UNIVERSIDAD CATÓLICA ANDRÉS BELLO

VICE-RECTORADO ACADÉMICO

ESTUDIOS DE POSTGRADO

ÁREA ACADÉMICA: GERENCIA

Postgrado en Sistemas de Información

Trabajo Especial de Grado

SISTEMA DE INFORMACIÓN PARA EL CONTROL DE PROYECTOS DE APOYO A LA INFANCIA Y ADOLESCENCIA.

Presentado por:

Pérez Cárdenas, María Gabriela;

para optar por el título de

Especialista en Sistemas de Información

Asesor:

Lic. Marlon Aurelio García Morales

Caracas, Septiembre del año 2004

APROBACIÓN DEL TUTOR

En mi carácter de Tutor del Trabajo de Especial de grado presentado por la ciudadana, María Gabriela Pérez Cárdenas, para optar al Grado de Especialista en, Sistemas de Información, cuyo titulo tentativo es: SISTEMA DE INFORMACIÓN PARA EL CONTROL DE PROYECTOS EN EL ÁREA DEL PROGRAMA DE APOYO A LA INFANCIA Y ADOLESCENCIA. CASO: CEDNA TÁCHIRA; y considero que dicho trabajo reúne los requisitos y meritos suficientes para ser sometido ala presentación publica y evaluación por parte del jurado examinador que se designe

En la Ciudad de San Cristóbal, a los 13 días del mes de Septiembre del año 2004:

Marlon Aurelio García Morales
C.I: 10.178.655

ÍNDICE GENERAL

LISTA DE CUADROS

LISTA DE GRÁFICOS

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

VICE-RECTORADO ACADÉMICO

ESTUDIOS DE POSTGRADO

ÁREA ACADÉMICA: GERENCIA

Especialización en Sistemas de Información

SISTEMA DE INFORMACIÓN PARA EL CONTROL DE PROYECTOS EN EL ÁREA DEL PROGRAMA DE APOYO A LA INFANCIA Y ADOLESCENCIA. CASO: CEDNA TÁCHIRA

Autor: María Pérez

Tutor: Marlon García

Fecha: Septiembre 2004

RESUMEN

El objetivo principal de esta investigación es el diseño físico de un Sistema de Información para el control de proyectos en el área del programa de apoyo a la infancia y adolescencia (PAIA) del CEDNA-Táchira. La metodología utilizada para el diseño de este prototipo es llamada Extreme programming (XP) o en español, Programación Extrema. Consiste en una serie de prácticas comunes en el desarrollo del software que unidas a la simplicidad del diseño y comunicación constante con el cliente, aseguran el desarrollo de aplicaciones livianas que sufren de cambios en especificaciones. El diseño de este sistema se realizó bajo una plataforma Apache utilizando el lenguaje php para la interconexión de las bases de datos creadas en MySql, todo bajo entorno Web. Además se utilizaron herramientas como Dreamweaver y Fireworks como soporte al lenguaje y diseño. El resultado final fue un prototipo llamado SPROY V 1.0 (Sistema de información para el control de proyectos CEDNA-Táchira), el mismo cumplió con los requerimientos del cliente y de necesitar modificaciones puede completarse con futuras iteraciones XP.

CAPITULO I
PLANTEAMIENTO DEL PROBLEMA

El CEDNA-Táchira se autodefine como (2004) “un organismo de naturaleza pública creado por mandato de la LOPNA, con personalidad jurídica propia, que ejerce funciones con plena autonomía de los demás órganos del Poder Público, respetando y promoviendo la descentralización”. La LOPNA (Ley Orgánica para la Protección del Niño y Adolescente) señala en el artículo 144 que “cada estado establecerá la organización interna del Concejo Estadal de Derechos”, destacando la autonomía de dichos órganos estadales.

En la actualidad este órgano que fundamenta sus acciones en la LOPNA y presenta un clima organizacional estable.

El CEDNA-Táchira trabaja en conjunto con el Sistema de Protección de la Niñez y Adolescencia (SPNA), a través de programas que se encargan de contribuir con la articulación de propuestas que garanticen la aplicación integral de los Derechos del niño, niña y adolescentes, y la intersectorialidad de los esfuerzos en políticas, acciones y recursos, a través del fortalecimiento de dichos órganos administrativos.

Uno de estos programas es el PAIA (Programa de apoyo a la infancia y adolescencia) se encarga de “apoyar las acciones que garanticen los derechos de los niños y adolescentes desde su concepción hasta los 18 años, contribuyendo con el proceso de consolidación de la nueva constitucionalidad de esta tarea y con el mejoramiento de la calidad de vida”. (Folleto: Programa de apoyo a la infancia y adolescencia).

Una de las funciones del programa es asesorar a las personas o asociaciones que deseen formular proyectos dirigidos a beneficiar a los niños y adolescentes del Estado Táchira tomando en cuenta la situación de vulneración de derechos del estado. Hasta ahora esta parte del programa ha funcionado sin ningún contratiempo; sin embargo, la actividad relacionada con la generación de los informes de proyectos necesita un grado mayor de estandarización y velocidad de producción; además de esto el principal detalle con estos informes es la falta de velocidad para la actualización de los datos. La carga de los datos es realizada a través de una hoja de cálculo en Excel y se realiza cada vez que necesitan generar en el informe a partir de un libro de asistencias en el que se asientan los datos de los nuevos proyectos y la información adicional resultado de las reuniones con los encargados de los proyectos actuales. También estos informes se solicitan en diferentes épocas del año y sin fecha fija; provocando en muchos casos la suspensión de las actividades de ese día por parte del personal del programa para completar a tiempo esta actividad.

Alguno de los errores encontrados en los informes que están involucrados con la carga manual de datos como pérdida de información y datos inconsistentes.

Además de esto no cuentan con una plataforma tecnológica totalmente eficiente como para unificar la data en un servidor; y la mayoría de sus proyectos tecnológicos tienen que ser apoyados por la gobernación. Los problemas continúan con un cambio de sede para Julio del presente año.

De seguir esta situación se acumularían los errores en dichos informes que a su vez generan más datos incorrectos que pueden afectar la credibilidad en dichos proyectos y la institución.

En vista de la situación en esta área de la institución y la necesidad de alcanzar mayor productividad, se recomienda la creación de un sistema de información automatizado que permita almacenar los datos de todos los proyectos de una forma ordenada y generar informes de una forma rápida y

sencilla para aprovechar los recursos y mejorar los procesos involucrados. A su vez la creación del sistema facilitará el acceso a la información y proporcionará un medio seguro para el almacenamiento de datos evitando la repetición inapropiada y la inconsistencia de datos.

Finalmente esta investigación busca obtener una respuesta a lo siguiente: ¿La propuesta del Sistema de información para el control de proyectos en el área del programa de apoyo a la infancia y adolescencia; alcanzará un software de alta calidad que brinde la solución deseada y que cumpla las expectativas planteadas en un corto tiempo?.

OBJETIVO GENERAL

Diseño físico de un Sistema de Información para el control de proyectos en el área del programa de apoyo a la infancia y adolescencia (PAIA) del CEDNA-Táchira.

OBJETIVOS ESPECÍFICOS

1. Diagnosticar las necesidades del CEDNA-Táchira, especialmente en el área del Programa de Apoyo a la infancia y Adolescencia.
2. Estudiar la metodología Extreme Programming para adaptarse a las necesidades planteadas.
3. Generar el diseño físico del Sistema de información.

JUSTIFICACIÓN

El sistema de información propuesto garantizará la carga segura de datos, además de el rápido acceso por claves de búsqueda a la información necesaria para el monitoreo de los proyectos. Funcionará con una base de datos en un ordenador, el cual también contendrá todo el sistema; el mismo tendrá el papel de servidor para la aplicación que podrá ser accedido además por otro ordenador. La principal funcionalidad del sistema consiste en generar el informe detallado del estado de los proyectos, cantidad de instituciones y municipios atendidos por el PAIA. El aporte principal del sistema esta directamente relacionado con el incremento de la productividad en esta área del programa y proporcionara una ubicación fácil de la información cuando se realicen asesorias a las personas encargadas de cada proyecto. También garantizará la entrega puntual de dichos informes y la seguridad de los datos almacenados.

Siguiendo los estatutos de la institución, los beneficiarios van a ser los niños que podrán obtener ayuda de los proyectos que son controlados por el personal que utilizará este software como una herramienta más para elevar su productividad.

Para finalizar, la principal ventaja de la creación del sistema de información automatizado es la facilidad con que se pueden cargar los datos y generar el informe, actualizando constantemente los datos de los proyectos.

CAPITULO II
MARCO TEÓRICO

El siguiente capitulo muestra las bases teóricas que sirvieron de soporte para la realización del trabajo especial de grado. Incluye conceptos o explicaciones en profundidad adecuada para cada uno de estos temas: sistemas de información, teoría de proyectos, desarrollo rápido, metodologías ágiles, extreme programming, hardware, software, herramientas para el desarrollo de software, Ley orgánica para la protección del niño y del adolescente, Concejo estadal de derechos del niño y del adolescente, plan de desarrollo social, entidades de atención y banco interamericano de desarrollo. Con las siguientes líneas se promete al lector un conocimiento general de los ejes principales de la investigación.

ANTECEDENTES

Al realizar una investigación de este tipo y para cualquier trabajo que implique la creación de un sistema, se necesitan saber antecedentes que muestren resultados comparables para poder guiar los aspectos primordiales en el sentido correcto. Las investigaciones anteriores sirven como un compendio de conocimientos que soportan el trabajo de grado y muestran los caminos más exitosos para culminar con éxito para evitar errores y contratiempos.

A continuación una serie de investigaciones que están ligadas con los sistemas de información y la metodología Extreme programming; se pondrá atención especial a esta metodología por ser la elegida para el desarrollo del proyecto.

En el trabajo de grado de Colmenares Luís (1989) realizado en la UNET y titulado *Sistema de información de mantenimiento pausterizadora Táchira C.A*. se muestra un claro interés por relacionar dicho sistema con la totalidad de la empresa, mostrando en varias ocasiones propuestas para mejorar el desempeño de los operadores del sistema basándose en simples estructuraciones de las actividades. La función principal del trabajo de grado fue la de elaborar un sistema automatizado de información de mantenimiento tal que presente características como: sencillez, flexibilidad, utilización de mecanismos de control, alta eficiencia y con la capacidad de determinar el nivel de confiabilidad de la maquinaria de la planta.

El sistema generaba la programación anual de las actividades de mantenimiento de la planta, con aspectos claves como introducción de datos, consultas y modificaciones. El aporte a la investigación se centra en el estudio de factibilidad que presenta en forma sencilla los recursos de la empresa y la relación con el sistema; es dividido en aspecto económico y aspecto técnico que permite tener una visión global del entorno empresarial en el área de aplicación. También se tomará en cuenta de esta investigación las recomendaciones que al final resaltan que hay que tomar a la organización como un solo sistema.

En cuanto a la metodología, Cleary Patricia (2002) en su tesis titulada *Adopting Extreme Programming – The Benefits and Obstacles* (Adoptando Programación Extrema – Los Beneficios y Obstáculos) aborda el tema de la programación extrema comparándola en principio con la metodología clásica de cascada. El objetivo principal de la tesis es comprobar que Extreme programming mejora el desempeño del equipo de trabajo de un proyecto de desarrollo de software y también evalúa la dificultad en la aplicación de la misma. Para ello analizaron 7 proyectos con ítems como en características, aproximación a la implantación, beneficios y costos. Como resultado se obtuvieron una suma de resultados positivos de tipo subjetivo. Sin embargo, se reconoce que hay pocas pruebas cuantitativas, ya que por la relativa

novedad, se necesitan hacer más estudios cuantitativos para verificar su validez. Esta investigación es de gran importancia y va a funcionar como punto clave para la realización de este trabajo de grado, por que presenta todas las características a investigar además de contar con una detallada explicación de la metodología a emplear, además enfoca su aplicación en diferentes proyectos, lo que facilita la comprensión de todos los pasos.

En el documento llamado *Extreme Programming in Research Environment*, del Centro de Investigación Langley NASA (2002), muestran la adaptación de la metodología Extreme Programming en un contexto científico en donde el cliente y el desarrollador son los mismos. Este documento funcionará para apoyar la comprensión de la metodología especialmente en el área tan mencionada de la adaptabilidad de la misma en diferentes ambientes de proyectos pequeños.

También contiene imágenes de como debe lucir el entorno de trabajo de la pareja de programadores, una de las características más importantes de la metodología. El resultado de esta investigación fue positivo, pero al mismo tiempo resaltan que el punto clave es cumplir con las reglas, confiar en los valores de XP (ahora en adelante Extreme Programming) y la interactividad con el cliente.

En una investigación llevada a acabo en Finlandia por Kääriäinen, Kosekela, Takalo, Abrahamsson, Kolehmainen (2003) llamada *Supporting requirements engineering in extreme programming: managing user stories* (Soportando requerimientos de ingeniería en programación extrema: gestión de historias de usuarios), se desarrolló un sistema para el ingreso de las historias de los usuarios (que en XP son llevadas en tarjetas de papel) usando también con conceptos como Requirements managment (RM) y Configuration managment (CM). El sistema fue creado de tal manera que fuera adaptable a la metodología ligera de XP facilitando para no interferir con las demás normas impuestas en la misma. La herramienta se adaptaba como un plug-in en la herramienta de desarrollo Eclipse para brindar un

ambiente integrado en los proyectos XP. Esta investigación es de utilidad para profundizar un poco en las historias de los usuarios y comprender más a fondo los procedimientos de recolección y mantenimiento de las mismas durante la aplicación de la metodología; también, funcionará como un comienzo para buscar herramientas de desarrollo de software que encajen con XP.

BASES TEÓRICAS

SISTEMAS DE INFORMACIÓN

Antes de definir que es un sistema de información hay que tratar la palabra "sistema" e "información" por separado.

Un sistema según O'Brien (2001:40) es "un grupo de elementos interrelacionados o que interactúan conformando un todo unificado". Tiene 3 componentes fundamentales: Entradas, procesamiento y salidas. Pero también hay que incluir otros componentes para hacerlo más eficiente: retroalimentación y control.

En cuanto a la información, Sweden (2002: sp) dice que:

> Whenever we wish to express what we know, we can only do so by uttering messages of one kind or another – oral, written, graphic, gestual or even through 'body language'. Such messages do not carry 'knowledge', they constitute 'information', which a knowing mind may assimilate, understand, comprehend and incorporate into its own knowledge structures.1

La información aporta un conocimiento extra al usuario, en cambio el dato no representa algo significativo y no ofrece algún aporte adicional.

[1] Traducción al (Español): En cualquier momento que deseamos expresar lo que conocemos podemos solo hacerlo a través de mensajes de una forma u otra – oral, escruta, gráfica, gestual o incluso a través del 'Lenguaje corporal'. Estos mensajes no poseen 'Conocimiento', ellos constituyen 'información', la cual puede ser asimilada, comprendida e incorporada como parte de las estructuras de conocimiento por una mente conciente

Finalmente teniendo claro que representan estas palabras por separado hay que definir Sistemas de información; según Whitten (1996), un sistema de información es “una disposición de componentes integrados entre sí cuyo objetivo es satisfacer las necesidades de información de la organización” (p. 39). Más adelante señala que entre los componentes del sistema se pueden encontrar las personas, actividades, datos, redes y tecnología. También hace mención a que los sistemas por sí solos no requieren computadores para funcionar, pero el poder de las tecnologías de información es la predominante en los sistemas de información modernos, y en general se les puede llamar indistintamente “s*istemas de información basado en ordenadores*” o “s*istema de información informático”.* Por su parte Senn (1990) llama a los sistemas de información que utilizan computadores *“Sistemas de información computarizados”.*

Los Sistemas de información deben cumplir 3 actividades generales: recibir datos de fuentes internas o externas de la empresa, actúa sobre los datos para producir información y produce información para el futuro usuario. En cuanto a si los sistemas de información necesariamente están vinculados a las computadoras, el autor resalta que no necesariamente, pero si se presenta en la mayoría de los casos.

En un análisis un poco más orientado a la gerencia O’Brien (2001:9) dice que “un sistema de información (SI) o (IS), information System, es una combinación organizada de personas, hardware, software, redes de comunicaciones y recursos de datos que reúne transforma y disemina información en una organización”. En este caso incluye redes de comunicaciones y recursos de datos a los conceptos anteriores, pero en general estos recursos extras van en función de la tecnología de la información. El autor en este caso afirma que “La tecnología de información se ha convertido en una necesidad estratégica. Crea en ella, actúe con base a ella o conviértase en un acontecimiento tangencial en la historia” (2001:6).

Las opiniones de los expertos muestran que hay que apoyarse en las tecnologías existentes para desarrollar propuestas eficientes, sin descuidar los otros recursos presentes en el sistema.

Componentes de un sistema de información

Es muy importante tener en cuanta los componentes principales de los sistemas de información, a continuación O'Brien (2001:43) los expone de esta forma:

> Un sistema de información depende de los recursos humanos (usuarios finales y especialistas en SI), hardware (máquinas y medios), software (programas y procedimientos), datos (bases de datos y conocimiento) y redes (medios de comunicaciones y soporte de redes) para desempeñar actividades de entrada, procesamiento, salida, almacenamiento y control que conviertan los recursos de datos en resultados de información.

Mas adelante cuando se discuta el desarrollo rápido se profundizará en estos componentes del sistema de información.

TIPOS DE SISTEMAS DE INFORMACIÓN

O'Brien (2001) clasifica los sistemas de información en: Sistema de apoyo a las operaciones y sistemas de apoyo gerencial.

Los sistemas de apoyo a las operaciones están identificados con las actividades comerciales de una empresa y se encargan específicamente de procesarlas de forma eficiente, también controlan procesos industriales, respaldan comunicaciones y actualizan bases de datos. Estos sistemas a su vez se dividen en:

- Sistemas de procesamiento de transacciones (TPS: Transaction processing system): Se encargan de procesar y registrar transacciones comerciales.
- Sistemas de colaboración empresarial: Son sistemas que a través de las tecnologías de información integran a las personas en un trabajo conjunto.

Los sistemas de apoyo gerencial están ligados más a la toma de decisiones, aportan información que ayuda a los gerentes a tomar decisiones basadas los resultados de la unión de varios datos. Estos sistemas tienen origen en los MIS o Sistemas de Información gerencial (MSI, Management information Systems), palabra de moda en los 60's que identificaba los sistemas de procesamiento de datos en instituciones y que todavía tiene vigencia. Entre sus características esta que hace énfasis en la orientación Gerencial y en el marco de sistemas. Dentro de los sistemas de apoyo gerencial destacan varios sistemas de información para respaldar diferentes enfoques gerenciales. O'brien (2001: 61) menciona los siguientes

- Sistemas de información gerencial: (Management Information Systems, MIS), estos sistemas proporcionan información a los gerentes basados en bases de datos recuperadas de los sistemas de procesamiento de transacciones de la empresa, también pueden utilizar datos externos para completar informes que son hechos a la medida de la gerencia.
- Sistema de apoyo a las decisiones: Estos sistemas van un poco más lejos que solo mostrar la información, cuentan modelos de decisión y bases de datos especializadas que ofrecen posibles respuestas y recomendaciones a los gerentes. En realidad los gerentes pueden decidir si aceptar la propuesta del sistema, desecharla o incorporar parte de la solución del sistema en la toma de decisiones.
- Sistemas de información ejecutiva: Proporcionan información variada, generalmente con gráficos y son muy sencillos de usar. Son creados especialmente para llenar las expectativas de información de los

gerentes. La información esta orientada en base a puntos claves para cumplimiento de los objetivos de la empresa y puede generarse de tanto de bases de datos internas como externas.

Entre otras clasificaciones se podría mencionar a los

- Sistemas expertos *(ES, Expert systems)* que tienen su punto fuerte en la utilización de (IA) Inteligencia artificial (AI, Artificial intelligence). La inteligencia artificial se basa en una serie de conocimientos de un experto en algún área específica que son almacenados en una base de datos de conocimiento y que a través de módulos de software realizan inferencias para generar respuestas a las preguntas realizadas por los usuarios.
- Sistema de gerencia del conocimiento *(KMS, Knowledge management systems)*: Como la tendencia de la gerencia en estos últimos tiempos es hacia la gerencia del conocimiento, era de suponerse que existieran sistemas capaces de manejar e incentivar la creación del conocimiento en una empresa. A esta recopilación se le llama el Know-how empresarial y generalmente se genera de sitios Web, foros de discusión que facilitan el aprendizaje y la diseminación del conocimiento.
- Sistemas de información estratégica*:* Puede ser cualquier tipo de sistema de información (TPS, MIS, DSS, etc.). Su función principal es funcionar como herramienta para desarrollar “productos, servicios y capacidades” que ayuden a la empresa a ubicarse en una zona de ventaja competitiva en comparación con sus competidores. Deben estar estrechamente ligados con los objetivos empresariales.
- Sistemas de información empresarial: Según el autor estos sistemas “respaldan las aplicaciones operacionales y gerenciales de las funciones empresariales básicas de una empresa” (65). Ejemplo:

Contabilidad, finanzas, gerencia de recursos humanos, marketing y gerencia de operaciones.

- Sistemas de información integrada: Simplemente integran los sistemas anteriores en una aplicación; también son llamados *sistemas de información compuestos o interfuncionales*.

Como se puede apreciar hay una gran variedad de sistemas, también se pueden mencionar los sistemas de información para oficinas que en palabras de Senn (1990:14) es una "clase especial de un sistema de procesamiento de información que puede usarse en el medio de las oficinas... para aumentar la aptitud tanto de los gerentes como del personal de operación para elaborar correspondencia, reportes, relaciones y documentos especiales". Es decir, estos evolucionaron y están directamente ligados con los procesadores de textos, son sistemas que han surgido de la automatización de las oficinas y que han surgido gracias a la incorporación de las tecnologías de información (ahora en adelante TI) en las empresas. Actualmente muchos de estos sistemas pueden integrarse con otros sistemas presentes en la empresa brindando una mejor adaptabilidad del usuario. En este caso la clasificación se asemeja a los sistemas empresariales de O'Brien.

Whitten (1996:39) incluye una subdivisión de acuerdo a la cantidad de usuarios que pueden utilizar un sistema: sistemas de información personales y sistemas de información multiusuario. En cuanto a los personales dice que "son aquellos que han sido diseñados para satisfacer las necesidades de información personal de un solo usuario". Los sistemas de información multiusuario por el contrario "han sido diseñados para satisfacer las necesidades de información de grupos de trabajo".

CICLOS DE VIDA DE SISTEMAS DE INFORMACIÓN

En el siguiente párrafo citado de Whitten (1996:95) se puede apreciar un sencillo concepto de ciclo de vida de los sistemas de información:

> El ciclo de vida del desarrollo de sistemas (VDS) es un proceso por el cual los analistas de sistemas, los ingenieros de software, los programadores y los usuarios finales elaboran sistemas de información y aplicaciones informáticas. IBM y otras compañías se refieren este término como ciclo de desarrollo de aplicaciones.

Existen diferentes tipos de ciclos de sistemas de información que pueden funcionar dependiendo del proyecto y la empresa en la que se desarrolla:

Cascada pura

Según McConnell (1997:149), “El modelo en cascada esta dirigido por documentos: es decir, los productos principales del trabajo que se pasan de etapa en etapa son documentos. En el modelo de cascada pura, las etapas son también discontinuas, es decir, no se solapan”. Este ciclo clásico permite seguir todos los pasos del desarrollo del sistema en forma ordenada, pero no permite regresar fácilmente a una etapa anterior para modificar algún requerimiento. Entre sus etapas el autor menciona: concepto de software, análisis de requerimientos, diseño global, diseño detallado, comunicación y depuración, prueba del sistema. Además funciona mejor cuando el producto esta bien definido y sus requerimientos no necesitan cambios.

Ventajas: casi todos los analistas están familiarizados con este ciclo, puede minimizar gastos en la planificación y se enfrenta a la complejidad de un proyecto de forma ordenada

Desventajas: el cambio de los requerimientos implica un gasto mayor y puede generar retardo en la entrega; no proporciona resultados factibles hasta el final sin embargo la documentación proporciona una visión del

estado y finalmente puede representar excesiva documentación para un proyecto de desarrollo rápido.

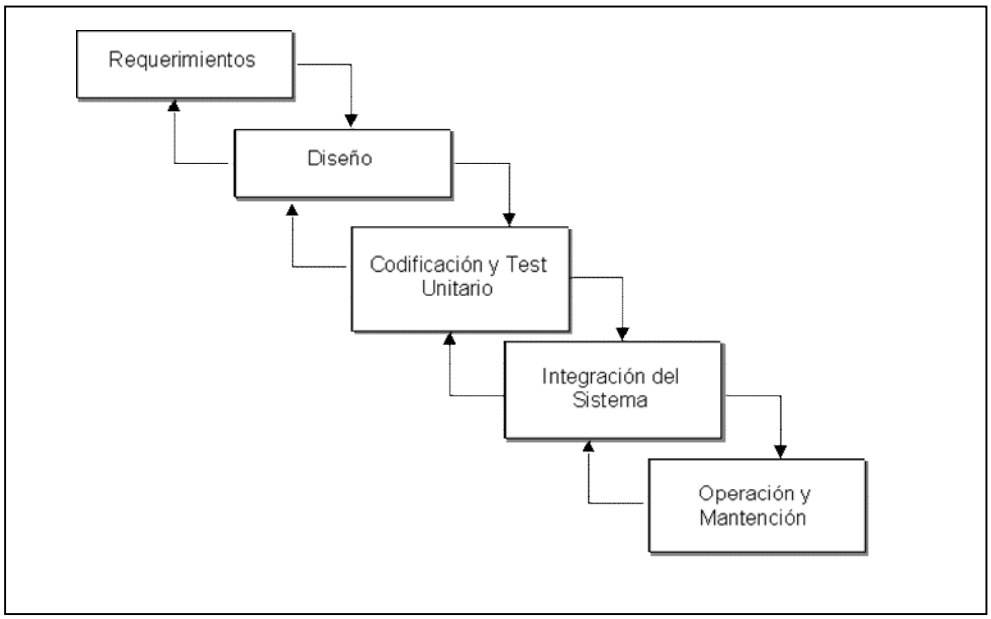


Gráfico 1.- Ciclo de vida de cascada. Tomado de "Ciclo de vida del software "por Aldarias (sf).

Codificar y corregir

Tal vez uno de los más utilizados y menos efectivo. Consiste en obviar aspectos importantes como el análisis del sistema y diseño para ir directamente a la programación. En la mayoría de los casos terminan en un total fracaso y son totalmente impredecibles, tanto así que depende más de la suerte que de la habilidad del programador para desarrollar el sistema. Es llamado también code-and-fix por sus siglas en inglés.

Sin embargo tiene dos ventajas, McConnell (1997:152) dice que: "En primer lugar, no conlleva ninguna gestión: no se pierde tiempo en la planificación, en la documentación, en el control de calidad, en el cumplimiento de los estándares o en cualquier otra actividad que no sea la codificación pura". Como no se pasa directamente a la programación "se

pueden mostrar inmediatamente indicios de progreso" También "requiere poca experiencia: cualquier persona que haya escrito alguna vez un programa de ordenador está familiarizada con el modelo".

