Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería del Software Educativo

The Methodologies of Agile Development like an Opportunity for the Engineering of Educative Software

Ailin Orjuela Duarte, MSc., Mauricio Rojas C., MSc.
Grupo de investigación CICOM, Universidad de Pamplona, Colombia
aorjuela@unipamplona.edu.co, mrojas@unipamplona.edu.co

Recibido para revisión 3 de Abril de 2008, Aceptado 19 de Mayo de 2008, Versión final 24 de Mayo de 2008

Resumen—La ingeniería del software educativo se viene convirtiendo en un área de estudio con altos niveles de aplicación debido a que cada vez los procesos de desarrollo de sistemas de software educativo se llevan a cabo empleando metodologías de la Ingeniería del software.
Las aplicaciones de software educativo desarrolladas en épocas pasadas en la gran mayoría de los casos surgían de esfuerzos de docentes con algunos conocimientos en informática o en su defecto de ingenieros con algunos intereses en el sector educativo.
Otro factor importante a analizar es que las metodologías ofrecidas por la ingeniería del software educativo son demasiado pesadas, entonces debido a este argumento en el presente documento se estudian los enfoques tradicionales, los enfoques modernos y las metodologías ágiles para ofrecer una propuesta metodológica menos pesada para la ingeniería del software educativo.

Palabras clave: Ingeniería del Software, Metodología Ágil, Modelo Pedagógico, Diseño Educativo, Diseño Computacional.

Abstract —Educational software engineering is becoming a research field with high levels of application due to the fact that the development processes of educational computer software are increasingly applying software engineering methodologies.
Most previous educational software applications arose from either the efforts of teachers with some knowledge of computing or systems engineers with interests in the educational sector.
Another important issue which needs to be analyzed is the fact that, in the past, educational software engineering methodologies were too heavy. Due to this argument, in this work both traditional and modern methodologies as well as fast and effective methodologies will be studied in order to put forward a more manageable methodological proposal for educational software engineering.

Keywords: Software Engineering, Fast Methodology, Pedagogical Model, Educational Design, Computer Design.

I. INTRODUCCIÓN

Este trabajo tiene como prospectiva dos objetivos, el primero de ellos pretende brindar una descripción del marco teórico de referencia de las metodologías de desarrollo ágil presentando algunas como: XP, CRYSTAL, SCRUM. El segundo objetivo busca analizar algunas características esenciales de estas metodologías para adaptarlas al contexto de la ingeniería del software educativo.

En la comunidad de la ingeniería del software, se está viviendo con intensidad un debate abierto entre los partidarios de las metodologías tradicionales (referidas como "metodologías pesadas") y aquellos que apoyan las ideas emanadas del "Manifiesto Ágil" [1].

Por un lado, para muchos equipos de desarrollo el uso de metodologías tradicionales les resulta muy lejano a su forma de trabajo actual; considerando las dificultades de su introducción e inversión asociada en términos de formación y compra de herramientas. Por otro, las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, basados en nuevas tecnologías.

Estas metodologías están especialmente orientadas para proyectos que necesitan de una solución a la medida, con una elevada simplificación sin dejar de lado el aseguramiento en la calidad del producto. Las metodologías ágiles se centran en el factor humano y el producto software; es decir, ellas le dan mayor valor al individuo, a la colaboración del cliente y al desarrollo incremental del software con iteraciones muy cortas.

En cuanto a los objetivos, la investigación documental se orientó hacia la identificación de metodologías de diseño, que contienen los métodos, las herramientas y los procedimientos específicos para la construcción de software. Después de

Revista Avances en Sistemas e Informática, Vol.5 No.2, Junio de 2008, Medellín, ISSN 1657-7663

esta fase se articulan las características de las metodologías ágiles para proponer algunas orientaciones para el proceso de desarrollo de software educativo.

El resto del presente documento está organizado como sigue. En la sección 2 se introducen las principales características de las metodologías ágiles, recogidas en el Manifiesto y se hace una comparación con las tradicionales. La sección 3 se centra en eXtreme Programming (XP), presentando sus características particulares, el proceso que se sigue y las prácticas que propone y se citan otras metodologías ágiles, enumerándose sus principales características. La sección 4 presenta algunas características de las metodologías de desarrollo ágil aplicadas al proceso de desarrollo de software educativo. Finalmente aparecen las conclusiones y referencias.

II. METODOLOGÍAS DE DESARROLLO ÁGIL

En este segmento del artículo se presentan los aspectos más relevantes de las metodologías de desarrollo ágil, entre los apartes que se describen están los antecedentes, el manifiesto ágil, comparación entre metodologías ágiles vs. Tradicionales.

A. Antecedentes

En febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace el término "ágil" aplicado al desarrollo de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto [3].

Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades.

Tras esta reunión se creó The Agile Alliance [8], una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía "ágil" [5].

B. Manifiesto ágil

En marzo de 2001 diecisiete críticos de los modelos de mejora del desarrollo de software basados en procesos, convocados por Kent Beck, quien había publicado un par de años antes Extreme Programming Explained, libro en el que exponía una nueva metodología denominada Extreme Programming, se reunieron en Salt Lake City para tratar sobre técnicas y procesos para desarrollar software. En la reunión se acuñó el término "Métodos Ágiles" para definir a los métodos que estaban surgiendo como alternativa a las metodologías formales (CMMI, SPICE) a las que consideraban excesivamente "pesadas" y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas previas al desarrollo.

Los integrantes de la reunión resumieron los principios sobre los que se basan los métodos alternativos en cuatro postulados, lo que ha quedado denominado como Manifiesto Ágil [5].

El manifiesto ágil está fundamentado en los siguientes valores:

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

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

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

- Responder a los cambios más que seguir estrictamente un plan. La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (en los requisitos, en la tecnología, en el equipo) es otro factor que determina el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta[5].

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

Los principios del manifiesto ágil son:

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

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

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

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

5. 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.

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

desarrollo.

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

8. Los procesos ágiles promueven un desarrollo sostenible.

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

10. La simplicidad es esencial.

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

12. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento [5].

C. Comparación entre metodologías ágiles y metodologías tradicionales

En la tabla 1 se muestran las principales diferencias de las metodologías ágiles con respecto a las tradicionales. Estas diferencias hacen referencia de igual manera a aspectos organizacionales y al proceso de desarrollo [5].

Tabla 1. Comparación de las metodologías tradicionales y las ágiles

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.
Preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Reglas de trabajo impuestas internamente (por el equipo).	Reglas de trabajo impuestas externamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
Flexibilidad en los contratos debido a la respuesta a cambios.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones en determinadas etapas del proceso.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos trabajando en diferentes tareas.
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.

