Revisión de metodologías de desarrollo de software

*Erik A. Sacre Mahnert**

Resumen

El siguiente trabajo muestra parte de una investigación realizada por el autor. Esta pretende recopilar algunas de las metodologías de desarrollo de software existentes en el mercado. Para ello, se consultó principalmente internet, tanto en sitios de acceso público, como en aquellos de carácter académico, con acceso restringido. Un resumen explicativo de cada una de las metodologías revisadas, con las referencias bibliográficas respectivas, se presenta al lector de modo que pueda profundizar en la que más le interese. Finalmente, se exponen las conclusiones derivadas al respecto del estudio.

Introducción

Muchas de las empresas que desarrollan software en nuestro país lo hacen sin aplicar ninguna metodología formal. Es posible que esto se deba a que se desconocen las más recientes propuestas de desarrollo

* Ingeniero de Ejecución en Informática, Universidad Técnica Federico Santa María; Magíster en Ciencias, Mención Computación, Universidad de Chile. Director de la Carrera de Ingeniería de Ejecución en Informática, UCINF.

de software que aparecen en la literatura, pues las metodologías comúnmente aceptadas en la comunidad de desarrolladores y empresas son las tradicionales, referidas a aquellas que se aplican en el desarrollo de sistemas de software convencionales. Sin embargo, con la inserción de internet y las tecnologías abiertas, estas metodologías muchas veces no se pueden aplicar directamente en el desarrollo de sistemas modernos [13].

La maduración de una metodología requiere mucho tiempo. Es así como las metodologías tradicionales, algunas de los años 70, aún se aplican. No obstante, la evolución del hardware, que implica mayores exigencias al software y viceversa, ha dejado atrás un método científico en la construcción de sistemas computacionales. En otras palabras, el "boom" de la tecnología no va a la par con las metodologías de desarrollo, que garantizan un óptimo uso de los recursos. Muchos desarrolladores construyen aplicaciones recurriendo solamente a su experiencia en los diseños y en la mantención, y otros pocos adaptan metodologías de la década de los 70 a tecnologías modernas, lo cual no es eficiente.

Lo aprendido en el pasado simplemente no puede ser aplicado en este nuevo tipo de aplicaciones. En efecto, lo que antes era utilizado no puede ser adaptado a las nuevas reglas de negocios y realidades tecnológicas. De hecho, como medida, se replantearon las disciplinas de la generación de negocios antecesoras, pero irónicamente se adoptaron prácticas que en la mayoría de los casos fallan en las nuevas generaciones [13].

Muchos de los atributos de calidad para sistemas basados en Web, tales como facilidad de navegación, accesibilidad, escalabilidad, mantenibilidad, usabilidad, compatibilidad, interoperabilidad y seguridad, no son especificados con la debida consideración durante el desarrollo. Gran cantidad de las aplicaciones Web, además, fallan en aspectos culturales, de privacidad, morales y legales [7].

Construir un sistema basado en Web requiere, en sus comienzos, de una disciplina mental diferente a la que es necesaria en la construcción de un sistema tradicional. Por ejemplo, el control y reporte de errores de la aplicación se debe extender a lugares distantes, la cantidad de usuarios

es incierta, la mantención muchas veces debe ser en línea, la seguridad ante los intrusos se debe extenuar a límites más allá de los requeridos por los sistemas tradicionales. Por otra parte, la navegación (hiperdocumentos), que es una característica esencial de los sistemas Web, no está presente en otros tipos de sistemas, sin olvidar aspectos de diseño gráfico y contenido, que hacen a las aplicaciones Web muy diferentes de las aplicaciones tradicionales.

Es así como muchas de las aplicaciones basadas en Web ya construidas y en operación, como también otras en proceso de construcción, utilizan metodologías tradicionales de construcción de aplicaciones, las cuales no están orientadas al Web, debido principalmente a que fueron concebidas antes de los años 90. Esto trae como consecuencia que el proceso de construcción y mantención sea muy costoso y con la eventualidad de presentar gran cantidad de errores, llegando incluso al fracaso de muchos proyectos.

Una investigación reciente en Estados Unidos determinó que el 53% de los proyectos basados en Web no cumplen con las funcionalidades requeridas y el 52% son de pobre calidad, y no hay razones para pensar que en Chile estas cifras sean mejores [6].

Considerando también que los sistemas basados en Web son dinámicos, la extensión natural del producto final de la *construcción* es la *mantención*, la cual muchas veces se entiende solamente como un "colocar información en una página" con una desordenada interacción.

Identificación del problema

El principal problema que tiene el ingeniero de software es decidirse por una forma de hacer bien las cosas. Algunos intentos de ello han resultado en metodologías y, otros, en recetas o guías para lograr un buen producto. Dado también que la Ingeniería de Software tiene muchas variantes —es decir, cada autor propone un enfoque diferente—, muchas veces confunde más que contribuye, pues al existir tanta variedad de formas de hacer las cosas, los desarrolladores de software fácilmente pueden elegir un camino equivocado en su aplicabilidad e, incluso, inventar

sus propios métodos, ya sea porque no se entienden los métodos existentes o simplemente porque no son aplicables a la realidad de la empresa y al equipo de desarrollo en cuestión. La pregunta clave, entonces, es: ¿Cómo enfrentar estos proyectos Web de una manera tal que podamos garantizar su éxito sin salirnos de los parámetros de tiempo, presupuesto y calidad previamente fijados?

Varios autores de renombre trabajan con el fin de presentar propuestas que permitan a los ingenieros de software poder desarrollar aplicaciones de una manera tal que podamos garantizar su éxito sin perder aquellos parámetros de tiempo, presupuesto y calidad previamente fijados. Sin embargo, todos estos intentos han tenido diversos grados de éxito y fracaso. Por ejemplo, una manera exitosa de organizar esquemáticamente los datos de una aplicación es el Modelo de Datos, el cual ha tenido diferentes versiones, todas mundialmente aceptadas. Pero los autores aún no se han puesto de acuerdo en un método adecuado para su utilización, entendiendo esta postura como integración de este y otros esquemas de desarrollo.

Las tendencias metodológicas actuales pretenden centrarse en el conocimiento de las personas, a diferencia de las metodologías más antiguas, donde el pilar básico era la documentación, ya sea para independizar los procesos de los desarrolladores o para lograr una comunicación formal entre el equipo y los usuarios. En cambio, las tendencias hoy en día tienen sus fundamentos en la integración laboral de los equipos de trabajo, favoreciendo el desarrollo personal y profesional, en gratos ambientes de trabajo, los cuales son controlados tan hábilmente que se refuerzan más los fundamentos.

Descripción de propuestas para desarrollar aplicaciones basadas en Web

A continuación se describen brevemente algunos de los métodos, modelos y procesos que existen en la actualidad. Cada uno de ellos se explica en forma resumida en este capítulo y las referencias para los detalles de su aplicación se indican respectivamente. Para mayores detalles de aplicabilidad, el lector deberá remitirse a la literatura de sus correspondientes autores, pues no es el fin

de este trabajo profundizar en cada método, modelo y proceso.

El objetivo de la revisión de las diferentes descripciones, efectuada anteriormente y considerada como relevante para enmarcar esta investigación, es entender que, si bien hay definiciones coherentes entre un mismo autor y entre varios autores, las propuestas metodológicas examinadas en este trabajo dan cuenta de que aun los mismos autores de estas propuestas no tienen clara esta terminología. Por ejemplo, se abusa de la palabra método y metodología, considerándose como sinónimos. Otros autores usan indistintamente las palabras "marco de trabajo", "método" y "metodología", y en algunos otros casos separan definidamente los conceptos de herramientas, modelos y procesos.

