguridad, etc. El Framework organiza toda la funcionalidad del sistema operativo en un espacio de nombres jerárquico de forma que a la hora de programar resulta bastante sencillo encontrar lo que se necesita.

Para ello, el Framework posee un sistema de tipos universal, denominado Common Type System (CTS). Este sistema permite que el programador pueda interactuar los tipos que se incluyen en el propio Framework (biblioteca de clases de .Net) con los creados por él mismo (clases). De esta forma se aprovechan las ventajas propias de la programación orientada a objetos, como la herencia de clases predefinidas para crear nuevas clases, o el polimorfismo de clases para modificar o ampliar funcionalidades de clases ya existentes como se puede apreciar en la figura 4.15 (Villar 2004).

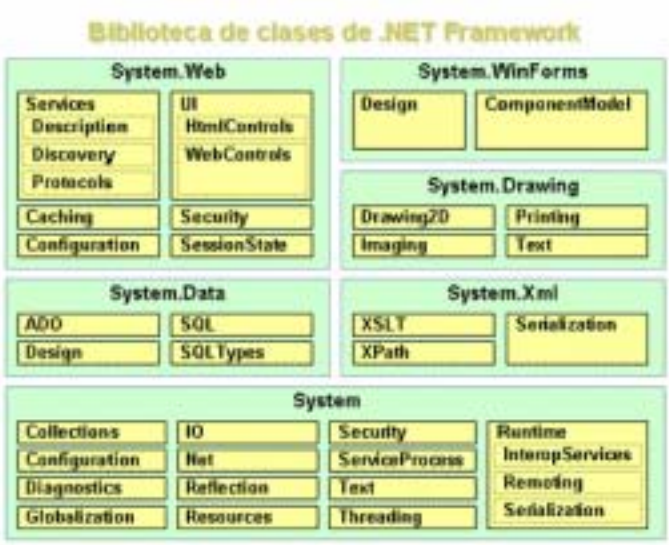

Figura 15 Biblioteca de clases de .Net

La biblioteca de clases de .Net Framework incluye, entre otros, tres componentes clave:

- ASP.NET para construir aplicaciones y servicios Web.
- Windows Forms para desarrollar interfaces de usuario.
- ADO .NET para conectar las aplicaciones a bases de datos.

La forma de organizar la biblioteca de clases de .Net dentro del código es a través de los espacios de nombres (namespaces), donde cada clase está organizada en espacios de nombres según su funcionalidad. Por ejemplo, para manejar ficheros se utiliza el espacio de nombres System.IO y si lo que se quiere es obtener información de una fuente de datos se utilizará el espacio de nombres System.Data.

La principal ventaja de los espacios de nombres de .Net es que de esta forma se tiene toda la biblioteca de clases de .Net centralizada bajo el mismo espacio de nombres (System). Además, desde cualquier lenguaje se usa la misma sintaxis de invocación, ya que a todos los lenguajes se aplica la misma biblioteca de clases.

4.5.5. Ensamblados

Uno de los mayores problemas de las aplicaciones actuales es que en muchos casos tienen que tratar con diferentes archivos binarios (DLL´s), elementos de registro, conectividad abierta a bases de datos (ODBC), etc.

Para solucionarlo el Framework de .Net maneja un nuevo concepto denominado ensamblado. Los ensamblados son ficheros con forma de EXE o DLL que contienen toda la funcionalidad de la aplicación de forma encapsulada. Por tanto la solución al problema puede ser tan fácil como copiar todos los ensamblados en el directorio de la aplicación.

Con los ensamblados ya no es necesario registrar los componentes de la aplicación. Esto se debe a que los ensamblados almacenan dentro de si mismos toda la información necesaria en lo que se denomina el manifiesto del ensamblado. El manifiesto recoge todos los métodos y propiedades en forma de meta-datos junto con otra información descriptiva, como permisos, dependencias, etc.

Para gestionar el uso que hacen las aplicaciones de los ensamblados .Net utiliza la llamada caché global de ensamblados (GAC, Global Assembly Cache). Así, .Net Framework puede albergar en el GAC los ensamblados que puedan ser usados por varias aplicaciones e incluso distintas versiones de un mismo ensamblado, algo que no era posible con el anterior modelo COM.

4.5.6. Ventajas de .Net

A continuación se resumen las ventajas más importantes que proporciona .Net Framework:

- **Código administrado**: El CLR realiza un control automático del código para que este sea seguro, es decir, controla los recursos del sistema para que la aplicación se ejecute correctamente.

- **Interoperabilidad multilenguaje**: El código puede ser escrito en cualquier lenguaje compatible con .Net ya que siempre se compila en código intermedio (MSIL).

- **Compilación just-in-time**: El compilador JIT incluido en el Framework compila el código intermedio (MSIL) generando el código máquina propio de la plataforma. Se aumenta así el rendimiento de la aplicación al ser específico para cada plataforma.

- **Garbage collector**: El CLR proporciona un sistema automático de administración de memoria denominado recolector de basura (garbage collector). El CLR detecta cuándo el programa deja de utilizar la memoria y la libera automáticamente. De esta forma el programador no tiene por que liberar la memoria de forma explícita aunque también sea posible hacerlo manualmente (mediante el método disponse() liberamos el objeto para que el recolector de basura lo elimine de memoria).

- **Seguridad de acceso al código**: Se puede especificar que una pieza de código tenga permisos de lectura de archivos pero no de escritura. Es posible aplicar distintos niveles de seguridad al código, de forma que se puede ejecutar código procedente del Web sin tener que preocuparse si esto va a estropear el sistema.

- **Despliegue**: Por medio de los ensamblados resulta mucho más fácil el desarrollo de aplicaciones distribuidas y el mantenimiento de las mismas. El Framework realiza esta tarea de forma automática mejorando el rendimiento y asegurando el funcionamiento correcto de todas las aplicaciones.

El desarrollo e instalación de aplicaciones de forma rápida es una ventaja competitiva para cualquier empresa o institución, en este caso, para el desarrollo de este trabajo, es importante poder dar al cliente un servicio eficaz, independientemente del tipo y soporte que se brinde a las aplicaciones. En este proceso es importante tanto la portabilidad como la escalabilidad de los productos resultantes. Cada sistema desarrollado debe ser factible ser escalado en todas sus fases de desarrollo, es decir, desde un primer prototipo con pequeños casos de prueba a la versión final completa. El problema reside en que aplicaciones con múltiples niveles son difíciles de describir porque conectan un conjunto amplio de recursos y habilidades de los desarrolladores. En el entorno heterogéneo de las redes informáticas de hoy en día, un ingeniero puede tener que integrar recursos de una gran variedad de sistemas distintos. La experiencia ha demostrado que dicha integración puede ocupar hasta un 50% del tiempo del ciclo de vida de un sistema. J2EE ofrece una capa estándar que funciona encima de otros sistemas (como por ejemplo sistemas de gestión de bases de datos, monitores de transacciones, servicios de nombres y de directorios, etc.), que facilita su integración. Dicha integración a través del modelo de componentes de J2EE permite la generación de aplicaciones y sistemas para la solución de los problemas relacionados con los requisitos del mundo actual.

J2EE (*Java2 Entreprise Edition*) ofrece un conjunto de especificaciones y técnicas que proporcionan soluciones completas, seguras, estables y escalables para el desarrollo, despliegue y gestión de las aplicaciones en múltiples niveles de funcionalidad basadas en servidores. J2EE reduce el costo y la complejidad de desarrollo, lo cual redunda en rapidez de desarrollo.

La plataforma J2EE define un estándar para el desarrollo de aplicaciones de múltiples niveles (servidores Web de aplicaciones, de bases de datos, etc.). Gracias a que su funcionamiento se basa en componentes modulares que incluyen un conjunto de servicios predefinidos, se simplifica la tarea de la producción de sistemas. J2EE extiende las ventajas de la plataforma Java 2 (como por ejemplo, seguridad, la portabilidad de programas, el acceso a las bases de datos, etc.) con la integración de recursos como Enterprice Java Beans, Servlets Java, Java Server Pages y la tecnología XML.

4.6. Bibliografía

Alvarez, M. A. (2004). Breve historia de PHP. **2004**.

González, A. (2002). Introducción a J2ME. **2004**.

McCallum, G. (2004). Construir Aplicaciones EJB con JBoss, Lomboz y Eclipse. **3 Agosto 2004**.

Méndez, M. e. C. R. M. and D. F. G. Carballeira (2000). Arquitectura de la Máquina Virtual Java. **2004**.

Villar, R. C. (2004). Análisis comparativo entre Microsoft® .NET y Sun® J2EE.

CAPÍTULO 5

Análisis y sistema

5. Análisis y sistema

Partiendo del análisis realizado para la presentación de una plataforma de educación a distancia, de la detección de los roles necesarios para interactuar con el sistema (administrador, profesor y alumno) y de las necesidades detectadas hacia cada rol, se presenta la siguiente propuesta, buscando definir un producto que satisfaga las necesidades del usuario de forma eficiente y predecible. En nuestro caso, el objetivo es analizar y desarrollar un sistema educativo innovador que permita diversificar y ampliar las ofertas de estudio, impartiendo una enseñanza fuera de las aulas, ya sea en casa, en lugar de trabajo o en cualquier otro sitio adecuado, logrando así la compaginación de la limitación de acceso a la enseñanza convencional con la oferta sin tiempos ni espacios fijos.

Este proyecto ha sido diseñado y desarrollado para operar en web, lo que permite intercambiar y compartir información, así como brindar servicios en cualquier momento y desde cualquier lugar en forma segura y controlada. Asimismo, una de las principales ventajas de un sistema de información en Web es la independencia del cliente, es decir, no se requiere la instalación de la aplicación en cada uno de los equipos donde vaya a utilizarse el sistema, sólo es necesario contar con una conexión a Internet eficiente y un navegador de Internet (browser) actualizado, lo que facilita su uso a los diferentes participantes del Sitio.

Por otra parte, en la actualidad los usuarios están más familiarizados con las aplicaciones y sistemas de información con interfaces gráficas, y con el uso cada vez más extendido de Internet (caracterizado por interfaces gráficas atractivas y fáciles de usar), lo que ayuda al proceso de adaptación a los sistemas en Web, facilitando su uso y adopción.

Cada sistema a desarrollar debe ser tratado con la metodología que mejor se adapte a los objetivos de un producto final de calidad. Es por ello que, basados en el análisis realizado en los capítulos anteriores, se decide tomar como punto de partida un proyecto que es el resultado de un esfuerzo comunitario para diseñar, crear y facilitar un nuevo Entorno de Colaboración y Aprendizaje (CLE) para la educación. El nombre del proyecto es SAKAI (Sinchronised Architecting of Knowledge Acquisition Infraestructure) un sistema realizado por Universidades destacadas del mundo y que permite alcanzar el objetivo de este trabajo, el cual se denominó C@D-CCADET.

Así, nuestra propuesta se realiza sobre la base del primer proyecto todas las adecuaciones de las actividades y roles definidos en los capítulos anteriores. Para fines prácticos de este trabajo sólo describimos el rol de administrador, ya que el administrador es quien da de alta las materias dentro de la aplicación, es decir, construye la plataforma; aunque una vez dadas de alta es el profesor el responsable de la administración de éstas.

5.1. Diagrama Técnico del Sistema

La arquitectura del sistema está formada por los elementos que se muestran en la figura 5.1 y que se detallan a continuación:

5.1.1. Cliente

El propósito de este proyecto es correr como una aplicación cliente/servidor. Mientras la mayoría de los clientes podría ser cualquier equipo UNIX o WINDOWS, el sistema se adapta a cualquier navegador consciente de que la aplicación puede ser utilizada en muchas situaciones.

La parte de presentación al cliente se hace utilizando un lenguaje como HTML, por lo cual los clientes podrán comunicarse directamente con los servicios brindados por el sistema, pues estarán habilitados para tales transacciones.

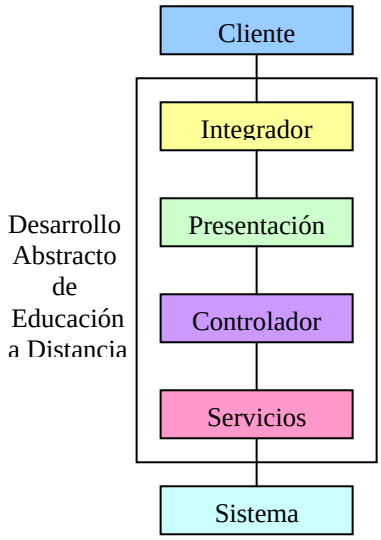

Figura 1 Elementos del sistema.

5.1.2. Integrador

Las salidas de las aplicaciones del proyecto pueden ser combinadas usando un servidor integrador de aplicaciones. Este integrador asigna y maneja la presentación de las pantallas y ciertas transacciones de la interfaz de usuario. El integrador además garantiza accesibilidad usando una combinación de elementos estándar UI (User Interfase) en la capa de presentación e integración.

5.1.3. Presentación

La capa de presentación combina datos de las herramientas del sistema y la descripción de la interfaz de usuario para crear fragmentos de ideas antes de ser presentado o entregado al usuario. La descripción de interfaz de usuario esta conformada idealmente en un recurso externo de software y hace uso de elementos estándar de interfaces de usuario diseñadas para brindar al usuario un sistema sencillo de utilizar.

5.1.4. Controlador

La herramienta del sistema es una capa en donde la presentación y el procesamiento de datos propio a los servicios que se brindan están estrechamente unidos. Esta capa brinda el código que permite responder a las peticiones de usuario, las cuales pueden o no modificar los datos en servicios ofrecidos. Finalmente puede proporcionar servicios para agregar datos a la capa de presentación.