De acuerdo a la tabla 1 se puede observar que las metodologías de desarrollo ágil son más orientadas a procesos de desarrollo de software de pocas semanas de desarrollo y con bajos niveles de formalización en la documentación requerida.

A continuación se describen con mayor nivel de especificación las principales semejanzas y diferencias de las metodologías ágiles y las metodologías pesadas:

Prácticas de desarrollo: En las metodologías de desarrollo ágil se procura realizar los procesos de software de acuerdo a las prácticas que le han dado resultados al grupo. En las metodologías pesadas se desarrolla de acuerdo a las normas sugeridas por los estándares de desarrollo.

Adaptación al cambio: En las metodologías ágiles por la misma capacidad de reacción son más adaptables a los cambios, por el contrario en las metodologías pesadas por el nivel de formalidad en la fase de requerimientos son más resistentes al cambio.

Control: Por su capacidad de adaptación el proceso se hace menos controlado que en las metodologías tradicionales que ejercen mayor control en el proceso por su nivel de formalización.

Documentación: En las metodologías ágiles no se hace énfasis en la documentación ni en los artefactos a diferencia de las metodologías pesadas.

Equipo de trabajo: En las metodologías ágiles existen bajo número de participantes y roles, por el contrario en las metodologías pesadas se sugieren los roles que proporcionan las normas.

En conclusión de acuerdo a la tabla se da prioridad: a los individuos y las interacciones más que a los procesos y a las herramientas, a los sistemas funcionando antes que a la documentación detallada, a la colaboración con el cliente antes que la negociación de contratos, a la respuesta al cambio antes que seguir el plan.

Revista Avances en Sistemas e Informática, Vol.5 No.2, Junio de 2008, Medellín, ISSN 1657-7663

III. PRINCIPALES METODOLOGÍAS ÁGILES

Las metodologías ágiles resuelven los problemas surgidos, posteriormente, a la masificación del uso del computador personal, dado que las expectativas y necesidades por parte de los usuarios se hicieron más urgentes y frecuentes. Fue así como a comienzo de los 90 surgieron propuestas metodológicas para lograr resultados más rápidos en el desarrollo de software sin disminuir su calidad.

En este segmento del artículo se describen las características esenciales de las metodologías ágiles más utilizadas como son XP, CRYSTAL y SCRUM.

A. Programación extrema (extreme programming, xp)

Antecedentes

En épocas anteriores en las cuales sólo había pantallas de texto, no había entornos de ventanas, las aplicaciones eran más sencillas que las actuales. Era suficiente uno o dos programadores, que de acuerdo a sus conocimientos y en un plazo razonable de tiempo eran capaces de hacer programas útiles. El programador era una especie de "mago" que hacía su aplicación y sólo él sabía cómo funcionaba. Si acaso, después de hacer el programa hacía un "flujograma" de los antiguos, no había UML, orientación a objetos ni cosas por el estilo.

Con el paso de los años las aplicaciones fueron cada vez más exigentes, los computadores disponían de más memoria y aparecieron los entornos de ventanas.

Lo anterior originó que los programas aumentaran su tamaño y se complicaran enormemente, requiriendo varios desarrolladores y tiempos más largos. Todo esto llevó consigo la necesidad de más organización y documentación. En este contexto, aparecieron las metodologías de desarrollo pesadas (las que se basan en escribir mucha documentación). Antes de hacer un programa, fue necesario escribir que se quería hacer, de forma que se aseguraba un acuerdo entre el cliente y los desarrolladores acerca de lo que se quería hacer.

Luego se escribía lo qué se iba a hacer, para que los programadores trabajaran con un objetivo común y organizado.

De alguna forma, toda esta organización y documentación quitó parte del encanto de la programación. Los programadores, en vez de dedicarse a lo que se supone que les gusta: programar; debían dedicar gran parte de su tiempo a hacer y leer documentos, reuniones, entre otros.

Tratando un poco de recuperar los viejos tiempos, en el que había menos papeleo, las cosas eran más sencillas y los programadores hacían lo que les gustaba, nació una nueva metodología de desarrollo, la programación extrema [2].

Definición

XP 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 programadores, 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 [2].

Características

A continuación se describen las características esenciales de XP organizadas en los apartados siguientes: historias de usuario, roles, proceso y prácticas.

Las Historias de Usuario

Corresponden a la técnica utilizada para especificar los requisitos del software. Se trata de formatos en los cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [2].

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 [2].

Una historia de usuario en forma general puede contener los siguientes ítems, los cuales pueden variar de acuerdo al equipo de desarrollo. Estos aspectos pueden ser: 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.

Roles XP

La propuesta original de Beck incluye los siguientes roles [2]:

- 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.

- **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.

Proceso XP

El ciclo de desarrollo consiste en los siguientes pasos:

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

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

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

4. El programador construye ese valor de negocio.

5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos.

De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

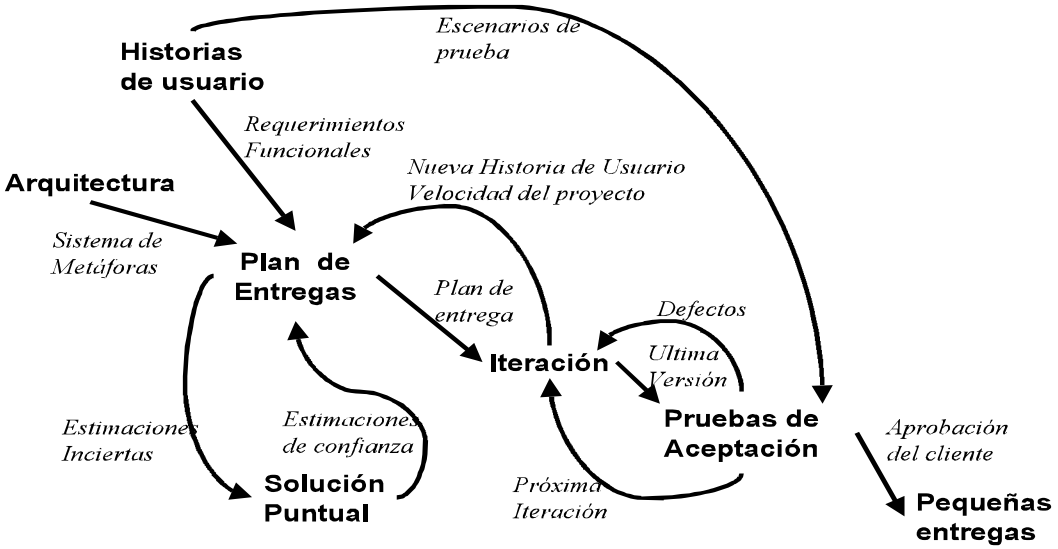


Figura 1. Proceso XP

El ciclo de vida ideal de XP consiste de seis fases [2]: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

Prácticas XP

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas [2]:

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

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

- **Metáfora**. El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).