Las propuestas revisadas se resumen en la tabla siguiente:

TABLA: PROPUESTAS ESTUDIADAS

Propuesta	Autores
Crystal (Clear)	Cockburn, A.
DRA (Desarrollo Rápido de Aplicaciones)	Martin, J.
Espiral	Boehm, B.
eXtreme Programing (XP)	Beck, K.
Incremental	Mills, H.D.
Iweb	Pressman, R.
Lineal Secuencial (Cascada)	Royce, W.
RMM (Relationship Management Methodology)	Isakowitz, T. et al.
RUP (Rational Unified Process)	Rumbaugh, J. et al.

Lineal Secuencial

Autor: Winston Royce
Referencias de publicación:
[15], [13], [16]

RESUMEN:
Llamado algunas veces "ciclo de vida básico", "método cascada", "método cascada clásico", este es un enfoque sistemático y secuencial para el desarrollo del software, que comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento. Las etapas son: ingeniería y modelado de sistemas/información, análisis de los requisitos del software, diseño, generación de código, pruebas y mantenimiento.

OBSERVACIONES:
El modelo Lineal es el más antiguo y más extensamente utilizado en la Ingeniería de Software. Sin embargo, la crítica ha puesto en duda su eficiencia. Por ejemplo, en un proyecto rara vez se sigue una secuencia, que es lo que propone este método. Asi-

GRÁFICA DESCRIPTIVA:

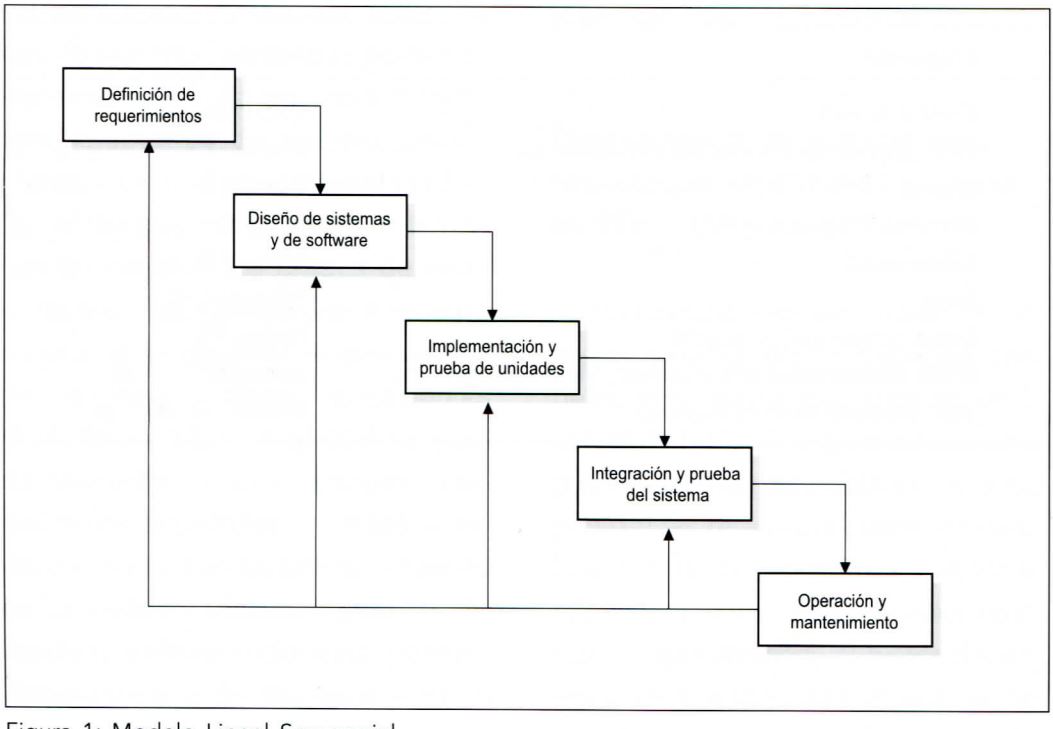

Figura 1: Modelo Lineal Secuencial

mismo, es muy difícil contar con todos los requisitos al comienzo del proyecto, condición necesaria para aplicar este método. Por último, el cliente debe tener paciencia, porque tendrá el producto al final. Este modelo se sigue usando en proyectos breves en áreas bien conocidas por el equipo de desarrollo. Su aplicación es útil en proyectos con pocos cambios y con pocas novedades.

DRA (Desarrollo Rápido de Aplicaciones o RAD)

Autor: J. Martin

Referencias de publicación:
[13], [10]

RESUMEN:

DRA es un modelo de desarrollo de software Lineal Secuencial que enfatiza un ciclo de desarrollo extremada-

GRÁFICA DESCRIPTIVA:

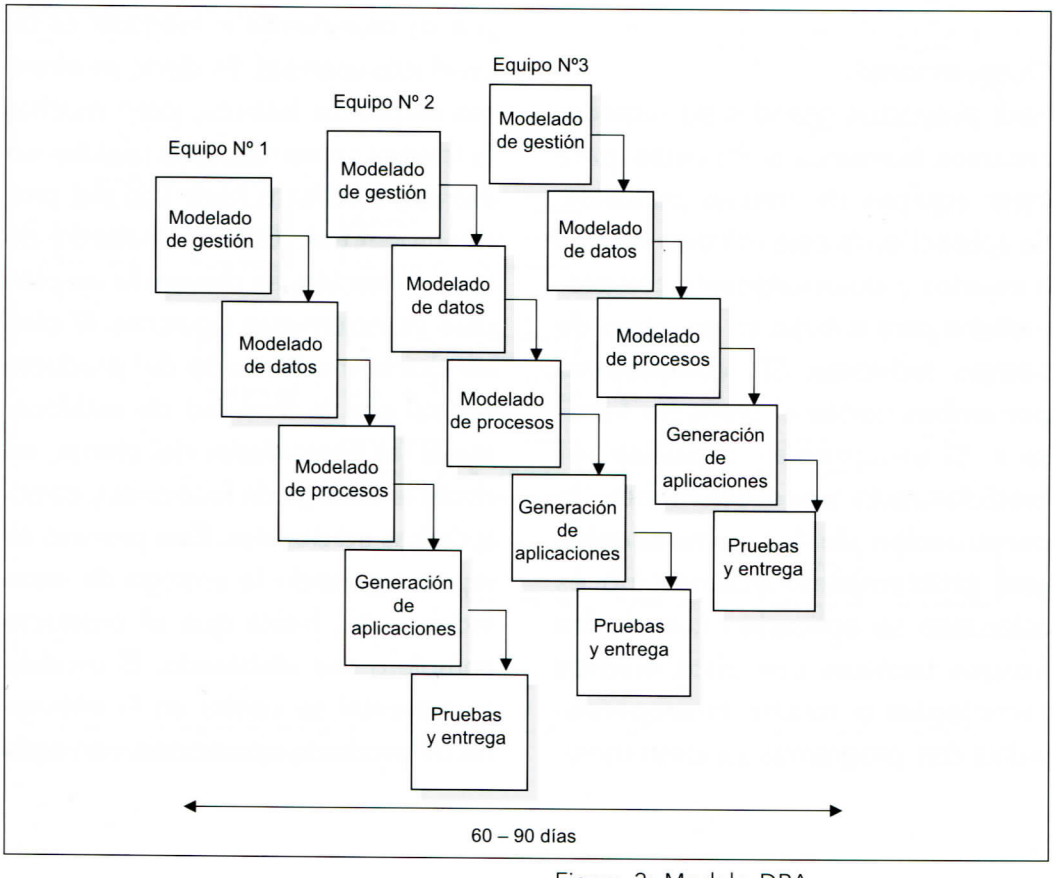

Figura 2: Modelo DRA

mente corto. RAD es una adaptación a alta velocidad del modelo Lineal Secuencial en el que se logra el desarrollo rápido por medio de la utilización de una construcción basada en componentes. Si se comprenden bien los requisitos y se limita el ámbito del proyecto, RAD permite al equipo de desarrollo crear un sistema completamente funcional dentro de períodos breves (por ejemplo, 60 ó 90 días). El enfoque RAD comprende las siguientes etapas:

OBSERVACIONES:
Para proyectos grandes se requiere recursos humanos suficientes para crear equipos de trabajo paralelos. La aplicación de este método implica a clientes y desarrolladores comprometidos para trabajar en un marco de tiempo reducido. Sin compromiso por ambas partes el proyecto fracasará. Si un sistema no puede ser modularizado adecuadamente, la construcción de los componentes será problemática. Además, no es adecuada su aplicación cuando los riesgos técnicos son altos (nuevas tecnologías o mucha interoperatividad con programas ya existentes).

Incremental

Autor: H.D. Mills (IBM)
Referencias de publicación:
[13], [16], [12]

RESUMEN:
El modelo Incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software. Cuando se utiliza este método, el primer incremento a menudo es un producto esencial. Es decir, se afrontan requisitos básicos, pero muchas funciones suplementarias quedan sin extraer. El cliente hace uso del producto central. Como resultado de esta utilización, se desarrolla un plan para el incremento siguiente. El plan afronta la modificación del producto central con la finalidad de satisfacer mejor las necesidades del cliente, así como la entrega de funciones y características adicionales. Este proceso se repite siguiendo la entrega de cada incremento, hasta que el producto completo sea elaborado. El modelo Incremental se centra en la entrega de un producto operacional con cada

incremento. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para la evaluación. Este tipo de desarrollo es particularmente útil cuando la dotación de personal no está disponible para una implementación completa en la fecha límite que se haya establecido para el proyecto. Los primeros incrementos se pueden llevar a cabo con menos personas.

OBSERVACIONES:

Los incrementos deben ser relativamente pequeños y cada uno debe entregar alguna funcionalidad del sistema. Una variación de este modelo es la programación extrema (eXtreme Programming).

GRÁFICA DESCRIPTIVA:

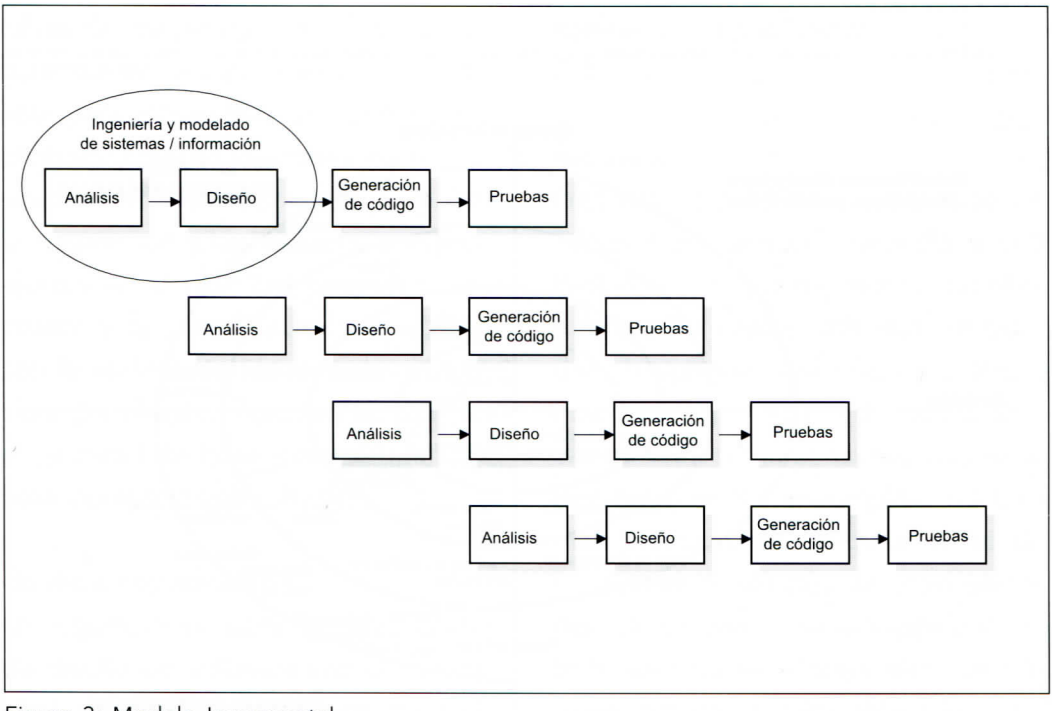

Figura 3: Modelo Incremental

Espiral

Autor: Barry Boehm
Referencias de publicación:
[13], [16], [2]

RESUMEN:
Modelo evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo Lineal Secuencial. Proporciona el potencial para el desarrollo rápido de versiones incrementales del software. En el método Espiral, el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones, la versión Incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones se producen versiones cada vez más completas del sistema diseñado, el cual se divide en un número de actividades de marco de trabajo, también llamadas regiones de tareas.