Tal vez es de utilidad para proyectos pequeños para las demostraciones cortas o prototipos desechables, pero no es del todo recomendable para proyectos grandes.

Espiral

Es totalmente más elaborado que el modelo anterior. McConell (1997:153) lo define como:

> ...un modelo de ciclo de vida orientado a riesgos que divide un proyecto software en miniproyectos. Cada miniproyecto se centra en uno o más riesgos importantes hasta que todos éstos estén controlados... después de que todos los riesgos estén controlados finaliza del mismo modo que el modelo de ciclo de vida en cascada.

Este ciclo también es llamado "Rollo de canela" funciona con iteraciones; define al principio los riesgos, se realiza un plan para manejarlos y después se establece una aproximación a la siguiente iteración. Cada iteración representa un nivel del sistema y a medida que se llenan las expectativas se pasa de nivel.

El mismo autor divide las iteraciones en 6 pasos:

1. Determinar objetivos, alternativas y límites.
2. Identificar y resolver riesgos.
3. Evaluar las alternativas
4. Generar las entregas de esta iteración, y comprobar que son correctas.
5. Planificar la siguiente iteración
6. Establecer un enfoque para la siguiente iteración (si se decide ejecutarla).

Las ventajas de este ciclo es que es que los riesgos bajan a medida que se inviertan más recursos.

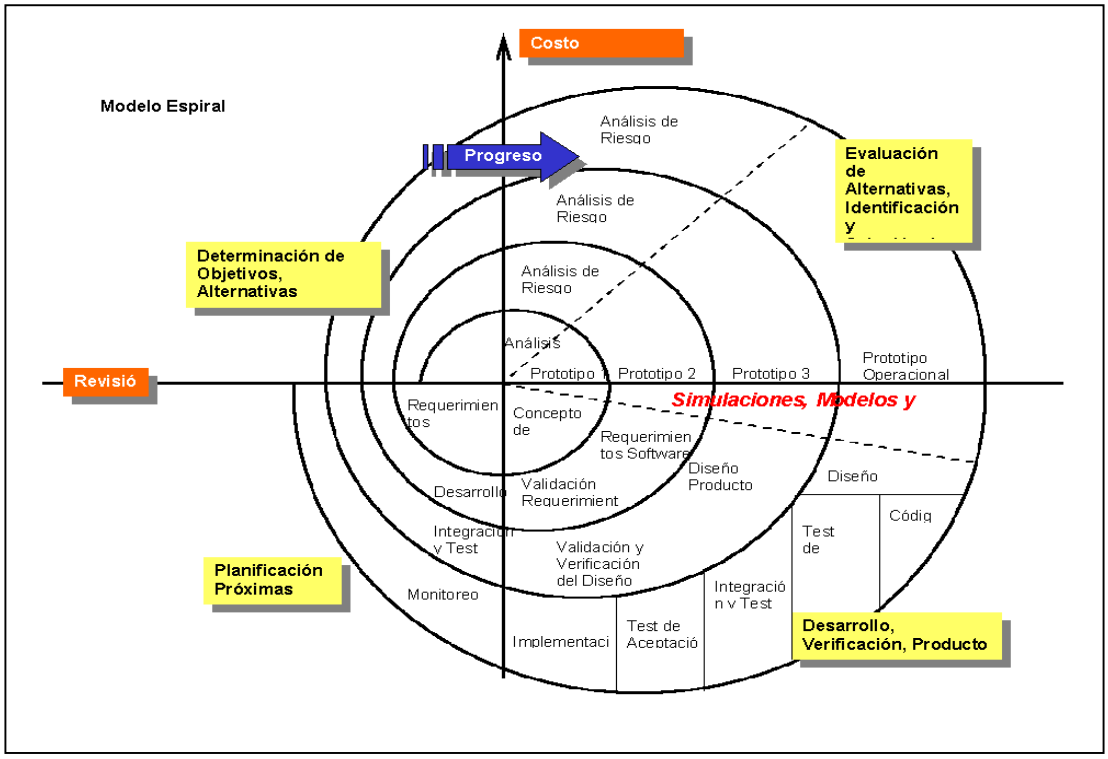


Gráfico 2.- Ciclo de vida del modelo en espiral. Tomado de "Ciclo de vida del software" por Aldarias (sf)

La desventaja es que es un modelo complicado, requiere conocimientos profundos especialmente para fijar los objetivos a medir para pasar a la siguiente espiral.

Cascadas modificadas

El modelo de cascada presenta problemas por que la mayoría de las etapas son "secuénciales disjuntas"; por eso hay otros modelos derivados de este que en cierto modo corrigen este error al solapar las etapas, reducir énfasis sobre la documentación o permitir más gestión.

***Sashimi* (Cascadas con fases solapadas)**

McConnell (1997:157) señala que este modelo "sugiere un grado mayor de solapamiento: por ejemplo, sugiere que se debería tener bien hecho el diseño global y quizás a medio hacer el diseño detallado antes de considerar completo el análisis de requerimientos". El mismo autor afirma que no es necesaria tanta documentación como en la cascada modificada, por que se puede ofrecer una "continuidad personal" en la concepción del software. Pero también dice que este modelo no esta "exento de problemas"; puede que el solapamiento no ayude a identificar bien los "hitos" del sistema, las actividades en paralelo pueden generar falta de comunicación y si es un proyecto pequeño y definido es preferible trabajar con el modelo clásico de cascada.

Cascada con subproyectos

En el modelo clásico hay que definir completamente el diseño global del sistema antes y no deberían hacer modificaciones en ninguno de los pasos que se culminaron. La arquitectura del sistema puede dividirse a su vez en otros subsistemas que pueden tener diferentes grados de complejidad. Cada subsistema puede tener características propias pero solo un diseño general; por ello se desarrollan las etapas de concepto del software, análisis de requerimientos y diseño global para todo el proyecto. Pero después se desarrolla el diseño detallado, codificación - depuración y prueba del sistema para cada uno de estos subsistemas. Todo esto culmina en una prueba final para todo el proyecto.

La principal desventaja de este modelo es que pueden existir dependencia entre estos subsistemas; que generalmente se eliminan para desarrollarlos como proyectos aparte.

Cascada con reducción de riesgos

Según el mismo autor en el desarrollo de proyectos informáticos no se puede ignorar los riesgos, de lo contrario estaríamos pecando de cometer un error clásico. Este modelo modifica la cascada solo un poco colocando un espiral para reducir el riesgo en lo alto de la cascada para controlar el riesgo de los requerimientos. McConnell (1997:158) explica que:

> Puede desarrollar un prototipo de interfaz de usuario, utilizar cuadernos, tener entrevistas con los usuarios, crear cintas de video donde los usuarios interactúan con un sistema más antiguo, o utilizar otros métodos que considere apropiados para la identificación de los requerimientos.

El defecto de este modelo es que los riesgos no solo se presentan durante el análisis de requerimiento, así que puede surgir algún contratiempo durante el desarrollo que nunca fue previsto y que pudo evitarse.

Prototipo Evolutivo

McConnell (1997:159) "El prototipado es un modelo de ciclo de vida en el que se desarrolla el concepto del sistema a medida que avanza el proyecto". Esto permite una visión de como puede quedar el sistema calmando la ansiedad del cliente y mejorando la capacidad del desarrollador para verificar los requerimientos.

Generalmente se inicia con las partes más visibles del sistema y se va modificando el producto dependiendo de la respuesta del cliente ante dicho prototipo. Una de las condiciones para utilizar este modelo es que los requerimientos sean cambiantes, cuando no se puede especificar el área de aplicación del sistema o si el sistema necesita la utilización de algún algoritmo desconocido por el desarrollador.

La desventaja es que por no saber cuanto se puede tardar incorporando los requerimientos del usuario desde el principio, es casi imposible saber cuanto tiempo durará el proyecto. Puede que los desarrolladores no culminen

el proyecto hasta que el cliente este satisfecho o hasta que se acabe el dinero; también puede que se convierta en un modelo de codificar y corregir si no se toman las medidas comunes de los demás modelos aunque sea en un nivel bajo.

Entrega por etapas

En este modelo el cliente también puede ver en etapas el desarrollo del sistema. Lo que lo diferencia del prototipado es que desde el principio se conocen los principales aspectos del sistema. Este modelo también se conoce como “implementación incremental”. McConnel (1997:161) explica el modelo de esta forma:

> ...con la entrega por etapas se atraviesan los pasos del modelo en cascada pasando por la definición del concepto del software, análisis de requerimientos y creación del diseño global de una arquitectura para el programa completo que se intenta construir. A continuación se procede a realizar un diseño detallado, la codificación, depuración y prueba dentro de cada etapa.

La ventaja de este modelo es que si se planifican las entregas de la forma más adecuada, el cliente puede disfrutar de gran porcentaje de la funcionalidad del sistema antes de terminarlo, esto también facilita la adaptabilidad al mismo. También disminuye la incertidumbre que puede presentarse en algunos clientes por no ver resultados.

La desventaja señalada por el actor es que dicho modelo no funcionaría sin una adecuada planificación para los “niveles técnicos” y los “niveles de gestión”. Un error visible puede ser que alguna etapa planificada a realizarse más adelante sea necesaria para alguna de las etapas superiores.

Diseño por planificación

Es muy parecido al de entrega por etapas, la diferencia es que tiene una fecha concreta límite para la entrega del sistema. Es útil para evitar que el producto siga el camino crítico; McConnell (1997:162) dice que:

> ...uno de los elementos críticos del modelo de ciclo de vida es que prioriza las prestaciones y planificar sus etapas de tal forma que las primeras contengan las prestaciones de mayor prioridad. Las prestaciones de más baja prioridad se dejan para más tarde. Si llega a la fecha de entrega antes de finalizar todas las etapas, no es conveniente dejar sin realizar las prestaciones críticas debido a que se ha gastado tiempo de implementación en otras menos críticas.

La desventaja es que una vez culminado el tiempo se entrega el sistema tal como esta y tal vez sin algunas prestaciones de menor prioridad que no pudieron realizarse a tiempo. En fin todo depende de la confianza que se tenga a la planificación desde el comienzo.

Entrega evolutiva

Es una unión entre prototipado y entrega por etapas. McConnell (1997:154) explica que “Se desarrolla una versión del producto, se muestra al cliente, y se refina el producto en función de la realimentación del cliente”. La diferencia de este modelo con el prototipado evolutivo depende de que si se ha tomado una planificación que concuerda con los requerimientos del usuario, que probablemente no van a ser modificadas durante el desarrollo.

Tiene la misma desventaja que el prototipado en cuanto a que se sigue la interacción hasta que se agote el presupuesto o que el cliente quede satisfecho.

Según el mismo autor (1997:165), “los modelos de ciclo de vida que soportan desarrollo incremental incluyen los modelos en espiral, prototipado evolutivo, entrega por etapas y entrega evolutiva”.

Diseño por herramientas

Finalmente se completa la explicación de los ciclos de vida de los sistemas con el diseño por herramientas, que no es más que buscar las herramientas existentes en el mercado que ayuden a construir un sistema relativamente funcional a partir de estándares comunes en el sistema con algunas de las capacidades que ellas presentan. Lo malo de utilizar este modelo es que no se tiene un control total del desarrollo del sistema y muchas de las prestaciones del sistema no pueden ser generadas. También depende mucho de los productos existentes del mercado que son ofrecidos como una panacea tecnológica y requieren un tiempo extra para analizar sus verdaderas capacidades. Estas herramientas incluyen “librerías de código y clases, generadores de código, lenguajes de desarrollo rápido, y otras herramientas software que reducen de manera espectacular el tiempo de implementación”.

También esta la opción de adquirir el software comercial existente que en algunos caso es mucho más rentable y menos costoso, empero, esto involucra que no el sistema no va a ser del todo adaptable a la organización.

DESARROLLO RÁPIDO

Los principios que motivan al desarrollo de proyectos informáticos son variados al igual que las metodologías empleadas en los mismos; sin embargo cada uno de estos métodos no es perfectamente adaptable para cada proyecto. Pero la principal preocupación de las personas involucradas es la de poder realizar el proyecto de acuerdo a las fechas estipuladas; lo cual se puede controlar a través de una buena planificación.

Según McConnell (1996:xix) en la realización de un proyecto de software hay que hacer énfasis en que durante la planificación se tomen en cuenta los siguientes objetivos: “menor tasa de defectos, mayor velocidad de ejecución, mayor aceptación por parte del usuario, mejor facilidad de mantenimiento,

menor coste o menor tiempo de desarrollo". Todo esto para conseguir un equilibrio que aporte las mejores condiciones a un proyecto exitoso.

En la actualidad están en boga la creación de proyectos informáticos utilizando el concepto de "desarrollo rápido". Lo ideal en el desarrollo de este tipo de proyectos es no centrarse en el código, sino en las estrategias y teorías que lo forman. Sin embargo, son percibidos de manera diferente por cada miembro del proyecto. Para un programador es programar más rápido, para un ingeniero es utilizar una herramienta case con participación intensa del usuario (RAD), para el programador de mercado es un prototipo rápido con la última versión de un lenguaje de programación y para el directivo es un método recomendado en alguna publicación de negocios.

En realidad lo más importante para llevar este tipo de proyectos es cumplir con dos cosas: Seleccionar métodos eficaces y seleccionar métodos orientados específicamente a alcanzar los objetivos de planificación. Según McConnell (1996; 6), entre los métodos orientados a la planificación se encuentran: (1) Métodos que mejoran la velocidad de desarrollo: que aceleran la entrega del software, (2) Métodos que reducen el riesgo de planificación: que procuran evitar retrasos y (3) Métodos que hacen visible el progreso: que permiten disipar la incertidumbre de los demás componentes de un proyecto.

¿Que es el desarrollo rápido?

El desarrollo rápido no implica simplemente planificar las cosas para que se realicen de una forma más rápida, tiene más relación con una estrategia global para un proyecto de Software que toma en consideración todos los elementos del mismo. Se pretende con esta estrategia realizar un producto de calidad en el menor tiempo posible, pero para ello se deben seguir una serie de pasos y reglamentos que garanticen el desarrollo rápido. Si se cumplen las reglas de juego es mucho más probable que funcione la estrategia.

Según McConnell (1996:13) esta estrategia se divide en cuatro partes:

1. Evitar errores clásicos.
2. Aplicar las bases del desarrollo.
3. Gestionar los riesgos para evitar un retorno catastrófico.
4. Aplicar métodos orientados a planificación.

Esta división también es llamada por el autor como los pilares fundamentales del desarrollo rápido, de allí también dice que no se pueden descuidar ninguno de los cuatro ni tampoco prestar más importancia a alguno en particular, en dado caso se fortalecería las ventajas que la empresa tiene en cada uno de estos pilares.

También de gran importancia son las cuatro dimensiones de la velocidad de desarrollo que según el mismo autor son: Personas, Proceso, Producto y Tecnología. Estas dimensiones son las que influyen directamente en el desarrollo de cualquier proyecto de software y al igual que los pilares no se puede descuidar ninguna dimensión. Continuamente se ve en un proyecto a personas trabajando en actividades (Procesos) para el desarrollo de un producto utilizando alguna herramienta (en este caso la tecnología).

A continuación una breve descripción de los tipos de estrategia basándose en lo expuesto por McConnell.

- *Evitar errores clásicos:* Por mucho tiempo a lo largo del desarrollo de proyectos informáticos se han incurrido en los mismos errores una y otra vez; ellos están involucrados directamente con las dimensiones de la velocidad de desarrollo; de esta manera se pueden ver errores relacionados con personas, procesos, producto y tecnología. A continuación una lista breve de estos errores que no se pueden pasar por alto.
- *Aplicar las bases del desarrollo:* Las metodologías utilizadas a lo largo de los años se centran en realizar un proyecto siguiendo patrones. También hay ciertas reglas que hay que cumplir en relación a las

bases del desarrollo, estas bases son creadas especialmente para evitar los desastres y estandarizar las acciones. Las bases que hay que tomar en cuenta son las siguientes: peopleware, bases de gestión (estimación, planificación, seguimiento y medidas), bases técnicas (gestión de requerimientos, diseño, construcción, gestión de la configuración del software), bases de control de calidad y finalmente seguir las instrucciones.

- *Gestionar los riesgos para evitar un retorno catastrófico:* Los riesgos son comunes y frecuentes en cualquier proyectos, por esta razón hay que controlarlos, localizarlos y eliminarlos de ser posible antes de que interfieran con la planificación del proyecto. Estos riesgos pueden generarse de cualquiera de las dimensiones del desarrollo rápido, así como también del el ambiente externo a la empresa.
- *Aplicar métodos orientados a planificación:* Para aplicar estos métodos hay que seleccionar primero el tipo de desarrollo rápido que se desea y también hay que conocer si en realidad es necesario aplicar un desarrollo rápido. Por ello a continuación se expone los tipos de desarrollo rápido.

METODOLOGÍAS ÁGILES

El término “Ágil” tiene sus inicios en una reunión de 17 expertos de la industria del software en febrero del año 2001, celebrada en Utah Estados Unidos. Su motivación era alejarse de las metodologías tradicionales que hacían énfasis en la documentación y procesos; para conseguir el desarrollo rápido de software. En esta misma reunión surgió “The Agile Alliance3”, dedicada a promover el surgimiento del desarrollo ágil de software y apoyar a las empresas en conseguirlo.

En un trabajo de investigación titulado Canos, Letelier y Penádes (2003) describen la filosofía ágil:

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.
II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
IV. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
V. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
VI. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
VII. El software que funciona es la medida principal de progreso.
VIII. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
X. La simplicidad es esencial.
XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
XII. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

Analizando este tratado se puede apreciar que coincide con las ideas de McConnell (1997:13) la metodología debe apoyarse en los pilares del

desarrollo rápido estos son: “(1) Evitar errores clásicos, (2) Bases de desarrollo, (3) Gestión de riesgos, (4) Métodos orientados a la planificación”. En ciertas áreas claramente estas metodologías simplemente pautan las reglas a seguir para el desarrollo de software bajo un esquema diferente.

Entre las metodologías ágiles se pueden citar Extreme Programming, SCRUM, Crystal, entre otros. Tal vez el primero, uno de los más conocidos y los más difundidos por su creador Kent Beck

Metodologías Agiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Cuadro 1.- Diferencias entre metodologías ágiles y no ágiles. Tomado de “Metodologías ágiles en el desarrollo del software” por Canos, Letelier y Penádes (2003)

EXTREME PROGRAMMING

Esta metodología fue creada por Kent Beck para la plantilla del proyecto C3 de Chrysler, concluido exitosamente en el año 1997. *Extreme programming* puede traducirse como “*Programación Extrema*” comprende una serie de metodologías de desarrollo de software llamadas “Ágiles” o “Livianas” que tienen la capacidad de realizar proyectos en muy poco tiempo.

Los puntos claves de esta metodología son*:* *Comunicación*: entre los desarrolladores y el cliente, *Simpleza*: se evita agregar elementos innecesarios, *Feedback:* estimaciones constantes y pruebas automáticas, *Coraje*: se necesita tomar riesgos controlados. Lo más importante de esta tecnología es que esta orientada a la gente y no al proceso. Sin embargo, no por ser una metodología ágil elimina aspectos como la documentación; por el contrario es más intensa y subyacente en el código fuente; también es adaptable y todo cambio es bienvenido.

Welicki (2003:sp) dice que XP no es innovador en todo sentido, simplemente reúne las prácticas comunes de tal forma en que se lleven acabo y asegurando que las prácticas se soportan entre sí en el mayor grado posible. Esta afirmación es verdadera, sin embargo la unión de estas prácticas acompañadas de ciertas reglas, es la que le da la solidez a la metodología.

En un texto citado por Canos, Letelier y Penádes (2003:sp)

> es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Esta metodología de más o menos 5 años ha sido utilizada en compañías como: Bayerishe Landesbank, Credit Swiss Life, Daimer Chryler, First Union National Bank, Ford Motor Company and UBS. Extreme programming (XP de ahora en adelante) le da el poder al desarrollo de responder a los cambios constantes del usuario incluso los cambios a última hora.

También enfatiza el trabajo en equipo: Gerentes, desarrolladores y clientes dedicados al desarrollo de un software de calidad. Los

programadores se comunican con el cliente de una manera más fluida, manteniendo diseños limpios y simples. Entregan el sistema tan pronto como sea posible e implementan cambios como son sugeridos. Esto brinda la adaptabilidad a los cambios en los requerimientos del cliente.

Uno de los principales problemas del desarrollo de software es que los clientes no saben expresar muy bien todos los requerimientos desde el principio, este es el principal problema a solucionar con la práctica de XP.

Según el site oficial la práctica de XP ayuda también a mitigar el riesgo e incrementar el éxito. La metodología fue creada para trabajar con grupo de programadores entre 2 y 10; es decir no se puede utilizar en proyectos con un staff muy grande. En general se utiliza esta metodología en proyectos pequeños de altos riesgos.

Los grupos de trabajo tienen que trabajar codo a codo (los programadores en pares), preguntando, negociando alcances, horarios y creando pruebas de funcionalidad. Otro de los requerimientos es realizar unidades automatizadas y pruebas funcionales. Se necesita aplicar una pequeña ingeniería de pruebas en algunos dominios, la forma más sencilla es cambiar el diseño de manera que sea más sencillo de probarlo.

Es bueno mencionar que XP los programadores no se dedican exclusivamente a la generación de código. Las actividades del par incluyen análisis, diseño, implantación y pruebas, añadiendo valor a cada uno de ser necesario.

El trabajo en equipo es inútil si no se ponen en prácticas las necesarias al código. XP sugiere la realización de estas pruebas para garantizar la calidad y el valor del sistema. Las pruebas se llevan a cabo de esta manera: pruebas, correcciones. Se repiten el ciclo hasta que el código pase la prueba y lo que garantiza funcionalidad.

Enfrentar el problema

Beck (1999:12) señala como principal punto de partida que el principal problema que debe afrontar la metodología es la entrega del software y el valor del mismo: “Software development fails to deliver, and fails to deliver value. This failure has huge economic and human impact. We need to find a new way to develop software”.[2]

El problema del desarrollo del software es cumplir con los objetivos trazados de manera eficiente. Sin embargo, en la mayoría de proyectos de desarrollo de software hay riesgos que pueden impedirlo:

- Tropiezos en la fecha de entrega: decir al cliente que no el software no esta culminado
- Cancelación del proyecto: depuse de las fallas de entrega, finalmente el proyecto es cancelado
- El sistema se vuelve amargo: es exitoso hasta que en algún tiempo si se necesitan algunos cambios, puede ser tan costoso que el sistema debe ser remplazado.
- Tasa de defectos: los defectos son tales que el programa no se utiliza.
- Negocio malentendido: El software se produce, pero no soluciona el problema para el cual fue creado.
- Cambios de negocios: El software se pone en producción, pero el problema a solventar fue remplazado por otro más preciso que alcanza los objetivos de la empresa.
- Falsas características enriquecidas: El software posee características novedosas que no van a generar dinero

[2] Traducción al (Español): El desarrollo del software falla en la entrega, y falla en entregar valor. Esta falla tiene un gran impacto económico y humano. Necesitamos encontrar una nueva forma de desarrollar software.

- El Staff se cansa: después de algún tiempo trabajando constantemente en el sistema, los programadores comienzan a odiarlo y se van.

La metodología afronta estos riesgos de la siguiente manera:

- Tropiezos en la fecha de entrega: Utiliza ciclos pequeños, utilizando para las iteraciones un ciclo de uno a cuatro semanas de acuerdo a los requerimientos de los usuarios. Además otra iteración cuya actividad de uno a tres días asegura la resolución de problemas.
- Cancelación del proyecto: El cliente debe seleccionar la presentación más viable para el negocio.
- El sistema se vuelve amargo: Usa pruebas que son realizadas por cada cambio para asegurar la calidad.
- Tasa de defectos: Elimina estos defectos utilizando pruebas para códigos de programadores como función por función y pruebas escritas por clientes en base a las características del programa.
- Negocio malentendido: El cliente es parte activa del equipo, refinando así los requerimientos.
- Cambios de negocios: Durante un ciclo de entrega, el cliente puede hacer modificaciones que son puestas en marcha de una manera casi imperceptible.
- Falsas características enriquecidas: Solo las actividades de alta prioridad se incluyen en el proyecto
- El Staff se cansa: Los programadores tienen la responsabilidad de estimar y completar su trabajo; lo que permite que el ajuste a su horario y el cumplimiento del mismo. También promueve el contacto con el equipo reduciendo la soledad. Otra práctica de la metodología es apoyar a los miembros nuevos a aceptar más

responsabilidades y así tener un modelo en contra de “Staff Turn - over” (Abandono por parte del equipo).

Al conocer los objetivos que persigue esta metodología se explicará a continuación la serie de pasos que la conforman. En el sitio web dedicado a esta metodología, http://www.extremeprogramming.org, aparecen claramente estos “pasos” como normativas a seguir:

PASOS EXTREME PROGRAMMING

A continuación se muestra una gráfica que muestra el ciclo de vida de la metodología XP y posteriormente se describirá cada uno de sus pasos. Dicha metodología se tomó del sitio Web oficial http://www.extremeprogramming.org, la redacción de cada uno de los pasos se deriva de la información encontrada en este sitio que esta totalmente en inglés.

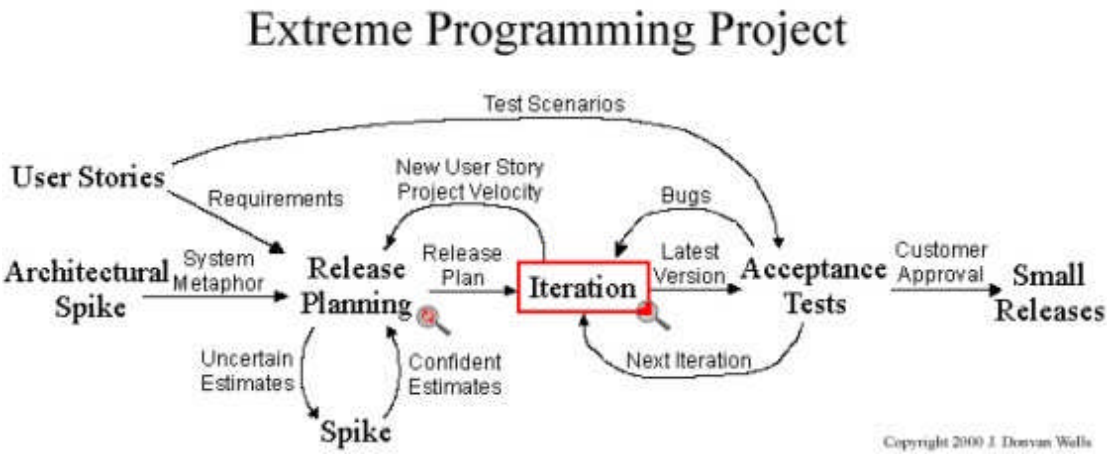


Gráfico 3.- Representación gráfica de la metodología Extreme Programming. Tomado de “Extreme Programming: A gentle introduction” por Don Wells (2003)

***Historia de usuarios*:** Estas son utilizadas para crear tiempos estimados y a su vez son la documentación del sistema. Son escritas por los clientes y representan las cosas que el sistema debe hacer por ellos. El formato involucra 3 oraciones escritas en lenguaje del cliente sin términos técnicos. Los usuarios solo proveen el menor detalle para realizar una estimación de bajo riesgo del tiempo que necesitará para ser implementada. Cuando se llega al tiempo de la implementación los desarrolladores irán con los clientes para recibir una descripción detallada de los requerimientos cara a cara. Después se estima cuanto tardará en realizarlas. En todo momento se deben evitar detalles tecnológicos en estas historias.