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

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

Revista Avances en Sistemas e Informática, Vol.5 No.2, Junio de 2008, Medellín, ISSN 1657-7663

- **Refactorización (Refactoring)**. Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo [13].

- **Programación en parejas**. Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores).

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

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

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

- **Cliente in-situ**. El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.

- Estándares de programación. XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos esquemas de programación para mantener el código legible.

El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada puesto que se apoyan unas en otras. La mayoría de las prácticas propuestas por XP no son novedosas sino que en alguna forma ya habían sido propuestas en ingeniería del software e incluso demostrado su valor en la práctica. El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

B. Crystal

En este segmento del artículo se describen los aspectos principales de la metodología Crystal. En primera instancia se especifican los antecedentes de la metodología, continuando con definiciones que ayudan a estructurar la fundamentación teórica y se termina con las características esenciales de los diferentes tipos de Crystal.

Antecedentes

En los inicios de 1990, en un estudio realizado en IBM se llegó a los siguientes acuerdos (Cockburn, 2001):

– Los equipos exitosos enfatizaban que no habían seguido métodos formales ni herramientas CASE y que habían estimulado la comunicación y los test.

– Los equipos con problemas no entendían sus fallas o si habían cumplido con los métodos formales.

La conclusión: Menos énfasis en la documentación exhaustiva y más en versiones que corran y puedan ser probadas. Lo primero son promesas, lo segundo hechos. Cada proyecto necesita sus propios métodos.

Alistair Cockburn en lugar de partir solamente de su experiencia personal para construir una teoría de cómo deben hacerse las cosas, complementa su experiencia directa con la búsqueda activa de proyectos para ver cómo trabajan.

Él ha explorado a fondo los métodos ágiles, haciendo énfasis en la familia de metodologías Crystal. Es una familia porque cree que los diferentes tipos de proyectos requieren diferentes tipos de metodologías. Él mira esta variación a lo largo de dos ejes: el número de personas en el proyecto, y las consecuencias de los errores. Cada metodología encaja en una parte diferente, de modo que para un proyecto de 40 personas que puede perder dinero discrecionalmente tiene una metodología diferente a la de un proyecto vital de seis personas [8].

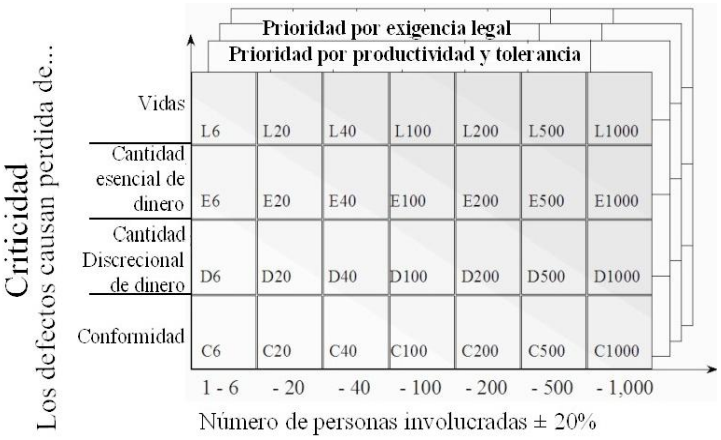


Figura 2. Clasificación de las diferentes metodologías Cristal [8]

La familia de metodologías Crystal comparten con la XP una orientación humana, pero esta centralización en la gente se hace de una manera diferente. Alistair considera que las personas encuentran difícil seguir un proceso disciplinado, así que más que seguir la alta disciplina de la XP, Alistair explora la metodología menos disciplinada que aun podría tener éxito, intercambiando conscientemente productividad por facilidad de ejecución. Él considera que aunque Crystal es menos productivo que la XP, más personas serán capaces de seguirlo.

Alistair también pone mucho peso en las revisiones al final de la iteración, animando al proceso a aplicar técnicas de mejoramiento continuo en forma automática. Su aserción es que el desarrollo iterativo está para encontrar los problemas

temprano, y entonces permitir corregirlos. Esto pone más énfasis en la gente supervisando su proceso y afinándolo conforme desarrollan [8].

Definiciones

Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros) [9].

Características

Las personas, como dispositivos activos, tienen modos de éxito y modos de fallo. Los siguientes son los principales [10]:

- Cuando el número de personas aumenta, también aumenta la necesidad de coordinar.

- Cuando el potencial de daños se incrementa, la tolerancia a variaciones se ve afectada.

- La sensibilidad del tiempo en que se debe estar en el mercado varía: a veces este tiempo debe acortarse al máximo y se toleran defectos, otras se enfatiza la auditoria, confiabilidad, protección legal, entre otros.

- Las personas se comunican mejor cara a cara, con la pregunta y la respuesta en el mismo espacio de tiempo.

- El factor más significativo es “comunicación”.

Los métodos se llaman Crystal evocando las facetas de una gema: cada faceta es otra versión del proceso, y todas se sitúan en torno a un núcleo idéntico. Hay cuatro variantes de metodologías: Crystal Clear para equipos de 8 o menos integrantes; Amarillo, para 8 a 20; Naranja, para 20 a 50; Rojo, para 50 a 100. La más exhaustivamente documentada es Crystal Clear (CC), y es la que se ha de describir a continuación. CC puede ser usado en proyectos pequeños. El otro método elaborado en profundidad es el Naranja, apto para proyectos de duración estimada en 2 años. Los otros dos aún se están desarrollando. Como casi todos los otros métodos, CC consiste en valores, técnicas y procesos. Los siete valores o propiedades de CC son:

1. Entrega frecuente. Consiste en entregar software a los clientes con frecuencia, no solamente en compilar el código. La frecuencia dependerá del proyecto, pero puede ser diaria, semanal o mensual.

2. Comunicación osmótica. Todos juntos en el mismo cuarto. Una variante especial es disponer en la sala de un experto diseñador senior y discutir respecto del tema que se trate.

3. Mejora reflexiva. Tomarse un pequeño tiempo (unas pocas horas cada o una vez al mes) para pensar bien qué se está haciendo, cotejar notas, reflexionar, discutir.

