Arquitectura de software para modelado de programación extrema

Graciela Lara-López, Marco A. Pérez-Cisneros y Jorge F. Hernández-Andrade

Universidad de Guadalajara. División de Electrónica y Computación. Departamento de Ciencias Computacionales. Av. Revolución 1500. Col Olímpica. Guadalajara. Jalisco. S.R. C.P 44430. México
graciela.lara@red.cucei.udg.mx, marcopc@cucei.udg.mx, jhandrade@red.cucei.udg.mx
Paper received on 01/08/08, accepted on 10/09/08.

Resumen. La producción formal de software plantea el uso de metodologías tradicionales y herramientas que se aplican por separado. Por tanto, la aplicación de una herramienta integrada que permitiera una interacción entre las distintas etapas de ciclo de vida del software y el personal involucrado dentro del desarrollo, puede representar una contribución de impacto muy importante. Este artículo presenta una propuesta para diseñar una herramienta que modela la programación extrema, apegada a sus cuatro valores y doce principios. En segundo término, este artículo compara esta nueva arquitectura con la única herramienta que existe en el mercado, conocida como Agilian© de Visual Paradigma, por medio de la implementación en un caso práctico, donde programación extrema resulta ser más eficiente que las metodologías tradicionales.

1 Introducción

Cada empresa de diseño de software emplea metodologías para desarrollar sus sistemas y para la administración de proyectos de software. Por lo general se considera un ciclo de vida bastante largo, donde el proceso comienza por la entrevista, requerimientos, diseño, desarrollo, codificación, pruebas y mantenimiento [3].

Es un problema constante para un líder de proyecto administrar sus requerimientos, costos, tiempos, personal, etc. En ocasiones no solo se dirige un proyecto, por lo que existe el riesgo de perder por un instante la documentación o alguna de las versiones ya realizadas, generando cambios o pérdidas de avances.

Sin el uso de la programación extrema, los programadores invierten muchos recursos para la construcción del software, normalmente utilizando lenguajes de programación estructurados, con nula posibilidad en la reutilización de bibliotecas. Esto repercute directamente en el proceso de mejoras y mantenimiento del sistema. Así mismo, en algunas ocasiones no existe un control de versiones que responda eficazmente a los cambios en los requerimientos ordenados por el cliente.

El auge de empresas dedicadas al desarrollo de sistemas, el surgimiento de nuevos lenguajes de programación y las computadoras con un alto poder de desempeño, han resultado en una evolución acelerada en los métodos de desarrollo de software aun con descuidos en etapas de desarrollo.

© M. G. Medina Barrera, J. F. Ramírez Cruz, J.H. Sossa Azuela, (Eds.).
Advances in Intelligent and Information Technologies
Research in Computing Science 38, 2008, pp.309-319.

En el caso de algunas organizaciones el proceso de documentación de los sistemas no cuenta con bitácoras (historial) de problemas encontrados y soluciones efectuadas, tienen problemas en las versiones de los módulos, no existen diagramas de apoyo para el desarrollo, etc. Ante esta problemática, nace UML (Lenguaje Unificado de Modelado) [1], un lenguaje diseñado para el desarrollo de software, cuyo objetivo es modelar todas las partes que forman un sistema como son las entradas, los procesos y los resultados de éste. Con la creación del lenguaje unificado nacen los llamados procesos ágiles, que se caracterizan por una comunicación constante con el cliente convirtiéndolo en parte del grupo de trabajo. Dichos sistemas también atribuyen especial importancia a la documentación del desarrollo mismo, y utilizan características derivadas de filosofías y no de metodologías. La mayor parte de sus aportaciones al desarrollo de software son expuestas en forma de recomendaciones que permiten llevar a cabo mejoras. Una característica final es su independencia del generador de código.

Los cambios de requerimientos durante el desarrollo de sistemas traen consigo problemas serios a tomar en cuenta como lo son: postergación de fechas de entrega, largos procesos de pruebas, modificaciones y mantenimiento, menos comunicación con el cliente y por tanto, menos entendimiento en los requerimientos.