5.1.5. Servicios

Un servicio es una colección de clases las cuales controlan los datos y definen el comportamiento de dicho servicio. Estos datos pueden o no ser persistentes a través del uso de sesiones. Los datos son modelados y representados a través de estándares definidos. El comportamiento es representado utilizando Interfaces de Programación de Aplicaciones (API). Unos servicios pueden ser llamados por otros servicios creando dependencias entre ellos. Los servicios son modulares, reusables y portables en el ambiente del sistema.

5.1.6. Sistema

El sistema reside en un servidor el cual puede incluir servicios web, manejadores de bases de datos, sistemas operativos, archivos y repositorios empresariales y de oficina, etc.

En este trabajo para la definición y desarrollo del sistema se uso UCD (User-Centered Design), ya que se tuvo un análisis profundo de entendimiento al usuario, apuntando a una presencia activa directa de éste en el proceso de desarrollo del mismo, logrando un sistema intuitivo y por lo tanto, de fácil uso para ellos.

5.2. Definición del Problema

En este proyecto se realizaron investigaciones de campo sobre las necesidades del usuario, se observaron usuarios y se realizaron encuestas hasta definir una propuesta que los apoyara de manera eficiente. Se analizó y se propuso la idea de desarrollar un sistema educativo innovador que permitiera diversificar y ampliar las ofertas de estudio, impartiendo una enseñanza fuera de las aulas, ya sea en casa, en el lugar de trabajo o en cualquier otro sitio adecuado, para con ello lograr la compaginación de la limitación de acceso a la enseñanza convencional con la oferta de otra alternativa.

En las entrevistas se preguntó al usuario lo que hacía y cómo le hacía, y esto se corroboró observando la manera en que trabaja. En un segundo momento se le inquirió acerca de las cosas que les gustan y disgustan de su trabajo, así como la manera de comunicarse dentro de él. Como resultado de una extensa investigación se detectaron los roles que compondrían el sistema: Administrador, Profesor, Alumno y con ello se puntualizaron las actividades requeridas para proporcionarle al usuario un sistema que le permitiera alcanzar sus expectativas de una educación a distancia, de fácil uso que permita interactuar con otros usuarios en tiempo real.

Es por ello que de acuerdo a lo concluido en nuestros capítulos previos y a las necesidades del usuario se realizó el análisis para la implementación de un sistema que le proporcione las herramientas necesarias para alcanzar su objetivo educativo, un sistema innovador de fácil uso que le brinde diversos apoyos que refuercen su estudio, como asesorías en línea que le permitan interactuar en tiempo real con el profesor y aclarar sus dudas, un foro de discusión a través del cual podrá aprender acerca de algún tema expuesto y aportar ideas que impulsen su educación, una agenda que le permita planear y/u organizar sus actividades de tal manera que tenga siempre presente sus compromisos académicos, una sección de anuncios emitidos por la administración de la institución que le permita estar al día con la información relevante de la misma, un apartado de recursos donde se le proporcione material de apoyo de alguna materia específica, documentos, url´s, entre otros y en donde a su vez el pueda colocar algún recurso para compartir, un módulo de tareas a través del cual pueda ser evaluado por el profesor y le permita ver el

resultado de su aprendizaje con los comentarios y/u observaciones sobre la misma, una sección que le permita compartir archivos con el profesor, una sección que le permita enviar correos electrónicos al correo registrado para la materia, una sección que permita consultar noticias de interés para la materia, y por último una sección que le permita consultar los datos de la materia que esta cursando además de una sección de ayuda para el manejo del sistema

Una vez ya detectadas y definidas las necesidades del usuario y de haber realizado el análisis tecnológico de tal manera que con el paso del tiempo pueda ser mejorado y/o modificado de acuerdo a las nuevas necesidades del sistema, se decide desarrollar un sistema portable de fácil aprendizaje que con el apoyo de otras tecnologías y perfectamente cimentado en la ingeniería de software, se presenta un proyecto completamente modular de fácil entendimiento al analista y desarrollador de sistemas. Se transcribe a diagramas de caso de uso por rol de usuario y se inicia la implementación.

Durante la investigación acerca del desarrollo de algunos módulos y ya presentado el inicio de este proyecto, y con la continua retroalimentación acerca de los debates de la educación a distancia, se encuentra un proyecto que se apega a nuestro estudio, es un proyecto cimentado en la ingeniería de software y cuyo análisis es plasmado en UML, y cuyo diseño es Centrado en el Usuario, este proyecto es impulsado por universidades destacadas por su excelencia académica como lo es la universidad de Michigan, la Universidad de Indiana y el Instituto de Tecnología de Massachusetts (MIT), el nombre del proyecto es Sakai Project, apoyados en la información y cimentación del mismo, se instaló, probó y analizó completamente la funcionalidad, por lo cual nuestro sistema denominado C@D-CCADET (Centro de Aprendizaje y Desarrollo del Centro de Ciencias Aplicadas y Desarrollo Tecnológico) se apoya en Sakai para hacer posible la implementación de nuestro proyecto de educación a distancia como resultado de un par de años de investigación abarcando la funcionalidad de los tres roles definidos para el proyecto, y del cual a continuación se presenta el diagrama (figura 28) de navegación del sitio el cual representa esquemáticamente la composición del mismo.

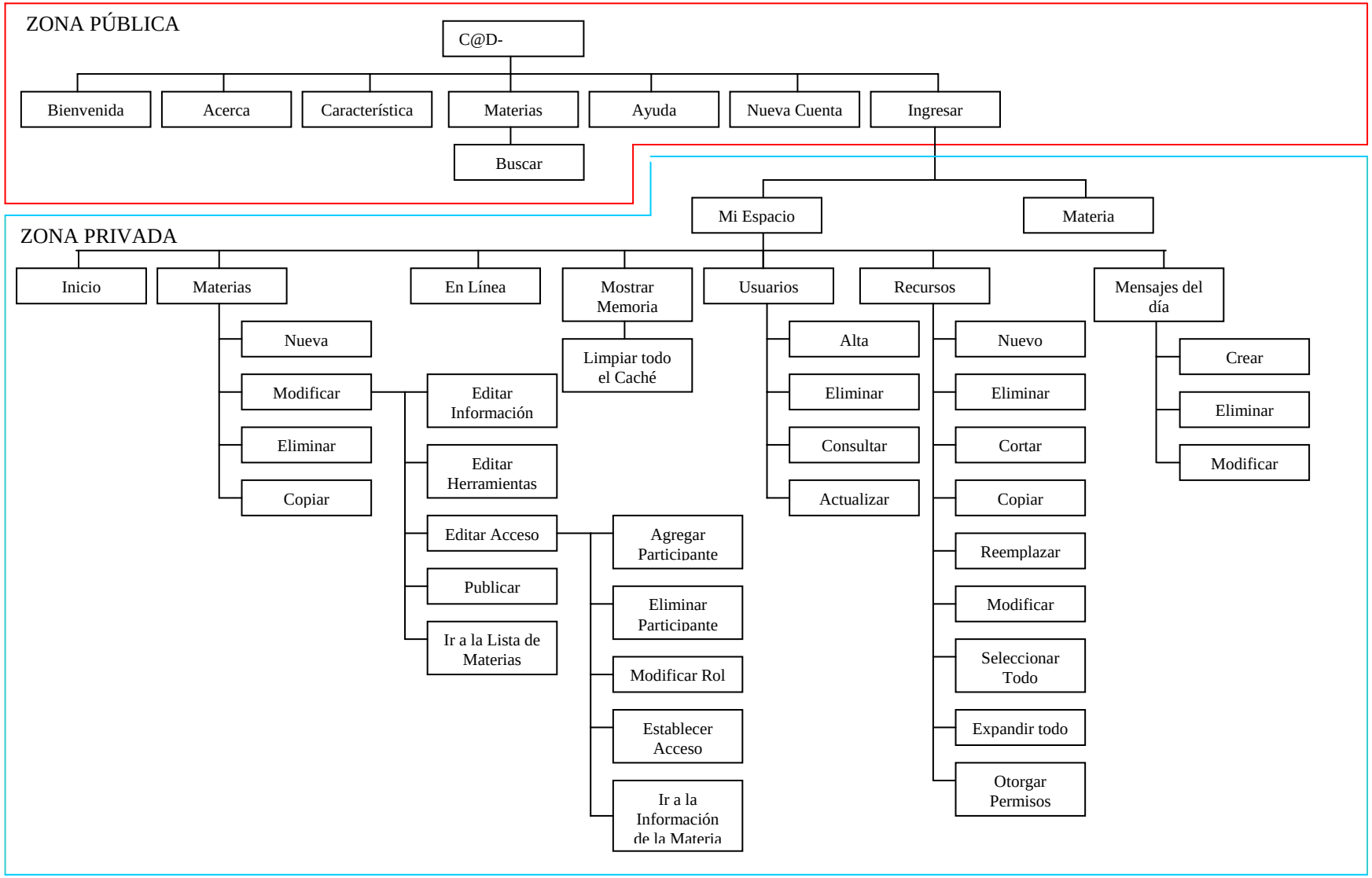

Figura 2 Diagrama de navegación

Como se puede observar en el diagrama la pantalla principal del sitio denominada ZONA PÚBLICA (a la cual puede ingresar cualquier usuario, aún sin estar registrado en el sistema con el simple hecho de acceder a la URL) es la pantalla que muestra información del sitio así como las materias que se imparten dentro de él, y existe una zona denominada ZONA PRIVADA a la que se puede acceder una vez que se cuenta con un usuario y clave dentro del sitio. El detalle de la funcionalidad completa del sistema se detalla más delante.

A continuación se detalla la parte referente a MATERIA (figura 29 y 30), que no es más que la representación de cada una de las materias dentro del sistema. El diagrama de navegación es mostrado en dos partes debido a lo extenso que es, cabe mencionar que se muestra todas las actividades que se pueden realizar mismas que son restringidas de acuerdo al rol del usuario.