Beck (2000:49) en su libro presenta un ejemplo de ficha (customer story and task card) en la cual pueden reconocerse los siguientes contenidos: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas y una lista de seguimiento con la fecha, estado cosas por terminar y comentarios. A efectos de planificación, las historias pueden ser de una a tres semanas de tiempo de programación (para no superar el tamaño de una iteración). Las historias de usuario son descompuestas en tareas de programación (task card) y asignadas a los programadores para ser implementadas durante una iteración.

Architectural Spike: Es creado para solucionar problemas técnicos o de diseño. Es un programa sencillo para explotar posibles soluciones. Construye un sistema que solo presta atención al problema bajo investigación e ignora los demás. La meta es la reducción de riesgos o problemas técnicos. Cuando surgen estos problemas se coloca un par de programadores en ello durante una o dos semanas para reducir el riesgo.

Choose a System Metaphor (Seleccione una metáfora de sistema): Mantiene al equipo en la misma página al nombrar clases y métodos consistentemente. Al nombrar objetos se entiende más fácilmente el diseño del sistema y la reutilización de código. Esto habla de la estandarización de código en el sistema y de que se recomienda seleccionar un sistema de nombres para los objetos de manera que todos puedan identificarlos sin tener mayor conocimiento del sistema.

Release Planning (Planificación de prestación): Una reunión del plan de prestación es usada para crear un plan de prestación el cual representa el proyecto total. Este plan es utilizado para crear los planes de iteración para cada una de ellas.

Es importante que la gente dedicada al área técnica solo se dedique a su área, al igual que la gente dedicada a la parte financiera. Este plan de prestación posee unas reglas que permite a los involucrados en el proyecto realizar sus propias decisiones. Las reglas definen un método para negociar un horario que todos puedan cumplir.

La esencia de la reunión del plan de prestación es que el grupo de desarrolladores puedan estimar cada una de las historias de los usuarios en términos de semanas de programación. Una semana ideal incluye cuanto tiempo se puede tardar en implantar la prestación de la historia sin incluir trabajo extra pero incluyendo las pruebas. El cliente decide que historia es más importante o tiene la mejor prioridad.

Las historias de los usuarios son escritas en tarjetas, juntos desarrolladores y clientes mueven las tarjetas en la mesa para crear un conjunto de historias a ser implantadas como la primera prestación (o la próxima).

Se puede planificar por tiempo u alcance. La velocidad del proyecto es utilizada para determinar cuantas historias pueden ser implantadas antes de la fecha prevista (tiempo) o cuanto tiempo tomará terminar un conjunto de

historias (alcance). Cuando se planifica en función del alcance se divide el total de semanas estimadas en las historias de los usuarios por la velocidad del proyecto para determinar cuantas iteraciones hay que realizar para obtener una prestación.

Las iteraciones individuales son planificadas con mayor detalle justo después de cada inicio de iteración y no antes. Nunca subestime los estimados de las historias de los usuarios, negocie hasta que los desarrolladores, clientes y managers puedan estén de acuerdo con el plan de prestación.

La filosofía básica de la planificación de prestación es que el proyecto tiene que ser cuantificado por cuatro variables; alcance, recursos, tiempo y calidad. Alcance es cuanto hay por hacer. Los recursos es cuanta gente esta disponible. El tiempo es cuando la prestación estará lista. Y calidad es que tan bueno va a ser el software y que tan bien probado será.

La gerencia puede seleccionar 3 de las 4 variables de proyecto, el desarrollo tiene la restante. Nótese que bajando el nivel en calidad tiene un impacto imprevisto para las otras variables.

Release Plan (Plan de prestación): Después que se escriben las historias de los usuarios se puede usar un reunión de planificación de prestación para crear un plan de prestación. El plan de prestación especifica que historia de usuarios se van a implementar para cada prestación del sistema y fechas para las prestaciones. Esto da un conjunto de historias para que los clientes seleccionen durante la iteración de la reunión de planificación para ser implementada durante la próxima iteración. Estas historias seleccionadas son traducidas en tareas de programación individuales para ser implementadas durante la iteración para completar las historias.

Las historias también están traducidas en *pruebas de aceptación* durante la iteración. Estas pruebas de aceptación se ejecutan durante esta

iteración, e iteraciones subsecuentes para verificar cuando las historias se terminan correctamente y que continúen trabajando correctamente.

Cuando la velocidad del proyecto cambia drásticamente por un par de iteraciones o en algunos casos después de varias se debe fijar fecha a una reunión de planificación de prestación con los clientes y crear un nuevo plan de prestación.

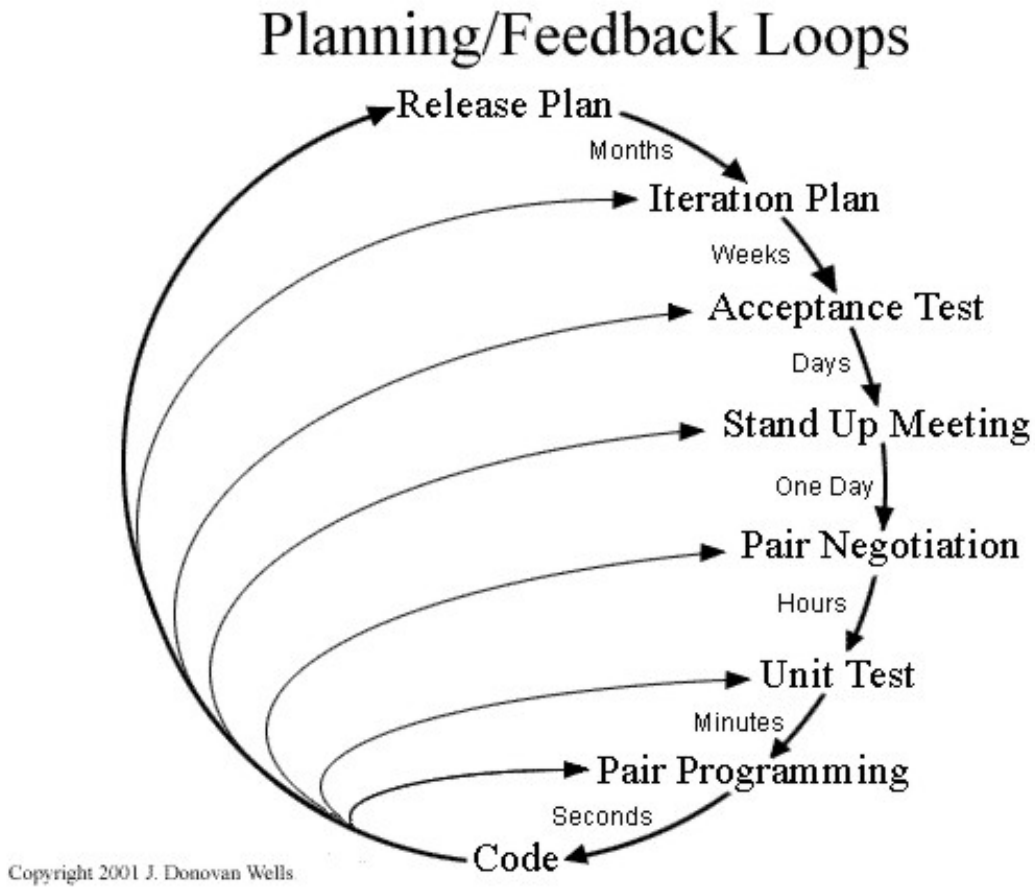


Gráfico 4.- Planning/Feedback Loops (Planificando / ciclos de retroalimentación). Tomado de "Extreme Programming: A gentle introduction" por Don Wells (2003)

***Acceptance Test (Pruebas de aceptación)*:** Las pruebas de aceptación se crean a partir de las historias de los usuarios. Durante una

iteración las historias de los usuarios seleccionadas durante la reunión de planificación de iteración serán transcritas a pruebas de aceptación. El cliente especifica escenarios para las pruebas cuando una historia de usuario ha sido correctamente implantada. Una historia puede tener una o más pruebas de aceptación, cualquier cosa que sea necesaria para garantizar la funcionalidad.

Las pruebas de Aceptación son sistemas de pruebas de cajas negras. Cada uno representa algún resultado esperado del sistema. Los clientes son responsables de verificar la veracidad de las pruebas de aceptación y de repasar la puntuación de las pruebas para decidir cual de las pruebas fallas tienen mayor prioridad. Las pruebas de aceptación son también utilizadas como pruebas de regresión antes del lanzamiento del producto.

Una historia de usuario no se considera completa hasta que pase la prueba de aceptación. Esto significa que se deben crear nuevas pruebas de aceptación en cada iteración o el equipo de trabajo no reportará nada de progreso.

Asegurarse de la calidad o Quality assurance (QA) es una parte esencial de XP. En algunos proyectos QA es realizada por un grupo separado, mientras que en otros QA se integrará en el equipo de desarrollo. En ambos casos XP se creó para tener una relación cercana con QA.

Las pruebas de aceptación deben ser automatizadas entonces pueden ejecutarse seguido. El Store de las pruebas de aceptación es publicado para el equipo. Es responsabilidad del equipo fijar un horario para arreglar las iteraciones de las pruebas fallas.

El nombre de pruebas de aceptación cambió de prueba de funcionalidad. Este refleje mejor el intento de garantizar que los requerimientos de usuarios fueron incorporados y el sistema es aceptable.

Daily Stand Up Meeting (Reunión diaria): La mayoría de los asistentes a una reunión de proyecto no participan sino escuchan los

resultados. Una gran suma de tiempo de desarrollo es desperdiciado para ganar una trivial comunicación. Que participen muchas personas en la reunión puede traer gastos en el proyecto y también puede tornarse en una pesadilla.

La comunicación entre el equipo es la función de esta reunión diaria *Stand up* (de pie). Una reunión todas las mañanas es utilizada para comunicar problemas, soluciones y promover el enfoque del equipo. Todos se disponen en círculo para evitar discusiones. Es más eficiente tener una reunión corta en la que todos tengan que asistir que muchas reuniones con pocos desarrolladores para cada una.

Cuando se tiene este tipo de reuniones la asistencia a otras reuniones puede basarse en quien es necesario y puede contribuir. Ahora es posible evitar planificar más reuniones. Con asistencia limitada, muchas reuniones son realizadas en un sitio espontáneo en frente de la computadora, donde el código puede verse y al mismo tiempo se pueden probar ideas.

Pair programming (Par de programación): Todo el código incluido en la producción de la prestación es creado por dos personas trabajando juntos en la misma computadora. La programación en pares incrementa la calidad de software sin impactar el tiempo de entrega. Esto es un contador intuitivo, pero dos personas trabajando en la misma computadora agrega mejor funcionalidad que si trabajaran separados, excepto que sería mejor en calidad. Con el incremento de calidad se esperan grandes ahorros a largo plazo en el proyecto.

La mejor manera de programar en pareja es sentarse uno al lado del otro en frente del monitor. Deslizando el Mouse y el teclado hacia delante y atrás. Una persona escribe y piensa tácticamente sobre el método creado, mientras que la otra persona piensa estratégicamente del como el método encaja dentro de la clase. Toma tiempo adaptarse a la programación en par así que no es fuera de lo común sentirse incomodo al principio.

Unit test (Pruebas unitarias): Cuando se crea una prueba al comienzo, después el código, se apreciará que es más fácil para crear el código. El tiempo combinado que se toma para crear una prueba y crear el código para hacerla pasar, toma el mismo tiempo así como arreglarlo. Pero si ya se tienen las pruebas unitarias, no es necesario crearlas después del código, ahorrando así un poco de tiempo ahora y otro poco después.

Crear una prueba unitaria ayuda al desarrollador a considerar que es necesario hacer en realidad. Los requerimientos son afianzados firmemente por las pruebas. No pueden existir mal entendidos en una especificación escrita en código ejecutable.

También se tiene feedback inmediato mientras se trabaja. No esta claro cuando los desarrolladores han terminado todas las funcionalidades necesarias. Puede suceder un arrastre de alcance como extensión y condiciones de errores son consideradas. Si se crean las pruebas unitarias primeros se conocerá cuando se ha terminado: si todas las pruebas corren.

También hay un beneficio al diseño del sistema. Y es difícil realizar pruebas unitarias a algunos sistemas. Estos sistemas son construidos en código primero y probados después, comúnmente por diferentes equipos. Al crear las pruebas primero el diseño será influenciado por el deseo de probar todo lo de valor para el cliente. El diseño reflejará esto al ser más fácil de probar.

Hay un ritmo para desarrollar la primera unidad de pruebas de software. Se debe crear una prueba para definir algunos pequeños aspectos del problema a la mano. Entonces se crea el código más simple que hará pasar la prueba. Después se crea una segunda prueba, ahora se agrega al código lo que se acabó de crear para hacer pasar la nueva prueba, pero no más; no hasta llegar a la tercera prueba. Se continúa hasta que ya no hay nada más que probar.

El código que se creará es simple y conciso, implementado en solo las características que deseen los programadores encargados. Otros desarrolladores pueden ver como funciona este nuevo código al buscar en las pruebas. Las entradas cuyos resultados son indefinidos estarán visiblemente ausentes del sistema.

Velocidad del proyecto: Es una medida que permite ver la velocidad de desarrollo del proyecto. Se calcula utilizando los estimados de las historias de los usuarios que se terminaron durante la iteración. También se totalizan los estimados para las tareas finalizadas durante la iteración. Las dos medidas son utilizadas para la planificación de la iteración.

Durante la reunión de planificación de iteración los clientes pueden seleccionar el mismo número de iteraciones iguales a la velocidad del proyecto medida en la iteración anterior. Estas historias se descomponen en tareas técnicas y el grupo puede anotarse para el mismo número de tareas iguales a la velocidad de proyecto de la iteración previa.

Este mecanismo simple permite a los desarrolladores recuperarse después de una iteración difícil. La velocidad del proyecto aumenta al permitir a los desarrolladores preguntarles a los clientes por otra historia cuando su trabajo este completo (sin tareas pendientes).

Cuando hay altas y bajas en la velocidad del proyecto se puede utilizar un release planning meeting (reunión de plan de prestación) para re-negociar el plan de prestación. Es de esperarse que la velocidad del proyecto varíe de nuevo cuando el sistema se pone en producción a través de tareas de mantenimiento.

La velocidad del proyecto es tan detallada y medida como se espera que sea exacta. No debe dividirse la velocidad del proyecto entre la longitud de la iteración o la cantidad de gente. Este número no es bueno para comparar la productividad de dos proyectos. Cada equipo de trabajo tendrá diferentes tendencias para estimar historias y tareas, algunas estimaciones más altas

que otras. Al final no importa, rastreando el total del monto del trabajo hecho durante cada iteración es la llave para mantener el proyecto en movimiento a un paso fiable.

El problema con cualquier proyecto es la estimación inicial. Colectar muchos detalles no hacen el primer estimado más que una suposición. En vez de crear grandes documentos trabaje con el verdadero alcance del proyecto. Hay que considerar el tiempo que se invertirá en crear una especificación detallada haciendo un par de iteraciones de desarrollo. Se medirá la velocidad del proyecto durante la exploración inicial para lograr mejores resultados en especificar la medida completa del proyecto.

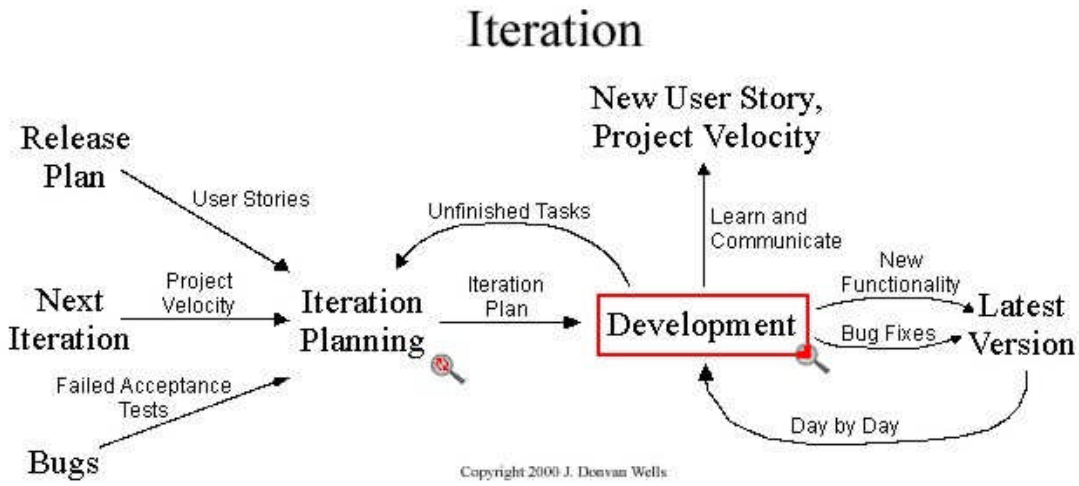


Gráfico 5.- . Iteración en un ciclo XP. Tomado de "Extreme Programming: A gentle introduction" por Don Wells (2003)

Interactive Development (Desarrollo interactivo): Agrega agilidad al desarrollo del proceso. Divide horario de desarrollo en una docena de iteraciones de una a tres semanas de duración. Mantener constante la duración de las iteraciones es el latido vital del proyecto. Esta constante es la que hace el progreso de medición y planificación simple y confiable en XP.

No se deben programar las tareas anticipadamente. En vez de esto hay que realizar una reunión de planificación de iteración al comienzo de cada iteración para planificar que es lo que realizará. La planificación just-in-time (justo a tiempo) es una manera fácil de mantenerse al día con los cambios de requerimientos.

También es en contra las reglas mirar más allá y tratar de implantar algo que no esta programado para esta iteración. Habrá suficiente tiempo para implantar esa funcionalidad cuando se convierta en la historia más importante en el plan de prestación.

Hay que tomar seriamente las fechas de entrega, haciéndole seguimiento al progreso durante una iteración. Si tal vez parece que no se puede terminar todas las tareas se establecerá otra reunión de planificación de iteración, re-estimando y eliminando algunas de las tareas.

Se debe concentrar los esfuerzos en completar las tareas más importantes seleccionadas por el cliente, en vez de tener varias tareas incompletas seleccionadas por los desarrolladores.

Puede parecer algo tonto si las iteraciones son solo de una semana de duración para hacer un nuevo plan, pero es gratificante al final.

Bugs: Para evitar los bugs (errores) en el sistema se crearon las pruebas de aceptación de fallas. Dichas pruebas ayudan al cliente a definir el problema y comunicarlo a los programadores. Los programadores tienen en que enfocarse y saben cuando el problema es solucionado. Al proporcionar estas pruebas de aceptación de fallas, los desarrolladores pueden crear pruebas unitarias para mostrar el error desde el punto de vista del código fuente. Cuando las pruebas unitarias corren al 100% la prueba de aceptación se realiza otra vez para saber si la falla ha sido remediada.

Planificación de iteraciones: En esta etapa se realiza una reunión en donde se divide las actividades de programación en iteraciones de una

semana o más. Las historias de los usuarios se seleccionan del plan de prestación tomando en cuenta la más importante para el cliente. Las pruebas de aceptación fallos que tienen que arreglarse también son seleccionadas. El cliente selecciona las historias de usuarios estimadas que al sumarse igualan la velocidad del proyecto para la última iteración.

Las historias de los usuarios y las pruebas son divididas en tareas de programación que las soporten. Las tareas son escritas en “index cards” (cartas indexadas) como las historias de los usuarios. Mientras que las historias de los usuarios están en lenguaje del cliente, las tareas están en lenguaje de desarrolladores. Las tareas duplicadas se pueden eliminar. Estas “task cards” serán el plan detallado para la iteración.

Los desarrolladores deciden con que tarea trabajar y estiman cuanto tiempo les tomará realizarla. Las personas no pueden intercambiarse y la persona que realizará la tarea tendrá que estimar cuanto durará realizarla.

Cada tarea deberá ser estimada para 1, 2 o 3 semanas de programación. Un día ideal de programación significa cuanto tiempo toma en terminar una tarea sin ninguna distracción. Las tareas que duren menos de un día pueden agruparse juntas. Las tareas que duran más de 3 días deberán dividirse mucho más.

Ahora la velocidad del proyecto es usada otra vez para determinar si la iteración esta sobre el límite o no. Totalizado las estimaciones del tiempo en días ideales de programación de las tareas, no puede exceder la velocidad del proyecto de la iteración previa. Si la iteración es muy larga entonces el cliente debe seleccionar que historias deben ser eliminadas para alcanzar la iteración anterior.

Si la iteración es muy corta entonces otra historia puede ser aceptada. La velocidad en tareas diarias (planificación de iteración) anula la velocidad en historias de semanas (planificación de prestación) hasta que sea más exacta.

Es alarmante ver como son recortadas las historias de los usuarios. Pero no hay que entrar en pánico, hay que recordar la importancia de la *unidad de pruebas* y la *refactorización (refactoring)*. Una deuda en estas áreas puede retrazar el desarrollo. Hay que evitar añadir funcionalidades antes de ser planificadas solo se debe añadir las que se necesitan por ese día (añadir extras puede causar retrasos).

No hay que verse tentado a cambiar las estimaciones de las historias o tareas. El proceso de planificación se apoya en la realidad de estimaciones consistentes, hacerlas un poco bajas puede crear más problemas.

Se debe mantener un ojo en la velocidad del proyecto; tal vez se necesite re-estimar todas las historias y re-negociar el plan de prestación cada 3 o 5 iteraciones, esto es normal. Hasta ahora mientras se implementen las historias más valiosas en primer lugar, se estará haciendo lo mejor para sus clientes y administración.

Un desarrollo interactivo puede agregar agilidad al proceso de desarrollo. En el proceso de planificación de tiempo no se debe planificar tareas específicas de programación muy alejadas de la iteración actual.

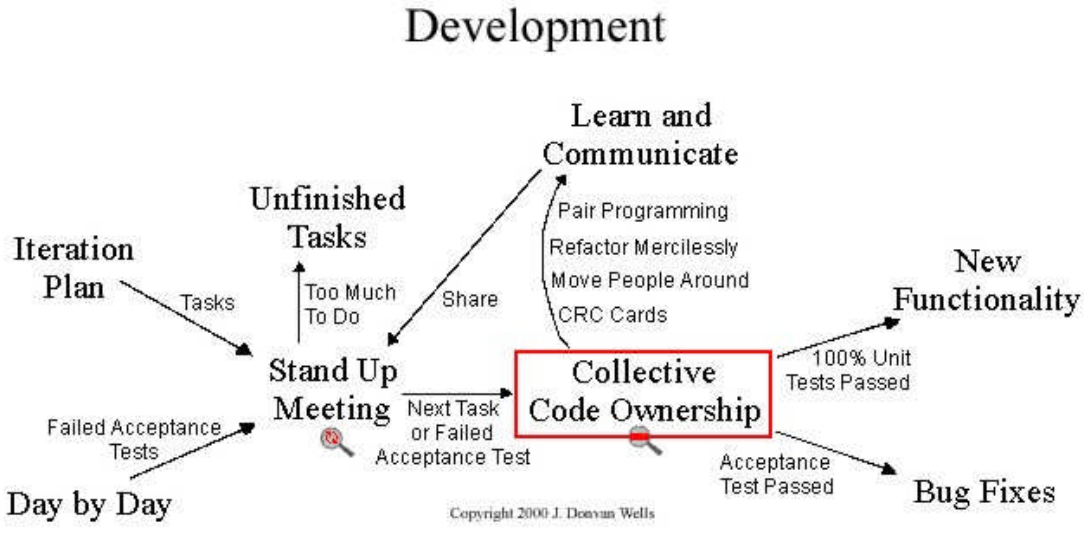


Gráfico 6.- Ciclo de desarrollo de una tarea. Tomado de "Extreme Programming: A gentle introduction" por Don Wells (2003)

Refractor Mercilessly: Los programadores se aferran al diseño del software mucho después que se volvieran poco dóciles. Por eso mismo se continúa usando código que no se puede conservar por que funciona y de alguna manera se tiene miedo de modificarlo. Pero en realidad es efectivo conservarlo?, XP toma la esencia de no hacerlo. Cuando se elimina redundancia, se elimina funcionalidad innecesaria y rejuvenece diseños viejos que se están refactorizando. La refacturización a través del ciclo completo del proyecto ahorra tiempo e incremente la calidad.

Refactor Mercilessly se utiliza para mantener un diseño simple y para evitar desorden innecesario y complejidad. Mantiene el código limpio y conciso para que sea más entendible, modificable y extendido. Se realiza todo lo que se espera de una vez y solo una vez. Al final toma menos tiempo producir un sistema que esta bien pulido.

Hay ciertas prácticas Zen en *refactoring*. Es difícil al principio por que ser capaz de ese diseño perfecto del cual se ha previsto y aceptar el diseño que fue descubierto usando *refactoring*. Ante todo hay que observar que el diseño previsto es una buena guía pero es ahora obsoleto.

Una oruga esta perfectamente diseñado para comer gran cantidad de follaje pero no puede conseguir una pareja, tiene que hacer refactoring a si mismo convirtiéndose en mariposa antes de ser diseñado para surcar los cielos en busca de otros de la misma especie. Se debe dejar ir las nociones de lo que el sistema debe hacer o no y tratar de ver el nuevo diseño que emerge ante los ojos de los desarrolladores.

Move people around (Mover la gente alrededor): Mover a la gente puede evitar serios problemas de pérdida de conocimiento y cuellos de botella. Si solo una persona del equipo puede trabajar en un área específica y esa persona se va; o simplemente se tienen un montón de cosas por hacer en esa sección, encontrará que el progreso del proyecto se ha reducido a una velocidad lenta.

El entrenamiento es también importante considerando que las compañías están tratando de evitar islas de conocimiento, que son susceptibles a pérdidas. Mover al personal en el código base junto a la combinación de programación en parejas hace por si solo el entrenamiento del personal. En vez de una sola persona que lo sepa todo de alguna sección del código, todos los miembros del equipo deben conocer gran parte del código en cada sección.

El equipo es más flexible si todos conocen suficiente sobre cada parte del sistema a trabajar. En vez de tener poca gente sobresaturada de trabajo mientras que otros miembros del equipo no tienen casi hacer, es decir, todo el equipo debe ser productivo. Cualquier cantidad de desarrolladores puede ser asignados a la parte más caliente del sistema. La flexibilidad del balance del carga de este tipo es el sueño de todo gerente.