Con la programación extrema, estos cambios no varían radicalmente los tiempos de entrega, ya que el contacto con el cliente es directo. De esta forma, el área de oportunidad que se pretende abordar es la comunicación. La programación extrema propone iniciar con el trabajo de los requerimientos esenciales del sistema desde el punto de vista del cliente para desglosar la estructura del mismo, evitando programar partes que pueden llegar a ser innecesarias para el sistema final.

Es muy frecuente que los avances, versiones y errores no sean documentados adecuadamente. Es precisamente aquí, donde la programación extrema puede ser de importante utilidad. En este trabajo se aborda la necesidad de que los desarrolladores de software cuenten con una herramienta para modelar la programación extrema. Actualmente es una de las filosofías de trabajo con mayor seguimiento y control en el desarrollo de sistemas.

2 Estudio del Arte

Los procedimientos de la ingeniería de software son la conjunción de los métodos y las herramientas para llevar a cabo el desarrollo de sistemas. Algunos métodos de ingeniería de software indican como construir técnicamente el código. Los métodos abarcan un amplio campo de tareas que incluyen la planificación y la estimación de proyectos, el análisis de requerimientos del sistema, el diseño de estructuras de datos, la arquitectura de programas, los procedimientos algorítmicos, la codificación, las pruebas y el mantenimiento. Las herramientas de la ingeniería de software dan un soporte automático para dichos métodos.

UML (Unified Modeling Language), es un lenguaje utilizado para la administración y obtención de calidad en el desarrollo de software. Fue creado al unificar los métodos Booch, Objectory de Iván Jacobson y OMT (Object Modeling Technique) de James Rumbaugh [5] para entender, diseñar, configurar, mantener y controlar la información del software en desarrollo. Este lenguaje utiliza un modelo gene-

ral del sistema, por medio de gráficos que permiten visualizar, especificar, construir y documentar un software.

En los últimos años nacen varias metodologías llamadas en un principio ligeras, y conocidas finalmente como Ágiles, las cuales cambian poco a poco la forma en la que los desarrolladores de software crean sus sistemas. Ágile propone una estructura con innovación continua que responde de manera eficaz los cambios de los requerimientos del sistema, ofrece una estructura robusta pero no demasiado grande y proporciona flexibilidad así como estabilidad en virtud de considerarse más una filosofia que una metodología [4]. Cabe mencionar que la Programación Ágil no es un lenguaje consolidado como lo es UML.

En 1999 nace una nueva disciplina de desarrollo de software llamada Programación Extrema creada por Kent Beck [7], la cual se basa en la simplicidad, la comunicación y el reciclado continuo de código. Los objetivos de la programación extrema son: la satisfacción del cliente, toma en cuenta todas las partes implicadas en el proyecto y los cuatro valores que menciona Beck son: *Comunicación, Sencillez, Retroalimentación y Valentia.* La programación extrema es más liviana y ágil y esta orientada más a las personas que a los procesos. Toma en cuenta las siguientes doce reglas: programación por parejas y detallada; cualquier miembro del equipo se puede emparejar con cualquiera; clientes en sitio; el equipo de trabajo debe trabajar no más de 40 horas a la semana; establecer un estándar de codificación; planificación; realizar pequeñas versiones con fechas preestablecidas; desarrollo del software; diseño sencillo y funcional; remodificación; documentar; hacer pruebas e integración de codificación e información y documentación de apoyo.

Cabe mencionar que sólo existe una herramienta de software en el mercado para la programación extrema, llamada: Agilian© de la compañía Visual Paradigm, la cual tiene sus propias características y diferencias con la nueva propuesta.

3 Metodología empleada.

Para desarrollar esta arquitectura se necesita en primer lugar, entender la metodología de la programación extrema, su forma de trabajo, su forma de relacionar sus cuatro valores de tal forma que sea sencilla de utilizar sin perder al mismo tiempo su relación con sus doce prácticas.

Se comenzará primero por mostrar las clases consideradas para el diseño de la arquitectura, segundo se describen los cuatro módulos que la conforman y las clases que los integran.

3.1 Arquitectura General de la Programación Extrema

La Figura. 1, muestra la arquitectura propuesta para la programación extrema en su diagrama de clases.