4. Seguridad personal. Hablar con los compañeros cuando algo molesta dentro del grupo.

5. Foco. Saber lo que se está haciendo y tener la tranquilidad y el tiempo para hacerlo.

6. Fácil acceso a usuarios expertos. Tener alguna comunicación con expertos desarrolladores.

Crystal Clear no requiere ninguna estrategia o técnica, pero siempre es útil tener unas cuantas a mano para empezar. Las estrategias comunes a otras Metodologías Ágiles, son:

1. Exploración de 360°. Verificar o tomar una muestra del valor de negocios del proyecto, los requerimientos, el modelo de dominio, la tecnología, el plan del proyecto y el proceso.

2. Victoria temprana. Es mejor buscar pequeños triunfos iniciales que aspirar a una gran victoria tardía.

3. Esqueleto ambulante. Es una transacción que debe ser simple pero completa.

4. Rearquitectura incremental. Se ha demostrado que no es conveniente interrumpir el desarrollo para corregir la arquitectura. Más bien la arquitectura debe evolucionar en etapas, manteniendo el sistema en ejecución mientras ella se modifica.

5. Radiadores de información. Es una lámina pegada en algún lugar que el equipo pueda observar mientras trabaja o camina. Tiene que ser comprensible para el observador casual, entendida de un vistazo y renovada periódicamente para que valga la pena visitarla.

En cuanto a las técnicas, se favorecen:

1. Entrevistas de proyectos. Se suele entrevistar a más de un responsable para tener visiones más ricas.

2. Talleres de reflexión. El equipo debe detenerse treinta minutos o una hora para reflexionar sobre sus convenciones de trabajo, discutir inconvenientes y mejoras y planear para el período siguiente.

3. Planeamiento Blitz. Una técnica puede ser el Juego de Planeamiento de XP. En este juego, se ponen tarjetas indexadas en una mesa, con una historia de usuario o función visible en cada una. El grupo finge que no hay dependencias entre tarjetas, y las alinea en secuencias de desarrollo preferidas. Los programadores escriben en cada tarjeta el tiempo estimado para desarrollar cada función. El patrocinador del usuario escribe la secuencia de prioridades, teniendo en cuenta los tiempos referidos y el valor de negocio de cada función. Las tarjetas se agrupan en períodos de tres semanas llamados iteraciones que se agrupan en entregas, usualmente no más largas de tres meses.

4. Estimación Delphi con estimaciones de pericia. En el proceso Delphi se reúnen los expertos responsables y proceden como en un remate para proponer el tamaño del sistema, su

Revista Avances en Sistemas e Informática, Vol.5 No.2, Junio de 2008, Medellín, ISSN 1657-7663

tiempo de ejecución, la fecha de las entregas según dependencias técnicas y de negocios y para equilibrar las entregas en paquetes de igual tamaño.

5. Encuentros diarios de pie. La palabra clave es "brevedad", cinco a diez minutos como máximo. No se trata de discutir problemas, sino de identificarlos.

6. Miniatura de procesos. Una forma de presentar Crystal Clear puede consumir entre 90 minutos y un día. La idea es que la gente pueda "degustar" la nueva metodología.

7. Gráficos de quemado. Su nombre viene de los gráficos de quemado de calorías de los regímenes dietéticos; se usan también en Scrum. Se trata de una técnica de graficación para descubrir demoras y problemas tempranamente en el proceso, evitando que se descubra demasiado tarde que todavía no se sabe cuánto falta. Para ello se hace una estimación del tiempo faltante para programar lo que resta al ritmo actual, lo cual sirve para tener dominio de proyectos en los cuales las prioridades cambian bruscamente y con frecuencia. Esta técnica se asocia con algunos recursos ingeniosos, como la Lista Témpana, llamada así porque se refiere al agregado de ítems con alta prioridad en el tope de las listas de trabajos pendientes, esperando que los demás elementos se hundan bajo la línea de flotación; los elementos que están sobre la línea se entregarán en la iteración siguiente, los que están por debajo en las restantes. En otras Metodologías Ágiles la Lista Témpana no es otra cosa que un gráfico de retraso. Los gráficos de quemado ilustran la velocidad del proceso, analizando la diferencia entre las líneas proyectadas y efectivas de cada entrega.

8. Programación lado a lado. Mucha gente siente que la programación en pares de XP involucra una presión excesiva; la versión de Crystal Clear establece proximidad, pero cada quien se enfoca a su trabajo asignado, prestando un ojo a lo que hace su compañero, quien tiene su propia máquina. Esta es una ampliación de la Comunicación Osmótica al contexto de la programación [10].

Hay ocho roles nominados en CC: Patrocinador, Usuario Experto, Diseñador Principal, Diseñador-Programador, Experto en Negocios, Coordinador, Verificador, Escritor. En Crystal Naranja se agregan aun más roles: Diseñador de IU (Interfaz de Usuario), Diseñador de Base de Datos, Experto en Uso, Facilitador Técnico, Analista/Diseñador de Negocios, Arquitecto, Mentor de Diseño, Punto de Reutilización. A continuación se describen los artefactos de los que son responsables los roles de CC:

1. Patrocinador. Produce la Declaración de Misión con Prioridades de Compromiso (Tradeoff). Consigue los recursos y define la totalidad del proyecto.

2. Usuario Experto. Junto con el Experto en Negocios produce la Lista de Actores-Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe familiarizarse con el uso del sistema, sugerir atajos de teclado, modos de operación, información a visualizar simultáneamente, navegación.

3. Diseñador Principal. Produce la Descripción Arquitectónica. Se supone que debe ser al menos un profesional de Nivel 3. (En Metodologías Ágiles se definen tres niveles de experiencia: Nivel 1 es capaz de "seguir los procedimientos"; Nivel 2 es capaz de "apartarse de los procedimientos específicos" y encontrar otros distintos; Nivel 3 es capaz de manejar con fluidez, mezclar e inventar procedimientos). El Diseñador Principal tiene roles de coordinador, arquitecto, mentor y programador más experto.

4. Diseñador-Programador. Produce, junto con el Diseñador Principal, los Borradores de Pantallas, el Modelo Común de Dominio, las Notas y Diagramas de Diseño, el Código Fuente, el Código de Migración, las Pruebas y el Sistema Empaquetado. Un programa en CC es "diseño y programa"; sus programadores son diseñadores-programadores. En CC un diseñador que no programe no tiene cabida.

5. Experto en Negocios. Junto con el Usuario Experto produce la Lista de Actores-Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe conocer las reglas y políticas del negocio.

6. Coordinador. Con la ayuda del equipo, produce el Mapa de Proyecto, el Plan de Entrega, el Estado del Proyecto, la Lista de Riesgos, el Plan y Estado de Iteración y la Agenda de Visualización.