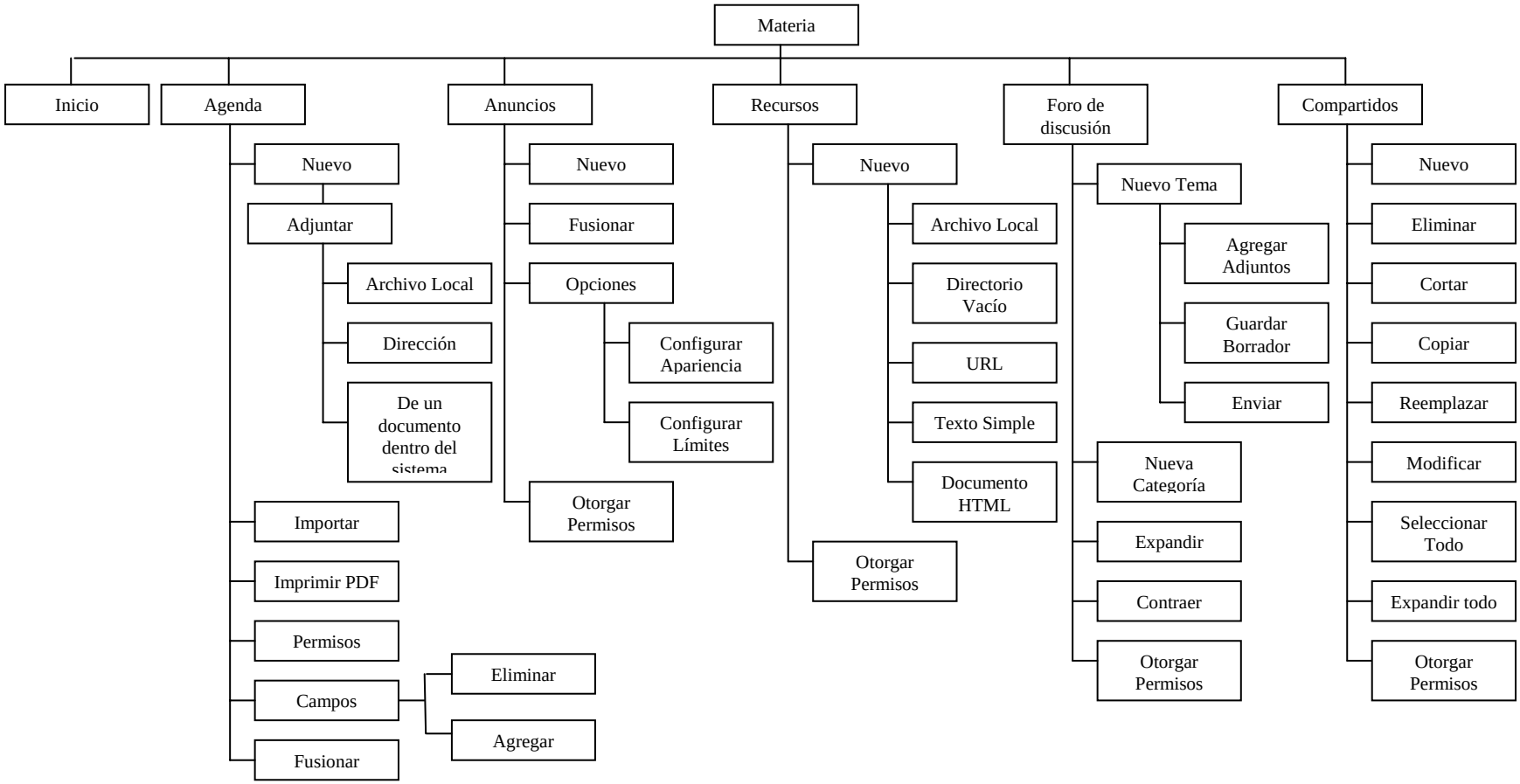

Figura 3 Diagrama de navegación de una Materia

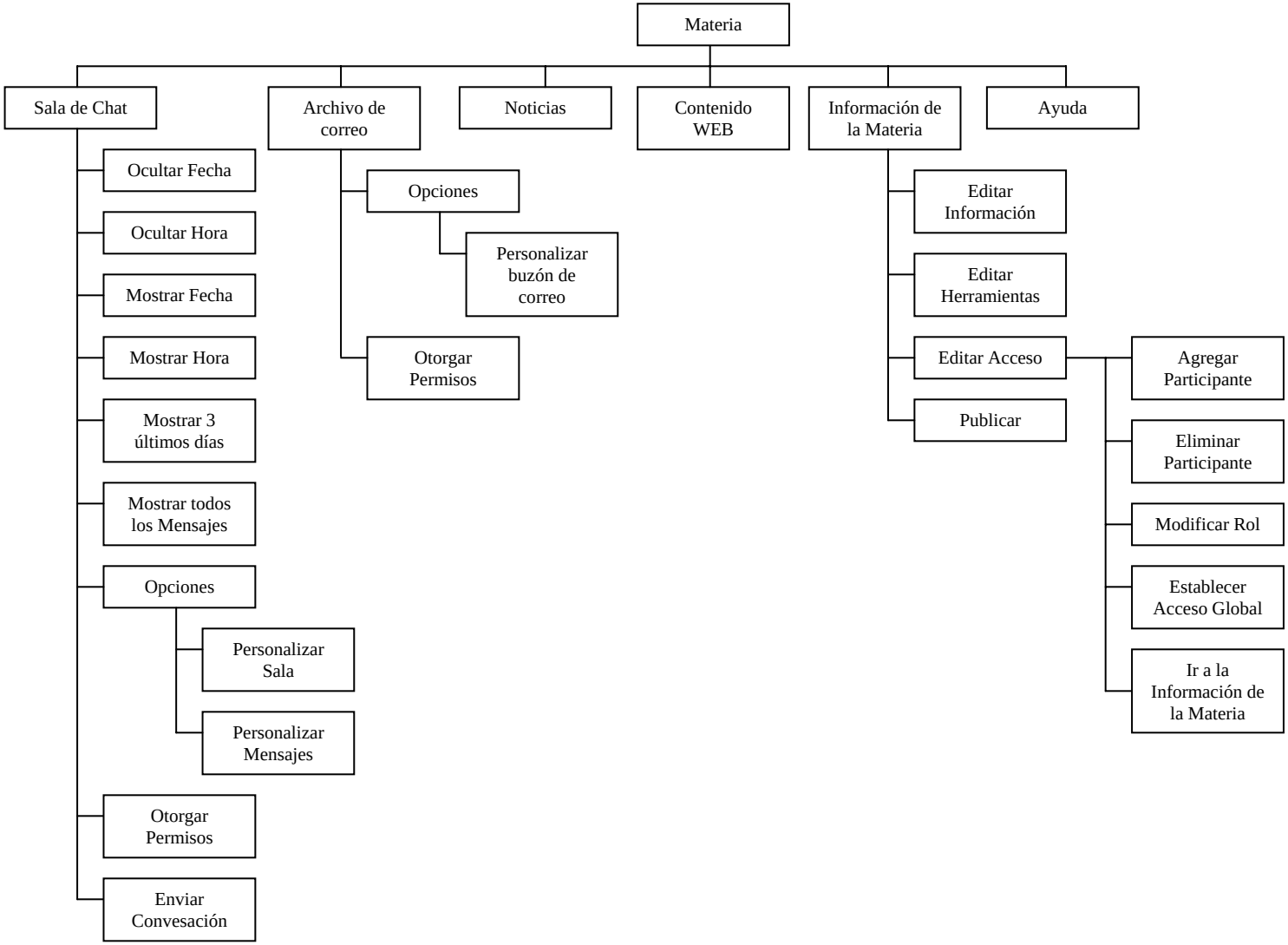

Figura 4 Diagrama de navegación de una Materia

En el desarrollo del presente trabajo de tesis únicamente nos enfocaremos en el rol de Administrador debido a que es él el que construye la plataforma, aunque cabe mencionar que el sistema ha sido adecuado en su totalidad para los tres roles existentes (Administrador, Profesor y Alumno).

5.3. Arquitectura del sistema C@D-CCADET

En la actualidad los usuarios están más familiarizados con las aplicaciones y sistemas de información con interfaces gráficas, y con el uso cada vez más extendido de Internet (caracterizado por interfaces gráficas atractivas y fáciles de usar), lo que ayuda al proceso de adaptación a los sistemas en Web, facilitando su uso y adopción.

Como puede observarse en el siguiente diagrama (figura 31), el sistema tiene tres grandes componentes:

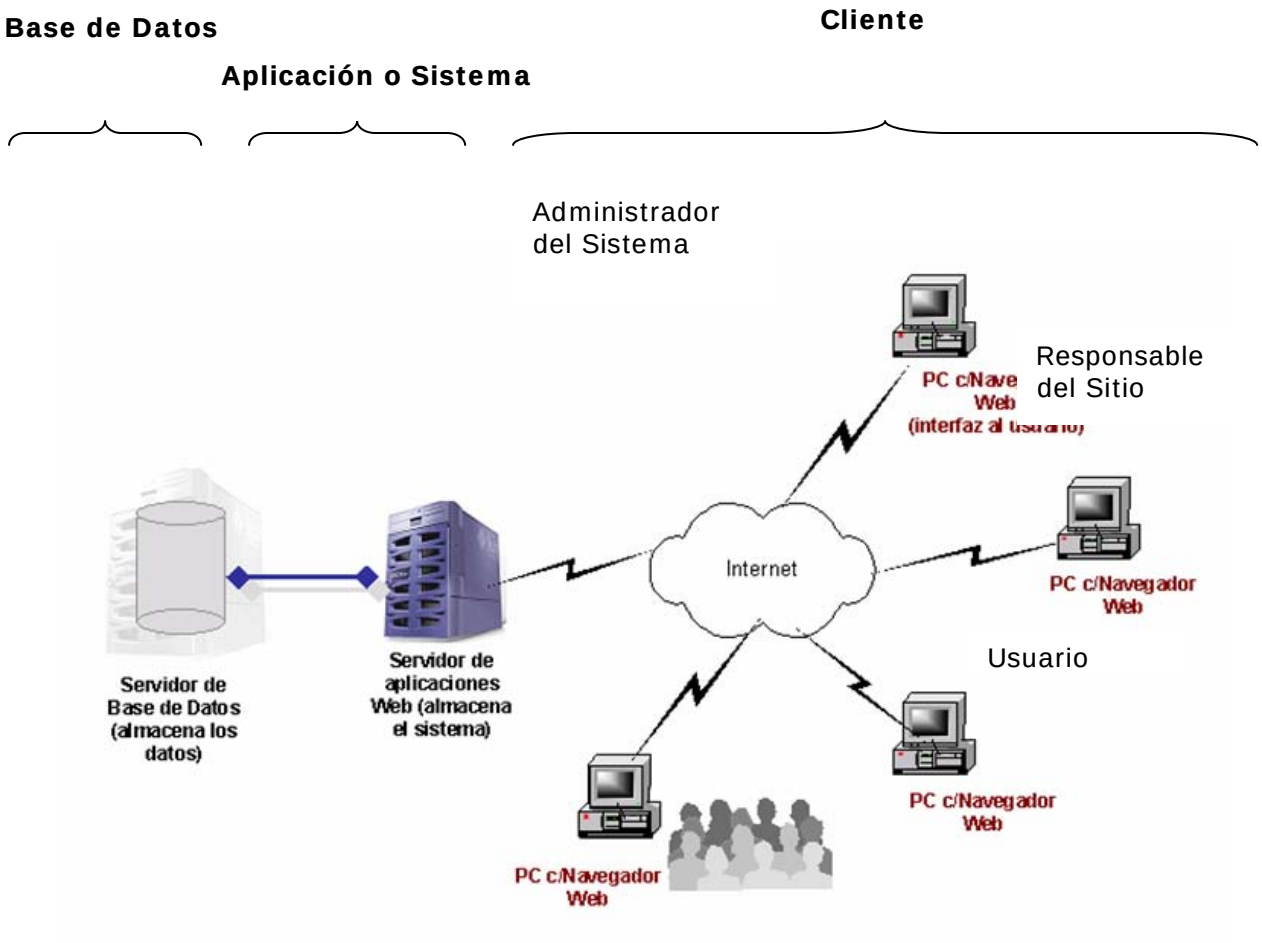

Figura 5 Componentes del sistema

5.3.1. La Base de Datos

Para almacenar físicamente la información manipulada por el sistema es necesario contar con una base de datos que proporcione la infraestructura necesaria para la toma de desiciones, ya que de acuerdo a la definición de la misma se podrá tener características competitivas con otros sistemas agilizando el manejo de la información. Para el presente trabajo manejamos una base de datos completamente orienta a objetos y es centrada en el documento, donde la información se almacena en formato XML o "native XML database", este tipo de bases de datos tiene repositorios con un formato "tipo XML", como puede ser DOM o INFOSET. DOM (Document Object Model) es el API para leer, crear y modificar documentos XML, crea un modelo de objetos en memoria, infoset (Information Set) es un modelo abstracto de datos XML, es la definición de términos. En este mismo "repositorio" (paquete de archivos) se almacenan los índices que se generan por cada documento XML almacenado.

Con esta base de datos no ocupamos SQL como lenguaje de consulta, en lugar de ello se utiliza Xpath.

Las ventajas que se provee con el XML es la flexibilidad que ofrece ya que la información se encuentra de manera estructurada y descriptivamente visual. La figura 32 muestra el diseño de la base de datos.

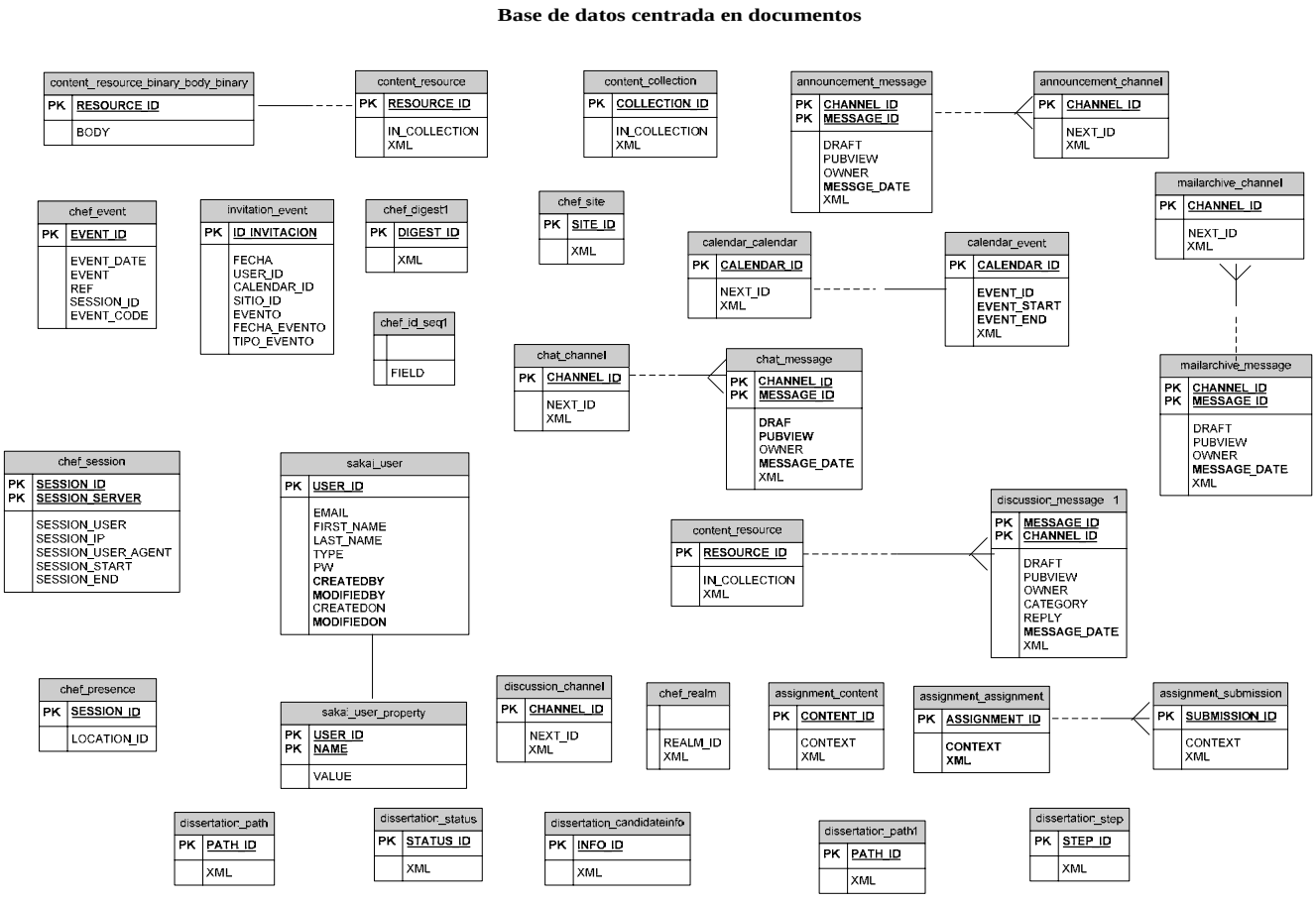

Figura 6 Diagrama de Base de Datos

5.3.2. La Aplicación o el Sistema

La aplicación se compone por una gran cantidad de programas que residen en el servidor de aplicaciones WEB. En el proyecto trabajamos con TOMCAT que es un "java conteiner" capaz de implementar servlets y jsp´s, básicamente cuando el Web Server recibe una petición de un Servlet le pasa el control al Tomcat para que este se encargue de instanciar las clases correspondientes en la JVM. El Web Server recibe una petición y reconoce el código java y se lo pasa al TOMCAT para que este se encargue de ejecutar el código correspondiente.