3.2 Descripción de cada componente

La arquitectura aquí propuesta esta formada por varias asociaciones, herencias y clases con sus atributos y operaciones, las cuales se describen a continuación:

- CLASE MANEJADOR DATOS, es la clase que representa la forma en que el sistema interactúa con el sistema operativo como se ejecuta nuestro sistema (Windows, Linux, etc), entre sus funciones están: el resguardo de la información y las conexiones entre las bases de datos y el sistema.

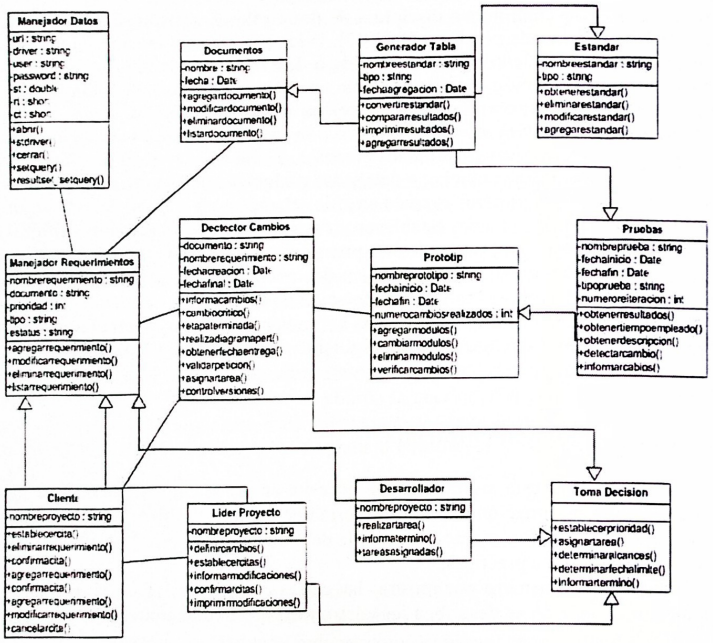

Fig. 1. Arquitectura ejemplificada en UML, en diagrama de Clases.

- CLASE DOCUMENTOS, permite describir la forma en que serán almacenadas todas las modificaciones que se van generando, y su tarea es llevar respaldo escrito de los avances de cada software.

- CLASE MANEJADOR REQUERIMIENTOS, es una de las clases principales del sistema, ya que contiene parámetros esenciales para la identificación de requerimientos, su nombre, prioridad o tipo, y contiene las operaciones más importantes para el cliente, las cuales son: agregar, modificar, eliminar o listar requerimientos.

- CLASE DETECTOR CAMBIOS, notifica avisos de cambios de un requerimiento, ya sea, agregación, modificación o eliminación para que el sistema evalúe ese requerimiento, le asigne prioridades, fechas probables de entrega, valida las peticiones y pasa el requerimiento a la clase de toma decisión.
- CLASE TOMA DECISIÓN, evalúa el requerimiento del sistema, determina sus alcances, fecha límite de entrega y asigna la realización de esa petición del cliente a cierto desarrollador, una vez que la clase detector de cambios ha recibido la solicitud.
- CLASE CLIENTE, es una de las partes mas importantes y es determinante que forme parte del grupo de trabajo. Esta clase contener los datos personales del cliente, las operaciones como establecer, cancelar o confirmar citas o agregar, modificar, eliminar o listar requerimientos.
- CLASE LÍDER DE PROYECTO, esta clase marca la diferencia entre cualquier integrante del grupo de trabajo y el líder del grupo. Posee acciones diferentes a las del cliente y del desarrollador ya que tiene la responsabilidad de establecer las citas con los clientes e informar de las modificaciones que va sufriendo su sistema.
- CLASE DESARROLLADOR, a diferencia del líder de proyecto tiene la obligación de realizar las tareas que se le van asignando e informar de su término, una vez que finaliza su tarea.
- CLASE GENERADOR TABLA, lo que hace esta clase es tomar estándares, convertirlos en una tabla, compararlos con nuestros resultados e imprimirlos para la futura cita con el cliente y poder tomar un punto de comparación entre nuestro sistema y los estándares.
- CLASE PRUEBAS, como su nombre lo dice, esta clase esta encargada de comparar los resultados obtenidos con los esperados después de realizar las pruebas pertinentes al prototipo o al software final.
- CLASE PROTOTIPO, una vez que se van agregando, modificando o eliminando módulos de programación al sistema que se esta desarrollando, se genera un nuevo prototipo, el cual ya esta actualizado, hasta los requerimientos mas prioritarios para el cliente.
- CLASE ESTÁNDAR, esta clase agrega, modifica o elimina los estándares que la clase generador tabla necesita para poder hacer las comparaciones con nuestro sistema.