Cada una de las regiones está compuesta por un conjunto de tareas,

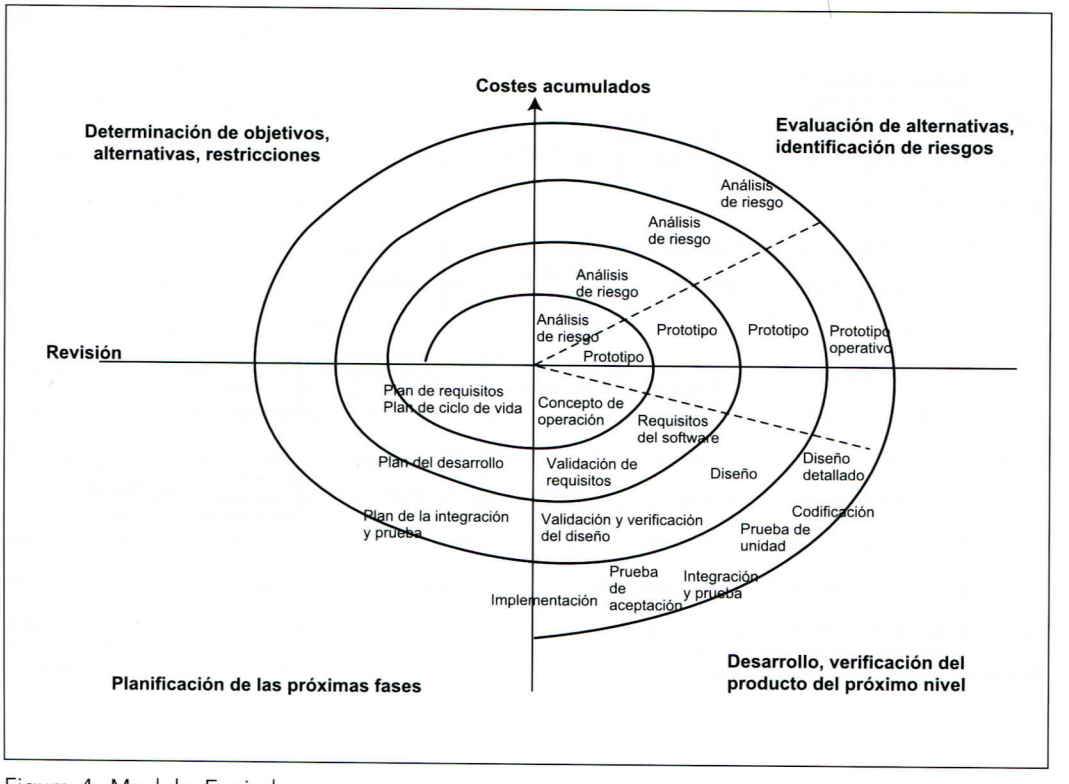

Figura 4: Modelo Espiral

que se adaptan a las características del proyecto. Para aquellos pequeños, el número de tareas de trabajo y su formalidad es bajo. Para proyectos mayores y más críticos, cada región de tareas contiene, a su vez, tareas de trabajo que se definen para lograr un nivel más alto de formalidad. Cuando empieza este proceso evolutivo, el equipo gira alrededor de la espiral en la dirección de las agujas del reloj, comenzando por el centro. El primer circuito de la espiral puede producir el desarrollo de una especificación de productos; los pasos siguientes en la espiral se podrían utilizar para desarrollar un prototipo y progresivamente versiones más sofisticadas del software. Cada paso por la región de planificación produce ajustes en el plan del proyecto. Los costos y la planificación se ajustan con la realimentación ante la evaluación del cliente. También se ajustan la cantidad de iteraciones requeridas para completar el software.

GRÁFICA DESCRIPTIVA:
La Figura 4 muestra las etapas del desarrollo de software como una espiral incremental, donde se van realizando mediciones o revisiones al final de cada vuelta de la espiral o de cada producto terminado.

OBSERVACIONES:
Puede resultar difícil convencer a los grandes clientes (sobre todo bajo licitaciones o grandes contratos) de que este enfoque es controlable. Requiere una considerable habilidad para la evaluación del riesgo (está explícita) y cuenta con esta habilidad para el éxito.

Iweb (Ingeniería Web)

Autor: Roger S. Pressman
Referencia de publicación:
[13]

RESUMEN:
Pressman establece el concepto de "Web enmarañada", referido a una Web-Internet que es muy susceptible a fallas, las cuales, además, se pueden propagar. Asimismo, plantea que para estar seguro de que la aplicación Web se ha construido adecuadamente, se deben emplear las mismas prácticas de aseguramiento de la calidad (SQA) que se usan en todos los procesos de la Ingeniería de Software: las revisiones técnicas formales valoran los modelos de análisis y diseño; las revisiones especializadas tienen en consideración la usabilidad, y la comprobación se

aplica para descubrir errores en el contenido, funcionalidad y compatibilidad del proyecto.

Roger Pressman define que las aplicaciones basadas en Web incluyen sitios Web completos, funcionalidad especializada dentro de los sitios Web y aplicaciones de proceso de información que residen en internet, en una intranet o en una extranet.

Iweb plantea técnicas para facilitar las especificaciones de la aplicación: el objetivo es identificar el problema, proponer elementos de solución, negociar diferentes enfoques y especificar un conjunto preliminar de requisitos de la solución en una atmósfera que permita alcanzar el objetivo.

La reunión se celebra en un lugar neutral y acuden tanto los clientes como los desarrolladores.

Se establecen normas de preparación y de participación.

Se sugiere una agenda lo suficientemente formal como para cubrir todos

GRÁFICA DESCRIPTIVA:

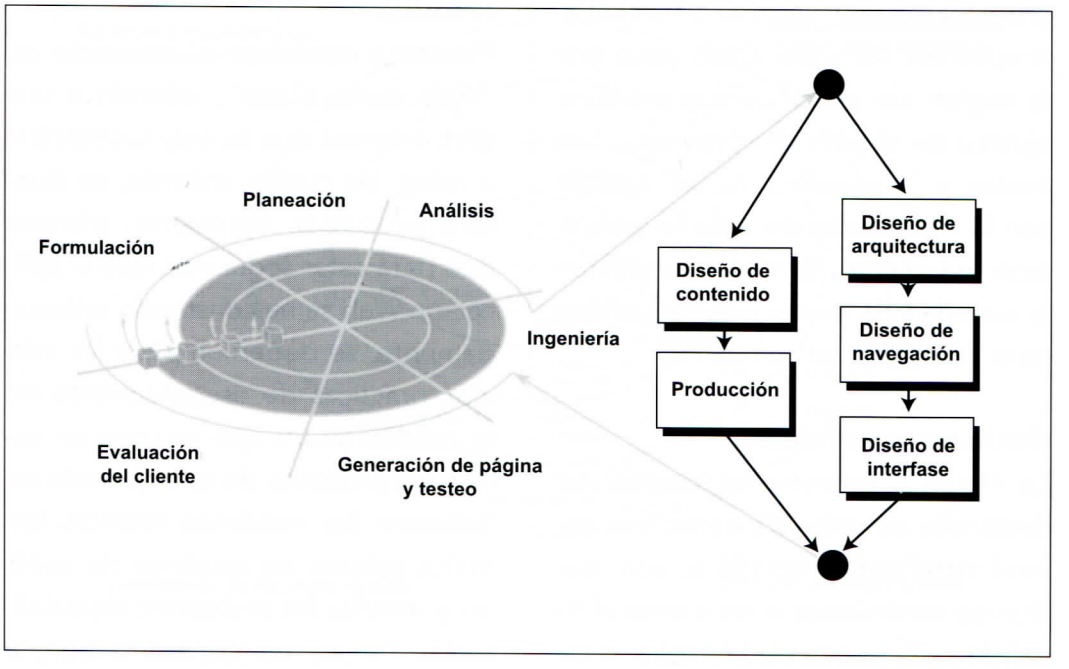

Figura 5: Modelo Iweb

los puntos importantes, pero lo suficientemente informal como para animar el libre flujo de ideas.

Debe nombrarse un coordinador (cliente o desarrollador) que controle la reunión.

Se usa un mecanismo de definición que puede ser hojas de trabajo, gráficos, carteles o pizarras.

Pressman plantea que los conceptos y principios de diseño se aplican a todas las aplicaciones Web. Cuando se crean aplicaciones basadas en Web se pueden reutilizar los métodos de diseño usados para los sistemas orientados a objetos. Los diagramas UML se pueden adaptar y utilizar durante la realización de las actividades de diseño de las aplicaciones Web.

Finalmente, Pressman señala que las aplicaciones Web son diferentes de otras categorías de software informático: son intensivas de red, están gobernadas por el contenido y están en continua evolución. La inmediatez que conduce a su desarrollo, la necesidad primordial de seguridad al funcionar, la demanda de estética y la entrega de contenido funcional son otros factores diferenciadores.

OBSERVACIONES:
La Ingeniería Web aplica un enfoque genérico que se suaviza con estrategias, tácticas y métodos especializados derivados de la Ingeniería de Software tradicional. Hace uso de un modelo de proceso iterativo e incremental porque la línea temporal del desarrollo para las aplicaciones Web es muy breve.

Programación Extrema (XP)

Autores: Kent Beck y Martin Fowler
Referencias de publicación:
[3], [4], [1], [9]

RESUMEN:
XP fundamenta su metodología de desarrollo en el problema planteado por Beck: "Todo el software cambia. Los requisitos cambian. El diseño cambia. La tecnología cambia. El equipo cambia. El negocio cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando este tiene lugar".