La plataforma de educación a distancia que presentamos se encuentra en tres capas:

- Capa presentación.
- Capa Lógica.
- Servicios de Aplicaciones (Application Services).

Dentro del sistema se cuenta con un grupo de servicios que pueden ser usados por otras herramientas o Servicios de Aplicaciones para interactuar con el sistema como sea necesario.

Dentro de la capa lógica el código esta escrito en Java que se ejecuta necesariamente en el servidor e interactúa con la capa de presentación usando JavaBeans. En la capa de presentación se encuentran los Java Server Faces (JSF´s) que nos permiten crear nuestros propios componentes de interface de usuario del lado del servidor de aplicaciones Web basadas en Java y que se basan en el modelo MVS (Modelo Vista Controlador), que no es más que un patrón que nos permite/obliga a separar la lógica de control, la lógica de negocio y la lógica de presentación y tener así un modelo que gestiona los datos, una vista que gestiona cómo se muestran esos datos y un controlador que determina qué modificaciones hay que hacer cuando se interactúa con un elemento.

Los Servicios Web (Web Services) son servicios que nos permiten que los programas escritos en lenguajes y plataformas diferentes puedan establecer comunicación de forma estándar.

5.3.3. El Cliente

El **"Cliente"** es el nombre que reciben cada uno de los equipos que los usuarios utilizan para interactuar con el sistema, es decir, para registrar y consultar la información. En cada uno de dichos equipos debe encontrarse instalado un software llamado genéricamente "Navegador Web", que se encarga de presentar el sistema al usuario.

Estos tres componentes utilizan Internet como medio de comunicación para la transmisión de la información como imágenes, gráficas, pantallas de captura, reportes, entre otros, lo que aprovecha al máximo las ventajas que se mencionaron con anterioridad.

5.4. El sistema

Derivado del análisis anterior y de la detección de los roles dentro del sistema los cuales son: administrador, profesor y alumno se presenta este trabajo acotándolo a la presentación únicamente del rol de administrador por ser muy extenso el proyecto, sin embargo se aclara que el sistema se encuentra totalmente adecuado para nuestra plataforma de educación a distancia.

El sistema consta de de dos principales zonas (mencionadas únicamente al principio de este capítulo), la zona pública que es una parte del sistema a la cualquier usuario que accese la URL a través del navegador podrá accesar, y la zona privada la cual contiene todas las materias a las cuales el usuario (profesor, alumno) tiene acceso así como el sitio de Mi Espacio, el cual es un sitio personal que nadie más tiene acceso más que el propietario del mismo.

5.4.1. Zona pública

Una vez que se ha llegado a la dirección del sistema, se muestra la pantalla la cual todo tipo de usuarios (administrador, profesor y alumnos) puede visualizar incluyendo a los NO dados de alta en el sistema, esta pantalla contiene las siguientes ligas mostradas en la figura 33.

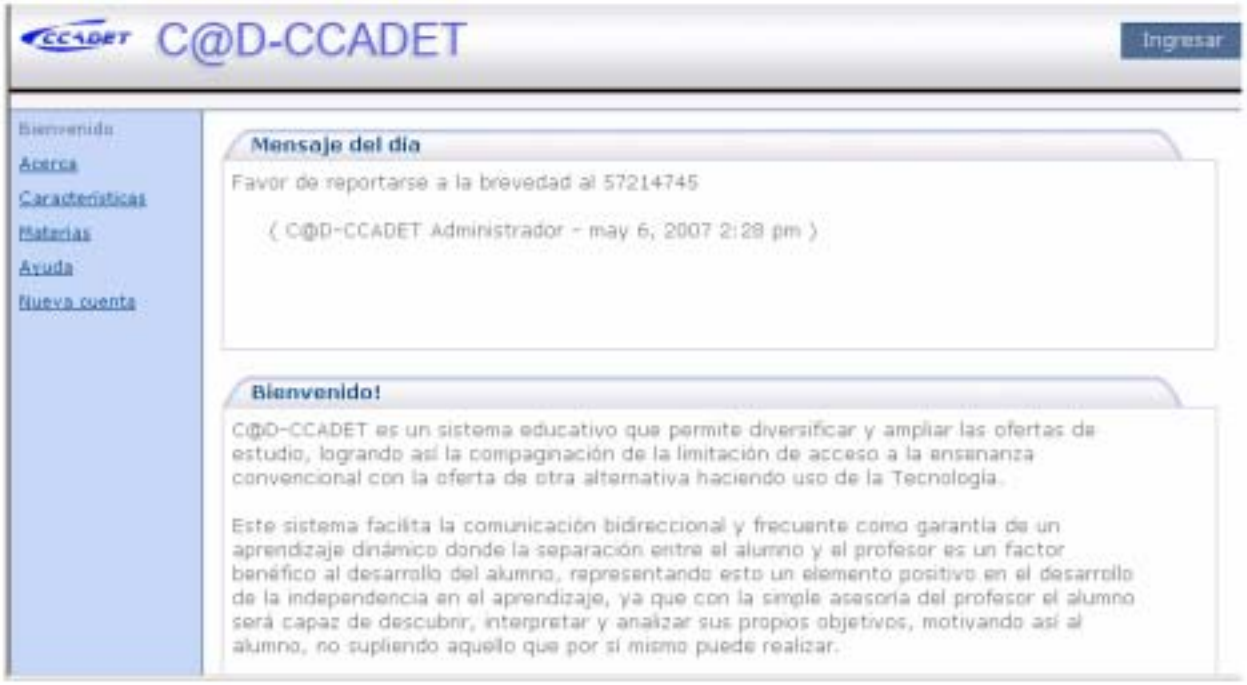

Figura 7 Pantalla de Zona Pública

- **Bienvenido**

Módulo que muestra la pantalla principal de La plataforma de Educación a Distancia, en la que se despliega el mensaje de bienvenida dado por el Sitio a los usuarios que accedan al mismo, ya sea para consultar únicamente o para inscribirse a alguna materia que haya resultado de su interés.

- **Acerca**

Muestra la información del sistema, sus colaboradores y la descripción del proyecto, así como la versión del mismo.

- **Materias**

Muestra información de cada uno de las materias dadas de alta en el sistema así como una descripción breve de la misma, lo cual permite al usuario que accesa al sistema con el interés de tomar alguna materia en el sitio evaluar la opción que le resulte de su interés.

- **Ayuda**
  Visualiza información de ayuda, preguntas frecuentes sobre los problemas más comunes que puede tener el usuario.

- **Nueva Cuenta**
  Permite darse de alta como usuario de tipo registered únicamente para probar el sitio Mi Espacio.

A continuación se detalla a fondo las acciones permitidas al administrador, cabe mencionar que el administrador cuenta con todos los permisos dentro del sistema.

5.4.2. Zona Privada

Es esta zona el sistema le presenta al administrador las Materias que ha dado de alta en el sistema así como toda la información referente a las misma, adicional a esto le presenta el sitio Mi Espacio el cual es propio del Administrador y a través del cuál se encarga de la administración del mismo.

5.4.2.1.Mi espacio

El administrador cuenta con todos los permisos del sistema, y desde que es autenticado, el sistema le proporciona a través del sitio de trabajo Mi Espacio las ligas a través de las cuales lleva a cabo la administración del sistema, y se muestran de manera resumida en el siguiente Mapa General (figura 34).

Mapa general de mi espacio para el rol de Administrador

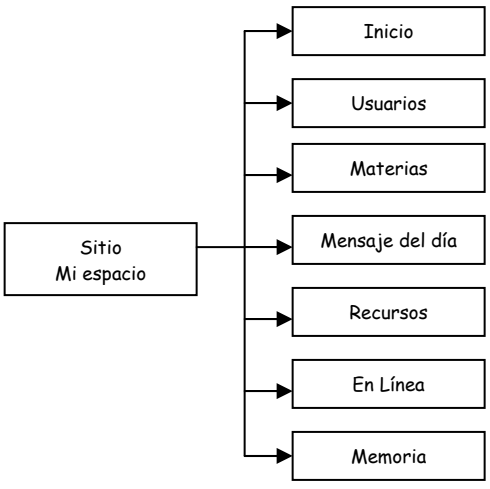

Figura 8 Mapa general de mi espacio

A continuación se muestra el diagrama de casos de uso propios del rol de Administrador (figura 35), que no es más que la representación de cómo opera el administrador con el sistema.

Diagrama de Casos de Uso “Administrador”

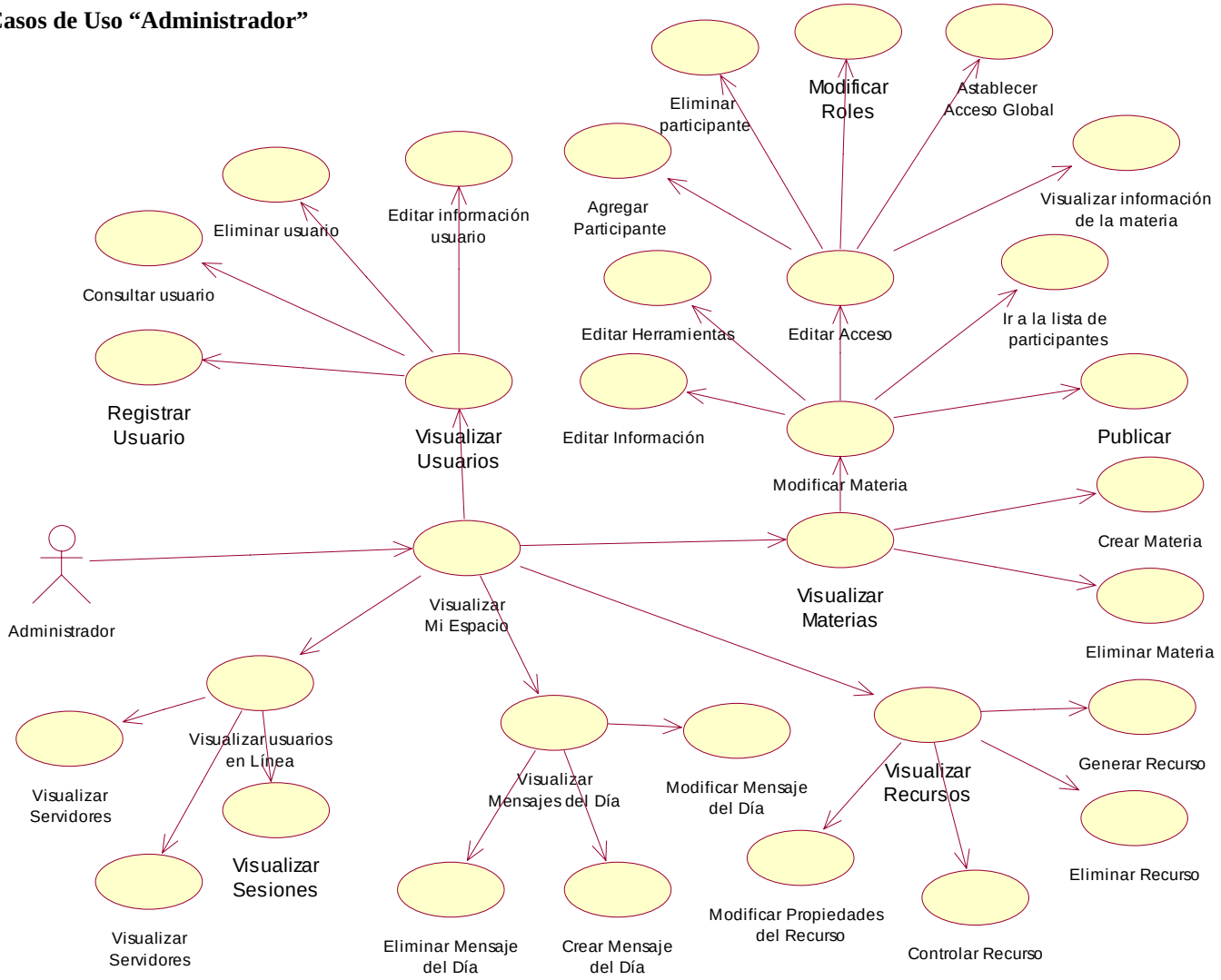

Figura 9 Diagrama Casos de Uso

5.4.2.2. Inicio

Dentro de este módulo al administrador le es posible consultar la sección de **Mensajes del día**, que contiene los mensajes enviados por el mismo a los usuarios del sistema, y la información del sitio **Mi Espacio**, que contiene información particular para cada usuario.

Dentro de esta página el administrador puede configurar la altura y el título de la sección, además de la información que se mostrará en pantalla.

5.4.2.3. Usuarios

Este módulo muestra al administrador la lista de los usuarios existentes en el sistema tal como se muestra en la figura 36, dentro de este módulo puede realizar las siguientes acciones:

- **Dar de alta un nuevo usuario**. Le permite crear un usuario dentro del sistema capturando la información solicitada.
- **Consultar y/o actualizar la información de un usuario existente**. Le permite consultar y actualizar información de un usuario.
- **Eliminar un usuario.** Le permite eliminar un usuario o varios de los que se encuentran registrados en el sistema.