La Figura 2, muestra la arquitectura propuesta para la programación extrema. Dentro de ésta se representan sus cuatro valores que se comentan adelante.

3.3 Módulo de comunicación

Promoviendo la comunicación se busca reducir los tiempos de retardo en revisiones y modificaciones, además de generar más confianza entre el grupo de trabajo y el cliente. Durante el proceso de comunicación, donde el cliente que es parte del grupo de trabajo, no se conforma con solamente contestar entrevistas, sino que mantiene comunicación con el grupo de desarrollo, hasta llegar a la fecha de entrega.

El diagrama que conforma la parte de la comunicación esta formado por las siguientes clases: Cliente, Detector Cambios, Líder Proyecto y Desarrollador.

3.4.1Módulo de Retroalimentación. Para la programación extrema es importante que se realice una adecuada organización de las versiones, prototipos y bibliotecas de módulos de cada sistema para que puedan ser reutilizados en futuros desarrollo. Las clases que conforma la parte de la retroalimentación son: Prototipo, Detector Cambios y Pruebas.

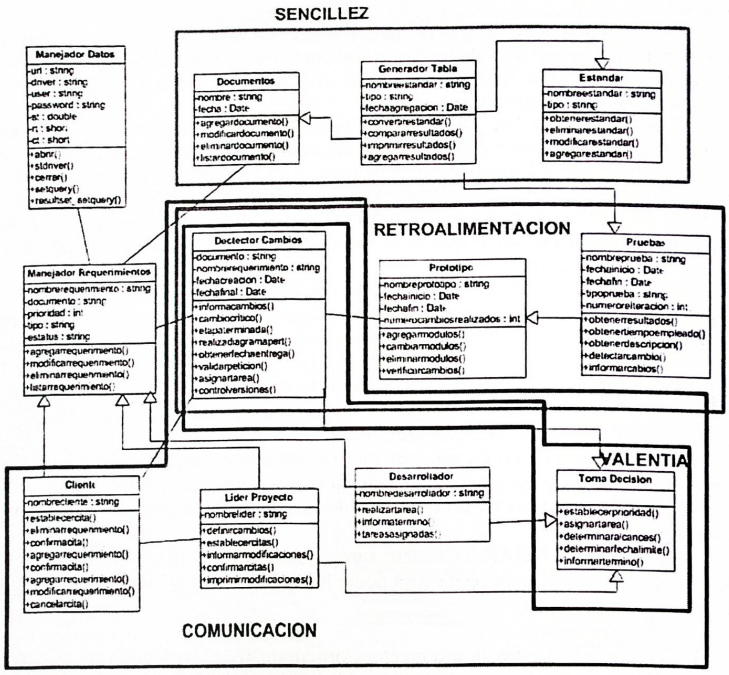

Fig. 2. Diagrama general de la arquitectura dividida en los valores de la programación extrema.

3.5 Módulo de Valentía

La valentía se refleja dentro de la arquitectura al demostrarnos que podemos hacer cualquier modificación sin tener que esperar mucho tiempo las aceptaciones del cliente, que gracias al módulo de comunicación, la toma de decisiones es rápida

y sencilla. Este valor se representa con las siguientes clases: Detector Cambios y Toma decisión.

3.6 Módulo de Sencillez

Propone que no se realicen tantos procesos y que los resultados sean entendibles y accesibles para todas las partes del grupo de trabajo, incluyendo al cliente. Para alcanzar la sencillez de la programación extrema, se necesita que la arquitectura sea fácil de entender como lo demuestran las siguientes clases: Estándar, Generador Tabla y Documentos.