7. Verificador. Produce el Reporte de Bugs. Puede ser un programador en tiempo parcial, o un equipo de varias personas.

8. Escritor. Produce el Manual de Usuario. El Equipo como Grupo es responsable de producir la Estructura y Convenciones del Equipo y los Resultados del Taller de Reflexión [10].

C. SCRUM

Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas Sprint, con una duración de 30 días. El resultado de cada Sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo del proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración [10].

Antecedentes

Esta metodología toma su nombre de una posición entrelazada en círculo que toman los integrantes de los equipos de rugby para tomar decisiones sobre el juego. Sus principios fundamentales fueron desarrollados en procesos de reingeniería por Goldratt, Takeuchi y Nonaka en la década de 1980 y aplicados al proceso de desarrollo de software por Jeff Sutherland en 1993, siendo formalizado con la colaboración de Ken Schwaber en una presentación en OOSPLA 96. Los principios de la misma han evolucionado con las contribuciones de varios autores fundamentalmente Cohn, Beedle y Schwaber [10].

El Scrum se ha desarrollado teniendo como principios fundamentales:

- El principio de incertidumbre de Ziv en la ingeniería del software: la incertidumbre es inherente e inevitable en el proceso de desarrollo de productos de software [16].

- Principio de requisitos indefinidos de Humphrey: para un nuevo sistema de software, los requerimientos no serán totalmente conocidos hasta que el usuario no lo haya usado [12].

En Scrum inicialmente planean el contexto y un estimado amplio de la entrega. Después desarrollan el estimado de la entrega basándose en desarrollo del ambiente del proyecto. El proceso de ciclo de vida de Scrum reconoce que el proceso de desarrollo fundamental está completamente indefinido y usa mecanismos de control para perfeccionar la flexibilidad, manipular lo impredecible y el control del riesgo [15].

Características del proceso

Las características esenciales del proceso de Scrum son:

- La primera y última fase (planificación y clausura) consisten en procesos definidos, donde todos los procesos, entradas y salidas están bien definidas. El conocimiento de cómo hacer estos procesos es explícito y se trata de hacer un repositorio de todas las actividades a realizar ("Backlog System").

- Se desarrollan iteraciones mensuales llamadas "Sprint". El equipo de desarrollo decide que funcionalidad incluir o no en cada iteración estimándose el tiempo necesario para terminar las tareas. La fase del Sprint es un proceso empírico. Muchos de los procesos en esta fase no están identificados o no son controlados.

- Los "Sprints" son no lineales y flexibles. Pueden ser usados procesos de conocimiento explícito. Cuando no existen estos conocimientos explícitos dentro del equipo, los errores y pruebas se usan para crear procesos de conocimiento.

- Dentro de los "Sprints" se realizan reuniones diarias de 15 minutos ("Scrum Meetings") en los cuales cada desarrollador da repuesta a tres preguntas:

1. ¿Qué hizo desde la última reunión?.

2. ¿Qué dificultades concretas tiene en el desarrollo de la tarea?

3. ¿Qué va a hacer hasta la próxima reunión diaria?

El proyecto es abierto hasta la fase de clausura. La entrega puede ser replanificada en cualquiera de las fases anteriores manteniéndose abierto a la complejidad del ambiente, la competencia, tiempo, calidad y presiones financieras, durante el desarrollo de dichas fases.

La entrega del proyecto es calculada sobre la influencia del ambiente.

- La metodología Scrum está diseñada para ser flexible durante el desarrollo de los sistemas. Provee de mecanismos de control para planificar una entrega del producto y manejar las variables en el progreso del proyecto. Esto permite la organización para cambiar el proyecto y las entregas en cualquier punto del desarrollo, entregando la versión más apropiada.

-La metodología Scrum libera a los desarrolladores a inventar las más ingeniosas soluciones durante el proyecto, mientras aprenden y el ambiente cambia. Los equipos de desarrolladores (idealmente alrededor de 7 miembros) pueden intercambiar los conocimientos acerca de los procesos de desarrollo, siendo esto una magnifica oportunidad de entrenamiento en el ambiente.

Fases

Las diferentes fases del Scrum consisten en:

Pre-juego

Planificación: definición de una nueva entrega basándose en un "Backlog" conocido junto a un costo y cronograma estimados. Si un nuevo sistema empieza a desarrollarse, esta fase consiste en conceptualización y análisis.

Arquitectura: Diseña cómo los artículos del Backlog son implementados. Esta fase incluye la creación ó modificación de la arquitectura del sistema y el diseño de alto nivel.

Juego

Desarrollo de los Sprints: desarrollo de una nueva funcionalidad en constante mira a las variables tiempo, requerimientos calidad, costo y competencia. La interacción con estas variables define el final de esta fase. Son múltiples los Sprints o ciclos usados para desarrollar el sistema. Dentro del Sprint la retroalimentación se obtiene con las reuniones diarias (Scrum-Meetings) y el control de la curva de progreso.

Post-juego

Clausura: preparación para la entrega, incluyendo la documentación final, prueba y entrega.

IV. ORIENTACIONES PARA LA INGENIERÍA DE SOFTWARE EDUCATIVO A PARTIR DE ALGUNAS CARACTERÍSTICAS DE LAS METODOLOGÍAS DE DESARROLLO ÁGIL

Esta sección tiene como objetivo articular diferentes tópicos de la ingeniería del software como son el enfoque clásico, los enfoques modernos y las metodologías de desarrollo ágil para proponer algunas orientaciones para el proceso de desarrollo de software educativo. Para cumplir con el objetivo se plantean una serie de fases o etapas por las cuales debe transitar el proceso de desarrollo de software educativo, entre las cuales proponemos:

1.Planeación

2. Obtención de requerimientos

3. Análisis

4. Diseño

5. Implementación

Revista Avances en Sistemas e Informática, Vol.5 No.2, Junio de 2008, Medellín, ISSN 1657-7663

6. Pruebas

7. Evaluación

Planeación

Durante la planeación se propone la conformación del equipo de trabajo y el establecimiento de las políticas de trabajo del proyecto, estas políticas deben estar enmarcadas en su gran porcentaje al fortalecimiento de los procesos de enseñanza y aprendizaje y a la incorporación de las nuevas tecnologías en los ambientes educativos. Es de vital importancia establecer políticas en cuanto al trabajo del equipo como son: el énfasis en la comunicación del equipo, el énfasis en la incorporación casi permanente del docente y/o estudiante en el equipo, el énfasis en el desarrollo incremental a través de iteraciones cortas. Es de especial importancia el factor de colaboración con el cliente que en este caso seria la institución educativa, el docente y/o el estudiante. Para la conformación del equipo de trabajo se proponen los siguientes perfiles:

- Un gerente de proyecto: Es el encargado de establecer, operacionalizar y controlar las políticas del proyecto.
- Un analista: Es la persona encargada de construir el modelo de análisis (funcional, datos y dinámico) del sistema de software.
- Un experto en requerimientos: Es la persona encargada de identificar las funcionalidades que necesita el cliente en el sistema.
- Un diseñador: Es la persona encargada de construir la arquitectura del sistema junto con sus respectivas maquetas y rutas de navegación.
- Un experto en Informática educativa: Es la persona encargada de emitir conceptos relacionados con los modelos pedagógicos apropiados para la construcción de software educativo.
- Un desarrollador o varios dependiendo del tamaño del proyecto: Son las personas encargadas de implementar las funcionalidades del sistema.
- El cliente (docente o estudiante): Son las personas que van a utilizar y probar el sistema.