Simplemente hay que motivar a todos en tratar de trabajar una sección del sistema al menos en cada iteración. La Programación en parejas lo hace posible sin perder productividad y asegura la continuidad del pensamiento. Una persona del par de programación puede ser cambiado mientras que el otro continúa con un nuevo compañero si es deseado.

CRC Cards (Cartas CRC): Usar Clases, responsabilidades y Cartas de colaboración (CRC) para diseñar el sistema como equipo. El mayor valor de las cartas CRC es permitir a la gente separarse del procedimiento de pensamiento y apreciar la tecnología de objetos. Las cartas CRC permiten a todo el equipo del proyecto contribuir al diseño. Entre más gente participe en el diseño del sistema se incorporarán un gran número de ideas.

Las cartas individuales CRC son usadas para representar objetos. La clase de estos objetos puede ser escrita en el tope de la carta, las responsabilidades son catalogadas abajo en el lado izquierdo, las clases de colaboración están listadas al lado derecho de cada responsabilidad. Se dice que “se puede escribir” por que una vez que las cartas son puestas para ser

llenadas, los participantes solo necesitarán pocas cartas con el nombre de la clase y ninguna carta escrita totalmente.

La sesión CRC se ejecuta con alguien describiendo el sistema y hablando de cuales objetos envían mensajes a otros objetos. Al pasar por las debilidades y problemas del proceso son fácilmente visibles. Diseñar alternativas puede ser explorado rápidamente al estimular el diseño propuesto.

Si hay mucha gente hablando y moviendo cartas entonces simplemente se debe limitar el número de gente levantada y moviendo cartas a dos. Cuando una persona se sienta, otra se levanta. Esto funciona en sesiones que se escapan de las manos, o cuando los equipos se vuelven rudos (un problema difícil finalmente es solucionado).

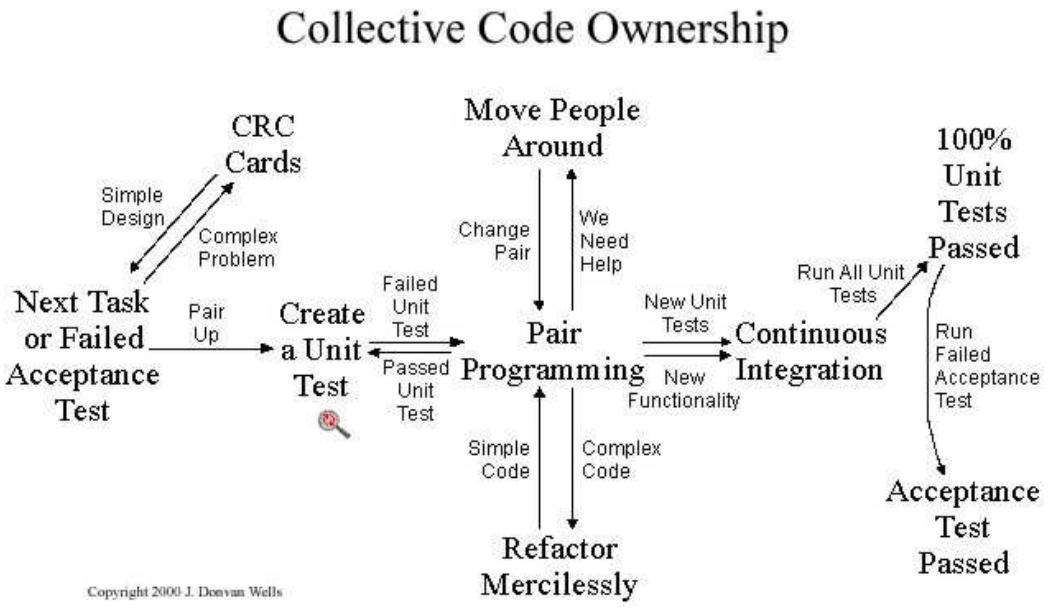


Gráfico 7.- Colective Code Ownership (Propiedad colectiva del código). Tomado de "Extreme Programming: A gentle introduction" por Don Wells (2003)

Una de las máximas críticas de las cartas CRC es la falta de diseños escritos. Generalmente no es necesario que las cartas hagan que el diseño

parezca obvio. Se requiere un registro permanente, una carta por cada clase puede escribirse por completo y ser retenida como documentación.

En el gráfico número siete se puede apreciar la representación gráfica del concepto de Propiedad colectiva. A continuación se explicarán cada uno de los ítems de este ciclo.

Next task or fail Acceptance Test
(Nueva tarea o prueba de aceptación falla)

Cuando se encuentra un error: Cuando se encuentra un error las pruebas son creadas para asegurarse de que no se repita. Un error en la producción requiere que la prueba de aceptación esta escrita para proteger el proyecto de él. Se debe crear una prueba de aceptación antes del debugging (buscar errores) para que los clientes puedan identificar el problema y comunicarlo a los programadores. Los programadores tienen pruebas fallas para enfocarse en sus propios esfuerzos y saber cuando el problema es solucionado.

Si falla una prueba de aceptación, los desarrolladores pueden crear pruebas unitarias para mostrar los defectos desde un punto de vista del código. Al fallar una prueba unitaria hay retroalimentación inmediata al esfuerzo del desarrollador cuando el error es reparado. Cuando las pruebas unitarias corren al 100% entonces la prueba de aceptación se puede ejecutar de nuevo para validad los errores corregidos.

Simple Disign (Diseño simple): La simplicidad del diseño es la clave; un diseño simple toma menos tiempo en ser terminado que uno complejo. Siempre se tiene que hacer la cosa más sencilla que pueda funcionar. Si se consigue algo que es complicado se remplaza con algo simple. También es más rápido y barato remplazar código complejo ahora, antes de perder más tiempo en ello. Se deben mantener las cosas lo más simple posibles al no

agregar funcionalidad después si esta programado. Sin embargo hay que estar alerta, por que mantener el diseño simple es un trabajo arduo.

Continuous integration (Integración Continua): Los desarrolladores deben integrar continuamente el código de la prestación en el repositorio de código cada pocas horas, cada vez que sea posible. En cualquier caso nunca espere por cambios por más de un día. La integración continua permite evadir divergencia o fragmentación de los esfuerzos de de los desarrolladores, cuando los desarrolladores no se están comunicando acerca de que puede re-utilizarse, o que puede compartirse. Todos necesitan con la última versión. Los cambios no pueden realizarse en código antiguo por que pueden causar dolores de cabeza.

Cada par de programadores es responsable por integrar su propio código cada vez que se presente una cambio razonable. Esto debe ser cuando las pruebas unitarias se ejecuten a un 100% o si alguna porción pequeña de la funcionalidad planeada es culminada. Solo un par integra en cada momento y después de pocas horas de codificación para reducir el problema a casi nada.

Casi todo el tiempo la integración continua evita o detecta problemas de compatibilidad de una forma temprana. La integración es una actividad "págame ahora y págame después". Siempre hay que trabajar en la última versión del sistema.

Roles XP

Los roles de acuerdo con la propuesta original de Beck (1999) son:

- Programador. El programador escribe las pruebas unitarias y produce el código del sistema.
- Cliente. Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias

de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

- Encargado de pruebas (Tester). Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- Encargado de seguimiento (Tracker). Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- Entrenador (Coach). Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- Consultor. Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.
- Gestor (Big boss). Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

HERRAMIENTAS PARA EL DESARROLLO DE SOFTWARE

Las herramientas para el desarrollo de software se incluyen dentro de las tecnologías utilizadas en el desarrollo. No por el hecho de que la tecnología o herramientas utilizadas en el desarrollo sean las más novedosas, esto garantice la consolidación del proyecto gracias a ellas. Se debe hacer un estudio precavido de las herramientas para asegurar que todos los componentes que ofrecen son funcionales en el desarrollo del sistema,

además hay que prestar atención a la curva de aprendizaje que puede traer problemas retrasando considerablemente el proyecto.

En palabras de McConnell (1997:373)

> Las herramientas para la productividad configuran la dimensión de la tecnología en el cuarteto de personas-proceso-producto-tecnología, y como tales desempeñan un papel importante en el desarrollo rápido. Adoptar una nueva herramienta puede ser una de las formas más rápidas de mejorar la productividad. Pero también puede ser una de las más arriesgadas.

Según el mismo autor estas herramientas se pueden aplicar en las siguientes áreas:

- Aplicaciones orientadas a SGBD: Herramientas especialmente creadas para las aplicaciones de bases de datos. Prácticamente se pueden conseguir para cualquier plataforma y funciona de tal forma que pueden facilitar carga de datos, reportes, etc.
- Aplicaciones personalizadas: Los lenguajes de 4GL pueden utilizarse para crear aplicaciones pequeñas con un entorno de diseño determinado, en el que el usuario no le importe estandarizar formatos de entrada, gráficos e informes.
- Prototipos desechables: Los lenguajes ideales para estos son los visuales, en los que puede mostrarse al cliente de que forma lucirá el sistema, para poder ampliar posteriormente sus capacidades. También estas herramientas se adaptan más a sistemas pequeños y sencillos.

Limitaciones de las herramientas de productividad

Puede que se seleccione la herramienta adecuada para apoyar el desarrollo de algún proyecto de desarrollo de software, pero puede también que esta herramienta termine generando más problemas que soluciones. Cada herramienta puede poseer ciertas características que la harán más

adaptable al proyecto, pero el caso es que si no se hacen las pertinentes evaluaciones de la misma, puede que al final no sea capaz de manejar algún requerimiento del sistema.

En caso de surgir algún contratiempo con la herramienta, en muchos de los casos es mejor re-comenzar parte del proyecto, allí subyace la gravedad de seleccionar mal alguna herramienta. En todo caso McConnell (1997:395) dice que “puede ignorar cualquier afirmación de poder mejorar la productividad más de un 25 por 100 al año por que no es realista”, esto a razón de un estudio que afirma que no se puede obtener más que este porcentaje.

Desarrollo rápido de aplicaciones (RAD: Rapid Application Design)

Según Whitten (1996:163) “Es una combinación de diversas técnicas estructuradas (especialmente, la ingeniería de información basada en datos) con técnicas de prototipos y de desarrollo conjunto de aplicaciones cuyo fin es acelerar el desarrollo de sistemas”.

El desarrollo rápido de aplicaciones también llamado RAD por sus siglas en ingles (Rapid Application Design), es el concepto de que los productos pueden desarrollarse rápido y con mejor calidad a través de:

- Recolectar información utilizando “workshops” o “focus groups”
- Hacer el prototipado temprano, reiteradas pruebas de usuario en el diseño
- El uso de componentes de software
- Una planificación rígida que retraza las mejoras del diseño al siguiente nivel de producción de la versión.
- Menos formalidad en los informes y otras formas de comunicación de equipo.

Algunas compañías ofrecen productos que proveen algunas o todas las herramientas RAD. Estos productos incluyen herramientas para recolección

de requerimientos, herramientas para prototipado, Herramientas de ingeniería integradas, lenguajes de desarrollo como java, comunicación groupware entre miembros de desarrollo y herramientas de pruebas. También esta muy relacionada con la programación orientada a objetos, que promueve la reutilización de software. (Obtenido de la red de redes <http://whatis.techtarget.com/definition/0,,sid9_gci214246,00.html>).

McConnell (1997:396) las cita entre las "panaceas que no se adaptan a la productividad", la describe como "un conjunto de métodos orientados a sistemas de información adaptados de algún modo a circunstancias individuales".

HARDWARE

El hardware representa la parte física de un sistema informático, es decir, todos los elementos materiales que lo componen. También se dice que es la parte tangible de una computadora.

En el sitio web de Sotware Libre (2004) se dice que el hardware son "…todos los dispositivos físicos utilizados en el procesamiento de datos, que en su conjunto forman una computadora o un Sistema de Cómputo. Por lo que podemos decir que hardware es todo lo que el usuario puede ver y tocar en un sistema de computación, por ejemplo, el monitor, el teclado, las unidades de disco, la impresora, el mouse, scanner, etc.."

El hardware más común en los sistemas de información son los microcomputadores, estaciones de trabajo, servidores, dispositivos de red y periféricos.

Sistemas de micro computadores

Según O'Brien (2001:130) los microcomputadores son "la categoría más importante de los sistemas computacionales para los usuarios finales. Aunque usualmente reciben el nombre de computador personal, o PC, un

microcomputador es mucho más que un computador pequeño utilizado por un individuo". Entre estos micro computadores se pueden mencionar las estaciones de trabajo, servidores de red y asistentes digitales personales (PDAs).

El sistema de un PC *(Personal Computer)* según Brooks (2002:25) posee los siguientes componentes: (a) *unidad del sistema*: El contiene los componentes primordiales del sistema, (b) *teclado*: dispositivo de entrada utilizado para introducir caracteres y comandos, (c) *mouse*: dispositivo usado en las interfaces gráficas para señalar o activar imágenes en el monitor, (d) m*onitor*: Un dispositivo de salida visual que muestra caracteres y gráficos en una pantalla, (e) *impresora:* dispositivo de salida que copia al papel datos y (f) *cornetas:* dispositivo de salida de audio.

Redes de computadoras

Tanenbaum (citado en cybercursos.net, 2003) dice que una red de computadores es "Una colección interconectada de computadores autónomos". Las redes involucran toda la plataforma física que respalda las telecomunicaciones, que a su vez se describe como el envío de información en cualquier formato de un lugar a otro utilizando medios electrónicos o que emiten luz. Los tipos de redes más conocidas son: Las redes WAN y LAN.

La Red de área ancha o WAN cubren una zona geográfica grande, según el currículo Cisco (1999:sp):

> Las WAN interconectaban las LAN, que a su vez proporcionaban acceso a los computadores o a los servidores de archivos ubicados en otros lugares. Como las WAN conectaban redes de usuarios dentro de un área geográfica extensa, permitieron que las empresas se comunicaran entre sí a través de grandes distancias.

Las redes de Área local o LAN en un material del currículo Cisco (1999:sp), se describen de esta manera:

> Las redes de área local (LAN) se componen de computadores, tarjetas de interfaz de red, medios del

> networking, dispositivos de control del tráfico de red y dispositivos periféricos. Las LAN hacen posible que las empresas que utilizan tecnología informática compartan de forma eficiente elementos tales como archivos e impresoras, y permiten la comunicación, por ejemplo, a través del correo electrónico. Unen entre sí: datos, comunicaciones, servidores de computador y de archivo.

Existen una gran cantidad de estándares para la construcción de estas redes. Cada uno de estos estándares cuenta con una gama específica de dispositivos de red y protocolos. A continuación una explicación de la tecnología cliente-servidor, una de las más populares.

Cliente-Servidor

Según Orfali,Harkey y Edwards (1998:12) la tecnología cliente-servidor es una "relación entre procesos ejecutados en apartados distintos. El proceso del servidor hace de este un proveedor de servicios. El cliente es un consumidor de servicios".

Cliente-servidor es "una frase que se usa para describir un modelo de interacción entre dos procesos, que se ejecutan en forma simultánea. Este modelo es una comunicación basada en una serie de preguntas y respuestas, que asegura que si dos aplicaciones intentar comunicarse, una comienza la ejecución y espera indefinidamente que la otra le responda y luego continua con el proceso" (Obtenido de la red de redes <http://www.linti.unlp.edu.ar/trabajos/tesisDeGrado/tutorial/redes/clteserv.htm>).

Estaciones de trabajo

Las estaciones de trabajo, o clientes citando a O'Brien (2001:131) "soportan aplicaciones con grandes demandas de cálculo matemático y de presentación gráficos, como el diseño asistido por computado (CAD, computer – aided – design) en ingeniería, o el análisis de inversiones y de portafolio en la industria de títulos – valores".

También se puede decir que "… son las máquinas o procesos que piden información, recursos y servicios a un servidor unido. Estas peticiones pueden ser cosas como proporcionar datos de una base de datos, aplicaciones, partes de archivos o archivos completos a la máquina cliente" (Obtenido de la red de redes <http://www.fismat.umich.mx/~elizalde/tesis/node21.html>)

Servidores

O'Brien (2001:131) describe a los servidores como "microcomputadores más poderosos que coordinan las telecomunicaciones que comparten recursos en redes de área local (LAN, local area network) pequeñas y sitios de Internet e Intranet".

La guía mIRC señala que un servidor es: "un ordenador de gran potencia, que se encarga de *prestar un servicio* a otros ordenadores… que se conectan a el.". (Obtenido de la red de redes <http://personal.redestb.es/guia_mirc/2.html>).

Orfali,Harkey y Edwards (1998:12) mencionan que según los tipos de servicio que pueden prestar los servidores, ellos pueden ser: servidores de archivos, servidores de bases de datos, servidores de transacciones, servidores groupware, servidores de objetos y servidores Web.

SOFTWARE

Difiere mucho del concepto de hardware, el software de un sistema informático es el conjunto de elementos lógicos necesarios para realizar las tareas encomendadas al mismo. Este dota al equipo de la capacidad para realizar cualquier tipo de trabajos. Y finalmente representa la parte intangible de las computadoras.

Según Zepeda (1997:sp) las características del software son

> (a) El software se desarrolla, no se fabrica en un sentido clásico, (b) No se "estropea", se deteriora. La diferencia

> radica en que cuando un componente de hardware se estropea, se sustituye por una pieza de repuesto; no hay piezas de repuesto para el software, ya que cada fallo en él, indica un error en el diseño o en el proceso mediante el cual se tradujo el diseño a código máquina ejecutable. Por esta razón, el mantenimiento del software es considerablemente más complejo que la del mantenimiento del hardware, (c) La mayoría del software se construye a medida, en vez de ensamblar componentes existentes, ya que éstos no existen en materia de software. Se puede comprar software ya desarrollado, pero sólo como una unidad completa, no como componentes que los diseñadores puedan reensamblar en nuevos programas.

También hay que mencionar que existe una gran cantidad de formas para clasificar el software; se puede clasificar: por su procedencia, sus aplicaciones, componentes, estado de desarrollo, forma de distribución y por su destino.

Bases de datos

Korth (2002:1), las define como ”…una colección de datos…” que “…contiene información relevante para una empresa”.

Para tener una visión más profunda de las bases de datos a continuación Aponte (s.f:12) afirma que:

> una base de datos es un conjunto estructurado de información condensada o elemental dispuesta para la gestión de una empresa. Este conjunto se describe los datos y representa las relaciones existentes entre datos. Debe ser accesible en forma batch o interactiva (convencional).

Existen dos tipos de arquitecturas básicas de bases de datos: bases de datos cliente-servidor y bases de datos distribuidas.

Una base de datos por si sola necesita de un mantenimiento especial, para eso se crearon los Sistemas de gestión de bases de datos. Estos sistemas permiten a un administrador de base de datos tener acceso a ellas y realizar tareas rutinarias de mantenimiento.

Continuando con las ideas de Aponte (sf:14) y ampliando el concepto de bases de datos un DBMS (por sus siglas en ingles: Database Management System) es “una recolección de numerosas rutinas de software interrelacionadas, cada una de las cuales es responsable de una tarea específica”.

Extendiendo un poco el funcionamiento de un DBMS, Gózalez (2000:5) menciona que:

> El primer objetivo de un SGBD es proporcionar un entorno que sea tanto práctico como eficiente de usar en la recuperación y el almacenamiento de la información de la base de datos. Otro de los objetivos principales de un SGBD es proporcionar al usuario una visión abstracta de la información, es decir, el sistema oculta detalles como los relativos a la forma de almacenar y mantener los datos, de tal forma que para que el sistema sea útil la información ha de recuperarse de forma eficiente.

Evidentemente este conjunto de herramientas sirven de soporte al administrador para realizar los cambios necesarios en una base de datos.

Sistema operativo

Los sistemas operativos según O’Brien (2001:193) son “El paquete de software de sistemas más importante para cualquier computador... consiste en un sistema integrado de programas que maneja las operaciones de la CPU, controla los recursos y las actividades de entrada/salida y de almacenamiento del sistema computacional, y proporciona diversos servicios de soporte, a medida que el computador ejecuta los programas de aplicaciones de los usuarios”.

Según Stallings (1991) (citado por García, Peralta y Fernández, 2004) “las funciones más comunes de un sistema operativo están las siguientes: operaciones de E/S, manipulación de sistemas de archivos, la detección de errores, etc”. Eventualmente todas estas tareas llevadas a cabo por el SO

facilitan las tareas del programador evitando que tomar en cuenta esto para la creación de software.

Al referirse a la arquitectura de estos sistemas operativos, Milenkovic (1994) (citado por García, Peralta y Fernández, 2004) señala el orden del sistema operativo (como se organiza respecto a si mismo) de la siguiente forma: Kernel molonítico, microkernel, máquinas virtuales y modelo cliente-servidor. En cuanto a los tipos de gestión de estos sistemas Tanenbaum (1991) (citado por García, Peralta y Fernández, 2004) muestra la siguiente clasificación básica:

- Sistemas por lotes: La característica principal es la falta de "interacción entre el usuario y el trabajo mientras éste se ejecuta. El trabajo se prepara y se envía. Tiempo después aparece la salida."
- Multiprogramación. "El *S.O.* escoge uno de los trabajos del depósito y comienza a ejecutarlo. En algún momento el trabajo tendrá que esperar, ya que el sistema ha pasado el control a otro programa y así sucesivamente. Mientras haya otro trabajo por ejecutar, la *UCP* nunca estará inactiva". En relación con los sistemas de multiprogramación estos a su vez se clasifican en
    - Sistemas de tiempo compartido
    - Sistemas de tiempo real
    - Sistemas combinados.
- Sistemas distribuidos: "Es un sistema débilmente acoplado, es decir, los procesadores no comparten ni memoria ni reloj, cada uno cuenta con su propia memoria local y se comunican a través de distintas líneas de comunicación".

Entre los sistemas operativos más utilizados en la actualidad se encuentran: Windows, Linux, SO y Unix. En cuanto a Linux, es un sistema operativo de código abierto que esta calando profundamente entre las empresas gracias a que es totalmente gratuito. Nacido del interés de un

estudiante llamado Linux Torvalds en un sistema operativo minix (creado por Tanenbaum), utiliza como herramienta GNU C Compiler y la distribución libre de código en comunidades de programadores para así conseguir popularidad y alcance.

TEORÍA DE PROYECTOS

Proyecto

Un proyecto según Gido y Clements (1999; 4) "tiene un objetivo especifico mediante un juego único de tareas interrelacionadas y el uso efectivo de los recursos". Por otra parte la definición del PMBOK del Project Managment Institute (2002;4), lo define como "un emprendimiento temporario realizado para crear un producto o servicio único".

Amabas definiciones coinciden en que los proyectos tienen como principal función la realización de un objetivo (que en algunos casos puede ser un producto). Dicho proyecto debe seguir una serie de actividades utilizando los recursos adecuados en un tiempo determinado.

Las características de un proyecto son reflejadas por el PMBOK (2002:4) de la siguiente manera:

> ellos pueden involucrar una sola persona o a muchos miles. Su duración varía desde unas pocas semanas hasta más de cinco años. Los proyectos pueden involucrar una sola unidad de la organización o pueden cruzar sus fronteras, como en las asociaciones y convenios para un proyecto determinado.

Por otra parte, Gido y Clements (1999; 4) mencionan los atributos que definen un proyecto:

- El objetivo: Tiene que ser bien definido y puede ser un resultado o producto esperado.
- Tareas interdependientes: Número de tareas no repetitivas que se realizan en orden para cumplir con el objetivo.

- Recursos: Incluyen personas, equipos, materiales, organizaciones y instalaciones.
- Tiempo limitado: la cantidad de tiempo que puede durar el proyecto, y se representa con una fecha de entrega tope.
- Puede ser un intento único: Son proyectos que no se han realizado con anterioridad. Ejemplo: el desarrollo de un nuevo producto.
- Cliente: La persona, personas, compañía o compañías que facilitan la información para la realización del proyecto. Generalmente tienen la última palabra y fijan el objetivo a alcanzar.
- Incertidumbre: Todos los proyectos se preparan en base a supuestos y estimados. Además de esto van acompañados de tareas y tiempos para su realización; que junto a los estimados proporcionan cierto grado de incertidumbre al no conocer si el proyecto estará a tiempo o no.

LOPNA

Hablar del CEDNA-Táchira sin hablar de la Ley Orgánica para la Protección del niño y del Adolescente (LOPNA) no podría representar sus raíces a cabalidad. Así pues, se mencionará a continuación algunos detalles de la creación de esta ley y su relación con el Consejo Estadal de Derechos del Niño y Adolescente.

Con esta ley se consiguió que Venezuela integrara a su legislación lo establecido con la Convención sobre los Derechos del Niño adoptada por la Asamblea General de las Naciones Unidas el 20 de noviembre del año 1989. Este cambio tuvo sus orígenes alrededor de 1990, donde Venezuela se comprometía a adoptar las medidas legislativas, administrativas y de otra índole para hacer efectivos los derechos establecidos en ella. Pero no es sino hasta 1995 que se realizan los cambios planteados para la reforma

participativa de la misma. Finalmente fue aprobada el 3 de septiembre del año 1998, y entró en vigencia a partir del 1ero de abril del año 1999.

"Esta nueva ley representa un cambio en la atención a la niñez y juventud. Significa pasar de una concepción del niño, niña y joven como objetos de compasión a sujetos plenos de derechos" (Obtenido de la red de redes <http://www.unicef.org/venezuela/frame-completo-vzla.htm>).

CEDNA-Táchira

El CEDNA-Táchira es un órgano dedicado a la defensa y garantía de los derechos de los niños, niñas y adolescentes; uno de sus principales aportes es estimular y fomentar proyectos que ayuden a garantizar dichos derechos. El Programa de Apoyo a la Infancia y Adolescencia (PAIA) fue creado para brindar soporte a dichos proyectos y además fortalecer a órganos como el SPNA (Sistema de protección del niño y el adolescente). Contando actualmente con 43 proyectos en espera a ser aprobados, el PAIA ayuda a formular con una mejor claridad y siguiendo los estándares de los organismos internacionales dichos proyectos para ponerlos en marcha. Esta tarea de asesorar a las personas encargadas de los proyectos puede llevar mucho tiempo y requiere de mucha paciencia.

Por otra parte el llamado SPNA, involucra una serie de órganos, programas, sanciones, servicios, entidades, que se encargan de velar y hacer cumplir los derechos de niños, niñas y adolescentes. Entre estas órganos es importante mencionar al Consejo Nacional de Derechos y los Consejos Estadales de Derechos. Según Yuri y Cristóbal (2002;6) estos se encargan de:

> Velar el cumplimiento de los derechos difusos y colectivos de los niños, niñas y adolescentes en la nación y en cada uno de los estados. Estos derechos... van más allá de los intereses individuales, por que son aquellos que deberían pertenecer y gozar todas las personas consideradas colectiva y socialmente, cuando están vinculadas entre sí por

> circunstancias de hecho que le afectan, positiva o negativamente... En definitiva... tienen como principal función la de vigilar el cumplimiento de estos derechos que son de todos.

También estos Consejos actúan en su zona territorial identificando violaciones a los derechos y deberes; proponiendo estrategias para los distintos órganos públicos y garantizando el respeto de los defendidos en la LOPNA.