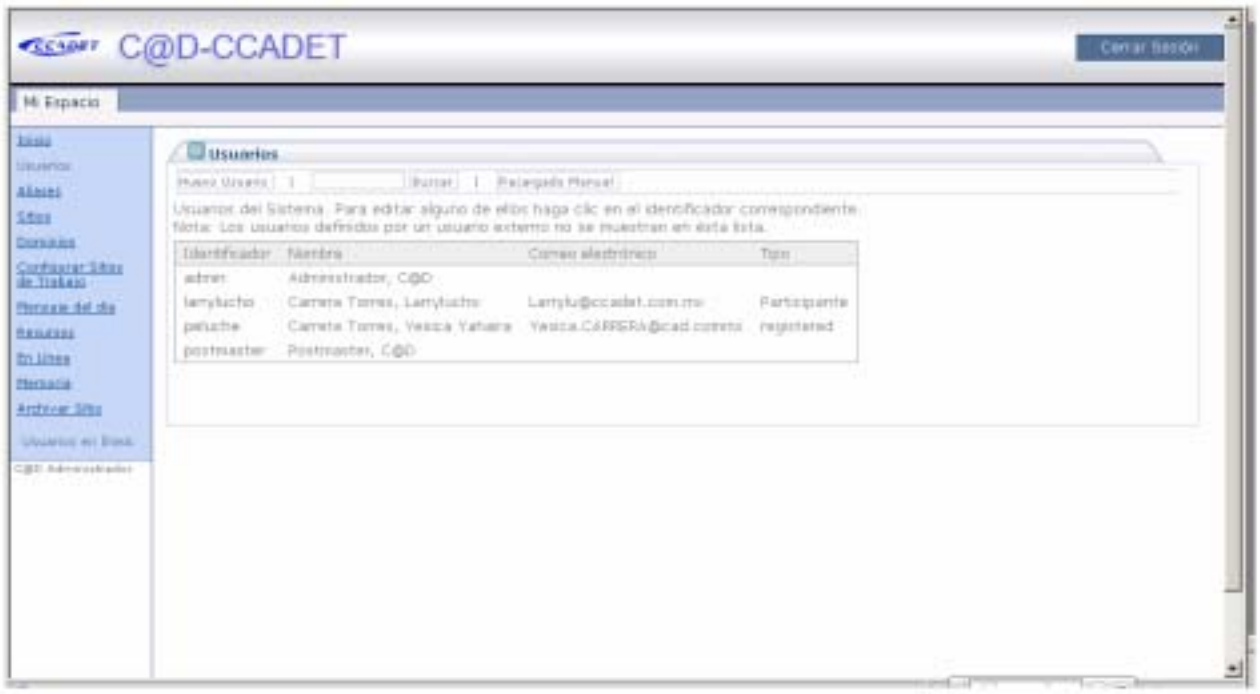

Figura 10 Pantalla de Usuarios

5.4.2.4.Materias

Este módulo es el más importante del sistema ya que es el que le permite al administrador la creación de materias y su manipulación, como eliminar, actualizar las propiedades y el acceso a la misma, dichas acciones se describen a continuación:

- **Nueva materia.** Le permite crear una nueva materia dentro del sistema a partir de plantillas con las herramientas configuradas de manera estándar.
- **Modificar materia.** Le permite modificar la información de las materias registradas dentro del sistema.
- **Eliminar materia.** Le permite borrar las materias registradas dentro del sistema.

La siguiente pantalla (figura 37) muestra detalles de cada una de las vistas de las materias registradas y a través de la cual le es posible al administrador seleccionar con la opción **vista**, las materias que se desean consultar. Para ver las materias disponibles sólo debe seleccionas el encabezado de cada característica de la misma.

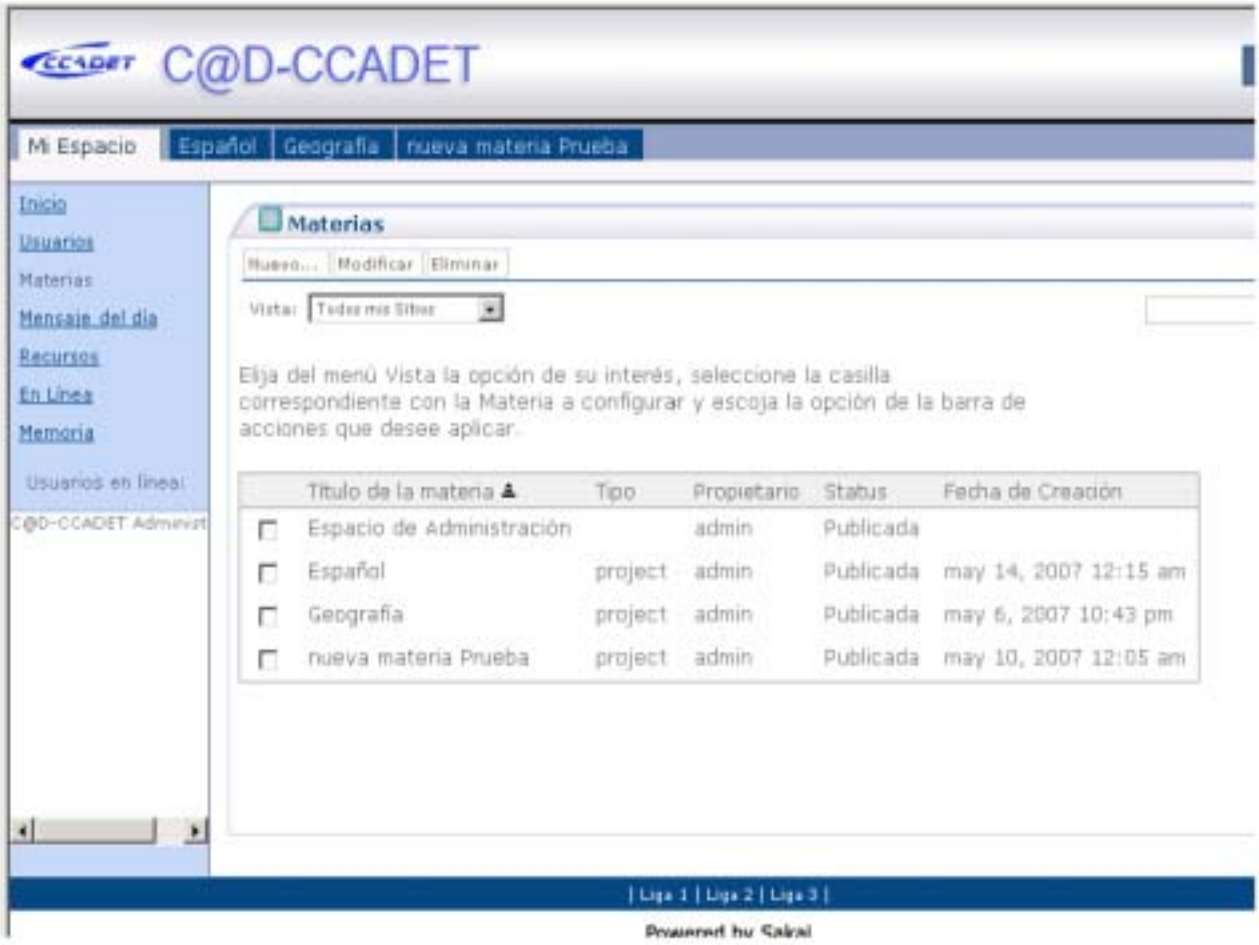

Figura 11 Materias

Las opciones disponibles son:

- **Páginas.** Le permite ver las materias en listas de quince y presenta las opciones de **Ver página anterior** y **Ver página siguiente**. Esta opción es útil cuando existen varias materias en el sistema.

- **Mostrar todo.** Le permite ver en una lista todas las materias de acuerdo a la vista seleccionada.

5.4.2.4.1. Nueva Materia

Esta opción es la que le permite al administrador crear una nueva materia en el sistema tal como se muestra en la figura 38.

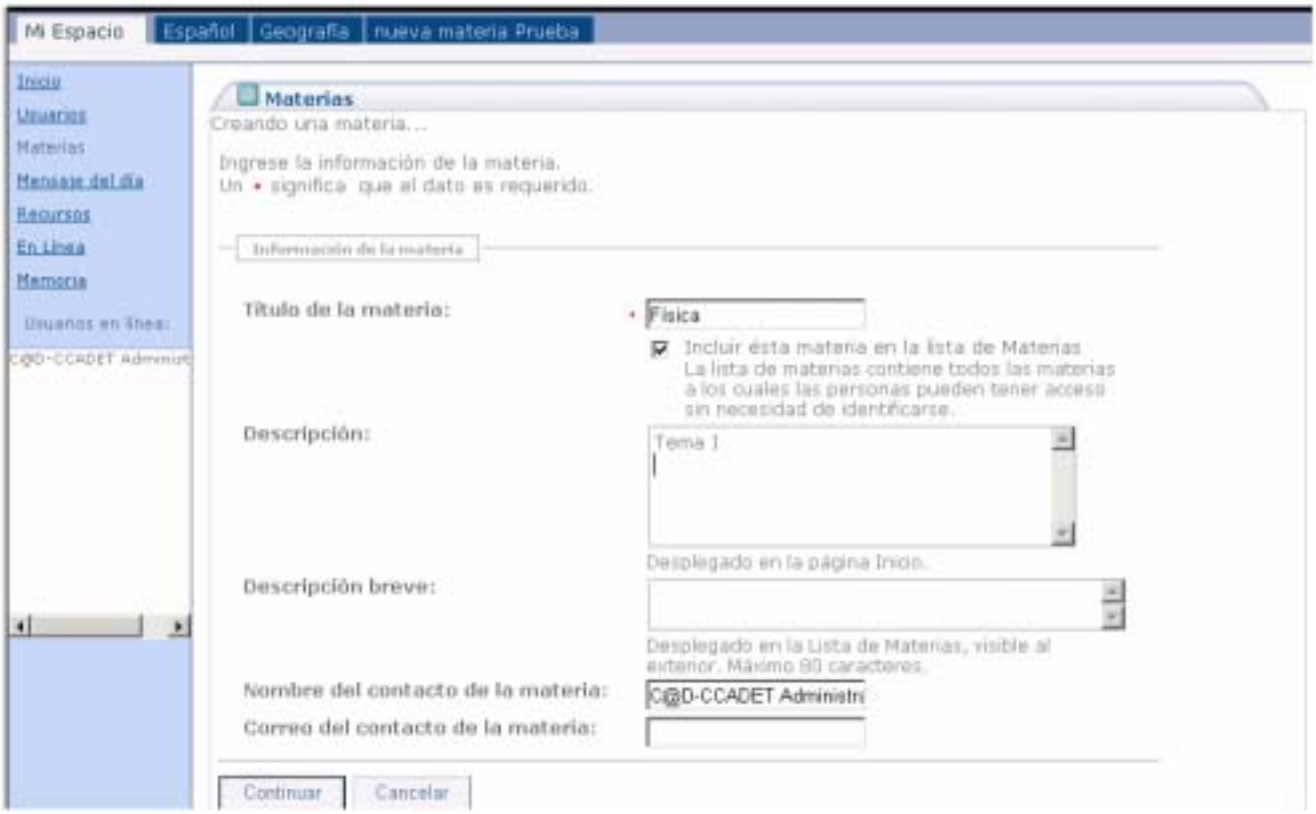

Figura 12 Alta de Materia

Una vez capturados los datos de la materia deberá seleccionar las herramientas con las cuales trabajará (agenda, anuncios, recursos, foro de discusión, compartidos, sala de Chat, archivo de correo, noticias, contenido Web, Información de la Materia y/o Ayuda) que puede ser una o varias. De ser seleccionada la herramienta **archivo de correo,** el administrador deberá capturar la dirección correspondiente a la lista de correo de la materia, la cual debe ser representativa del mismo.

Una vez seleccionadas las herramientas requeridas para la materia el administrador deberá confirmar al sistema la creación de misma tal como se muestra en la figura 39.

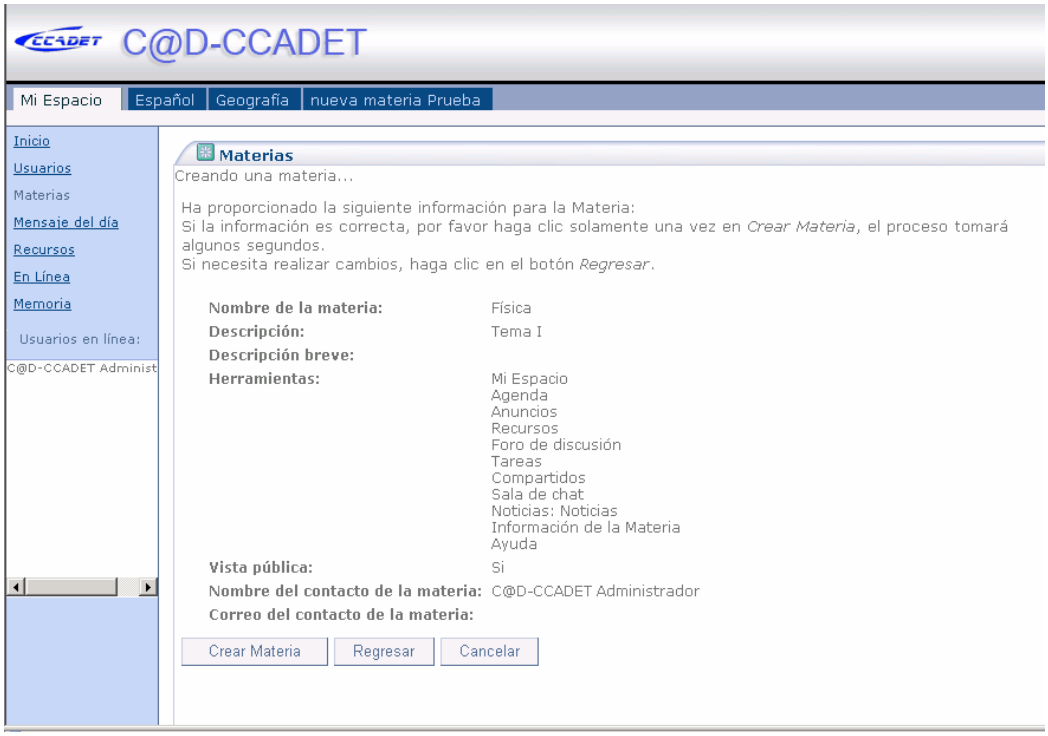

Figura 13 Confirmación de creación de Materia

Una vez creada la materia el administrador deberá definir si será o no publicada, como se muestra en la figura 40. Una materia publicada es aquella que está lista para ser utilizada, es decir, que se ha terminado de configurar, sino está publicada sólo el administrador tiene acceso a la materia.