Entre la definición de las políticas se puede empezar por definir los pilares conceptuales fundamentales sobre los cuales se deben construir los procesos de desarrollo de software educativo. A manera de propuesta se sugieren los siguientes pilares:

- Aspectos educativos.
- Aspectos tecnológicos.
- Aspectos conceptuales.
- Aspectos metodológicos.
- Aspectos organizacionales.

Es importante resaltar que en la fase de planeación se deben especificar los niveles de articulación entre los diferentes pilares como se muestra en la figura 7:

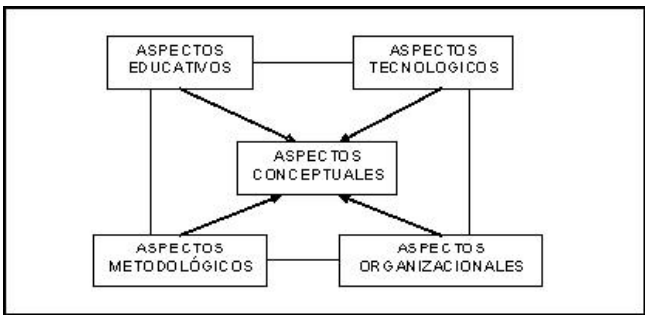


Figura 3. Pilares conceptuales fase planeación.

Entre los aspectos educativos se deben contemplar ítems como el modelo pedagógico que debe soportar el software educativo articulado con el uso de las nuevas tecnologías. En la especificación de este pilar conceptual es donde radica la mayor diferencia con una metodología utilizada para el desarrollo de un software tradicional.

De igual forma, en este segmento se deben especificar las estrategias didácticas a las cuales debe responder el software educativo.

Entre los aspectos tecnológicos se deben describir los componentes de hardware, software y comunicaciones que soportan la implementación del software educativo.

Entre los aspectos conceptuales se deben especificar los contenidos que se pretenden transmitir a través del software educativo.

Entre los aspectos metodológicos se deben describir todas aquellas actividades y estrategias que deben acompañar el uso del software educativo en el proceso de enseñanza-aprendizaje.

Los aspectos organizacionales deben detallar todas aquellas actividades relacionadas con la gestión que garanticen el correcto proceso de desarrollo e implementación del software educativo.

Obtención de requerimientos

Un requerimiento es una característica que debe tener el sistema o una restricción que debe satisfacer para que sea aceptado por el cliente. En esta fase nos enfocamos en la obtención de requerimientos basados en escenarios y casos de uso. Los desarrolladores obtienen los requerimientos a partir de entrevistas con los usuarios que en este caso pueden ser los docentes y estudiantes que van utilizar el sistema, luego desarrollan escenarios visionarios en donde se describe la funcionalidad que proporcionará el sistema futuro. El cliente y los usuarios validan la descripción del sistema revisando los escenarios y probando prototipos pequeños proporcionados

por los desarrolladores. Conforme madura y se estabiliza la definición del sistema, los desarrolladores y el cliente se ponen de acuerdo en una especificación del sistema en forma de casos de uso. Para cada caso de uso se deben especificar los flujos básicos y flujos alternativos en forma sencilla.

La obtención de requerimientos se enfoca en la descripción del propósito del sistema. El cliente, los desarrolladores y los usuarios identifican un área problema y definen un sistema que ataca el problema. A tal definición se le llama especificación del sistema.

Como producto de esta fase se debe producir un modelo de casos de uso acompañado de un documento donde se especifiquen los requerimientos no funcionales.

En metodologías tradicionales se trata de cumplir con requerimientos no funcionales sin que lleguen a convertirse en el eje central del producto software. Para la ingeniería del software educativo es de relevante importancia la determinación de requerimientos de escenarios y personajes debido a que estos elementos son los medios a través de los cuales el niño va a adquirir los contenidos del aprendizaje.

Análisis

El análisis da como resultado un modelo del sistema que pretende ser correcto, completo, consistente y verificable. El cliente y el usuario están involucrados, por lo general, en esta actividad, en especial cuando se necesita cambiar la especificación del sistema y cuando se necesita recopilar información adicional.

El análisis se enfoca en la producción de un modelo del sistema, llamada el modelo de análisis, que es correcto, completo, consistente y verificable. El análisis se diferencia de la obtención de requerimientos en que los desarrolladores se enfocan en la estructuración y formalización de los requerimientos obtenidos de los usuarios. Aunque puede ser que el modelo de análisis no sea comprensible para los usuarios y el cliente, ayuda a que los desarrolladores verifiquen la especificación del sistema producida durante la obtención de requerimientos.

Como producto de esta fase se debe producir un modelo de clases si es necesario acompañado del refinamiento del modelo de casos de uso generado en la fase anterior, estos modelos deben ser desarrollados solo a nivel de diagramas de UML solo se deben especificar con mayor detalle en caso de que los clientes lo requieren.

Diseño

El diseño de sistemas es la transformación del modelo de análisis en un modelo de diseño del sistema. Durante el diseño del sistema los desarrolladores definen los objetivos de diseño del proyecto y descomponen el sistema en subsistemas más pequeños que pueden ser realizados por equipos individuales. Los desarrolladores también seleccionan estrategias para la construcción del sistema, como la plataforma de hardware y software en la que se ejecutará el sistema, la estrategia de almacenamiento de datos persistentes, el flujo de control global, la política de control de acceso y el manejo de condiciones de frontera. El resultado del diseño del sistema es un modelo que incluye una descripción clara de cada una de estas estrategias, una descomposición en subsistemas y un diagrama de organización que representa la correspondencia entre el hardware y el software del sistema.

En forma específica en esta etapa se debe generar un modelo de arquitectura en forma general y una lista de requerimientos no funcionales u objetivos de diseño.