La metodología XP nos propone una serie de reducciones y técnicas que

persiguen un único fin: *la satisfacción del cliente*. Es una metodología que enfatiza el trabajo en grupo, un grupo formado por los jefes de proyecto, desarrolladores y clientes. En cada uno de los ciclos de desarrollo del proyecto hay diversas prácticas inherentes a la Programación Extrema. Esta metodología se basa en la implementación y entrega de pequeñas partes de programas. Hace énfasis en el perfeccionamiento constante del código, incluye al usuario en el equipo de desarrollo y plantea la programación en parejas, es decir, que dos programadores compartan un único computador, donde uno codifique y el otro reflexione acerca de la forma de implementar de la mejor manera el método, pensando, a su vez, en la estrategia y líneas adecuadas, posibles fallas, en cómo comprobar el código con los test y de qué modo es posible simplificar el sistema. Estos roles son intercambiables.

De lo anterior se desprende que XP parte de la base de que el código no es propietario de un programador, sino que es propiedad de todos, lo que permite que cualquiera pueda modificar una parte del mismo. Naturalmente, deben existir estándares de codificación, lo que posibilita que el código parezca construido por un único programador. Asimismo, en vez de diseñar todo el sistema al comienzo, plantea pequeños ciclos de desarrollo, lo que permite entregar la primera versión a producción en un par de meses, la cual va cambiando a sucesivas nuevas versiones de manera muy frecuente, desde un día a un mes. También plantea que el código a programar debe ser lo más simple posible, de forma que no gane ni pierda funcionalidad y su mantención sea más liviana. Cada pocas horas o al cabo de un día de programación, se inserta el sistema completo en una máquina de integración, separada de la de explotación. Si al integrar el sistema, aprueba todas los test y sigue funcionando correctamente, se dará por terminada esa tarea o ciclo y puede pasar a explotación. Si no funciona óptimamente tras un tiempo, los programadores deben botar el código agregado y partir de cero nuevamente.

Un pilar básico sobre el que se sustenta XP es que los test son automatizados. Para ello, XP plantea la construcción de un *framework* automático, como Junit [9] o cualquiera de sus versiones para diferentes lenguajes. La idea es preparar los test

antes del propio código que se desea probar, lo que posibilita una programación por intención, esto es, los códigos son escritos como si los métodos más costosos ya lo hubieran sido.

Lo novedoso de esta metodología es que plantea que el cliente debe estar en terreno todo el tiempo junto al equipo de desarrollo, para responder cualquier consulta que los programadores le planteen. Lo anterior ayuda a establecer prioridades, evitar ambigüedades y supuestos.

XP define cuatro variables para cualquier proyecto de software: costo, tiempo, calidad y alcance. Establece que tres de ellas pueden ser fijadas por el cliente. La variable que debe ser fijada por el equipo de desarrollo es el alcance, de manera que una vez fijadas las otras tres, el equipo de desarrollo podrá determinar el alcance mediante la estimación de las tareas a realizar para satisfacer los requisitos del cliente y la prioridad en la implementación de los requisitos, para dar la mayor funcionalidad posible al comienzo.

La metodología XP se puede complementar con un factor que plantea Alistar Cockburn y Jim Highsmith: las competencias individuales de cada programador, las cuales son consideradas un factor crítico para el éxito de su aplicación [3]. En otras palabras, el desarrollo rápido de un software depende de la calidad (competencias personales) del programador. Cockburn indica que las características de los programadores se resumen en cuatro: amigabilidad, talento, destreza y comunicación [3].

XP plantea tres enfoques que lo diferencian de las demás propuestas:

HISTORIAS DE USUARIO:

El enfoque de XP es que el usuario redacta sus propios requerimientos. Las historias de usuario servirán para crear el plan estimado de entrega. Las escriben los propios clientes, tal como ven ellos las necesidades del sistema. Por tanto, serán descripciones breves y expresadas en el lenguaje del usuario, sin terminología técnica.

Las historias de usuario, además, servirán para definir las pruebas de aceptación del proyecto.

El nivel de detalle de las historias de usuario debe ser el mínimo posible, lo suficiente como para permitir hacerse

una ligera idea de cuánto costará implementar el sistema. Cuando se llegue a la fase de implementación, los desarrolladores podrán acudir al cliente para ampliar detalles.

Por otra parte, los desarrolladores deberán hacer una estimación de cuánto tiempo, idealmente, les llevará implementar cada historia de usuario. Las condiciones ideales son aquellas en las que se codifica la historia de usuario sin otras distracciones y sabiendo exactamente qué es lo que hay que implementar. Como resultado debiéramos obtener un período ideal de una, dos o tres semanas. Más de tres semanas significa que debemos dividir la historia de usuario en partes. Menos de una semana significa que la historia de usuario es demasiado simple y tendremos que unir dos o más de ellas.

PROGRAMACIÓN EN PAREJAS:
Todo el código que vaya a incluirse en producción debe ser creado por dos personas que compartan el mismo puesto de trabajo. El desarrollo por parejas aumenta la calidad del producto sin perjudicar los tiempos de entrega. Parece contradictorio, pero dos personas trabajando juntas en el mismo computador son capaces de añadir la misma funcionalidad que ambas trabajando por separado y, además, logrando unas cotas de calidad mayores. El mejor método de desarrollo por parejas consiste en sentarse uno junto al otro, compartiendo computador, dejando que uno se encargue del desarrollo mientras el otro reflexiona sobre la mejor forma de afrontar los problemas. Esta actividad debería pasar de uno a otro periódicamente.

REUNIONES DE SEGUIMIENTO:
En una reunión típica de proyecto, la mayoría de los asistentes no aporta sino que se limita a escuchar. Una reunión de seguimiento tiene como propósito evitar esta situación y que la comunicación fluya entre todos los integrantes del equipo. La reunión de seguimiento de cada mañana debe llevarse a cabo para sacar a la luz los problemas y sus soluciones, así como centrar el objetivo del equipo. Es mejor congregar a una reunión a la que asistan todos los miembros del proyecto que ejecutar muchas reuniones con unos cuantos de los integrantes del equipo presentes.

La Figura 6 muestra el esquema de trabajo de XP. La planificación se lleva a cabo a lo largo de todo el proyecto.

Los ciclos de diseño, desarrollo y pruebas son breves e iterativos.

OBSERVACIONES:

XP es catalogado como una forma de trabajo ágil, que evita el que los programadores se distraigan en la documentación y en las tareas administrativas que demanda la ejecución de un proyecto. Estas tareas son realizadas en XP por el jefe de proyecto, quien debe velar para que los equipos de trabajo tengan la menor cantidad de distractores posibles. XP tiene muchos de sus fundamentos en las capacidades de las personas, por lo que es directamente dependiente de la afinidad, responsabilidad y conocimientos de los miembros del equipo, factores que, de no existir, podrían dificultar en algunos casos su aplicabilidad.