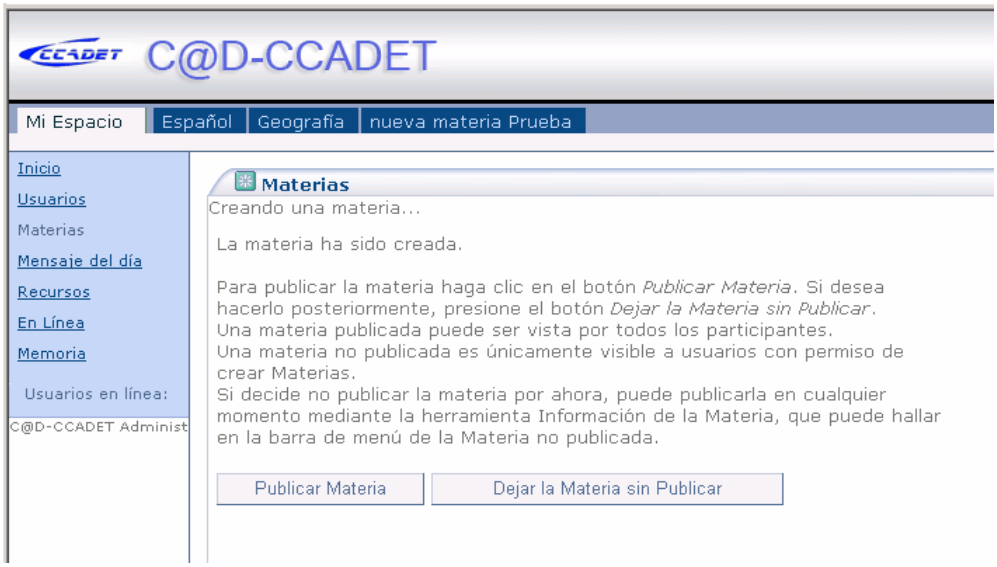

Figura 14 Publicación de Materia

Ya creada la materia se registra fecha de creación y se muestra en la pantalla de materias registradas como se muestra en la figura 41.

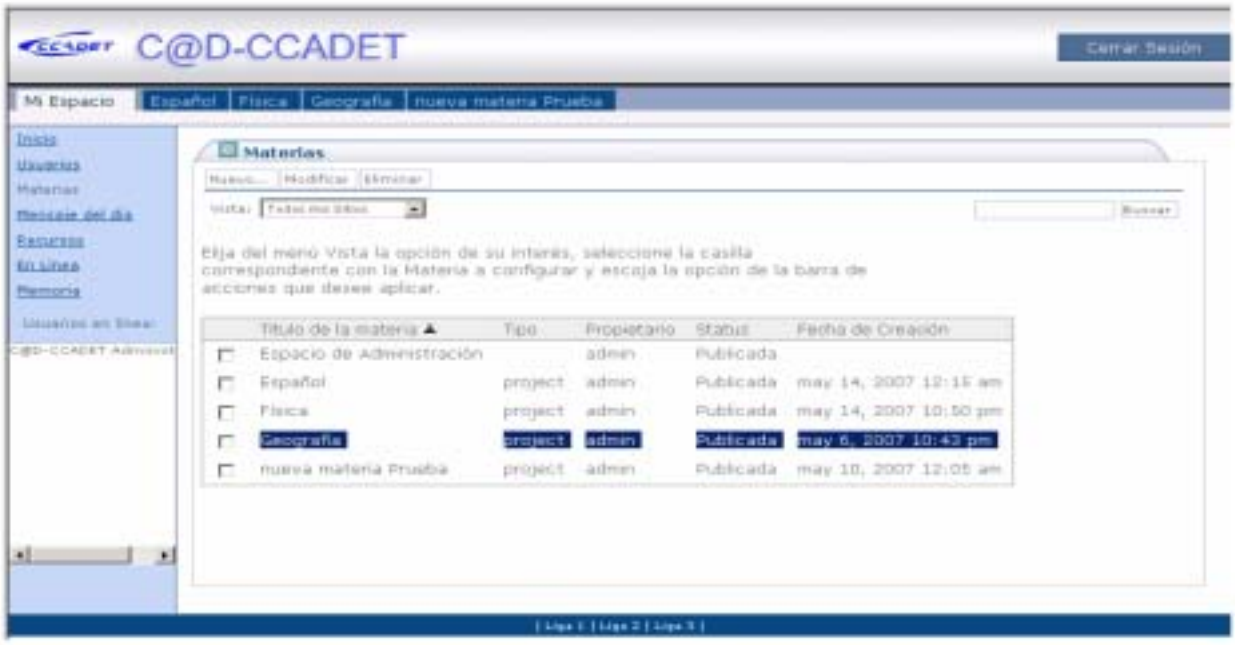

Figura 15 Materias registradas

5.4.2.4.2. Modificar materias

Esta opción le permite al administrador actualizar información de una materia tal como se muestra en la figura 42.

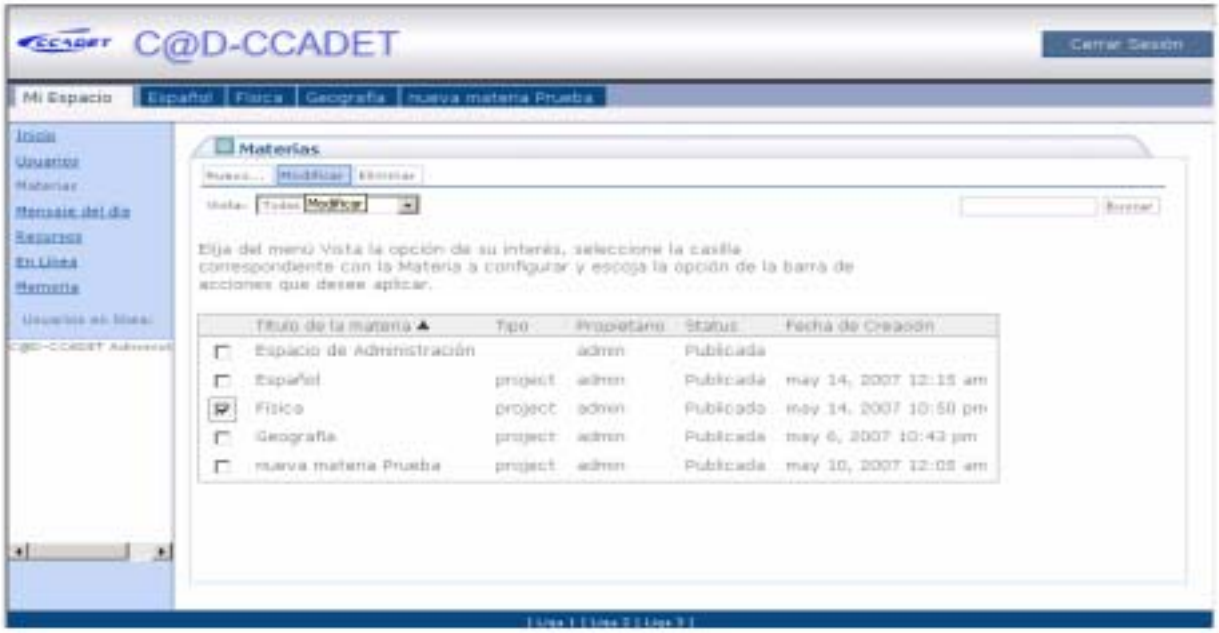

Figura 16 Modificar Materia

Una vez localizados los datos de la materia, el sistema le mostrará la pantalla correspondiente (Figura 43).

Figura 17 Presentación de Materia modificada

La opción de **Ir a la lista de materias** le permite regresar a la pantalla que presenta la lista de materias.

Las acciones a realizar dentro de este módulo son las siguientes:

- **Editar Información.** Esta opción le permite al administrador editar/modificar la información de la materia proporcionada al momento en que se creó (figura 44).

Figura 18 Edición de Información

- **Editar Herramientas.** Esta opción permite al administrador agregar o eliminar herramientas de una materia (figura 45).

Figura 19 Edición de Herramientas

- **Editar acceso. Descripción General.** Permite al administrador controlar a los usuarios y su acceso a la materia (figura 46).

Desde esta pantalla puede realizar las siguientes acciones:

- **Agregar participante.** Le permite agregar nuevos participantes a una materia.
- **Eliminar participante.** Le permite eliminar usuarios de una materia.
- **Modificar rol(es).** Le permite modificar los permisos de los usuarios sobre una materia de acuerdo al rol que poseen.
- **Acceso global.** Le permite establecer el control de acceso a una materia permitiendo definir si una materia será o no publicada.
- **Ir a información de la materia**. Le permite regresar a la pantalla anterior que muestra la información de la materia seleccionada.

Figura 20 Edición de acceso

- **Editar Acceso. Agregar Participante.** A través de esta opción el administrador podrá agregar participantes a una materia capturando la información requerida por el sistema tal como se muestra en la figura 47.

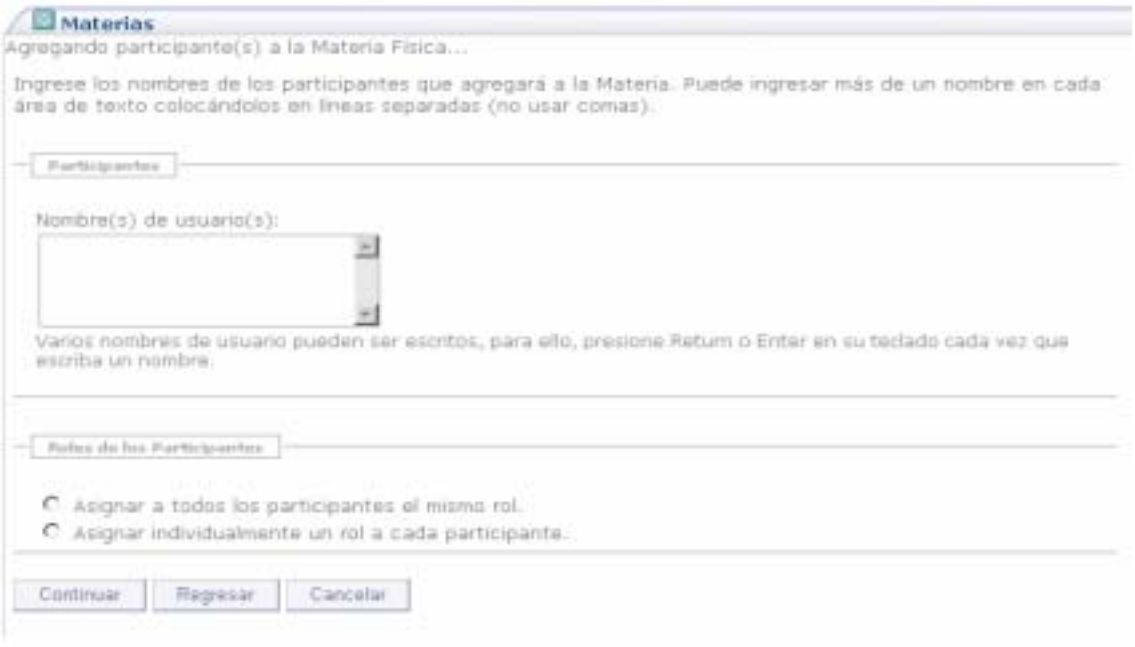

Figura 21 Agregar participante

Al concluir con la captura de datos necesarios el administrador debe asignar los roles correspondientes que puede ser de alumno (access) o profesor (maintain), adicional a esto el administrador puede asignar el mismo rol a todos los participantes de la materia o hacerlo de manera individual (figura 48).