En esta etapa de diseño se debe hacer énfasis en tres tipos de diseño adicional que son los siguientes:

1. Diseño educativo
2. Diseño de comunicación
3. Diseño computacional

Diseño educativo

Debe resolver los interrogantes que se refieren al alcance, contenido y tratamiento que debe ser capaz de apoyar el sistema. En este tipo de diseño se debe prestar especial atención al modelo pedagógico para diseñar las actividades del sistema de acuerdo al modelo o fusión de modelos sobre los cuales se quiere implementar el sistema.

A partir de las necesidades de los usuarios finales del sistema se debe implementar el sistema que responda a ellas y al contexto en el que se desenvuelve el usuario final.

Se debe utilizar una propuesta de modelo pedagógico que sirva como eje transversal en el proceso de enseñanza – aprendizaje que se va a soportar con el sistema de software. Este modelo pedagógico debe servir como medio para incorporar al usuario final los contenidos que necesita para alcanzar el perfil que se busca en un educando.

El modelo pedagógico a utilizar en la implementación del software educativo debe estar articulado con aspectos de relevante importancia como los contenidos, el ambiente o contexto en el cual se deben desarrollar los contenidos.

En forma general, se pueden visualizar los aspectos que se deben tener en cuenta en el diseño educativo de la siguiente manera:

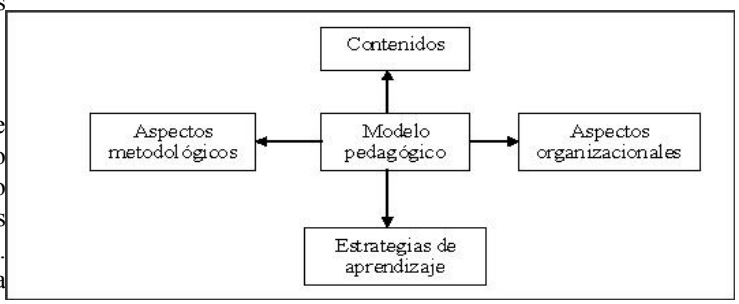


Figura 4. Elementos del diseño educativo.

Revista Avances en Sistemas e Informática, Vol.5 No.2, Junio de 2008, Medellín, ISSN 1657-7663

Como se puede observar en la figura anterior el diseño educativo está basado fundamentalmente en el modelo pedagógico a emplear o definido para la situación de aprendizaje.

Diseño de comunicación

En esta se sección se pretende especificar la forma como se comunica el usuario con el programa, estableciendo dispositivos y códigos o mensajes. Básicamente esta fase corresponde a lo que comúnmente se denomina diseño de interfaces.

En la figura se describen los aspectos que se deben tener en cuenta en el diseño de comunicación:

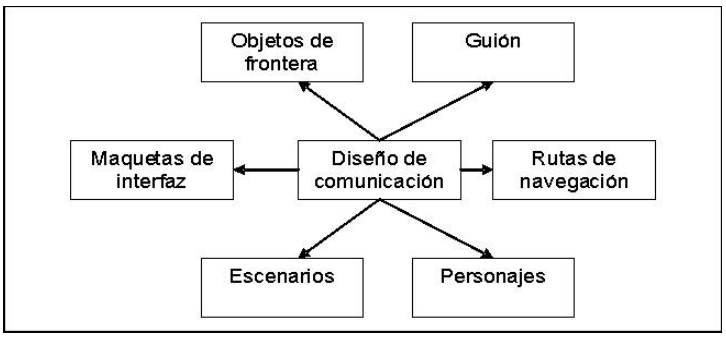


Figura 5. Elementos diseño de comunicación

Diseño computacional

Tomando como punto de partida las necesidades de los usuarios se establece qué funcionalidades debe cumplir el sistema de software educativo. En este segmento también se deben materializar los requerimientos no funcionalidades que surgen de los clientes, usuarios y del propio modelo pedagógico.

En el diseño computacional se deben describir los usuarios del sistema complementados con las funcionalidades a las cuales tienen acceso, para luego implementar módulos de acuerdo al diseño más apropiado que soporte los requerimientos funcionales y no funcionales del sistema.

En forma general, en el diseño computacional se deben materializar los modelos funcional, de clases y dinámico.

En la figura se describen los aspectos que se deben tener en cuenta en el diseño computacional:

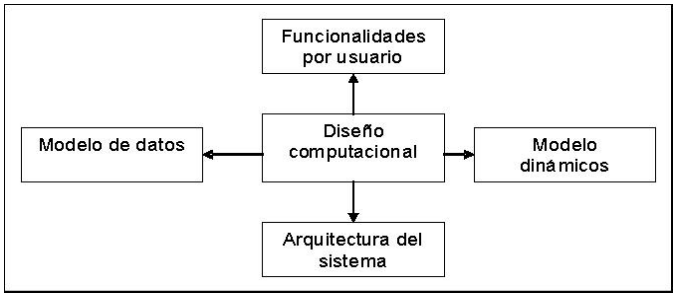


Figura 6. Elementos diseño computacional

En la figura se observa que para el diseño computacional se deben materializar las funcionalidades de todos los usuarios a través de diagramas de casos de uso, de igual forma se deben especificar el modelo de datos del sistema si es necesario y por último el modelo dinámico por medio de diagramas de secuencia.

Implementación

Durante la implementación se pretende traducir los diferentes modelos diseñados en las etapas anteriores en código fuente. En forma paralela se deben integrar cada uno de los subsistemas desarrollados en forma segmentada.

Pruebas

Las pruebas son el proceso de encontrar diferencias entre el comportamiento esperado, especificado por los modelos del sistema, y el comportamiento observado del sistema.

Las pruebas son el proceso de análisis de un sistema, o componente de un sistema, para detectar las diferencias entre el comportamiento especificado (requerido) y el observado (existente). El objetivo de las pruebas es aumentar la confiabilidad del sistema.

En términos educativos se recomienda realizar pruebas pilotos y pruebas de campo con potenciales usuarios del sistema y con otros desarrolladores diferentes a las del equipo.

Evaluación

Es de vital importancia para los procesos de enseñanza-aprendizaje determinar en la fase de requerimientos instrumentos de evaluación que pueden ser actividades dentro del sistema con los cuales se puedan medir el nivel de cumplimiento de los objetivos para cada uno de los contenidos que se pretenden desarrollar en el sistema.

CONCLUSIONES

Los métodos ágiles son una reacción a las formas tradicionales de desarrollo de software, admitiendo la necesidad de una alternativa al desarrollo de software orientado a la documentación y centrados en el proceso.