4 Resultados

Para demostrar que la arquitectura propuesta cumple con los estándares básicos requeridos por la programación extrema, se realizo un comparativo entre la única herramienta existente que es Agilian © de Visual Paradigma.

Se toman en cuenta los principios más significativos de la programación extrema para realizar el comparativo propuesto. En la Tabla 1 se muestra dicha comparación, atendiendo cada uno de los principios.

Con el objeto de enriquecer la discusión se desarrollo el siguiente caso práctico: El Centro de Enseñanza Técnica Industrial, CETI, es una Institución Educativa para dar una formación de calidad a Tecnólogos Profesionales e Ingenieros, con capacidades emprendedoras para la generación y aplicación de tecnologías. Dada la necesidad de desarrollar un sistema que permitiera la captura y preselección de cursos por parte del estudiante y el Departamento de Control Escolar, se decide desarrollar una solución integral de software. Sin embargo el proceso de desarrollo enfrentaba serías dificultades a los cambios de personal y diferencias de horarios de los empleados [8].

De aquí la necesidad de crear un mecanismo para garantizar que todas estas anormalidades no afectaran el desarrollo del sistema, esto da origen a la aplicación de la programación extrema.

Debido a la creciente demanda estudiantil, el sistema anterior ya no cumplía con las expectativas y sobretodo no permitía un servicio adecuado al estudiante. Para solventar esta problemática se buscaron herramientas de software, entornos de desarrollo, bases de datos y utilerías que unidas cumplieran, en su totalidad, todas las características de la programación extrema y compararlas con nuestra arquitectura. De esta forma se cumplen dos objetivos: Demostrar que nuestra arquitectura cumple con todas las especificaciones expuestas por Kent Beck [7] creador de la programación extrema y comprender la programación a través del uso de herramientas externas.

Se ilustran a continuación las herramientas de software y los resultados obtenidos de éstos para la resolución del caso de estudio del CETI.

Tabla 1. Comparativa entre Agilian © y la nueva arquitectura.

Principios Programación Extrema	Agilian ©	Nueva Arquitectura
Programación en Par	No propone ninguna recomendación	Propone que se lleve a cabo tal y como se expresa en sus practicas.
Definición de roles	Define roles, pero muy esporádicos	Define claramente cada rol para la asignación de tareas y responsabilidades
Comunicación con clientes	Propone tenerlo siempre comunicado	Incluye al cliente al grupo de trabajo, para facilitar los procesos de comunicación y entregas.
Aplicación de Estándares y Herramientas	Se centra en los estándares generales y herramientas ya existentes	Se apoya de estándares para comprobar la calidad del sistema, hasta el momento se apoya en herramientas pero lo ideal es llevar la arquitectura de este trabajo hasta una herramienta CASE.
Propiedad colectiva	No todos están enterados de las modificaciones	Con la recomendación de la programación extrema sobre la programación en parejas, todos están al tanto de las modificaciones
Pruebas	Se llevan a cabo constantemente	Se realizan a la par de cada modificación
Desarrollar de la manera mas sencilla	Recomienda realizar el modelado y el desarrollo de la manera más simple.	Una de sus Prácticas recomienda realizarlo de la manera más sencilla.
Formalizar los modelos para terceras personas	Propone realizar formatos estandarizados para la presentación a terceras personas	Cada persona involucrada es tomada en cuenta para el desarrollo por tanto no hace falta la realización de formatos

Tabla 2. Comparativa entre las herramientas utilizadas y la arquitectura propuesta.

	Herramienta Utilizada	Proceso obtenido	Módulo de la arquitectura
Manejo de procesos	Rational ClearCase del DB2	Durante éste proceso se llevo a	Detector de cambios y Mane-

	de IBM.	cabo un buen manejo de los cambios que sufría el sistema, todos los requerimientos llevaban un buen orden, una bitácora y un seguimiento desde su solicitud hasta su realización y aprobación por parte del cliente.	jador requerimientos
Colaboración	Project web	Unos de los más importantes valores de la programación extrema es la comunicación, con esta herramienta quedó demostrado que se puede llevar a cabo una buena relación con el cliente y hacerlo parte del grupo de trabajo.	Cliente, Líder de proyecto y Desarrollador
Pruebas	JUnit	La parte de Pruebas fueron avaladas por esta herramienta, la cual fue de gran ayuda y dejo en claro que se aprobaron satisfactoriamente.	Pruebas
Aseguramiento de la calidad	DevPartner	Durante este proceso se obtuvo la aprobación de los estándares de calidad aplicados al sistema, esta herramienta colaboro a que se demostrara su validez.	Estándar y Generador Tabla
Desempeño	Jmeter	Una vez que se tenian los resultados de las pruebas se compararon con	Pruebas y Generador Tabla