GRÁFICA DESCRIPTIVA:

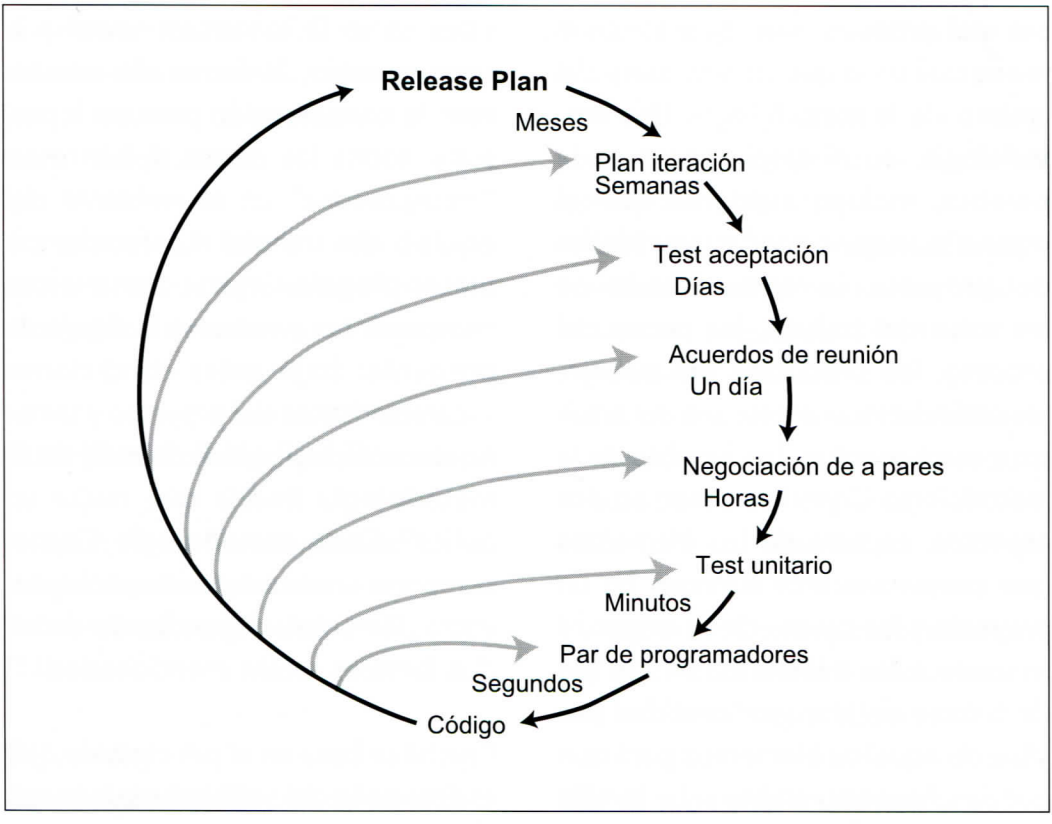

Figura 6: Modelo XP

Crystal (Clear)

Autor: Alistair Cockburn
Referencia de publicación:
[5]

Resumen:
Crystal es un método para desarrollar software, aplicable tanto a pequeños como a grandes proyectos, principalmente orientados al objeto. Crystal no solamente considera los aspectos de las notaciones, los diagramas, técnicas de diseño o pasos del proceso, sino que también revisa qué es lo que sucede fuera del ámbito de la metodología. Una metodología, en el amplio valor de la palabra, incluye aspectos que el equipo maneja, como las prioridades del proyecto, la representación de los roles del trabajo, los pasos del proceso, los productos del trabajo, los estándares, la estructura del equipo y otros asuntos. Las familias de la metodología Crystal se basan en dos aspectos: captura de los elementos que constantemente suceden en un proyecto y las causas de su origen, y traspaso a los miembros del equipo de trabajo de la mayor cantidad posible de aquellos elementos para que puedan hacer su trabajo. La familia Crystal es un esfuerzo para aunar los componentes prácticos del desarrollo de proyectos con los teóricos, vale decir, la consideración de cómo los humanos piensan y comunican los aspectos relevantes del desarrollo de software a través de una metodología. Esta plantea que un proyecto realizado por seis personas se debe tratar en forma diferente que un proyecto que abarca a 45 personas, aunque esto parezca obvio para la gente que desarrolla software. Un aspecto importante de la metodología es que revisa algunos elementos cognitivos, tales como la estructura social y la comunicación, la forma de interactuar, la comunicación persona a persona; todos los cuales deben estar "incorporados" en el ambiente del equipo de trabajo. La familia de metodologías Crystal toma estos componentes y resuelve la siguiente pregunta: bajo estas condiciones —características del proyecto y tamaño del equipo—, ¿cual debería ser la metodología liviana que mejor se aplica? Cada metodología Crystal comporta unas características de proyecto diferentes, dependiendo de los dos factores recién mencionados.

Crystal se basa en el principio de que el desarrollo del software es un juego cooperativo, en el cual los participan-

tes se ayudan entre sí para juntos lograr finalizar el juego: la entrega del software.

OBSERVACIONES:
El autor de la metodología Crystal describe la propia metodología por medio de historias. Básicamente, coloca un escenario donde el autor interactúa con una persona ficticia, que es la propia metodología Crystal. En este diálogo se compara la metodología con otras y se profundizan los enfoques prácticos de Crystal, donde uno de los objetivos es tratar de disminuir la brecha de aprendizaje del equipo, dependiendo del tamaño del proyecto.

Al momento de la impresión de este artículo, no se encontraba disponible una gráfica descriptiva de Crystal.

RUP (Rational Unified Process)

Autores: James Rumbaugh, Ivar Jacobson y Grady Booch
Referencia de publicación:
[14]

RESUMEN:
Rational Unified Process es un proceso de desarrollo de software basado en incrementos. El desarrollo a lo largo del tiempo se funda en iteraciones. Cada iteración se compone de cuatro fases: concepción, elaboración, construcción y transición. A su vez, en cada iteración se van completando y revisando las diferentes disciplinas o flujos de trabajo, tales como modelamiento del negocio, requerimientos, análisis y diseño, entre otros. RUP divide el proceso de desarrollo en ciclos, teniendo un producto al final de cada ciclo. Cada fase concluye con un hito bien definido donde deben tomarse ciertas decisiones.

Las seis mejores prácticas sobre las que se basa RUP son:
- Desarrollo iterativo del software,
- Administración de requerimientos,
- Uso de arquitecturas basadas en componentes,
- Modelamiento visual del software,
- Verificación de la calidad del software,
- Control de cambios.

Estas prácticas son operacionalizadas con RUP, obteniendo: roles, disciplinas, actividades y artefactos.

RUP puede ser implementado en pequeños y grandes proyectos, otor-

gando para ello cierta flexibilidad. Por ejemplo, si un proyecto en particular demanda el uso de un mapa o ruta de trabajo diferente a la provista por RUP, simplemente es posible producir o elaborar un nuevo mapa de trabajo acorde con el proyecto.
Las fases de RUP son:

Inicio o concepción. Esta etapa es importante, porque es aquí donde se definen los lineamientos del negocio antes de comenzar el proyecto. Durante esta etapa se establecen los casos del negocio para construir el software. La visión es un artefacto clave que se debe tener en cuenta aquí, como también es importante contar con una descripción de alto nivel del proyecto.