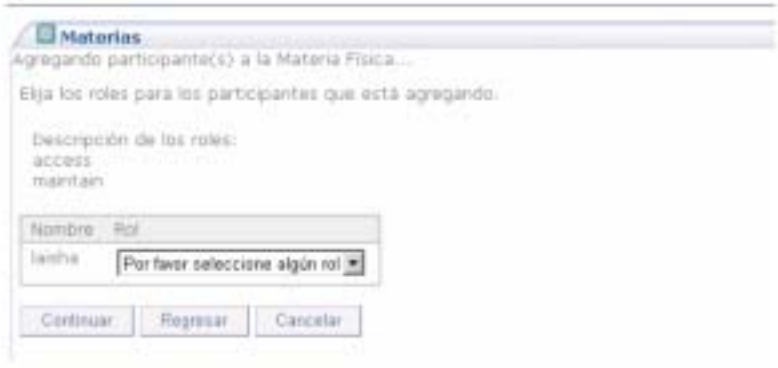

Figura 22 Asignación de rol

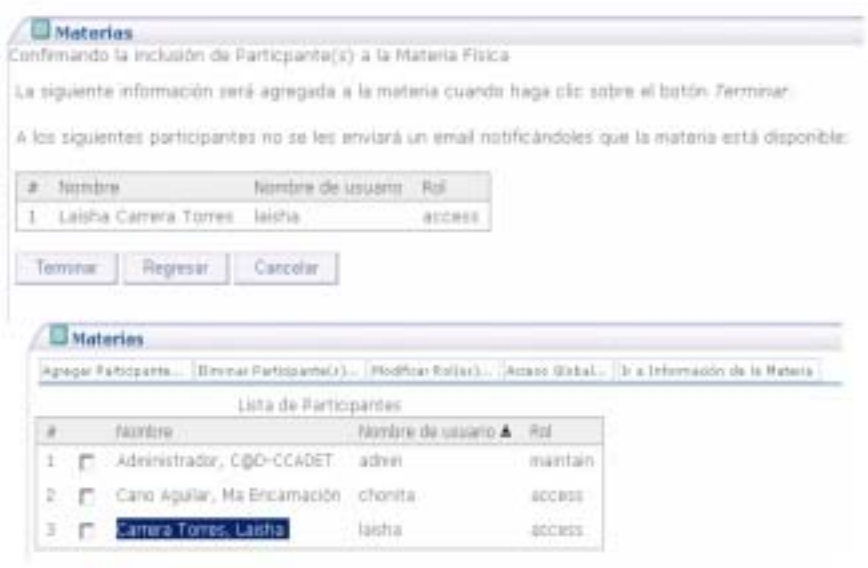

Figura 23 Confirmación asignación de rol

- **Editar Acceso. Eliminar Participante(s).** Una vez inscritos en una materia el administrador tiene la opción de eliminar un participante o varios, seleccionándolo(s) de la lista (fig 50).

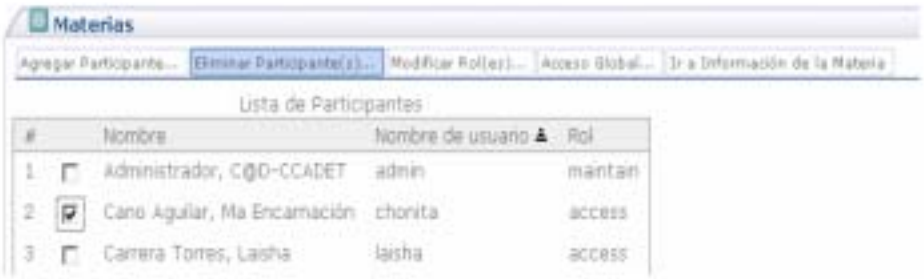

Figura 24 Eliminar participante

- **Editar Acceso. Modificar Roles.** Esta opción le permite al administrador modificar el rol de los usuarios de la materia seleccionando el(los) usuario(s) al(os) que se le(s) desea modificar el rol, y presionando el botón correspondiente a esta acción, tiene la opción de modificar los roles de varios participantes ya que el sistema presenta la opción de asignar un mismo rol a todos los usuarios seleccionados o asignar el rol de forma individual (figura 51).

Figura 25 Modificar rol

- **Publicar.** Esta opción le permite al administrador publicar la materia después de haber sido creada, o por el contrario, volverla no pública en caso de ser pública (figura 52).

Figura 26 Publicar Materia

5.4.2.5. Mensajes del día

Este es un módulo en donde el administrador publica mensajes (figura 53), dentro de este módulo es posible realizar las siguientes acciones:

- Crear nuevos mensajes. Una vez que captura los datos requeridos por el sistema, el administrador tendrá la opción de enviar el mensaje, ver la vista preeliminar, guardar un borrador o cancelar.

  NOTA: Las opciones Vista Pública, Adjuntos y Notificación vía correo electrónico no aplican para un mensaje del día.

- Eliminar mensajes. Esta opción permite al administrador eliminar un mensaje seleccionando la opción correspondiente a esta acción.

- Modificar mensajes. El administrador podrá modificar mensajes seleccionando el botón correspondiente a esta acción, el sistema le muestra una pantalla con la información del mensaje y la posibilidad de modificarla.

NOTA: Un mensaje es público desde el momento en que es creado, no puede ser enviado por correo.

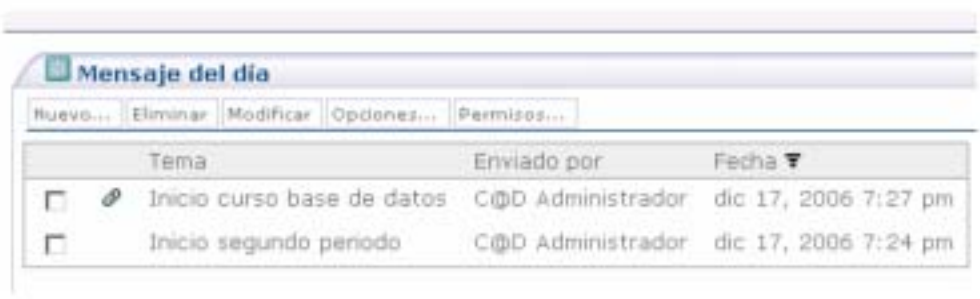

Figura 27 Mensajes del día

En esta pantalla con la opción de **Opciones** el administrador puede configurar la presentación de los mensajes del sistema, el contenido del mismo y cantidad de mensajes que desea sean mostrados por página.

Figura 28 Configuración presentación mensajes del día

Si en la pantalla anterior (figura 54) el administrador seleccionada la opción de **Mostrar todas las columnas**, le es posible ordenar los mensajes seleccionando el título de la columna por la que se desea ordenar (orden ascendente, orden descendente).

5.4.2.6. Recursos

Dentro de este módulo el administrador tiene la posibilidad de realizar las siguientes acciones (figura 55).

- **Generar recursos de distintos tipos.** Le permite crear un recurso dentro del sistema.
- **Eliminar los recursos.** Le permite eliminar recursos existentes dentro del sistema.
- **Controlar los recursos.** Le permite controlar de los recursos del sistema, con opciones como cortar, copiar o reemplazar.
- **Modificar propiedades de los recursos.** Le permite actualizar la información de los recursos registrados en el sistema.

Figura 29 Recursos

Con los siguientes directorios el administrador puede controlar los recursos:

- **Attachment**. Almacena los recursos que fueron anexados como parte de un anuncio, mensaje del día o mensaje en el foro de discusión.
- **Group** . Almacena todos los recursos de las materias existentes.
- **Group-user.** Almacena los recursos de la herramienta **Compartidos**.
- **Public.** Almacena los recursos públicos existentes en la zona pública del sistema.
- **User.** Almacena los recursos que cada usuario guarda en **Mi Espacio.**

5.4.2.6.1. Generar recursos

Esta opción permite al administrador crear un nuevo recurso (figura 56) en el sistema. Los recursos los cuales puede crear son:

- **Archivo local.** Máximo 20 Mb.
- **Directorio vacío**
- **URL**
- **Texto simple**
- **Documento HTML**

Para crear un nuevo recurso el administrador deberá presionar el botón correspondiente a esta acción y seleccionar el **tipo de recurso** que se desea generar.

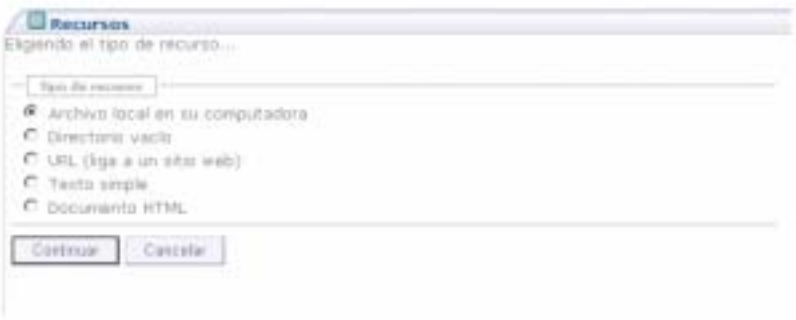

Figura 30 Crear Recurso

El administrador deberá indicar el número de elementos a crear, llenar pantalla de captura y para finalizar presionar **Publicar** (figura 57).

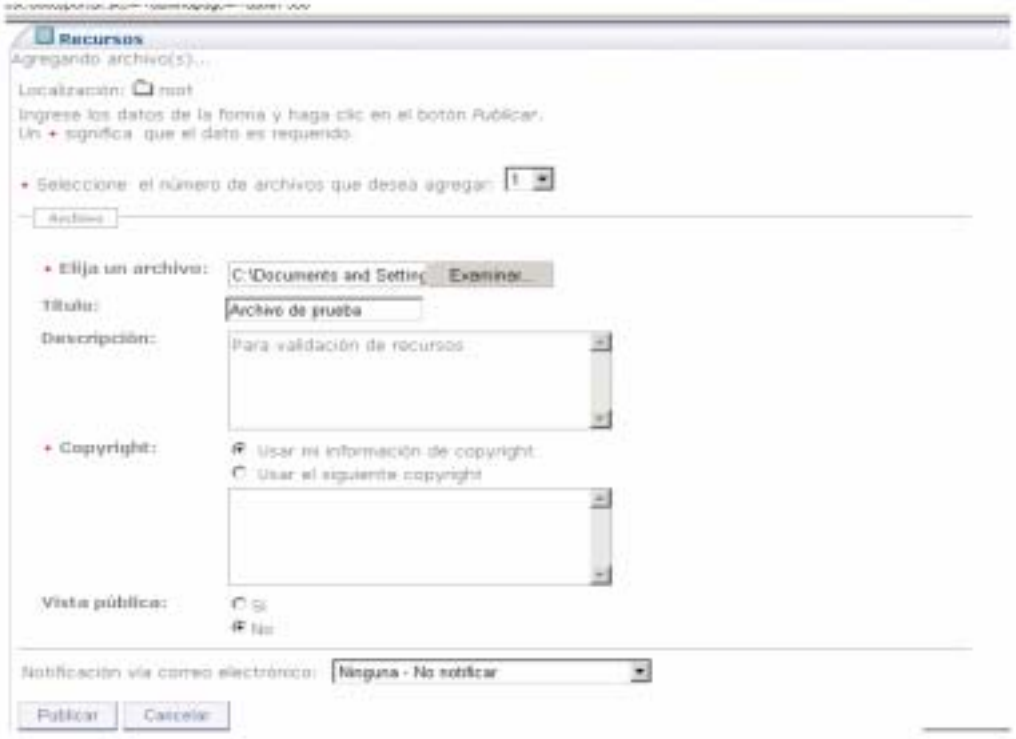

Figura 31 Publicar Recurso

5.4.2.6.2. Controlar los recursos

Una vez que el administrador localiza el documento en la página principal de recursos, las operaciones que podrá realizar son (figura 58):

- **Cortar**. Le permite mover archivos de directorios.
- **Copiar**. Le permite crear una copia de un archivo seleccionado y pegarlo en otro.
- **Pegar**. Le permite pegar los archivos que fueron cortados o copiados.
- **Pegar acceso directo.** Le permite crear una referencia a los archivos que fueron copiados. Esta acción la tendrá disponible después de haber copiado o cortado un recurso.
- **Reemplazar**. Le permite sustituir el archivo seleccionado por otro.
- **Seleccionar Todo.** Le permite seleccionar todos los archivos y directorios.
- **Deseleccionar Todo.** Le permite eliminar la selección de todos los archivos y directorios que previamente fueron seleccionados.
- **Expandir Todo**. Le permite ver el contenido de los directorios existentes.
- **Contraer Todo.** Le permite ver el nombre de los directorios sin ver contenido.

Si no existen recursos en este módulo, las únicas opciones disponibles para el administrador serán nuevo y permisos. Una vez generados nuevos recursos estarán disponibles.

NOTA: Para los directorios no están disponibles las opciones de **copiar**, **cortar**, **pegar** y **pegar acceso directo**.

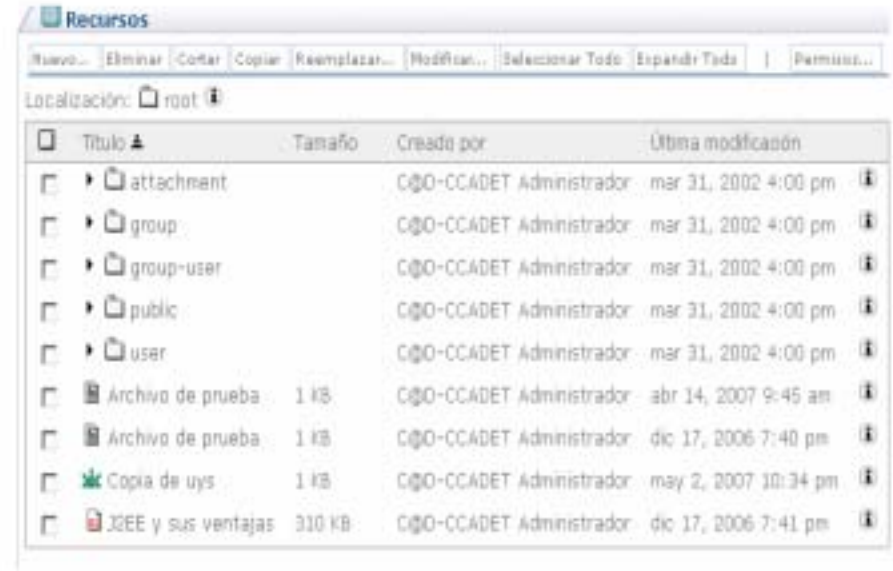

Figura 32 Operaciones sobre los Recursos

5.4.2.6.3. Modificar propiedades de los recursos

Esta opción permite al administrador actualizar la información de un recurso por medio de la selección del nombre del mismo y presionando el botón de **Modificar** (figura 59)**.**

Figura 33 Modificar Recurso

Al terminar de modificar el administrador deberá presionar el botón de **Publicar**, después el sistema le mostrará la pantalla de recursos existentes.