Estas entidades surgieron con el fin de garantizar la cobertura de la protección de los derechos de los niños, niñas y adolescentes a nivel nacional. Las siglas CEDNA significan *"Consejos Estadales de Derechos del Niño y Adolescente".* El CEDNA-Táchira es un organismo de carácter público que ejerce funciones en otros órganos del poder público respetando y promoviendo la descentralización.

La visión del CEDNA-Táchira extraída del folleto *Consejo Estadal de Derechos del Niño y del Adolescente* (sf) es la de:

> garantizar en la entidad la conformación, monitoreo, supervisión y control del Sistema Integral de Protección del Niño y del Adolescente (SIPNA-Táchira), en función del mejoramiento de los niveles de calidad de vida, bienestar y felicidad de dicha población, respondiendo efectiva y satisfactoriamente a sus necesidades frente a la realidad social imperante.

El CEDNA-Táchira cuenta con una serie de funciones entre las cuales esta la de promover la participación de la sociedad en actividades de divulgación, promoción, desarrollo o atención de los derechos y garantías de la LOPNA. A razón de llevar a cabo esta función se creó el PAIA, *Programa de Apoyo a la infancia y Adolescencia*. Respaldado por organismos como la Gobernación del Estado y el Ministerio de Salud y Desarrollo Social. Su principal objetivo es el de apoyar las acciones que garanticen los derechos de los niños y adolescentes desde su concepción hasta los 18 años. En el

folleto *Programa de Apoyo a la infancia y Adolescencia (PAIA) se* menciona que el cumplimiento de su objetivo lo realiza a través de:

> Financiamiento de proyectos consistentes con la nueva institucionalidad, enmarcados en la Constitución de la República Bolivariana de Venezuela y la LOPNA, patrocinando la prevención, vulneración de derechos, la atención integral de niños, niñas y adolescentes y el alcance de una plena condición de calidad de vida como seres humanos y ciudadanos.
> Apoyo a la gestión de los órganos administrativos que conforman el Sistema de Protección del Niño y del Adolescente (SPNA), mediante la intersectorialidad de esfuerzos y responsabilidades colectivas para alcanzar los objetivos.

Para Explicar mucho mejor las funciones del PAIA, el programa se encarga de apoyar la creación y desarrollo de proyectos que garanticen los derechos antes mencionados, a través de financiamientos como el del Banco Interamericano de Desarrollo y entes estadales.

Muchos de estos proyectos son liderados por organizaciones de la sociedad civil legalmente constituidas que nunca han elaborado proyectos de esta índole. En estos casos el PAIA se encarga de fortalecer los conocimientos a través de talleres y seminarios, además proporciona material necesario y asesorías individualizadas. Todo esto lleva tiempo y actualmente cuentan con aproximadamente con 43 proyectos vigentes.

En la mayoría de los casos el personal asignado como asesores de proyectos dentro del CEDNA-Táchira tienen como misión brindar apoyo técnico en cuanto a la formulación de proyectos sociales a las organizaciones a fin de dar cumplimiento a los criterios y/o requisitos necesarios para la obtener su aprobación

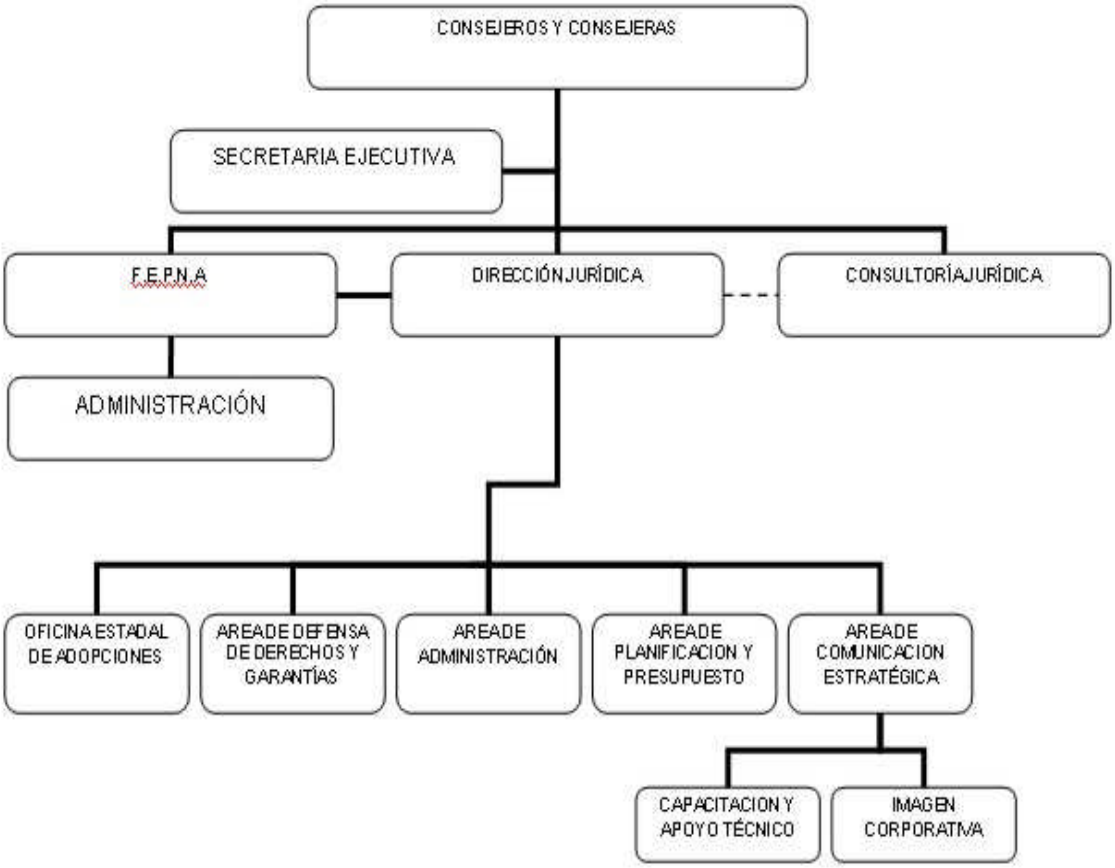


Gráfico 7 .- Estructura Organizacional CEDNA-Táchira

A continuación se mostrarán los objetivos, misión y visión proporcionados por el mismo CEDNA-Táchira.

Objetivos de la creación

El Concejo estadal de Derechos del Niño y del Adolescente del Estado Táchira es un órgano de naturaleza pública, deliberativa, consultiva y controlada, que con representación paritaria de entes del sector público y de la sociedad se encarga de acuerdo con su competencia geográfica, de velar por el cumplimiento de los derechos difusos y colectivos de los niños y adolescentes, consagrados en esta ley.

Misión

Son el órgano rector del sistema integral de Protección del Niño y adolescente que los reconoce como sujetos plenos de derechos y deberes sin discriminación alguna, a tal efecto ellos formulan, establecen, coordinan, implementan, supervisan y controlan la ejecución de las políticas públicas, de acuerdo con los lineamientos y principios expresados en la Convención internacional sobre los Derechos del niño, la constitución de la República Bolivariana de Venezuela, la Ley Orgánica para la protección del Niño y del Adolescente y demás leyes del ordenamiento jurídico venezolano vigentes aplicables, que permitan garantizar los derechos difusos y colectivos de los niños y adolescentes del Estado Táchira, del mismo modo velan conjuntamente con los demás entes y órganos del sistema integral de protección del niño y del adolescente por la optimización y mejoramiento de las condiciones necesarias para el desarrollo integral y armoniosa de la población infantil y juvenil. Cuentan con un equipo humano y técnico altamente calificado, comprometido con gran sensibilidad, disposición y mística de trabajo y con un conjunto de entes y órganos que con participación de la sociedad y el estado despliegan acciones de manera coordinada, sistemática y descentralizada.

Visión

El concejo Estadal de Derechos del Niño y del Adolescente del Estado Táchira, garantizará la conformación, monitoreo, supervisión y control del Sistema Integral de Protección del Niño y del Adolescente, en función del mejoramiento de los niveles de calidad de vida, bienestar y felicidad de los niños y adolescentes, y en atención a los principios expresados y consagrados en la Convención Internacional sobre los Derechos del Niño, al Constitución de la República Bolivariana de Venezuela, la Ley Orgánica para la Protección del Niño y del Adolescente y demás leyes del Ordenamiento

Jurídico venezolano vigente, respondiendo efectiva y satisfactoriamente a las necesidades de la realidad social imperante.

Objetivos

- Formular las políticas públicas y planes estadales de acción en materia de protección de niños y adolescentes, siguiendo los lineamientos generales del Concejo Nacional de Derechos del Niño y del Adolescente.
- Implementar las políticas públicas y planes estadales de acción en materia de protección y defensa de los derechos difusos y colectivos de los niños y adolescentes del Estado Táchira.
- Supervisar y controlar el desarrollo de las políticas públicas y las directrices establecidas para garantizar el efectivo ejercicio de los derechos de los niños y adolescentes.
- Consolidar a los órganos, entidades y servicios encargados de coordinar, orientar, supervisar y controlar los programas y acciones destinados a asegurar la protección integral de los niños y Adolescentes.
- Orientar a los órganos y entidades que conforman el Sistema Integral de Protección del Niño y del Adolescentes del Estado Táchira.
- Promover la descentralización administrativa a nivel municipal en miras a acercar las soluciones a los problemas de las comunidades y fortalecer las competencias y gestiones de los municipios, respetando la autonomía político territorial, diversidad cultural y particularidades de los mismos.
- Promover la participación de la sociedad y el fortalecimiento del núcleo familiar en miras a la incorporación progresiva de los niños y adolescentes a una ciudadanía activa.

- Revisa periódicamente los resultados de los programas que ejecutan los diferentes órganos y entidades que conforman el Sistema Integral de Protección del Niño y del Adolescente.
- Promover la intersectorialidad y corresponsabilidad en la creación de las políticas Públicas y Planes de Acción y asignación de recursos para la solución de problemas en materia de niños y adolescentes.

Principios y valores

El CEDNA – Táchira orienta sus pensamientos y acciones por los siguientes principios y valores:

Principios

- Ayuda y participación
- Igualdad
- Democracia
- Integridad y efectividad
- Libertad
- Respeto
- Comunicación y eficacia
- Confianza
- Eficiencia
- Solidaridad
- Sentido de pertenencia
- Reciprocidad

Valores

- Cooperación
- Equidad
- Diálogo
- Niños y Adolescentes

- Autonomía
- Tolerancia
- Asertividad
- Comprensión
- Honestidad
- Independencia
- Responsabilidad
- Armonía

CAPITULO III
MARCO METODOLÓGICO

El siguiente capítulo explica el tipo de investigación, población y herramienta de recolección de datos utilizadas para el desarrollo de la investigación así como también la metodología y planificación para llevar a acabo el proyecto. La información recolectada facilitará la visualización de los requerimientos del sistema y la planificación pretende fijar los puntos claves para la culminación del trabajo especial de grado.

TIPO DE INVESTIGACIÓN

Cuando se habla de investigación se hace referencia a una serie de métodos que se generan para dar solución a ciertos problemas que suelen presentarse en diferentes campos de la vida cotidiana del ser humano.

Extraído del Manual de la Universidad Pedagógica Experimental Libertador (2003:16) la investigación es proyecto factible por que la misma se basa en "la investigación, elaboración y desarrollo de una propuesta de un modelo operativo viable para solucionar problemas, requerimientos o necesidades de las organizaciones o grupos sociales".

DISEÑO DE LA INVESTIGACIÓN

La investigación también es de campo. Tomando lo señalado por Morales, Rubio y Gutiérrez (1999:41) en ella el investigador se basa en métodos que permiten recoger datos en forma directa de la realidad donde se presentan, en el sitio del acontecimiento; para este caso el instrumento que permitió la recolección de los datos fue la entrevista aplicada a una población de empleados que trabajan en el Programa de Apoyo a la infancia y Adolescencia.

POBLACIÓN Y MUESTRA

La UNA (1991:274) se explica que la población es “Cualquier conjunto de objetos que tengan alguna característica común observable”. También según la UNA (1991:273) la población puede ser de dos tipos: finita e infinita. La población finita esta “constituida por un determinado o limitado número de elementos” y la población infinita se observa cuando “es muy grande y se desconocen sus límites precisos”.

La población comprendió un grupo de cuatro profesionales que laboran en el CEDNA-Táchira en el área del Programa de Apoyo a la infancia y adolescencia. Su elección está ligada por el grado de conocimiento que tienen en relación al programa y sus procesos, además deben brindar los requerimientos del sistema en general.

Este tipo de población es finita y es una muestra pequeña, razón por la cual se seleccionó la misma cantidad de personas para aplicarles el instrumento.

TÉCNICAS DE RECOLECCIÓN DE DATOS

Senn (1998:133) explica que las técnicas para la recolección de datos son todos aquellos medios que tienen como objetivo reunir datos relacionados con los requerimientos, en este caso, del Software. Entre estos se incluyen las entrevistas que fue la técnica seleccionada en esta investigación.

La entrevista fue el instrumento de medición utilizado en la recolección de datos. Senn (1992:133) señala que “Las entrevistas dan a los analistas oportunidades para reunir información de las personas que han seleccionado debido a sus conocimientos del sistema que está bajo estudio. A menudo este método es la mejor fuente de información cualitativa (Opiniones, políticas, descripciones subjetivas de actividades y problemas)”. La entrevista seleccionada fue del tipo estructurado, que facilita mejor el análisis de datos y consiste en una serie de preguntas con términos uniformes para todos los entrevistados.

En el caso de la investigación este tipo de instrumento se adaptó a la perfección en vista que eran pocos los informantes y ellos no disponían de mucho tiempo para llenar cuestionarios.

En la investigación se completó el proceso de recolección de datos, obteniendo opiniones procedentes de la aplicación de una encuesta con un total de 12 preguntas o ítems del tipo abierto.

CAPITULO IV
ANÁLISIS E INTERPRETACIÓN DE RESULTADOS

Una vez recolectados los datos con el instrumento, se procede a analizarlos. De esta manera se persigue sintetizar los datos y comparar los resultados con los conocimientos teóricos que fundamentan el estudio. Según el material de la UNA (1991:383) “Esta práctica científica permite definir con antelación que datos serán capaces de rechazar o afirmar una hipótesis, qué resultados indicarán una u otra conclusión, etc., y de esta manera evitar sorpresas lamentables”. En una vista rápida a esta afirmación está claro que el análisis de los resultados es una actividad importante para evitar riesgos en cualquier proyecto.

Los datos obtenidos con el instrumento de recolección utilizado en esta investigación son en ámbitos generales del tipo cualitativo, por tanto, el análisis se enfocará a la interpretación de los resultados obtenidos de los entrevistados.

El instrumento posee 12 ítems (Ver anexo número 1), a continuación un análisis individual de las respuestas:

ÍTEM 1: ¿QUE CONTRATIEMPOS HA PRESENTADO EL SISTEMA ACTUAL?

Según los entrevistados, los problemas presentados por el sistema actual involucraban:

- Confusiones al generar informes con estándares diferentes, sin embargo poseen un estándar común, pero que de vez en cuando tienen que cambiar para ajustarlo a ciertas exigencias de quien solicita la información. Es evidente que se debe realizar un estándar para la generación de dichos informes que sea flexible y adaptable a las posibles modificaciones futuras.
- Otro de los problemas resaltados es la falta de disponibilidad de información actualizada y pérdida de tiempo en la búsqueda de la misma. Esto a consecuencia del control de asistencias de los proyectos que es totalmente manual. La solución más eminente es la creación de una base de datos que permita acceso rápido y actualizado a los datos.
- A la hora de generar el informe, el personal debe dejar sus actividades del día para terminarlo con éxito. Se percibe que en realidad esto puede causar algo de incomodidad en los empleados que pueden deben dejar a un lado su labor solo para culminar con esta actividad. Lo más fiable es generar el informe de forma automática con los datos de la base de datos que se mencionó anteriormente.

ÍTEM 2: ¿SIENTE LA NECESIDAD DE MEJORAR EL SISTEMA?

A esta pregunta todas las respuestas fueron afirmativas, lo que demuestra que la necesidad además de estar presente es una prioridad para el PAIA. También es evidente que están dispuestos a seguir con el proyecto de desarrollo del nuevo sistema, que en cierto modo ayudará a mejorar su productividad en un futuro.

ÍTEM 3: ¿QUE DESEA QUE HAGA EL SISTEMA?

Según la opinión de los entrevistados, el sistema debe generar informes detallados y actualizados de forma eficiente. Deberá poseer características

como status del proyecto, nombre de la organización, población que atiende, costo del proyecto, fecha de presentación del proyecto, fecha de aprobación, puntaje obtenido, beneficiarios (directos e indirectos). Adicionalmente resaltaron la creación de un nuevo reporte que generaría el número de organizaciones atendidas por mes, así como cuantas son entidades de atención y cuantas están ejecutando programas.

Esta pregunta fue de gran importancia, por que ayuda a determinar los requerimientos básicos del sistema además de conocer las necesidades de la institución.

ÍTEM 4: ¿LOS PARÁMETROS DEL SISTEMA SIGUEN LOS OBJETIVOS DE LA EMPRESA?

La respuesta a este ítem fue afirmativa; y explicaron que se asemejan por que el PAIA sigue una serie de planes operativos fijados por metas. Estas metas mensuales van orientadas a la cantidad de proyectos atendidos al mes; y el sistema permitirá su visualización de estas cantidades de forma rápida.

ÍTEM 5: ¿CON CUANTAS PERSONAS CUENTA PARA LA REALIZACIÓN DE ESTE PROYECTO?. NÓMBRELAS Y ESPECIFIQUE SUS HABILIDADES Y FUNCIONES EN PROYECTOS ANTERIORES.

Las personas mencionadas anteriormente, señaladas como parte de la población a estudiar son parte del personal que trabajará en el proyecto. En cuanto a sus habilidades, se mencionan como clave la capacidad de cada uno de los integrantes para asesorar proyectos del tipo de carácter social. Otra característica de este grupo de personas y que se puede observar es la excelente integración que tienen como miembros de un equipo y el apoyo que mutuo para la realización de las tareas. Evidentemente este grupo de

personas manejan una serie de habilidades sociales que permiten la continuación de cualquier tarea y que ayudan a mantener un buen clima organizacional. La función en los proyectos anteriores es del tipo variado, muchas de las tareas van dedicadas al análisis de situaciones, trascripción de datos y servicio de atención al cliente. Para el proyecto es necesaria su participación como fuente de información para generar las "historias de los usuarios" utilizadas en la metodología.

ÍTEM 6: ¿CON QUE RECURSOS CUENTA PARA REALIZAR EL SISTEMA? Y DE SER NECESARIO ¿CUALES NECESITARÍA?

Los recursos existentes en la institución asignados al PAIA son limitados, sin embargo cuentan con 2 computadores Pentium 4 con excelentes características para funcionar como estaciones de trabajo. El punto débil de esta plataforma es que no poseen una red para compartir datos y tampoco un servidor en donde se centralicen los mismos. También según sus comentarios presentan una débil seguridad ante virus informáticos y también tienen que compartir la única impresora con otro departamento del CEDNA-Táchira.

ÍTEM 7: DESCRIBA TODOS LOS PASOS DEL SISTEMA ACTUAL

1. En la entrevista realizada al equipo del PAIA describieron los pasos para la realización de dicho informe de la siguiente manera:
2. Solicitud del Estatus de los proyectos: Dichas solicitudes pueden presentarse a lo largo del año en cualquier época del año y son solicitados para presentar información en el plan operativo del CEDNA-Táchira o por la gobernación del Estado Táchira.
3. El siguiente paso es buscar informes anteriores: Esta actividad involucra buscar en el libro de asistencias los proyectos nuevos y entidades atendidas además de buscar el estado de los proyectos.

4. Actualizar datos en la hoja de cálculo: Este paso involucra comparar los datos de los informes anteriores con los datos reflejados en el libro de asistencias y sobrescribir los datos que tienen que actualizarse. Esta actividad fué asignada a un solo miembro del equipo (Rosangela), a razón de evitar errores de trascripción y repetición de datos.
5. Impresión del informe: Se procede a imprimir el informe utilizando para ello la impresora compartida, el archivo de la hoja de calculo se maneja en un disco 3 ½.
6. Finalmente el informe es revisado: Esta labor la cumple el Director Ejecutivo del CEDNA-Táchira.

ÍTEM 8: DE SEGUIR ESTA SITUACIÓN QUE PODRÍA PASAR

En este ítem, coincidieron en que puede generar pérdida y repetición de información además de errores en la trascripción de datos. En este punto el sistema deberá no puede evitar que alguien introduzca datos incorrectos, pero si puede evitar la pérdida y repetición de los mismos.

ÍTEM 9: QUE TIPO DE RIESGOS PUEDEN AFECTAR LA PUESTA EN MARCHA DE ESTE PROYECTO

Este ítem generó algo de confusión a la hora de ser respondido, pero una vez hecha la una aclaratoria, mencionaron que próximamente van a cambiar de cede y puede retrazar de alguna forma el proyecto. Este tipo de aclaratorias es clave para prevenir situaciones inesperadas y a corto plazo se tomará en cuenta en la planificación del proyecto.

ÍTEM 10:¿SE PODRÍAN ESTABLECER REUNIONES DIARIAS PARA SEGUIR EL ESTADO DEL PROYECTO?. A PARTIR DE QUE FECHA Y QUE CANTIDAD DE HORAS PUEDE DEDICAR AL PROYECTO (DISPONIBILIDAD).

La disponibilidad del personal para cualquier actividad relacionada con el sistema es casi nula, alegaron tener mucho trabajo para poder dedicar un horario fijo a la culminación del proyecto, empero, señalaron que están dispuestos a brindar información cuando sea necesario. La disponibilidad de los miembros del grupo es uno de los puntos claves para la culminación de cualquier proyecto, en especial en la generación de requerimientos y evaluaciones prototipo producido en la metodología. Se espera solucionar esto al trasladar el desarrollo del software a la institución.

ÍTEM 11: ¿ES POSIBLE FACILITAR UN ÁREA Y EQUIPO PARA REALIZAR EL SISTEMA?

Entre los miembros del grupo surgieron varios sitios en los que podría ubicar el centro de desarrollo que según la metodología debe estar ubicado muy cerca del cliente. Sin embargo el problema a corto plazo es que faciliten un computador para iniciar el trabajo, una alternativa es movilizar un equipo ajeno a la oficina para comenzar el desarrollo del sistema.

ÍTEM 12: ¿EXISTE ALGUNA PERSONA CON HABILIDADES DE PROGRAMACIÓN QUE PUEDA PARTICIPAR EL PROYECTO?

El único con este tipo de habilidades es Eduardo de Pablos, que posee conocimientos en Visual Basic; sin embargo el afirma no tener tiempo suficiente como para dedicarse a la programación del sistema. En vista de que la metodología empleada resalta la programación en pareja; es

necesario ubicar a un recurso humano capacitado en el área y con ánimos de unirse al grupo.

METODOLOGÍA

La metodología utilizada fue Extreme Programming y que se explicó en detalle en el capítulo anterior; consiste en una serie de pasos repetitivos que generará un modelo físico del sistema que permitirá al cliente visualizar en poco tiempo la aplicación.

Antes de emprender los pasos se desarrollaron las siguientes actividades adicionales a la metodología: (a) planificación del proyecto (estudios de factibilidad y riesgos) y (2) análisis de las herramientas de software disponibles en el mercado y la selección de las que más se adaptan a la curva de aprendizaje/tiempo.

En relación con la metodología, los pasos a seguir fueron los siguientes: (1) recolectar las historias de los usuarios, (2) estimación de esfuerzo para la implementación de dichas historias, (3) seleccionar lo que el cliente quiere construir de acuerdo con sus prioridades y las restricciones de tiempo, (4) construcción del valor del negocio y (5) repetición el paso número uno hasta obtener el diseño físico del sistema.

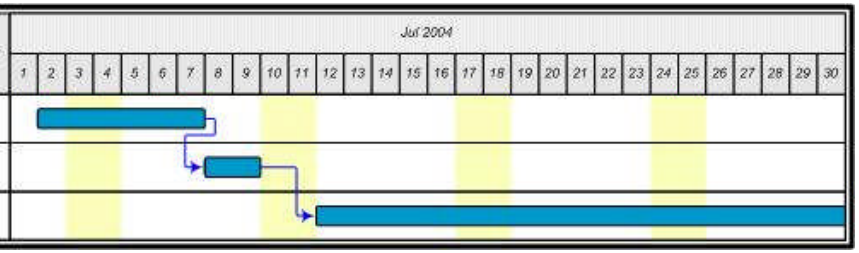
En la etapa de estimación y construcción se realizaron otras actividades imprescindibles en el desarrollo de sistemas, como la documentación y el diseño de estructuras de datos.

PLANIFICACIÓN

El siguiente cuadro muestra la planificación para el mes de julio del año 2004 que enmarca las actividades principales para el desarrollo del sistema en su totalidad. Empero, este cronograma esta sujeto a modificaciones.

PLANIFICACIÓN JULIO 2004

ID	Nombre de tarea	Comienzo	Fin	Duración
1	Planificación del proyecto	02/07/2004	07/07/2004	4d
2	Análisis de las herramientas de software	08/07/2004	09/07/2004	2d
3	Metodología Extreme Programming	12/07/2004	30/07/2004	15d



CAPITULO V
PROPUESTA

Este capítulo muestra la realización de la propuesta y los diseños generados para el desarrollo del sistema. En primer lugar hay que señalar que el proyecto no podía realizarse sin antes plantear una planificación adecuada al corto tiempo de desarrollo. A continuación la explicación de los pasos previos para obtener esta planificación.

PLANIFICACIÓN

Antes de poner en práctica la metodología se estimaron tiempos para la planificación preliminar y otras actividades tomando en cuenta los riegos durante las últimas semanas del mes de Agosto. La facilidad de realización y pronta culminación del prototipo depende en parte de esta planificación preliminar en donde se preparó todo lo necesario para brindar al equipo de desarrollo XP una experiencia interesante y lucrativa. Evidentemente el calendario planteado en el Capítulo III sufrió ciertas modificaciones.

A continuación las características que se tomaron en cuenta para poder realizar el calendario de actividades para la culminación de los objetivos planteados.

Breve exposición y presentación de la metodología

Se realizó un texto que resumía brevemente la metodología para poder que el programador y demás personas que formaban parte del proyecto tuvieran acceso a esta información. También, se procedió a explicar dicha

metodología al grupo de programadores participantes sin experiencia en la programación extrema. (Ver anexo número 2)

Herramienta - estudio de herramientas

Para el desarrollo del sistema, se tomaron en cuenta las herramientas que ofrecieran la menor de las curvas de aprendizaje. Fijándose en la entrevista a los programadores, se decidió realizar la aplicación bajo entorno Web. Es por ello que se decide utilizar PHP versión 4, MySql versión 1.4 y un servidor Apache versión 1.3. Acompañado también de Yapp (Yet another portal Program), un conjunto de librerías PHP que facilitan la productividad en tareas básicas como la conexión a una base de datos y creadas por uno de los programadores (Alexander Moreno).