Elaboración: El objetivo de la fase de elaboración es definir una arquitectura base del sistema, que servirá de fundamento para las actividades de diseño e implementación en la etapa de construcción. En RUP, las actividades de diseño en esta etapa se centran en la arquitectura del sistema y, para sistemas intensivos en software,

GRÁFICA DESCRIPTIVA:

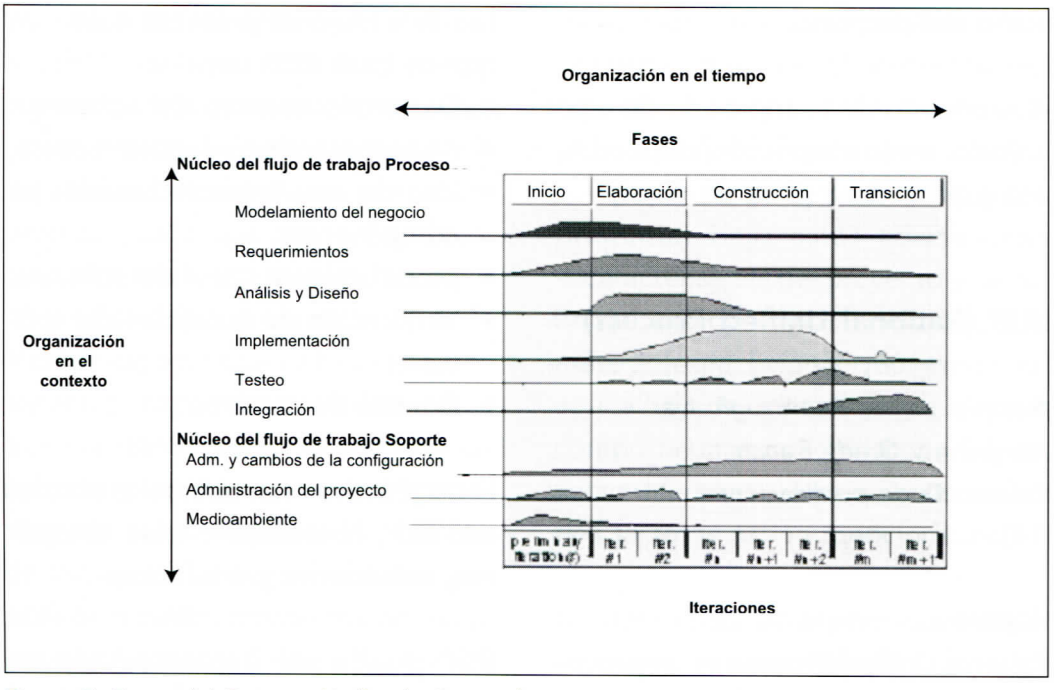

Figura 7: Fases del Proceso Unificado Rational

en la arquitectura del software. Se usa la arquitectura de componentes, que es una de las seis mejores prácticas del desarrollo de software adoptadas por RUP.

Construcción: El objetivo de la fase de construcción es completar el desarrollo del sistema. La fase de construcción es en algunos casos un proceso de manufactura, donde se da énfasis a la administración de recursos y al control de operaciones que optimizan los costos y la calidad. En este caso (RUP), la administración es un paso intermedio para la transición desde el desarrollo de la propiedad intelectual durante la concepción y elaboración, al desarrollo de un producto durante la construcción y transición.

Transición: El objetivo de esta fase es asegurarse de que el software esté disponible para los usuarios finales. Esta fase incluye el testeo del producto que será liberado, haciendo los ajustes necesarios para que el usuario lo pueda utilizar.

OBSERVACIONES:

RUP sigue un modelo iterativo que aborda las tareas más riesgosas del proyecto primero. Así se logra reducir los riesgos y tener un subsistema ejecutable tempranamente. RUP apoya el desarrollo basado en componentes, tanto nuevos como preexistentes. UML es la base del modelamiento visual de RUP.

RUP ayuda a planificar, diseñar, implementar, ejecutar y evaluar pruebas que verifiquen estas cualidades.

RMM (Relationship Management Methodology)

Autores: T. Isakowitz, E.A. Stohr y P. Balasubramanian.
Referencia de publicación:
[8]

RESUMEN:

La Metodología de Administración de las Relaciones (RMM) está concebida para ser aplicada para diseñar y construir aplicaciones hypermediales que tienen estructuras regulares o estables con información altamente cambiante. Se centra en las fases de diseño, desarrollo y construcción cíclica en el desarrollo de software. El nombre Relationship Management se fundamenta en que la hypermedia es un vehículo para administrar y relacionar diferentes objetos de información.

Algunos ejemplos de aplicaciones que se pueden desarrollar con esta metodología son los catálogos de productos, *front-end* de bases de datos tradicionales o aplicaciones legadas. Muchas de estas aplicaciones poseen contenido altamente volátil que requiere frecuentemente una actualización; en estos casos, es necesario automatizar los procesos implicados según sea necesario.

Esta metodología no es aplicable para aquellas situaciones en que las estructuras no están claras y el contenido no cambia frecuentemente.

Un modelo de datos es una agrupación de objetos lógicos usados para mostrar una abstracción del mundo real. Los modelos de datos son necesarios para expresar el diseño de una aplicación. En este contexto, hay que diferenciar los sistemas de hipertexto de las aplicaciones con hipertexto. Un modelo de datos para un sistema hipertexto detalla la arquitectura interna, pero es una pequeña parte del modelamiento de aplicaciones hypermedia.

RMM provee el Relationship Management Data Model (RMDM). RMDM provee un lenguaje para describir objetos de información y mecanismos de navegación en aplicaciones hypermedia. RMDM es la piedra angular de RMM. Las entidades son un gran número de atributos. Los *slices* son subconjuntos de ellos.

La navegación es soportada por seis tipos de accesos:

- Links unidireccionales
- Links bidireccionales
- Agrupamiento
- Índices condicionales
- Conditional Guided Tour
- Conditional Indexed Guided Tour

GRÁFICA DESCRIPTIVA:
La figura siguiente muestra el método RMM. En esta figura se aprecia que el foco del método es el diseño de entidades, apoyado por RMDM.

OBSERVACIONES:
RMM es una metodología que tiene sus fundamentos en el modelamiento de datos relacional y principalmente tiene su foco en el desarrollo de aplicaciones de pequeña envergadura, con mucho énfasis en la interfaz.