Los métodos ágiles de desarrollo de software han surgido muy recientemente como tendencias no ampliamente aceptadas aun, pero con buena probabilidad de lograr interesantes aportes en cuanto a metodologías de desarrollo de software. Estas tendencias se conocen también con el nombre de métodos livianos y son apropiados para pequeños equipos de desarrollo.

La agilidad se puede definir con dos caracterizaciones [11]:

- Agilidad es la habilidad para crear y responder a cambios de acuerdo a los beneficios en un ambiente de negocios turbulento.
- Agilidad es la habilidad para balancear flexibilidad y estabilidad.

En este sentido la agilidad de los métodos de desarrollo software se puede sintetizar en la capacidad para responder al

cambio y para simplificar los procesos.

La propuesta metodológica para el desarrollo de software educativo describe un conjunto de fases que permiten guiar el proceso de desarrollo de productos software que apoyen el proceso enseñanza-aprendizaje. Esta propuesta tiene la particularidad que permite articular aspectos educativos, tecnológicos, conceptuales, metodológicos y organizacionales de una manera rápida sin caer en principios de las metodologías tradicionales como es la orientación al proceso.

De igual forma, la propuesta permite adaptar procesos de desarrollo que requieran altos niveles de documentación y orientación al proceso, adicionalmente también permite adaptarse a proyectos de desarrollo de software educativo de gran tamaño.

De acuerdo a las características de las metodologías de desarrollo ágil como son el avance iterativo e incremental, permite tener rápidamente prototipos funcionales que pueden ser probados y evaluados en forma conjunta por un equipo conformado por clientes y desarrolladores.

La propuesta metodológica esta orientada a la generación de prototipos funcionales en cada iteración y no tanto a la producción de artefactos. Sin embargo, se puede adaptar a la generación de artefactos según exijan los requerimientos de los clientes.

En forma general se puede afirmar que la propuesta metodológica permite adaptarse a procesos de desarrollo de software educativo de diferentes tamaños, requerimientos y niveles de complejidad.

La propuesta metodológica o conjunto de fases de desarrollo ofrece una alternativa de solución al problema descrito en el hecho de que muchos proyectos de software educativo se llevan a cabo sin ningún eje conductor que permita orientar el proceso de desarrollo, en otras situaciones estos proyectos son de un tamaño pequeño que no permiten la utilización de metodologías tradicionales que son muy orientadas a la generación de artefactos y los presupuestos de tiempo no son tan largos.

La propuesta en mención tiene como uno de sus valores agregados la capacidad de adaptarse a diferentes tipos de proyectos de desarrollo de software educativo, es decir, a proyectos de diferentes tamaños, características, nivel de educación y de diferente tipo (multimedia, micromundo, videojuego, animación, diálogos animados).

En metodologías tradicionales empleadas por la ingeniería del software educativo no se le da el valor necesario a los aspectos educativos y/o didácticos, en características tales como el modelo pedagógico que deba soportar el producto de software educativo. En otros trabajos se sugieren algunos macro algoritmos que pueden guiar el proceso de desarrollo de software educativo basados en diferentes modelos pedagógicos lo cual se convierte en un conjunto de alternativas para los educadores y para los desarrolladores.

REFERENCIAS

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

[2] Beck, K.. "Extreme Programming Explained. Embrace Change", Pearson Education, 1999. Traducido al español como: "Una explicación de la programación extrema. Aceptar el cambio", Addison Wesley, 2000.

[3] Boehm, B. Turner, R. Balancing Agility and discipline. A guide for the Perplexed. Addison-Wesley. 2003.

[4] Bruegge, B. , Dutoit, A. Ingeniería de software orientada a objetos. Pearson Educación. México, 2002.

[5] Canós, J., Letelier, P. y Penadés, M. Metodologías Ágiles en el desarrollo de Software. Universidad Politécnica de Valencia, Valencia, 2003.

[6] Caro, G. Agile manifiesto y experiencias personales. Memorias Jornadas de Gerencia. ACIS. Bogotá 2004.

[7] Cockburn, A. Selecting a Project's Methodology, , Humans and Technology, IEEE SOFTWARE July/August 2000.

[8] Cockbun, A. .Agile Software Development.. Addison-Wesley. 2001.

[9] Fowler, M., Beck, K., Brant, J. "Refactoring: Improving the Design of Existing Code". Addison-Wesley. 1999.

[10] Gutierrez, Joaquin. Metodologías ágiles. Universidad Pablo de Olavide. 2007.

[11] Highsmith, J. Agile Project Management, 2003

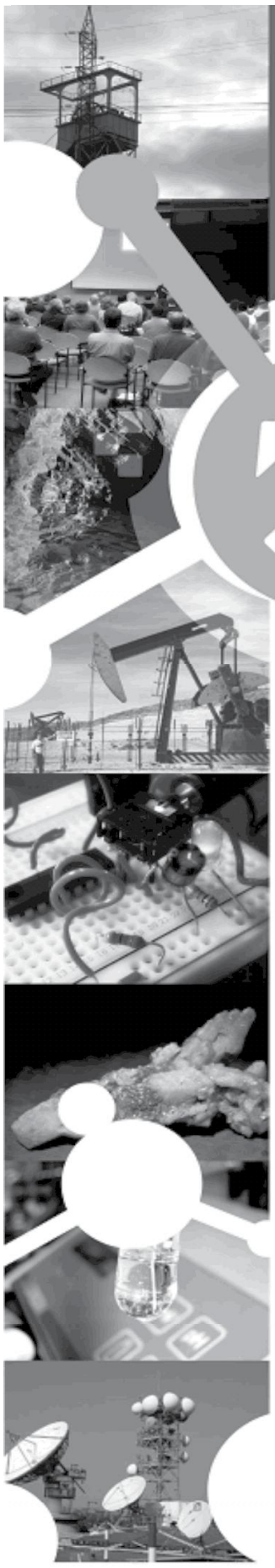
[12] Humphrey, W.S., Managing the Software Process, Addison-Wesley, Reading, MA, 1989.

[13] Poppendieck M., Poppendieck T. "Lean Software Development: An Agile Toolkit for Software Development Managers". Addison Wesley. 2003.

[14] Pressman, R. Software Engineering: A Practitioner's Approach. McGraw-Hill. 2005.

[15] Schwaber, k. Agile Project Management with Scrum (Microsoft Professional). Mar 10, 2004.

[16] Ziv, H. y D. J. Richardson: , ICSE97, XIX International Conference on Software Engineering, 23 de agosto de 1996.



UNIVERSIDAD
NACIONAL
DE COLOMBIA

SEDE MEDELLÍN

FACULTAD DE MINAS

120 años

TRABAJO Y RECTITUD