NOTA: Desde esta sección le es posible al administrador realizar cambios o actualizaciones a directorios, URL´s, archivos de texto, documentos HTML y accesos directos.

5.4.2.7.En línea

Permite al administrador ver de manera detallada las herramientas de las materias existentes en el sistema y algunas características específicas de las mismas, así como información referente a las sesiones actuales de los usuarios (figura 60) y a los servidores utilizados.

Figura 34 Usuarios en Línea

5.4.2.8.Memoria

Le muestra al administrador la memoria caché utilizada por el sistema y le permite limpiarla presionando el botón de **Limpiar todo el caché** (figura 61)**.**

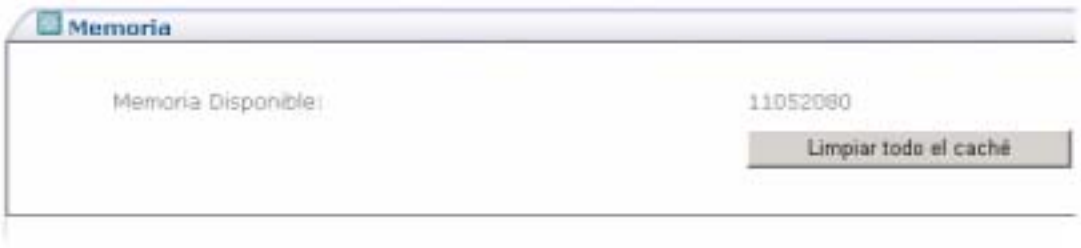

Figura 35 Limpiar Memoria Caché

5.4.2.9.Materias

Las materias son el elemento principal del sistema y son dadas de alta por el administrador que es el que define de acuerdo a las necesidades de las mismas las herramientas de las cuales dispondrá. Una vez generada la materia es responsabilidad del profesor la administración de la misma, aunque cabe recordar que el administrador tiene permisos sobre todo el sistema y puede realizar las actividades propias del profesor, es por ello que a continuación únicamente se describirá de manera breve cada una de ellas.

- **Inicio.** Pantalla principal de la materia, le muestra al usuario la información de la misma, los anuncios recientes, temas de discusión y los últimos mensajes de la sala de chat. Esta pantalla puede ser configurada por el usuario, dando clic en el botón **Opciones** de la sección correspondiente y mostrar la información que el usuario prefiera.

- **Agenda.** Muestra a los usuarios el calendario de actividades en diferentes formatos.

- **Anuncios.** Muestra a los usuarios un listado con los últimos anuncios que ha publicado el profesor, son útiles para informar a los participantes de la materia temas de interés, los anuncios pueden incluir múltiples anexos. Es posible hacer un borrador de un anuncio y salvarlo antes de ser enviado.

- **Recursos.** Permite al profesor y alumnos crear, borrar, y actualizar las propiedades de las carpetas y de los archivos, así como realizar acciones de edición para archivos (copiar, cortar, entre otras).

- **Foro de discusión.** Permite al usuario discutir y/o compartir información importante respecto a un tema o actividad de la materia, ya que permite el manejo de conversaciones estructuradas organizadas en categorías y temas. En esta herramienta el profesor puede brindar la opción a los alumnos de permitir crear sus propios temas de discusión. La presentación de las discusiones puede ser en filas o columnas.

- **Tareas.** Permite a los profesores crear, distribuir, recolectar y clasificar tareas en línea. Las asignaciones son privadas y los envíos de los estudiantes no son visibles para otros alumnos. La herramienta de tareas permite evaluar con puntos, marcar como entregado, con o sin calificación, o ser reenviadas, opción que se puede usar para evaluar borradores de proyectos finales o para permitir a los alumnos corregir una tarea para volverla a enviar al profesor.

- **Compartidos.** Permite al usuario compartir e intercambiar información entre el profesor y cada alumno de modo exclusivo.

- **Sala de Chat.** Permite tener comunicación en línea a los que se encuentren dentro del sistema, la comunicación es en tiempo real y permite más de una Sala de Chat. El sistema muestra los usuarios que están firmados en la materia, de esta manera los alumnos saben con quien pueden hablar. Por defecto los mensajes de la Sala de Chat se guardan y son visibles para todos los usuarios.

- **Archivo de correo.** Almacena los correos enviados a la lista de correo de la materia.

- **Noticias.** Permite al usuario (alumno/profesor) elegir una fuente de información para tener acceso desde su Sitio a las noticias más relevantes del día.

- **Contenido Web.** El usuario podrá tener acceso a Sitios web desde la materia en la que se encuentre.

- **Información de la materia.** Muestra información relacionada con las características de la materia.

- **Ayuda.** Muestra una ayuda general para los problemas más frecuentes, además de un acceso a la guía de ayuda al final de la pantalla.

Al final del menú se presenta la sección usuarios presentes.

Las herramientas que acabamos de mencionar se muestran de manera gráfica en la siguiente imagen (figura 62):

Mapa general de una materia.

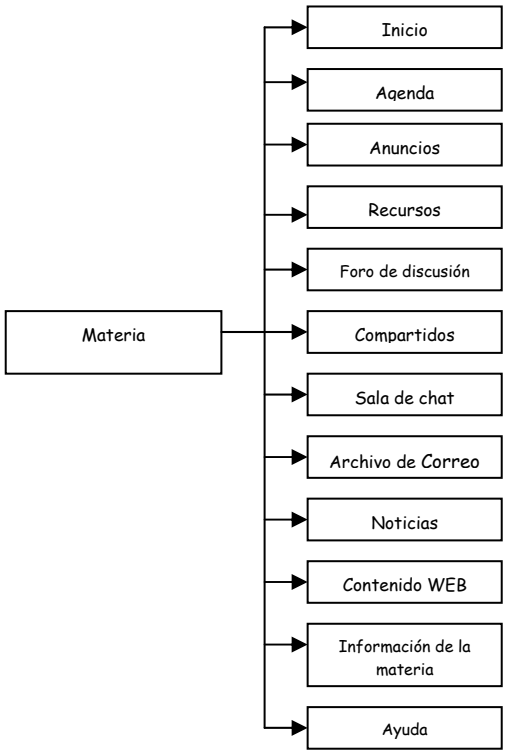

Figura 36 Mapa general de Materia

5.5. Cierre de capítulo

Como se mostró al principio del capítulo en el diagrama de navegación de la materia, cada herramienta cuenta con diversas acciones las cuales permiten la interacción entre los alumnos y el profesor, y como ya se mencionó el responsable de administrar una materia es el profesor asignado a ésta.

Este capítulo resume la funcionalidad de C@D-CCADET, proyecto que se presenta como una opción de educación a distancia que brinda la posibilidad de expresar, compartir e intercambiar conocimiento, esto con la implementación de planes y programas de estudio que apelen a la dualidad educativa entre educación presencial y a distancia, aportando elementos para una sociedad fructífera y proveedora de conocimiento capaz de trascender a través del pensamiento.

C@D-CCADET se presenta como una alternativa educativa a personas que buscan una renovación o mejoramiento sin sacarlos de su contexto laboral, social y familiar. Esto último permite superar la clásica barrera entre la escuela y la vida ya que el estudiante no es sacado de su medio.

Desde el punto de vista de los procesos curriculares C@D-CCADET acredita la experiencia adquirida y los conocimientos previos del estudiante, permitiendo un aprendizaje realmente significativo. El aporte de las teorías constructivas encuentra en la educación a distancia un excelente campo de aplicación. Este es un tipo de educación exigente, especialmente por las características que los usuarios deben desarrollar como pre-requisitos: capacidades de lectura compresiva, de identificación y solución de problemas, de análisis y de crítica, y por último, habilidad para investigar y comunicar adecuadamente los resultados. A pesar del que el estudio es individual no descarta el trabajo en pequeños grupos. Aún más, es recomendado.

En C@D-CCADET desde la perspectiva del docente, la educación a distancia no prescinde de éste. Tampoco deja de lado la relación profesor alumno, sólo cambia la modalidad y la frecuencia. De la función de enseñante el docente pasa a ser un facilitador del aprendizaje, un creador de situaciones con medios innovadores que permiten al alumno lograr cambios de conducta y el desarrollo de habilidades necesarias.

CAPÍTULO 6

Conclusiones

6. Conclusiones

Una de las mayores preocupaciones de la Educación a Distancia es lograr que cumpla con los mismos estándares de la Educación Presencial. Se dice de manera implícita que lo que se quiere es que esta nueva modalidad de educación no sea una modalidad en segundo término, sino que sea al mismo nivel que la educación tradicional. En la Educación a Distancia parte de la problemática es saber si el estudiante que esta presenciando el curso a través de Internet es el mismo que presenta exámenes y trabajos, sin embargo estas son las mismas inquietudes que se presentan en la educación presencial, ya que el estudiante que se encuentra en el aula no se sabe si esta sentado escuchando o no, por lo cual el aseguramiento de la educación pasa inevitablemente sobre asignaciones, tareas y/o exámenes; por lo que este mismo aseguramiento de calidad se puede dar en la educación a distancia y que desde mi punto de vista para lo cual hay que tomar precauciones de identificación o autenticidad del alumno. Es importante mencionar que actualmente en las aulas no se verifica la autenticidad del alumno y que se deja a la memoria del profesor recordar las caras de los alumnos. En los salones en que se examinan a los alumnos tan sólo se exige una identificación a fin de evitar que no se vaya a presentar un alumno por otro. Estos controles son bastante débiles y son compartidos entre la educación tradicional y la educación a distancia.

Considero que el punto anterior no es punto para rechazar la idea de la Educación a Distancia ya que una manera de garantizar la autenticidad del alumno es recabar información del alumno a lo largo de las sesiones interactivas (de haberlas) y después utilizar esta información de tal manera que permita validar que el estudiante que haya asistido a las sesiones sea el mismo que contesta el examen.

Otro punto que se ha criticado de la Educación a Distancia es la falta de interactividad y espontaneidad que existen en la educación presencial, sin embargo desde mi punto de vista esto no puede generalizarse, ya que esta interactividad y espontaneidad se da en el salón de clases dependiendo del profesor y de sus estrategias de enseñanza. En la Educación a Distancia la interactividad se logra mediante reuniones electrónicas de grupo, en las sesiones de “chat” e incluso a través del correo electrónico que es relativamente rápido.

Hay que considerar que actualmente la Educación a Distancia es más barata que la tradicional y esto lo asocian a una deficiencia educativa, lo cual debe ser seriamente analizado ya que considero que son muchos factores los que hay que considerar para lograr la “calidad” que se requiere, si bien es cierto el profesor puede llegar a muchos alumnos, lo contrario no es cierto. Con frecuencia los alumnos de Educación a Distancia participan más que cuando se encuentran en un aula, por lo que saturan el correo electrónico del profesor y en las sesiones interactivas se ha visto en la práctica que una participación de más de diez alumnos a la vez es prácticamente imposible si se requiere que todos participen.

En nuestro días la Educación a Distancia ha evolucionado con el uso de la tecnología por lo que en un futuro no muy lejano podemos prever el auge de la misma, sobre todo en las corporaciones ya que con esta modalidad se economizan viajes y viáticos, además de ayudar a las compañías para cumplir con los objetivos y metas a un menor costo.

Tomando en cuenta los avances tecnológicos podemos prever la conjunción de otras tecnologías de teleconferencia, como las de enseñanza asistida por computadora, sistemas abiertos, sistemas multimedia e inteligencia artificial que darán grandes resultados como el trabajo oral a través del reconocimiento de voz (y los costos que esto pueda traer al transmitir únicamente caracteres en vez de una transmisión directa de la voz, ya sea en forma analógica o digital), la búsqueda inteligente de recursos en la red y para monitoreo de la misma, traer al alumno lo que le interesa y clasificarlo, proporcionando material educativo comercial de calidad con multimedios y otras características atractivas como el uso de tres dimensiones, para que en buena manera el alumno pueda aumentar su capacidad de autoaprendizaje, por el momento cabe mencionar que muy poca material de este tipo existe en el mercado, organizaciones como el CREAD (Inter-American Distance Education Consortium) fomentan el intercambio de este tipo e material entre instituciones de educación superior para propósitos de Educación a Distancia.

Actualmente el empleo de simulación y realidad virtual para el aprendizaje es una revolución, ya que permite al alumno aprender participando como si estuviera en el lugar. El diseño del material educativo que emplea simulación y realidad virtual aún es muy costos, por lo que en un futuro no muy lejano con el surgimiento de lenguajes de programación y paquetes especializados esto podrá estar al alcance de la Educación a Distancia y con ello aumentar el potencial de la misma.

En resumen la Educación a Distancia ha abierto una nueva área no sólo de desarrollo y ofrecimiento educativo, sino también un área multidisciplinaria de investigación y desarrollo, que además del interés científico y técnico que conlleva, tiene grandes perspectivas de aplicación debido a la gran demanda de capacitación, educación a lo largo de la vida y educación superior (que por definición es costosa). Con la Educación a Distancia no prevemos que se vaya a acabar la educación presencial, ya que cada una tiene su propio mercado, especialmente la educación presencial seguirá enfocada a los niños. Por lo demás es posible cubrir con la Educación a Distancia los métodos tradicionales de enseñanza, y lograr una educación de calidad altamente efectiva, tanto en su extensión como en su interacción.