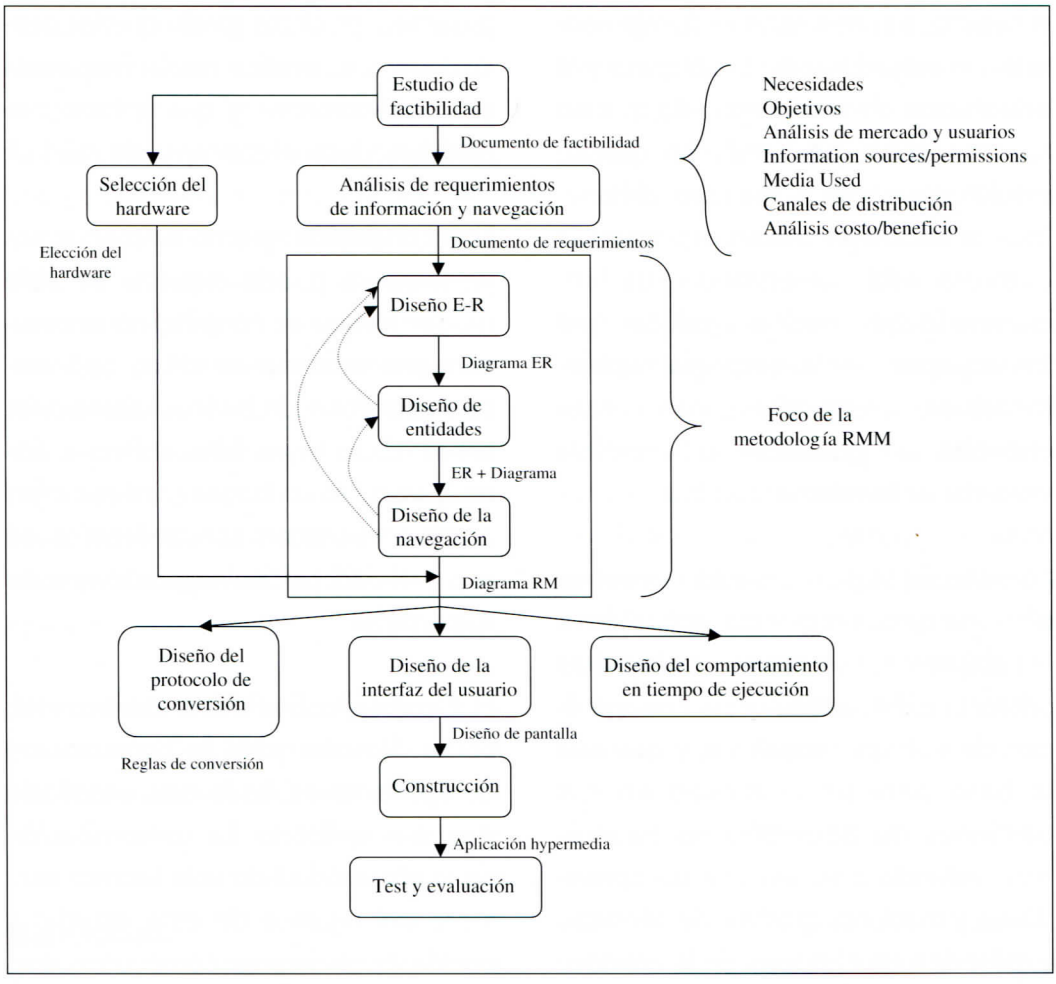

Figura 8: Método RMM

Conclusiones

Las propuestas de desarrollo de aplicaciones que están en la literatura del área, sugieren formas de abordar los desarrollos en una forma muy genérica y amplia, en muchos casos bajo una realidad que no es aplicable directamente en las empresas. Esto significa que los desarrolladores deben adaptar estas propuestas según las necesidades de cada organización. Muchas de estas adaptaciones pueden llevar al fracaso del

proyecto, por dos razones fundamentales: la inmadurez de la empresa y la adaptación de una propuesta que no fue concebida pensando en que se podía adaptar a la realidad chilena. Esta situación se acentúa cuando la empresa está comenzando sus funciones, lo que podría significar que los equipos en la empresa rápidamente se desmoralicen, con consecuencias tan graves en la organización como la misma quiebra.

Lo anterior sugiere que es necesario elaborar una propuesta aplicable a las empresas, con énfasis en aquellas en formación, inmaduras, con equipos de trabajo inexpertos, y que sea la base para un comienzo en sus gestiones de desarrollo no traumático, velando a su vez por un aprendizaje y madurez gradual de técnicas y métodos en el marco de la calidad.

Para que esta propuesta efectivamente ayude a las empresas en formación, debe utilizar las mejores prácticas y sugerencias de las propuestas revisadas. Además, deberá considerar las actividades más comunes que se realizan en la construcción de una aplicación. Estas actividades deberán estar basadas en algún estándar simple, pues este garantiza en cierto grado que las actividades concebidas de la propuesta sean las correctas y que a futuro se podrá madurar el concepto de calidad.

Una conclusión que no es menos importante se puede expresar de este modo: lo que es correcto no necesariamente es lo que se utiliza. La literatura está llena de buenas intenciones sobre cómo hacer bien software. Sin embargo, esas buenas intenciones muchas veces no son aplicables en una realidad particular, por lo que no son usadas.

El Desarrollo Evolutivo Incremental, como filosofía para la construcción de aplicaciones, es lo más enseñado y lo más aplicado. La comprobación de la efectividad de esta técnica está fuera del alcance de este estudio y podría considerarse como un nuevo proyecto que revise la efectividad de esta técnica para proyectos y/o empresas muy grandes o de otras áreas de desarrollo de la Ingeniería de Software. Sin embargo, los antecedentes de las encuestas indican que es la línea correcta.

Lo que necesita una empresa que está comenzando en la actividad informática, es una pequeña guía de

cómo hacer las cosas bien, sin burocracia, motivadoramente y con resultados a corto plazo. Los estándares reconocidos en el mercado proveen una guía de las actividades que deben realizarse a lo largo del ciclo de vida de un software. Sin embargo, estas actividades en algunos casos no son ejecutadas porque no se ajustan a la realidad del problema o debido a la inexperiencia de los equipos de trabajo. Los estándares no indican cómo llevar a cabo las actividades, sino más bien señalan qué actividades hay que ejecutar.

Para aplicar bien un estándar hay que entenderlo correctamente. Uno de los puntos claves de esta comprensión es la madurez de los equipos de trabajo y las empresas. Las empresas que están comenzando son inmaduras, por lo que aplicar ciertos estándares inapropiados puede llevar al fracaso del proyecto.

En relación a la forma de abordar un nuevo proyecto, es necesario contar en sus inicios con una adecuada claridad de los requisitos, pues de esta forma es posible dimensionar los esfuerzos para su construcción, sean estos recursos humanos y económicos, como también el método de desarrollo a emplear. Esto es aplicable tanto a proyectos Web como a proyectos no Web.

Bibliografía

[1] Beck, K. y M. Fowler. *Planning eXtreme Programming*. Boston: Addison-Wesley, 2001.

[2] Boehm, B. "A Spiral Model for Software Development and Enchancement". *Computer* Vol. 21 N° 5 (1988): 61-72.

[3] Cockburn, A. and J. Highsmith. "Agile Software Development: The People Factor". *IEEE Computer* (November, 2001): 131-33.

[4] Cockburn, A. "Agile Software Development Joins the 'Would-Be' Crowd". *Cutter IT Journal* Vol. 15 N° 1 (January, 2002): 6-12.

[5] ———. "Crystal 'Clear': A Human-Powered Software Development Methodology for Small Teams" [en línea] <http://members.aol.com/humansandt/crystal/clear/> [Consulta: 10 de diciembre de 2002].

[6] Cutter Consortium. "Access to the Experts" [en línea] <http://www.cutter.com/index.shtml> [Consulta: 10 de diciembre de 2002].

[7] Ginige, A. and S. Murugesan. "Web Engineering: An Introduction". *IEEE Multimedia* Vol. 8 N° 1 (2001): 14-18.

[8] Isakowitz, T. et al. "RMM: a Methodology for Structured Hypermedia Design". *Communications of the ACM* Vol. 38 N° 8 (August, 1995): 26-30.

[9] http://www.junit.org/news/article/index.htm [en línea] [Consulta: 17 de enero de 2003].

[10] Martin, J. *Rapid Application Development*. New York: Macmillan, 1991.

[11] Miller, R.W. "Demystifying Extreme Programming: 'XP Distilled' Revisited", Part 1 [en línea] <http://www-106.ibm.com/developerworks/java/library/j-xp0813/?loc=j> [Consulta: 10 de diciembre de 2002].

[12] Mills, H.D. and D. O'Neill. "The Management of Software Engineering". *IBM Systems Journal* Vol. 38 Nos. 2-3 (1999).

[13] Pressman, R.E. *Ingeniería de Software*. 5ta. ed. Madrid: McGraw Hill, 2002.

[14] Rumbaugh, James; Ivar Jacobson y Grady Boach. *El proceso unificado de desarrollo de software*. México DF: Addison Wesley, 2000.

[15] Royce, W. "Managing the Development of Large Software Systems: Concepts and Techniques". WESCON, Western Electronic Show and Convention. Agosto de 1970.

[16] Sommerville, Ian. *Ingeniería de software*. 6ta. ed. México DF: Addison Wesley, 2002.