		los resultados de una versión anterior del mismo sistema del CETI, demostrando que se obtuvieron mejoras en tiempo y costo.	
Administración del código fuente	Visual Source Safe	Con cada requerimiento agregado al sistema se llevo un riguroso proceso de documentación, esta herramienta ayudo mucho a este proceso.	Documentos
Fase de desarrollo	IBM Visual-Age	Para el proceso de desarrollo del sistema, esta herramienta colaboro a poder establecer los avances de cada modulo del sistema, una vez que se programaban, guardando sus versiones, y modificaciones que sufrían durante el proceso de termino.	Detector cambios y Prototipo

5 Conclusiones y Trabajos a Futuro

La validación de las características de la herramienta propuesta es aún subjetiva, pero consideramos que el diseño de esta arquitectura demuestra su robustez, permitiendo el desarrollo más eficiente, menos costoso y más simple para los desarrolladores y para el cliente. Cabe mencionar que la aplicación desarrollada del CETI con el diseño de esta arquitectura duro solo 2 meses. Comparando esto con los 4 años que ya se habían dedicado anteriormente, permite ver claramente una mejora considerable en el tiempo de desarrollo. A futuro se recomienda desarrollar procesos y módulos que fortalezcan esta arquitectura y la hagan independiente de estas propuestas, mediante el desarrollo de una nueva herramienta. Una de las principales herramientas que se utilizan en esta arquitectura son los diagramas de Pert [3], en los cuales nos apoyamos para la realización de la calendarización y asi estimar una fe-

cha de entrega del requerimiento. Se necesita que el sistema se empotre en un servidor web, para que tanto el cliente como el grupo de trabajo accedan desde cualquier parte del mundo con conexión a Internet, y revisen los documentos, los requerimientos, las citas, etc. Se necesita un sistema distribuido para distribuir información a todas las personas involucradas en la realización del proyecto. Sabemos de las tendencias de algunas empresas para contratar programadores externos que resuelven ciertas problemáticas y de ahí la conveniencia de tener este módulo que controla el flujo de información. El sistema distribuido contendrá la misma información del sistema independientemente distribuida en distintos servidores, pero tendrá el potencial de discernirla dependiendo de la persona que ingresa al sistema, es decir, el cliente no podrá tener acceso a ciertos documentos. Así mismo, el líder de proyecto no tendrá acceso a información que concierne solo al cliente. Se puede mejorar la parte de la asignación de requerimientos a los desarrolladores, que se hace manualmente. El líder de proyecto asigna dicha tarea, pero sería conveniente que el sistema determinará quien realiza que requerimiento en base a ciertos valores como: el tiempo con que cuenta el desarrollador, la experiencia del desarrollador en el tipo de sistema que se esta trabajando, la carga de trabajo y la capacidad de interactuar con otro desarrollador.

Referencias.

1. Kim Hamilton. O'reilly. (2006). Learning UML 2.0.
2. Lourdes Munich. Fundamentos de Administración. Trillas, Quinta edición.
3. Roger S. Pressman. (2005). Ingeniería de software un enfoque practico. Mc Graw Hill.
4. David Anderson. (2003). Prentice-Hall. Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results.
5. Booch G. (1999). Addison-Wesley. Editorial. El lenguaje unificado de modelado.
6. Craig Larman. (2004). Prentice – Hall. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3ra. Edition).
7. Michele Marchesi, Don Wells. Extreme Programming Perpectives. Addison-Wesley. 2002.
8. Esqueda Yañez Eliseo (2008). Diseño logístico para la administración de un proyecto de software usando la programación externa. Tesis de Maestria no publicada, Universidad de Guadalajara, Guadalajara, Jalisco.