Riesgos en la planificación

El mayor de los riesgos a la hora de planificar, es indudablemente el corto tiempo de desarrollo, no pudiendo escapar de esto se decidió como fecha tope de entrega final del prototipo el día Viernes 3 de septiembre del presente año. Postergando el proyecto en caso de fallas hasta el día Lunes 6 de Septiembre. (Ver anexo número 3)

Errores clásicos

Se conoce que la planificación demasiado optimista el proyecto forma parte de la lista de errores clásicos, sin embargo, las circunstancias ameritaban este tipo de planificación apresurada. Como punto clave para evitar más errores clásicos se imprimió una lista de ellos que estaba totalmente visible a los programadores, para recordar constantemente en que tipo de errores no debían incurrir.

Factibilidad

Para medir la factibilidad del proyecto se pueden tomar varias cosas en consideración, la primera es el factor económico. En este caso, el órgano destacó que no podía ofrecer ningún aporte económico. Sin embargo, el CEDNA-Táchira se va a ver beneficiado de forma positiva al adquirir el sistema. Al no invertir el mayor capital en su desarrollo, este deberá proporcionar ganancias directas cuando sea productivo. Se determinó que este proyecto tendrá resultados positivos relacionando la velocidad de respuesta de las tareas con la satisfacción del cliente.

Motivación

En vista de que en la mayoría de los proyectos la motivación de los desarrolladores es la fuente de dinero y considerando que en este caso no fue posible facilitarles esta opción, se planificó una motivación basada en entrega de golosinas a los programadores si las pruebas realizadas por iteración eran positivas.

EXTREME PROGRAMMING

Después de realizar la planificación se procedió a seguir los pasos de la metodología. En primer lugar se crearon los modelos de las fichas que recopilan las historias de los usuarios y las fichas de clases (Ver Anexo número 4).

A continuación se realizó la primera reunión con todos los participantes del proyecto. En esta reunión llamada en la metodología Realese planning (planificación de prestación) se especificaron los roles para cada uno de lo participantes, se planteó la idea general del sistema de nuevo, se recabaron las historias de los usuarios, los encargados del CEDNA-Táchira asignaron una locación con un computador para realizar el desarrollo del sistema y al

final planificó la siguiente reunión llamada en la metodología Interation planning (Planificación de iteración).

Los roles de los involucrados en el proyecto fueron los siguientes:

- Programadores. Alexander Moreno Delgado, Francisco Antonio Oliveros y María Gabriela Pérez (la programación en pareja se turnaba entre estos tres individuos, dedicándose también al diseño del sistema).
- Cliente. El CEDNA-Táchira
- Encargado de pruebas (Tester). Ing. María Gabriela Pérez.
- Encargado de seguimiento (Tracker). Ing. María Gabriela Pérez
- Entrenador (Coach). Ing. María Gabriela Pérez.
- Consultor. Lic. Marlon García (programación php).
- Gestor (Big boss). Gilma Valdúz.

En la reunión de planificación de iteración los desarrolladores dividieron las historias de los usuarios en tareas de programación sencillas y especificaron los tiempos de desarrollo para cada una de ellas. Se les pidió a los usuarios que clasificaran las historias dándole prioridad a las prestaciones más necesarias. Con los tiempos y prioridades se fijó una fecha de entrega para la primera prestación la cual consistía de la creación de la base de datos del sistema y creación de la interfaz de carga de datos. Para observar el cronograma de esta primera prestación diríjase el anexo número 4.

A continuación se muestra las historias de los usuarios, con prioridades:

Historia de usuario	***Prioridad***
Almacenar datos del proyecto: Estado, beneficiarios, organización, responsable, líneas, áreas, municipios, etc	1
Listado de proyectos aprobados por año	2
Listado de todas las organizaciones atendidas por año	3
Localización de responsables por proyecto	4
Mostrar el estado de un proyecto X	5

Cuadro 2.- Prioridades de las historias de los usuarios.

Para representar la cantidad de tiempo estimado para cada una de estas historias, los desarrolladores las desglosaron en actividades. A continuación una muestra de las actividades:

Historia	***Actividades***	***Tiempo de duración***
Almacenar datos del proyecto: Estado, beneficiarios, organización, responsable, etc	Diseñar la estructura de la base de datos	3 horas
	Crear la base de datos bajo MySql	8 horas
	Realizar pruebas con datos reales y consultas sql	2 horas

Cuadro 3.- Tiempo de duración de las historias de los usuarios.

La velocidad del proyecto involucra dos medidas: cantidad de historias de usuarios por iteración que fué aproximadamente de una historia por iteración; también se mide totalizando los estimados de las tareas de la iteracción, en el ejemplo anterior se totalizan 13 horas.

Según la metodología el siguiente paso es dividir estas tareas entre lo programadores designados. En vista de que solo se contaban con 2 programadores se decidió asignar todas las tareas a este equipo de programación en pareja. Este tipo de programación garantizó la fluidez de ideas y potenció la capacidad de los programadores a detectar errores antes de tiempo.

A continuación siguieron estos pasos individualmente para cada una de las tareas:

- Diseñaron pruebas e interfaces (diseños sencillos).
- Desarrollar la prueba primero que el código de la tarea.
- Desarrollar la tarea utilizando para ello las librerías de YAPP.
- Realizar las pruebas pertinentes
- Al final del día se podía ver en marcha parte de la prestación total del sistema.

El usuario tenía la facilidad de ver en marcha el proyecto y al final del día alguna parte del sistema estaba funcionando casi sin mayores detalles. De existir algún problema el usuario señalaba que había que corregir y de nuevo se realizaba una reunión de planificación de prestación para reformular las iteraciones y tiempos de entrega así como las pruebas pertinentes. De seguir teniendo problemas se planificaban otras iteraciones utilizando las pruebas unitarias (pruebas de código).

El resultado final es un prototipo sencillo de un sistema para el control de proyectos. El sistema a petición del cliente no es exclusivo para el área del PAIA; si no que también podía funcionar para otro programa llamado FEDNA.

Este cambio a última hora no perjudicó el proyecto en absoluto y demostró la adaptabilidad de la metodología a los cambios en requerimientos.

A continuación pueden observar parte de las imágenes capturadas en el proceso de desarrollo de la metodología XP. Más adelante se aprecian los diseños de las pantallas de la aplicación, reportes y estructuras de datos del sistema.

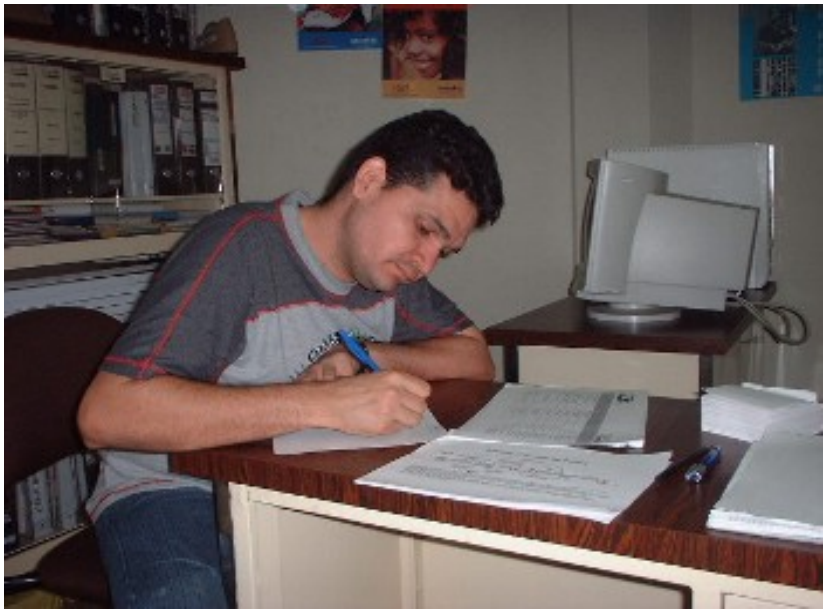
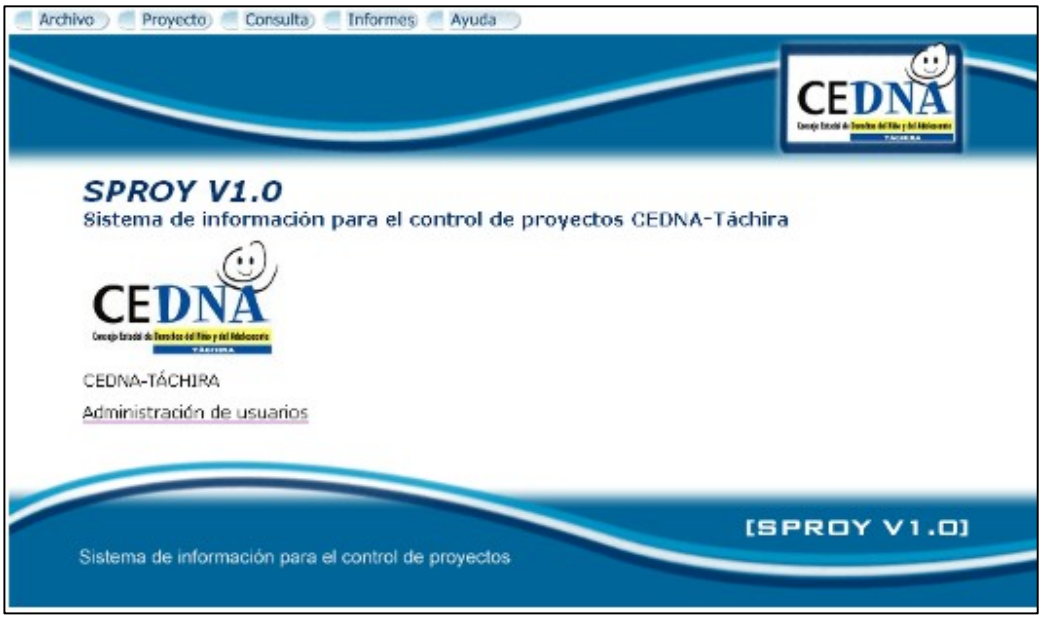


Imagen 1.- XP Realese Planning Meeting (Reunión de planificación de prestación)

Imagen 2.- Inicio de la primera iteración: Diseño de la estructura de la base de datos

Imagen 3.- 2da iteración: creación de la base de datos.

PANTALLAS



Página de inicio de la aplicación

Página para cargar Áreas de proyectos

Página para cargar líneas de un proyecto, esta relacionada directamente con el área.

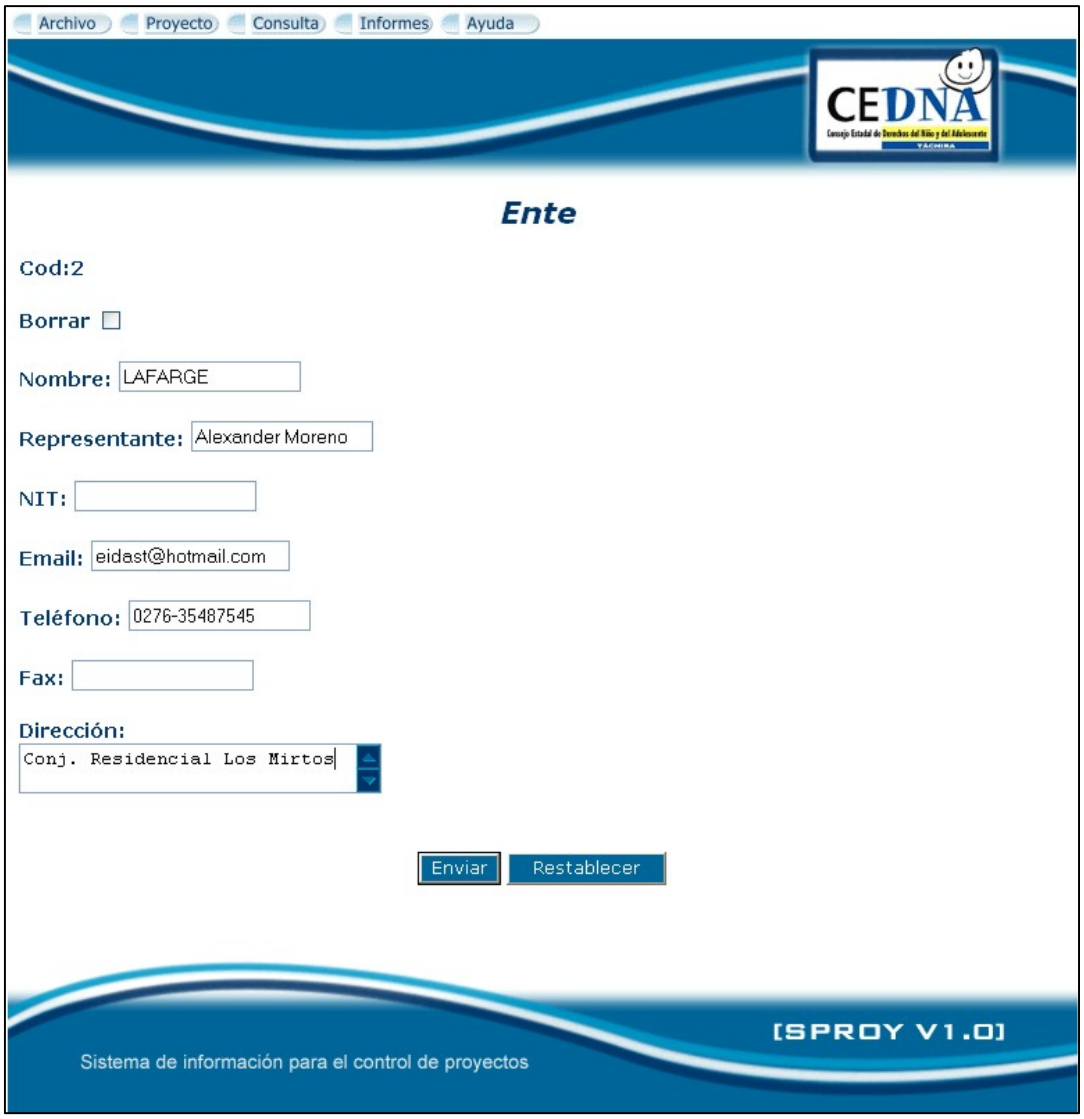
Página para cargar los diferentes estados de un proyecto

Página de administración de municipios

Página de administración de tipos de beneficiario

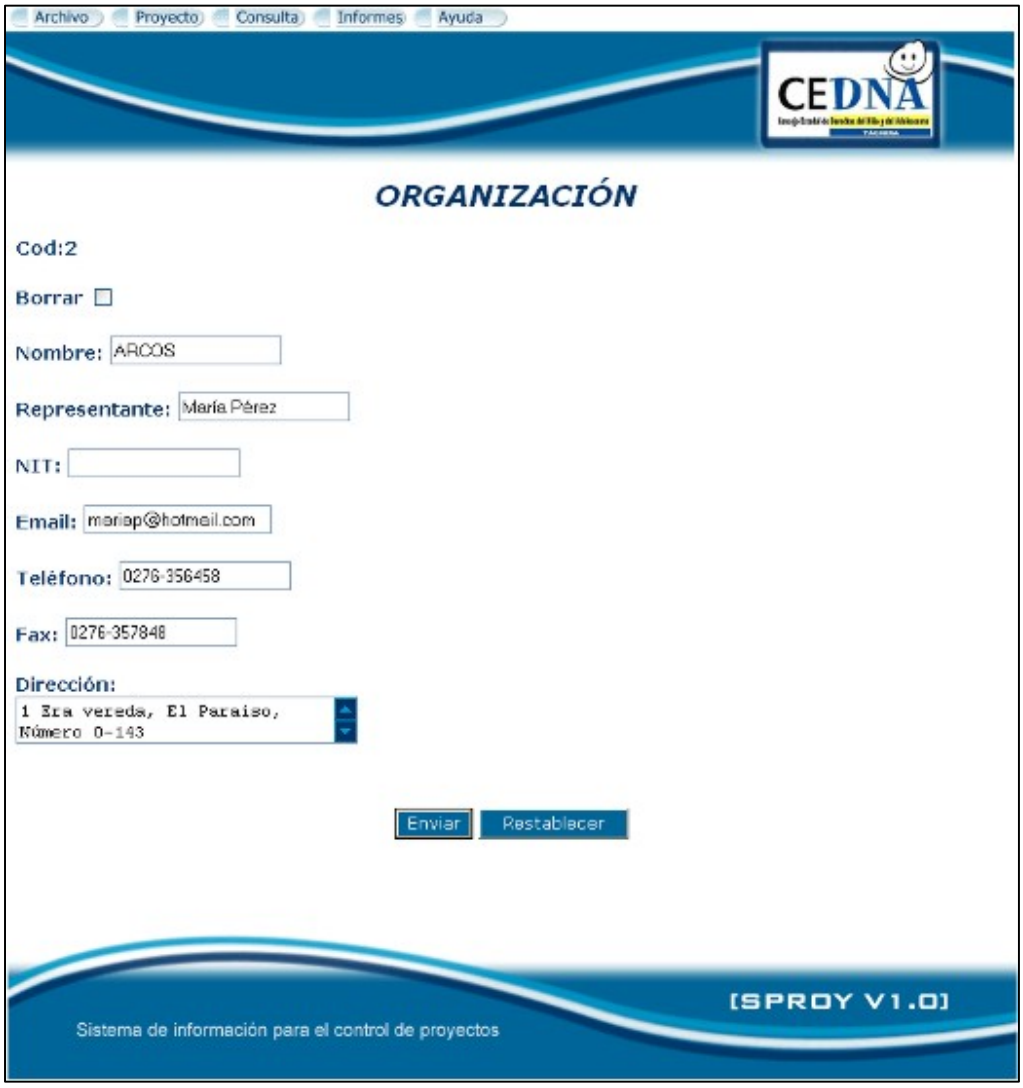
Página de administración de representantes

Página de administración de entes



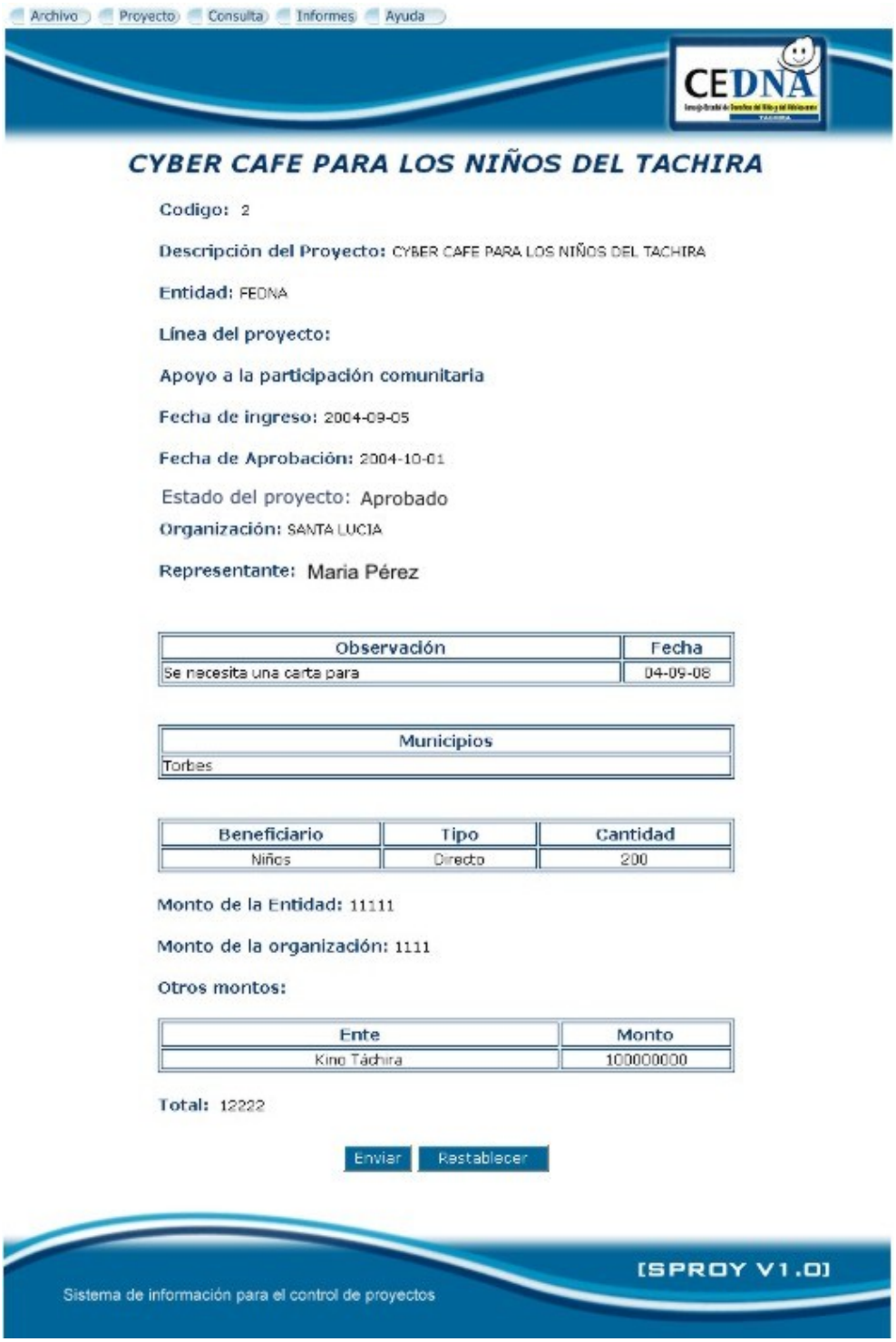
Página de administración de Ente (Continuación)

Página de administración de Organización



Página de Administración de Organización (Continuación)

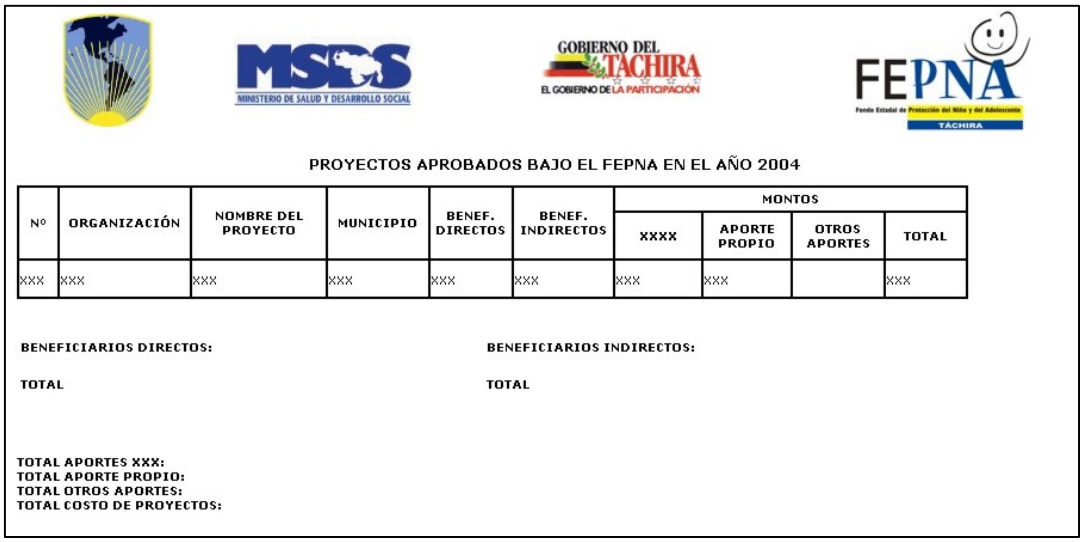
Página principal de administración de proyectos



Página de consulta de un proyecto

Página de administración de proyectos (Continuación)

Página del menú de informes de proyectos



PROYECTOS APROBADOS BAJO EL FEPNA EN EL AÑO 2004

Nº	ORGANIZACIÓN	NOMBRE DEL PROYECTO	MUNICIPIO	BENEF. DIRECTOS	BENEF. INDIRECTOS	MONTOS			
						XXXX	APORTE PROPIO	OTROS APORTES	TOTAL
xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx		xxx

BENEFICIARIOS DIRECTOS:

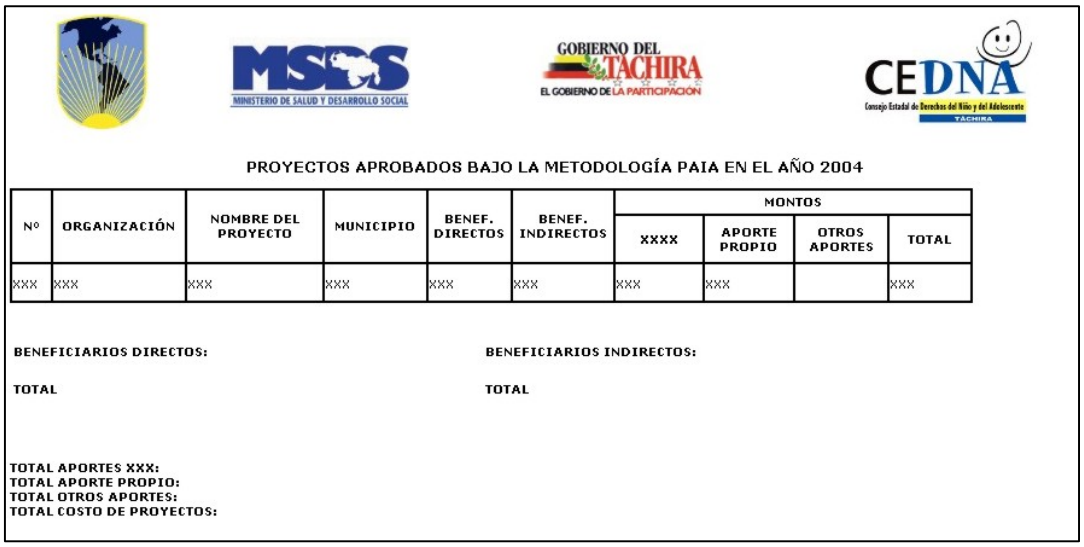
TOTAL

BENEFICIARIOS INDIRECTOS:

TOTAL

TOTAL APORTES XXX:
TOTAL APORTE PROPIO:
TOTAL OTROS APORTES:
TOTAL COSTO DE PROYECTOS:

Informe de proyectos aprobados FEPNA



PROYECTOS APROBADOS BAJO LA METODOLOGÍA PAIA EN EL AÑO 2004

Nº	ORGANIZACIÓN	NOMBRE DEL PROYECTO	MUNICIPIO	BENEF. DIRECTOS	BENEF. INDIRECTOS	MONTOS			
						XXXX	APORTE PROPIO	OTROS APORTES	TOTAL
xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx		xxx

BENEFICIARIOS DIRECTOS:

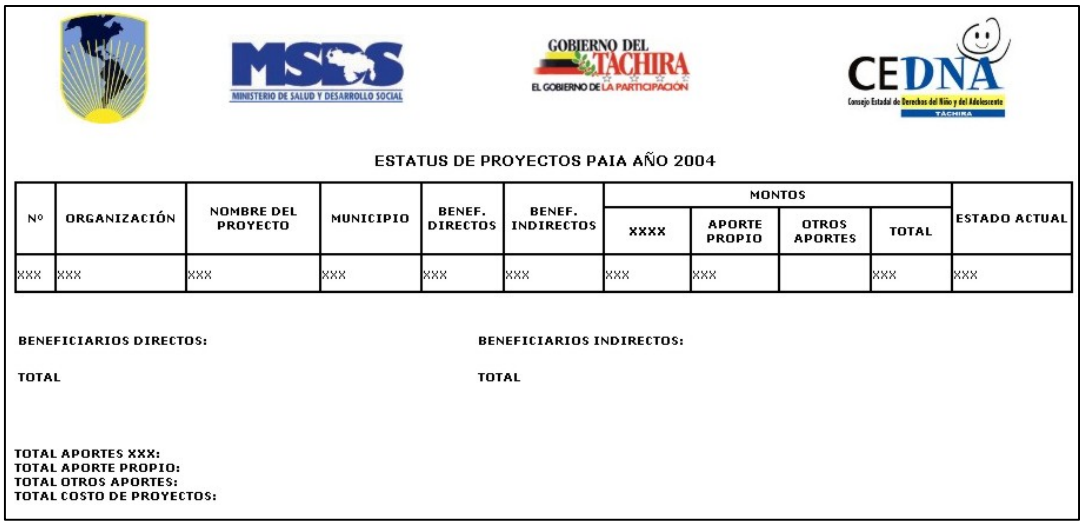
TOTAL

BENEFICIARIOS INDIRECTOS:

TOTAL

TOTAL APORTES XXX:
TOTAL APORTE PROPIO:
TOTAL OTROS APORTES:
TOTAL COSTO DE PROYECTOS:

Informe de proyectos aprobados PAIA



ESTATUS DE PROYECTOS PAIA AÑO 2004

Nº	ORGANIZACIÓN	NOMBRE DEL PROYECTO	MUNICIPIO	BENEF. DIRECTOS	BENEF. INDIRECTOS	MONTOS				ESTADO ACTUAL
						XXXX	APORTE PROPIO	OTROS APORTES	TOTAL	
XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX		XXX	XXX

BENEFICIARIOS DIRECTOS:

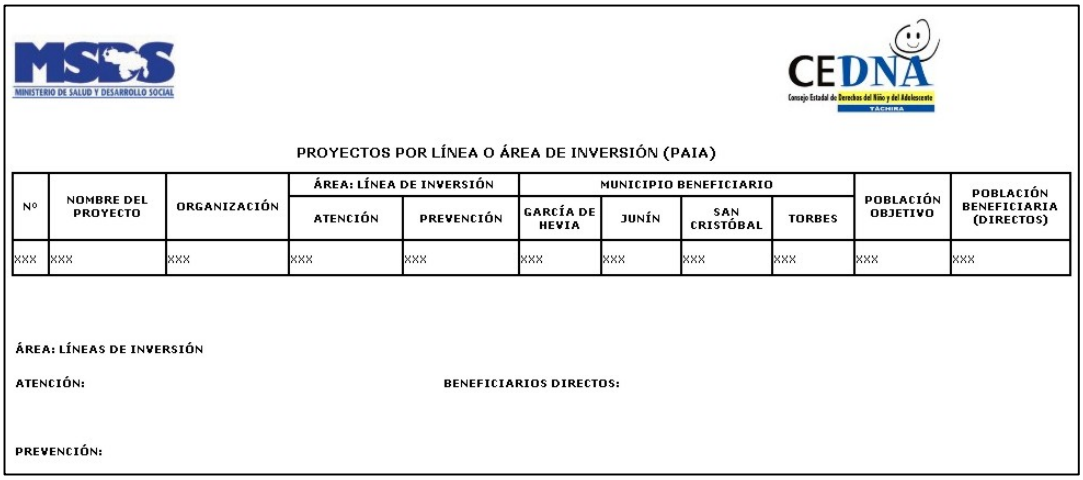
TOTAL

BENEFICIARIOS INDIRECTOS:

TOTAL

TOTAL APORTES XXX:
TOTAL APORTE PROPIO:
TOTAL OTROS APORTES:
TOTAL COSTO DE PROYECTOS:

Informe de Estatus de proyectos PAIA



PROYECTOS POR LÍNEA O ÁREA DE INVERSIÓN (PAIA)

Nº	NOMBRE DEL PROYECTO	ORGANIZACIÓN	ÁREA: LÍNEA DE INVERSIÓN		MUNICIPIO BENEFICIARIO				POBLACIÓN OBJETIVO	POBLACIÓN BENEFICIARIA (DIRECTOS)
			ATENCIÓN	PREVENCIÓN	GARCÍA DE HEVIA	JUNÍN	SAN CRISTÓBAL	TORBES		
XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX

ÁREA: LÍNEAS DE INVERSIÓN

ATENCIÓN:

BENEFICIARIOS DIRECTOS:

PREVENCIÓN:

Informe de Línea o área de inversión (PAIA)

PROYECTOS POR LÍNEA O ÁREA DE INVERSIÓN (FEPNA)

Nº	NOMBRE DEL PROYECTO	ORGANIZACIÓN	ÁREA: LÍNEA DE INVERSIÓN		MUNICIPIO BENEFICIARIO				POBLACIÓN OBJETIVO	POBLACIÓN BENEFICIARIA (DIRECTOS)
			ATENCIÓN	PREVENCIÓN	GARCÍA DE HEVIA	JUNÍN	SAN CRISTÓBAL	TORBES		
xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx

ÁREA: LÍNEAS DE INVERSIÓN

ATENCIÓN: BENEFICIARIOS DIRECTOS:

PREVENCIÓN:

Informe de Línea o área de inversión (FEPNA)

PAIA

ORGANIZACIONES ATENDIDAS

AÑO 2004

Nº	NOMBRE DEL PROYECTO	ORGANIZACIÓN QUE PRESENTA EL PROYECTO	CONTACTO	MUNICIPIO	DIRECCIÓN	TELÉFONO
xxx	xxx	xxx	xxx	xxx	xxx	xxx

CANTIDAD DE ORGANIZACIONES ATENDIDAS:

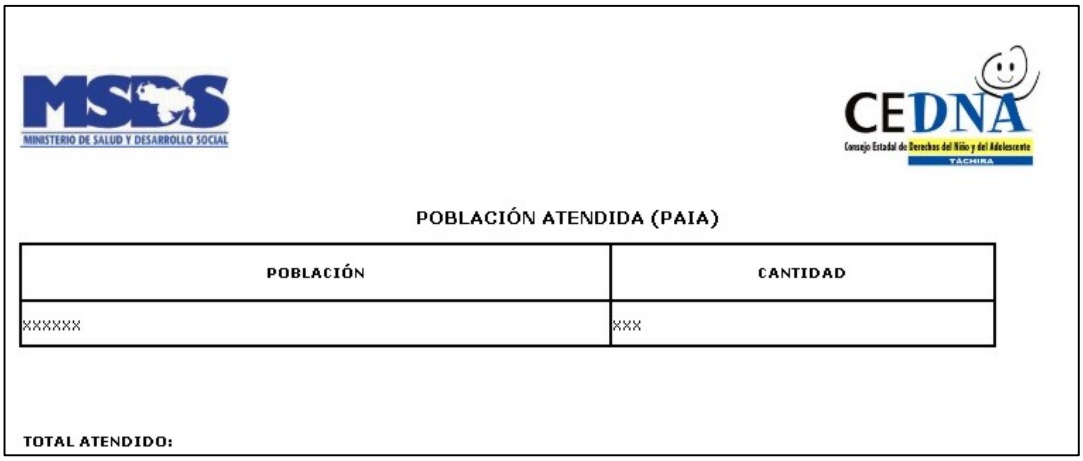
Informe de organizaciones atendidas (PAIA)

POBLACIÓN ATENDIDA (FEPNA)

POBLACIÓN	CANTIDAD
xxxxxx	xxx

TOTAL ATENDIDO:

Informe de población atendida (FEPNA)



POBLACIÓN ATENDIDA (PAIA)

POBLACIÓN	CANTIDAD
xxxxxx	xxx

TOTAL ATENDIDO:

Informe de población atendida (PAIA)

FEPNA

ORGANIZACIONES ATENDIDAS

AÑO 2004

Nº	NOMBRE DEL PROYECTO	ORGANIZACIÓN QUE PRESENTA EL PROYECTO	CONTACTO	MUNICIPIO	DIRECCIÓN	TELÉFONO
XXX	XXX	XXX	XXX	XXX	XXX	XXX

CANTIDAD DE ORGANIZACIONES ATENDIDAS:

Informe de organizaciones atendidas (FEPNA)

ESTATUS DE PROYECTOS FEPNA AÑO 2004

Nº	ORGANIZACIÓN	NOMBRE DEL PROYECTO	MUNICIPIO	BENEF. DIRECTOS	BENEF. INDIRECTOS	MONTOS				ESTADO ACTUAL
						XXXX	APORTE PROPIO	OTROS APORTES	TOTAL	
XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX		XXX	XXX

BENEFICIARIOS DIRECTOS:

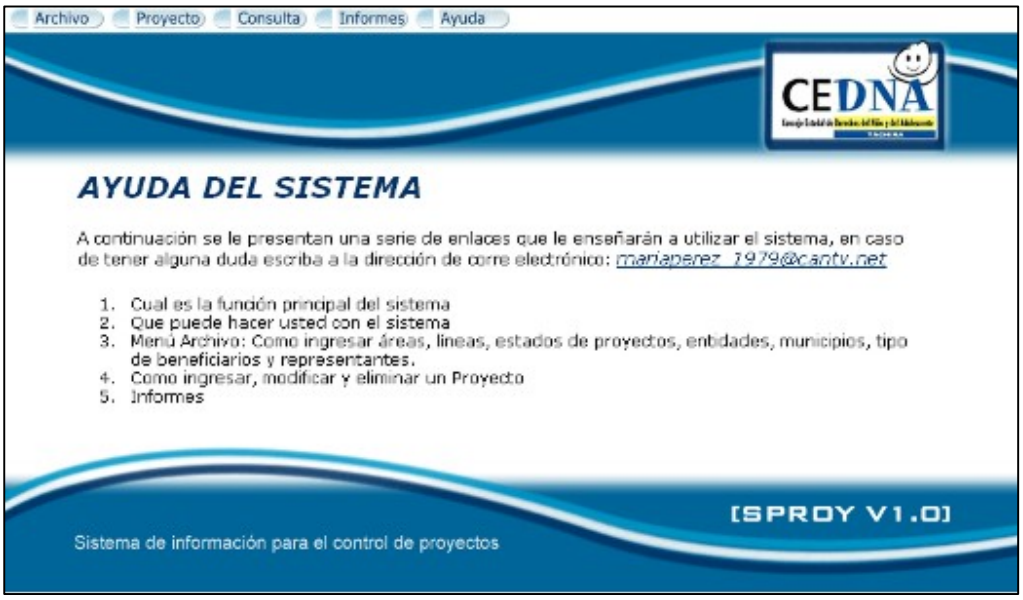
TOTAL

BENEFICIARIOS INDIRECTOS:

TOTAL

TOTAL APORTES XXX:
TOTAL APORTE PROPIO:
TOTAL OTROS APORTES:
TOTAL COSTO DE PROYECTOS:

Informe de estatus del proyecto (FEPNA)



Página de ayuda del sistema

Página de descripción del sistema

Página de administración de usuarios del sistema (niveles de seguridad)

ESTRUCTURAS DE DATOS

A continuación se presentan las estructuras de datos creadas para el sistema. Considerando que el sistema (prototipo) se realizó con una página Web dinámica, se agregan estructuras extras que interactúan con las librerías de YAPP.

Cedna_area

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
are_cod	int(8)		No		auto_increment
are_nam	varchar(50)		No		

Cedna_beneficiario

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
ben_cod	int(10)	UNSIGNED	No		auto_increment
ben_tb1_cod	int(10)	UNSIGNED	No	0	
ben_pro_cod	int(10)	UNSIGNED	No	0	
ben_tb2	int(10)	UNSIGNED	No	0	
ben_can	int(10)	UNSIGNED	No	0	

Cedna_constantes

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
con_cod	int(8)		No		auto_increment
con_nam	varchar(20)		No		
con_des	varchar(200)		Sí	*NULL*	
con_val	varchar(200)		No		

Cedna_contadores

Campo	*Tipo*	*Atributos*	*Nulo*	*Predeterminado*	*Extra*
con_cod	int(8)		No		auto_increment
con_nam	varchar(40)		Sí	*NULL*	
con_des	varchar(100)		Sí	*NULL*	
con_can	int(10)		Sí	*NULL*	

Cedna_entidad

Campo	*Tipo*	*Atributos*	*Nulo*	*Predeterminado*	*Extra*
ent_cod	int(8)		No		auto_increment
ent_nam	varchar(50)		No		

Cedna_estado

Campo	*Tipo*	*Atributos*	*Nulo*	*Predeterminado*	*Extra*
est_cod	int(8)		No		auto_increment
est_des	varchar(100)		No		

Cedna_linea

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
lin_cod	int(8)		No		auto_increment
cod_entidad	int(8)		No		
lin_nam	varchar(100)		No		
lin_are_cod	int(8)		No		

Cedna_municipio

Campo	*Tipo*	*Atributos*	*Nulo*	*Predeterminado*	*Extra*
mun_cod	int(10)	UNSIGNED	No		auto_increment
mun_nam	varchar(45)		No		
mun_pob	int(10)	UNSIGNED	No	0	

Cedna_modules

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
mod_cod	int(8)		No		auto_increment
mod_nam	varchar(40)		No		
mod_pre	varchar(20)		No		
mod_des	varchar(100)		Sí	*NULL*	
mod_url	varchar(100)		No		
mod_img	varchar(100)		Sí	*NULL*	
mod_pes	int(3)		Sí	*NULL*	

Cedna_observacion

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
obs_cod	int(10)	UNSIGNED	No		auto_increment
obs_des	varchar(45)		No		
obs_dat	date		No	0000-00-00	
obs_pro_cod	int(10)	UNSIGNED	No	0	

Cedna_organizacion

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
org_cod	int(10)	UNSIGNED	No		auto_increment
org_nam	varchar(100)		No		
org_dir	text		No		
org_ema	varchar(45)		No		
org_tel	varchar(45)		No		
org_fax	varchar(45)		No		
org_rep_cod	int(10)	UNSIGNED	No	0	
org_fec_ini	date		No	0000-00-00	
org_id	varchar(15)		No		

Cedna_proyecto

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
pro_cod	int(10)	UNSIGNED	No		auto_increment
pro_nam	varchar(45)		No		
pro_lin_cod	int(10)	UNSIGNED	No	0	
pro_est_cod	int(10)	UNSIGNED	No	0	
pro_fec_ini	date		No	0000-00-00	
pro_fec_apro	date		No	0000-00-00	
pro_org_cod	int(10)	UNSIGNED	No	0	
pro_ent_cod	int(10)	UNSIGNED	No	0	
pro_mon_e	float		Sí	*NULL*	
pro_mon_ap	float		Sí	*NULL*	
pro_mon_tot	float		Sí	*NULL*	

Cedna_representante

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
rep_cod	int(10)	UNSIGNED	No		auto_increment
rep_nam	varchar(45)		No		
rep_ape	varchar(45)		No		
rep_ced	varchar(15)		No		
rep_fec_nac	date		No	0000-00-00	
rep_dir	text		Sí	*NULL*	
rep_tel	varchar(45)		Sí	*NULL*	
rep_cel	varchar(45)		Sí	*NULL*	
rep_ema	varchar(80)		Sí	*NULL*	

Cedna_rpromun

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
rmu_cod	int(10)	UNSIGNED	No		auto_increment
rmu_pro_cod	int(10)	UNSIGNED	No	0	
rmu_mun_cod	int(10)	UNSIGNED	No	0	

Cedna_rproorg

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
ror_cod	int(10)	UNSIGNED	No		auto_increment
ror_pro_cod	int(10)	UNSIGNED	No	0	
ror_ent_cod	int(10)	UNSIGNED	No	0	
ror_mon	float		No	0	

Cedna_session

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
ses_cod	int(8)		No		auto_increment
ses_log	varchar(20)		Sí	*NULL*	
ses_ip	varchar(15)		Sí	*NULL*	
ses_url	varchar(200)		Sí	*NULL*	
ses_tim	datetime		Sí	*NULL*	

Cedna_tipob

Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
tb1_cod	int(10)	UNSIGNED	No		auto_increment
tb1_nam	varchar(45)		No		

Cedna_usuarios

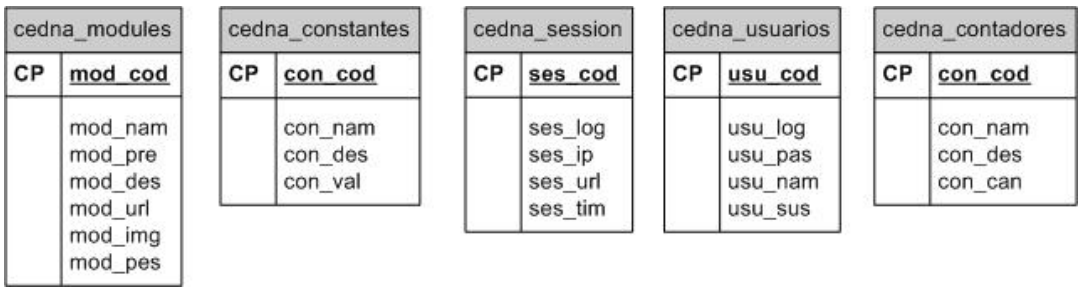
Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
usu_cod	int(8)		No		auto_increment
usu_log	varchar(20)		No		
usu_pas	varchar(50)		No		
usu_nam	varchar(100)		No		
usu_sus	int(1)	UNSIGNED	No	0	

Cedna_ente

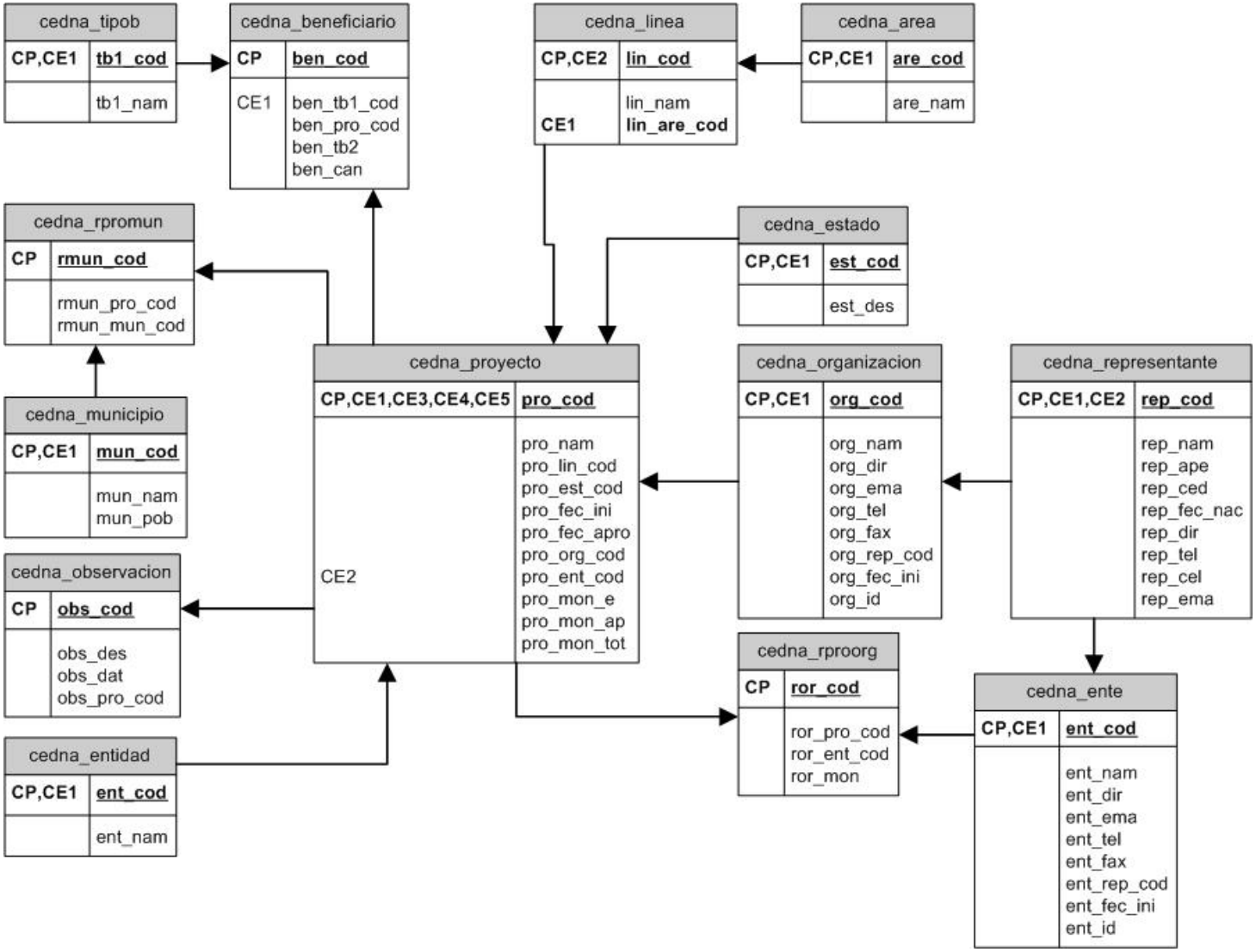
Campo	Tipo	Atributos	Nulo	Predeterminado	Extra
en_cod	int(10)	UNSIGNED	No		auto_increment
en_nam	varchar(100)		No		
en_dir	text		No		
en_ema	varchar(45)		No		
en_tel	varchar(45)		No		
en_fax	varchar(45)		No		
en_rep_cod	int(10)	UNSIGNED	No	0	
en_fec_ini	datetime		No	0000-00-00 00:00:00	
en_id	varchar(15)		No		

Diagrama Entidad-Relación

A continuación se muestran las tablas que almacenan datos importantes para las librerías YAPP y tablas sin relación.



La tabla de usuarios se utiliza para controlar la seguridad del sistema, la tabla modules (modulos) es utilizada directamente por la librería al igual que la tabla de sessión, constantes y contadores. Las tablas que presentan relación y que forman parte de la aplicación son las siguientes:


cedna_tipob
CP,CE1 tb1_cod
tb1_nam
cedna_beneficiario
CP ben_cod
CE1 ben_tb1_cod
ben_pro_cod
ben_tb2
ben_can
cedna_linea
CP,CE2 lin_cod
lin_nam
CE1 lin_are_cod
cedna_area
CP,CE1 are_cod
are_nam
cedna_rpromun
CP rmun_cod
rmun_pro_cod
rmun_mun_cod
cedna_estado
CP,CE1 est_cod
est_des
cedna_municipio
CP,CE1 mun_cod
mun_nam
mun_pob
cedna_proyecto
CP,CE1,CE3,CE4,CE5 pro_cod
CE2
pro_nam
pro_lin_cod
pro_est_cod
pro_fec_ini
pro_fec_apro
pro_org_cod
pro_ent_cod
pro_mon_e
pro_mon_ap
pro_mon_tot
cedna_organizacion
CP,CE1 org_cod
org_nam
org_dir
org_ema
org_tel
org_fax
org_rep_cod
org_fec_ini
org_id
cedna_representante
CP,CE1,CE2 rep_cod
rep_nam
rep_ape
rep_ced
rep_fec_nac
rep_dir
rep_tel
rep_cel
rep_ema
cedna_observacion
CP obs_cod
obs_des
obs_dat
obs_pro_cod
cedna_rproorg
CP ror_cod
ror_pro_cod
ror_ent_cod
ror_mon
cedna_ente
CP,CE1 ent_cod
ent_nam
ent_dir
ent_ema
ent_tel
ent_fax
ent_rep_cod
ent_fec_ini
ent_id
cedna_entidad
CP,CE1 ent_cod
ent_nam

CONCLUSIÓNES

- El diseño de un prototipo utilizando la metodología Extreme programming es totalmente adaptable a una aplicación web de proporciones pequeñas.
- La metodología también es adaptable a los cambios de requerimientos tan drásticos como los de estructura de los datos.
- El equipo de trabajo se siente a gusto al realizar las pruebas de calidad por si mismos y demuestran mayor interés en culminar un trabajo.
- La combinación de pair programming o programación en parejas mejora la productividad al conseguir soluciones en alguno de los dos programadores.
- Los diseños sencillos proporcionan una codificación sencilla y entendible por el programador.
- La facilidad de estar en el sitio del cliente evita la incertidumbre y ayuda a la corrección de errores a tiempo.
- Además de estas facilidades se consigue una mejor comunicación con el equipo de trabajo por las diversas reuniones en cada una de las iteraciones.

RECOMENDACIONES

- Si se piensa desarrollar una investigación o aplicar la metodología Extreme programming, hay que tener en cuenta que casi toda la documentación encontrada está en el idioma ingles y que poco se puede conseguir en español. Se recomienda conseguir una herramienta de traducción integrada como el Babylon para facilitar la lectura.
- En muchos de los casos hay un rechazo por parte de los programadores a aplicar cada uno de los pasos de la metodología que aseguran su éxito; se debe procurar estimular la capacidad comprender dichos pasos y así lograr mejores resultados.
- No se recomienda saltar alguno de los pasos; hay que realizar las iteraciones y reuniones necesarias.
- Es importante conservar el contacto con el cliente. Este es un punto crucial de la metodología.
- Hay que mantener siempre los diseños sencillos, así se evitan complicaciones en la codificación.

BIBLIOGRAFÍA CONSULTADA

1. CEDNA Táchira (2004). CEDNA, Consejo de Derechos del Niño y del Adolescente, Táchira, Venezuela: Autor.
2. CEDNA Táchira (2004). Programa de Apoyo a la infancia y Adolescencia Táchira, Venezuela: Autor.
3. MC CONNELL Steve (1997). Desarrollo y gestión de proyectos informáticos, Editorial Mc Graw Hill.
4. Universidad Pedagógica Experimental Libertador (1998). Manual de trabajos de grado de especialización y maestría y tesis doctorales Editorial FEDUPEL, Caracas, Venezuela.
5. YURI Emilio, CRISTÓBAL Cornieles (2002). Consejo Nacional de Derechos y consejos Estadales de Derechos, Editorial La Primera Prueba, Venezuela.
6. YURI Emilio, CRISTÓBAL Cornieles (2002). Programas de Protección y Entidades de Atención, Editorial La Primera Prueba, Venezuela.
7. WHITTEN Jefrey, BENTLEY Lonnie, BARLOW Victor (1996). Análisis y diseño de sistemas de información (Tercera Edición), Editorial Mc Graw Hill.
8. O´BRIEN, J. (2001). Sistemas de información gerencial. McGraw Hill Interamericana S.A.
9. GIDO Jack, CLEMENTS James (1999). Administración Exitosa de proyectos, Internacional Thomson Editores.
10. KORTH Henry, SILBERSCHTZ Abraham y SUDARSHAN S.(2002) Fundamentos de Bases de datos, Cuarta Edición Mc Graw Hill.
11. Ley Orgánica para la Protección del Niño y del Adolescente (1998). Gaceta Oficial de la República Bolivariana de Venezuela, 5.266, 2 de octubre de 1998.

12. UPEL (2003). Manual de trabajos de grado de especialización y maestría y tesis doctorales, FEDUPEL: Fondo Editorial de la Universidad Pedagógica Experimental Libertador, Morales, Rubio y Gutiérrez (1999)
13. SENN James (1998). Análisis y diseño de Sistemas de Información McGraw-Hill. México.
14. Hernández, Fernández y Baptista (1991)
15. KENDALL & Kendall (1991). Análisis y diseño de sistemas. Primera Edición, Editorial Prentice Hall. México.
16. Kent Beck (1999). Extreme Programming explained: Embrace change Editorial Addison-Wesley Professional; primera edición.
17. BECK Kent y FOWLER Matin (2000). Planning Extreme Programming Editorial Addison-Wesley Professional; primera edición.
18. UNA (1991). Técnicas de documentación e investigación II.Universidad Nacional Abierta.
19. ORFALI Robert, HARKEY Dan, EDWARDS Jeri (1998). Cliente/servidor: guía de supervivencia. Editorial Mc Graw Hill, Mexico.
20. BROOKS Charles (2003). A+ Traning Guide. Cuarta edición. Que Certification.
21. WALLACE Doug, RAGGETT Isabel y AUFGANG Joel (2002). Extreme programming for web projects. Editorial Addison Wesley
22. UNICEF Venezuela (2004). [On-line]. Disponible en: http://www.unicef.org/venezuela/frame-completo-vzla.htm
23. Extreme Programming (2004). [On-line]. Disponible en: http://www.hdp.cl/modules/news/article.php?storyid=266

24. WELLS Don (2004). Extreme Programming: A gentle introduction [On-line]. Disponible en:http://www.extremeprogramming.org/
25. Extreme Programming (2004) [On-line]. Disponible en: http://www.fing.edu.uy/inco/cursos/ingsoft/pis/GXP/index.htm
26. NAUMANN Berry, NAUMANN Anja (2004). rapid application development [On-line] . Disponible en: http://whatis.techtarget.com/definition/0,,sid9_gci214246,00.html
27. CANOS, LETELIER y PENÁDES (2003). Metodologías Ágiles en el desarrollo de Software [Resumen en línea], Universidad Politécnica de Valencia, Disponible en: www.willydev.net/descargas/prev/TodoAgil.Pdf [Consulta:2004, Junio 6]
28. LEÓN E. Welicki (2003). Implementando Extreme Programming en la plataforma .NET [Resumen en línea], Departamento de Lenguajes, Sistemas informáticos e Ingeniería del Software. Madrid, España. Disponible en: http://www26.brinkster.com/lwelicki/articles/Implementando%20Extreme%20Programming%20en%20la%20plataforma%20NET.pdf –[Consulta:2004, Junio 6]
29. ALDARIAS Paco (2004). Ciclo de vida de software [On-line]. Disponible en: http://www.inicia.es/de/pacofpr/fpt_tr1.ppt [Consulta:2004, Junio 6]
30. Cisco Networking Academy Programm (1999). CCNA Semestre 1 V2.1 [On-line]. Disponible: www.cisco.netacad.com (requiere permisología)
31. GARCÍA Lidón, PERALTA Luís y FERNÁNDEZ Samuel (sf) Sistemas operativos: un paseo por la historia [On-line]. Disponible: http://spisa.act.uji.es/~peralta/os/ [Consulta:2004, Junio 28]
32. Software Libre [On-line]. Disponible: http://www.tele-centros.org/tc-

toolkit2.0/fuente/conceptos/prin_conceptos1.htm
[Consulta:2004, Septiembre 4]

33. Cliente/Servidor.
[On-line]. Disponible:
http://www.linti.unlp.edu.ar/trabajos/tesisDeGrado/tutorial/redes/clteserv.htm
[Consulta:2004, Septiembre 2]

34. El Modelo Cliente-Servidor
[On-line]. Disponible:
http://www.fismat.umich.mx/~elizalde/tesis/node21.html
[Consulta:2004, Septiembre 2]

35. ZEPEDA ALDANA Consuelo (1997). Deficiencias legislativas en la protección de derechos de autor en software: propuesta de creación de la Ley de Derechos de Autor en Software [Resumen en línea]
Disponible:
http://www.justiniano.com/revista_doctrina/Derecho_de_autor_en_software.htm
[Consulta:2004, Septiembre 3]

36. MARTIN GONZALEZ Óscar (2000). Arquitecturas de sistemas de bases de datos [Archivo PDF]. Disponible: E-mule, ed2k://|file|ARQUITECTURAS.DE.SISTEMAS.DE.BASES.DE.DATOS.pdf|830439|BDC54FD538996B637FD97D0DF4E63342|/

37. Cybercursos.net (2004). [Página Web en línea]. Disponible: www.cybercursos.net [Consulta:2004, Septiembre 3]

38. La guía mIRC (2003).). [Página Web en línea]. Disponible: http://personal.redestb.es/guia_mirc/2.html [Consulta:2004, Septiembre 3]

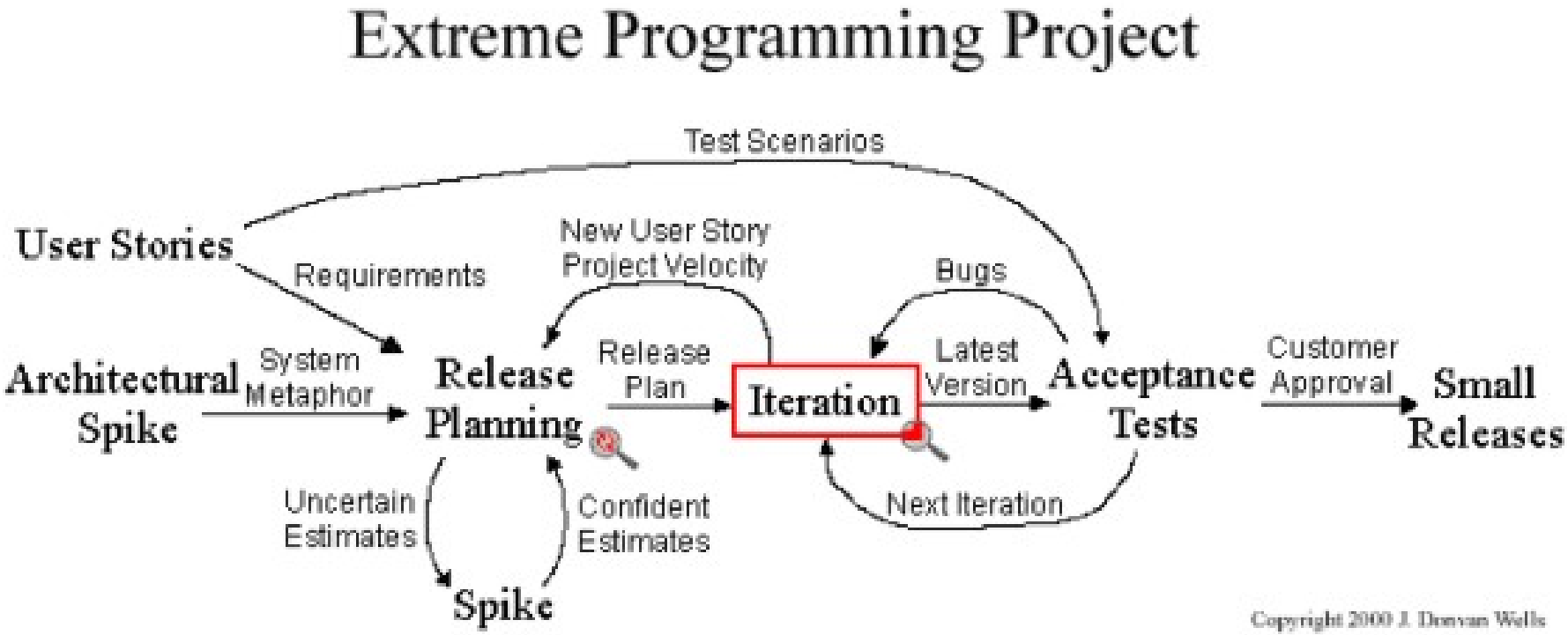
ANEXOS

ANEXO 1
MODELO DE ENTREVISTA

1. ¿QUE CONTRATIEMPOS HA PRESENTADO EL SISTEMA ACTUAL?
2. ¿SIENTE LA NECESIDAD DE MEJORAR EL SISTEMA?
3. ¿QUE DESEA QUE HAGA EL SISTEMA?
4. ¿LOS PARÁMETROS DEL SISTEMA SIGUEN LOS OBJETIVOS DE LA EMPRESA?
5. ¿CON CUANTAS PERSONAS CUENTA PARA LA REALIZACIÓN DE ESTE PROYECTO?. NÓMBRELAS Y ESPECIFIQUE SUS HABILIDADES Y FUNCIONES EN PROYECTOS ANTERIORES.
6. ¿CON QUE RECURSOS CUENTA PARA REALIZAR EL SISTEMA? Y DE SER NECESARIO ¿CUALES NECESITARÍA?
7. DESCRIBA TODOS LOS PASOS DEL SISTEMA ACTUAL
8. DE SEGUIR ESTA SITUACIÓN QUE PODRÍA PASAR
9. QUE TIPO DE RIESGOS PUEDEN AFECTAR LA PUESTA EN MARCHA DE ESTE PROYECTO
10. ¿SE PODRÍAN ESTABLECER REUNIONES DIARIAS PARA SEGUIR EL ESTADO DEL PROYECTO?. A PARTIR DE QUE FECHA Y QUE CANTIDAD DE HORAS PUEDE DEDICAR AL PROYECTO (DISPONIBILIDAD).
11. ¿ES POSIBLE FACILITAR UN ÁREA Y EQUIPO PARA REALIZAR EL SISTEMA?
12. ¿EXISTE ALGUNA PERSONA CON HABILIDADES DE PROGRAMACIÓN QUE PUEDA PARTICIPAR EL PROYECTO?

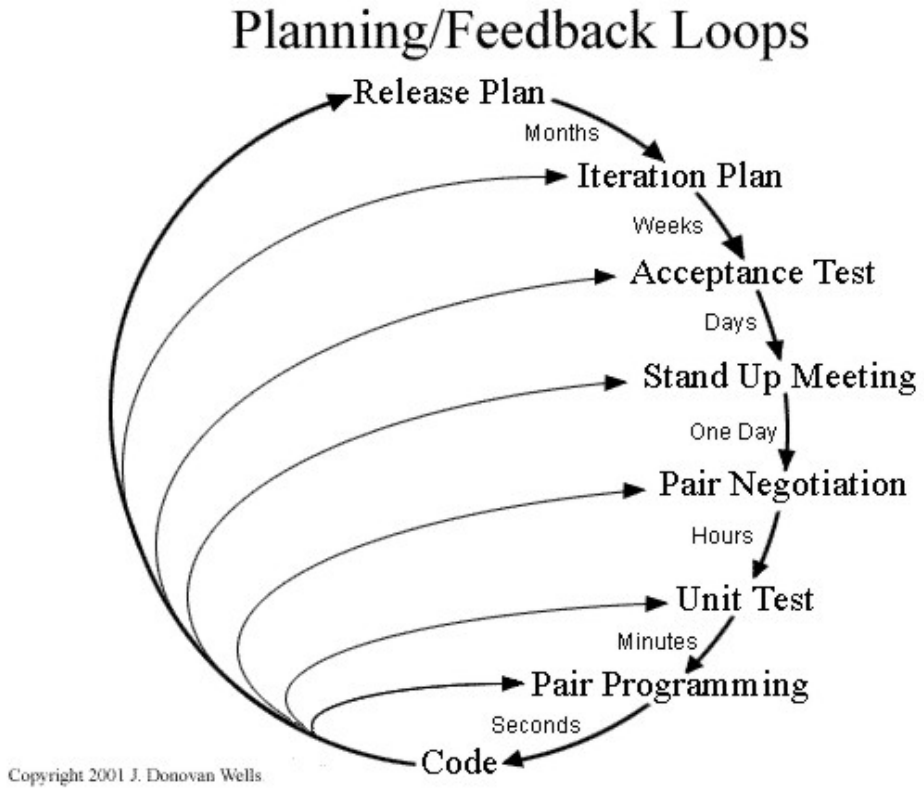
ANEXO 2

MODELO DEL TEXTO RESUMEN DE XP



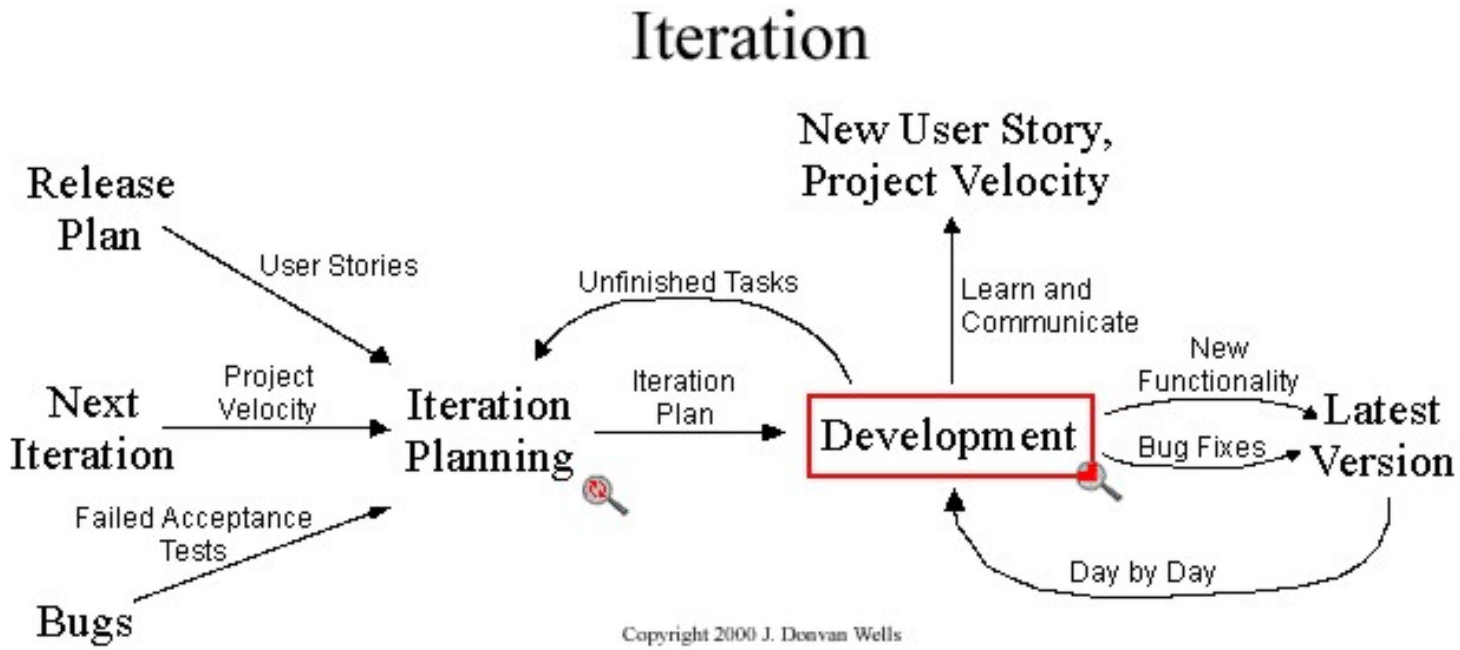
Historia de usuarios: El formato involucra 3 oraciones escritas en lenguaje del cliente sin términos técnicos
Architectural Spike: Construye un sistema que solo presta atención al problema bajo investigación e ignora los demás.
System Metaphor: se recomienda seleccionar un sistema de nombres para los objetos de manera que todos puedan identificarlos sin tener mayor conocimiento del sistema.
Release Planning: estimar cada una de las historias de los usuarios en términos de semanas de programación (pruebas), prioridad de las historias determinada por los clientes. (Selección de las historias a implantar) – (Filosofía basada en: variables; alcance, recursos, tiempo y calidad)
Release Plan: que historia de usuarios se van a implementar para cada prestación del sistema y fechas para las prestaciones, historias seleccionadas son traducidas en tareas de programación individuales para ser implementadas.(también están traducidas en *pruebas de aceptación* durante la iteración)

Velocidad del proyecto: en función del alcance se divide el total de semanas estimadas en las historias de los usuarios por la velocidad del proyecto para determinar cuantas iteraciones hay que realizar para obtener una prestación. Cuando cambia por un par de iteraciones se debe fijar fecha a una reunión de planificación de prestación con los clientes y crear un nuevo plan de prestación.



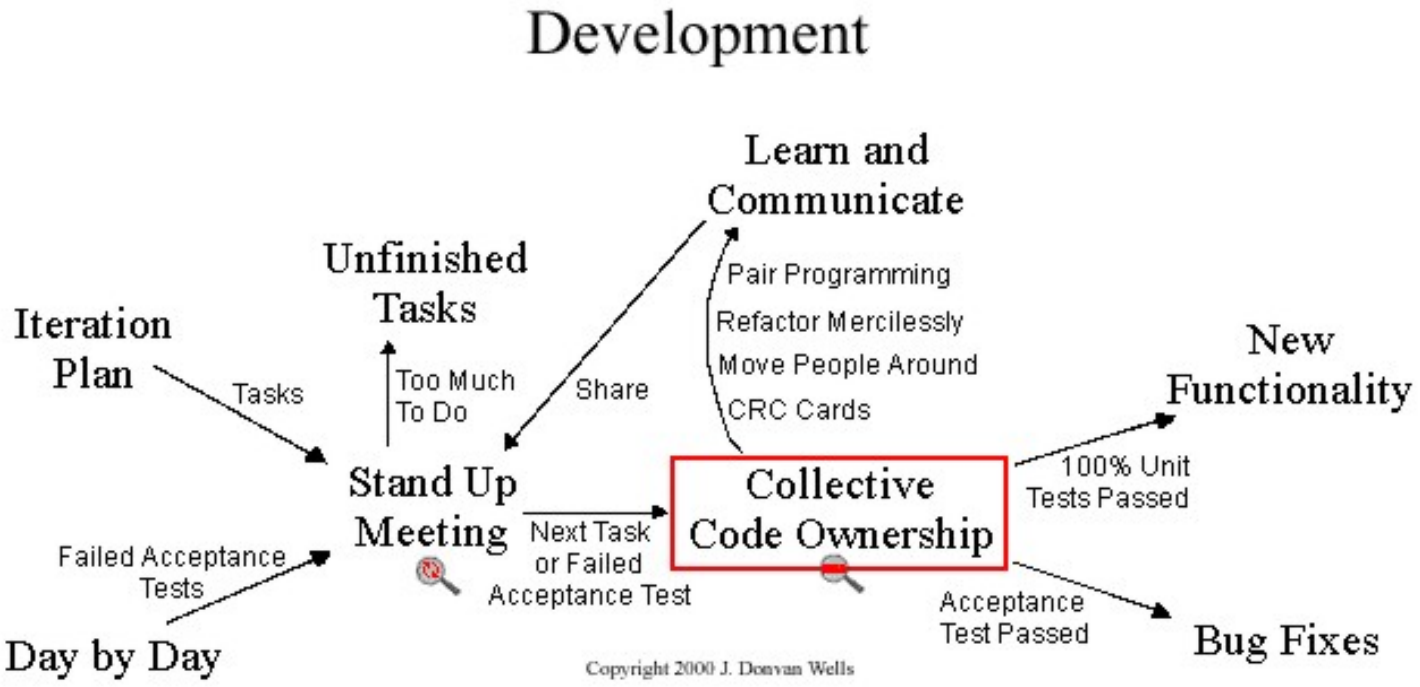
Acceptance Test: aceptación se crean a partir de las historias de los usuarios (una historia puede tener 1 o más pruebas). Son sistemas de pruebas de cajas negras. Cada uno representa algún resultado esperado del sistema. Los clientes son responsables de verificar la veracidad de las pruebas de aceptación y de repasar la puntuación de las pruebas para decidir cual de las pruebas fallas tienen mayor prioridad

Daily Stand Up Meeting: Una reunión corta todas las mañanas es utilizada para comunicar problemas, soluciones y promover el enfoque del equipo. (todos se disponen en círculo para evitar discusiones - el lugar puede ser espontáneo)
Pair programming: Todo el código incluido en la producción de la prestación es creado por dos personas trabajando juntos en la misma computadora. Una persona escribe y piensa tácticamente sobre el método creado, mientras que la otra persona piensa estratégicamente del como el método encaja dentro de la clase.
Unit test: Significa crear una prueba para el código primero antes de realizarlo, de esta manera el diseño guarda relación con lo que valora el usuario y el código termina siendo más sencillo



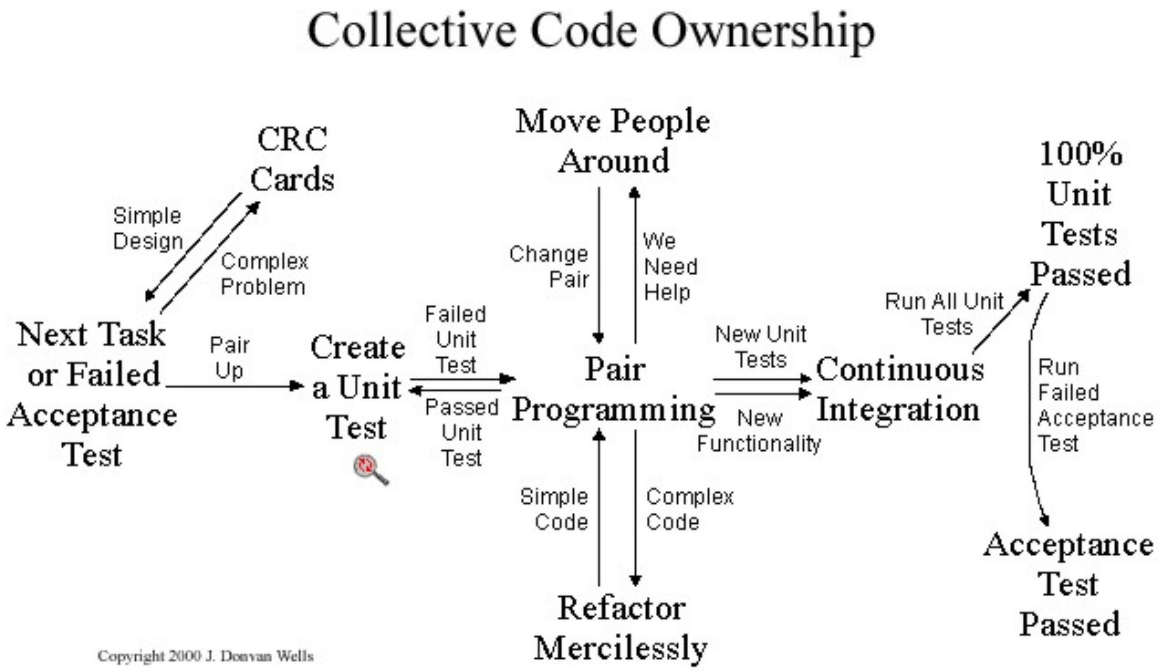
Interactive Development: Divide horario de desarrollo en una docena de iteraciones de una a tres semanas de duración. (Mantener constante la duración de las iteraciones, el latido del proyecto) hay que realizar una reunión de planificación de iteración al comienzo de cada iteración para planificar que es lo que realizará. es en contra las reglas mirar más allá y tratar de implantar algo que no esta programado para esta iteración. Hay que tomar seriamente las fechas de entrega

Bugs: Para evitarlos se crearon las pruebas de aceptación de fallas, ayudan al cliente a definir el problema y comunicarlo a los programadores. Al proporcionar estas pruebas de aceptación de fallas, los desarrolladores pueden crear pruebas unitarias para mostrar el error desde el punto de vista del código fuente.
Planificación de iteraciones: se realiza una reunión en donde se divide las actividades de programación en iteraciones de una semana o más Las pruebas de aceptación fallos que tienen que arreglarse también son seleccionadas. El cliente selecciona las historias de usuarios estimadas que al sumarse igualan la velocidad del proyecto para la última iteración. Las tareas son escritas en “index cards” (cartas indexadas) - Escritas en un lenguaje más técnico. Las tareas que duran más de 3 días deberán dividirse mucho más. Totalizado las estimaciones del tiempo en días ideales de programación de las tareas, no puede exceder la velocidad del proyecto de la iteración previa.



Refractor Mercilessly: se utiliza para mantener un diseño simple y para evitar desorden innecesario y complejidad. Mantiene el código limpio y conciso para que sea más entendible, modificable y extendido. se elimina redundancia, se

elimina funcionalidad innecesaria y rejuvenece diseños. Se debe dejar ir las nociones de lo que el sistema debe hacer o no y tratar de ver el nuevo diseño que emerge ante los ojos de los desarrolladores.
Move people around: Simplemente hay que motivar a todos en tratar de trabajar una sección del sistema al menos en cada iteración. La Programación en parejas lo hace posible sin perder productividad y asegura la continuidad del pensamiento.
CRC Cards: Usar Clases, responsabilidades y Cartas de colaboración (CRC) para diseñar el sistema como equipo. Las cartas individuales CRC son usadas para representar objetos. La clase de estos objetos puede ser escrita en el tope de la carta, las responsabilidades son catalogadas abajo en el lado izquierdo, las clases de colaboración están listadas al lado derecho de cada responsabilidad. La sesión CRC se ejecuta con alguien describiendo el sistema y hablando de cuales objetos envían mensajes a otros objetos. Al pasar por las debilidades y problemas del proceso son fácilmente visibles. Diseñar alternativas puede ser explorado rápidamente al estimular el diseño propuesto. Se requiere un registro permanente, una carta por cada clase puede escribirse por completo y ser retenida como documentación.



Next task or fail Acceptance Test: Se debe crear una prueba de aceptación antes del debugging para que los clientes puedan identificar el problema y comunicarlo a los programadores. Los programadores tienen pruebas fallas para enfocarse en sus propios esfuerzos y saber cuando el problema es solucionado.
Simple Disign: La simplicidad del diseño es la clave; un diseño simple toma menos tiempo en ser terminado que uno complejo
Continuous integration: Los desarrolladores deben integrar continuamente el código de la prestación en el repositorio de código cada pocas horas, cada vez que sea posible. Todos necesitan con la última versión. Los cambios no pueden realizarse en código antiguo.Casi todo el tiempo la integración continua evita o detecta problemas de compatibilidad de una forma temprana

ALGORITMO SIMPLE DE XP:

Mientras(sistema_es_útil) {

Captar requisitos

User Stories

Methaphor

Planificar

Release planning

Iteration **planning**

Desarrollar

Programming

Presentar la entrega

Releasing

}

ANEXO 3
PLANIFICACIÓN MODIFICADA