UNIVERSIDAD CATÓLICA DEL URUGUAY

DÁMASO ANTONIO LARRAÑAGA

FACULTAD DE INGENIERÍA

PROBLEMAS Y SOLUCIONES EN LA IMPLEMENTACIÓN DE EXTREME PROGRAMMING

A/I Ariel Erlijman Piwen

A/I Alejandro Goyén Fros

Memoria de Grado presentada como requisito parcial para la obtención del Grado de Ingeniero en Informática

Tutor: Ing. Fernando Machado

Montevideo, abril de 2001

Agradecimientos

Queremos agradecer a Fernando Machado, quien fue nuestro tutor, por habernos guiado durante todo este trabajo. Su experiencia en el área de desarrollo de software trajo invalorables aportes.

Las siguientes personas dedicaron parte de su tiempo libre a la lectura de este trabajo y se lo agradecemos enormemente: Daniel Conte, Cristian Inthamoussu, Jorge Roure y Pablo Salomón.

Queremos también agradecer a nuestras familias, amigos y novias por apoyarnos durante todo este tiempo. Sin ustedes, el viaje no hubiese sido divertido.

Finalmente, le agradecemos a los creadores de Extreme Programming por haber creado una metodología de desarrollo de software que tiene muy presente el lado humano de dicha actividad.

Dedicatoria

Dedico este trabajo a mi madre, a mi padre, a mi hermano y a Eliane por haberme acompañado y apoyado durante mi etapa académica. También se lo dedico a Marcelo Lichtenstein y a Alejandro Goyén, por haber compartido conmigo largas horas de estudio.

Ariel Erlijman Piwen

Este trabajo va dedicado a mis padres y a mi hermano.

Alejandro Goyén Fros

Índice

Glosario

AOP	Aspect Oriented Programming
BDUF	Big Design Up-Front
C3	Chrysler Comprehensive Compensation
CMM	Capability Maturity Model
FDA	Food and Drug Administration
FDD	Feature-Driven Development
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
MT	Metodologías de desarrollo Tradicionales
MVC	Model View Controller
OOPSLA	Object Oriented Programming Systems, Languages, and Applications
SEI	Software Engineering Institute
SPICE	Software Process Improvement and Capability dEtermination
TCR	Test-Code-Refactor
YAGNI	You ain’t gonna need it

1. LISTA DE TABLAS

2. LISTA DE FIGURAS

3. RESUMEN

Extreme Programming (XP) es una de las metodologías livianas para el desarrollo de software que ha tenido mayor repercusión y aceptación en los últimos años. Sin embargo, las organizaciones que han optado por utilizar esta metodología, se han enfrentado a diferentes problemas en lo que se refiere a su implementación. El propósito de este trabajo consistió en generar una guía práctica donde se exponen los problemas más comunes en la implementación de XP junto a las alternativas para solucionarlos. Diferentes foros de discusión en Internet fueron analizados para detectar los problemas más frecuentes y las soluciones propuestas. Los resultados de este análisis generaron una lista de 32 problemas que han surgido en la comunidad. Son presentados y analizados los seis problemas más frecuentes con el propósito de ayudar a futuras organizaciones en la adopción de XP.

4. ABSTRACT

In recent years, Extreme Programming (XP) has gained popularity as a successful software development methodology given its openness to change in today's world of dynamic business rules and requirements. However, the organizations that have adopted this methodology have experienced different problems in what refers to its implementation. The purpose of this work consisted in generating a practical guide that reveals the most common problems in the implementation of XP, together with alternative solutions. Different discussion forums on the Internet were analyzed to detect the most frequent problems and proposed solutions. The results of this analysis created a list of 32 problems that have appeared in the community. The six most frequent problems are presented and analyzed with the objective of helping organizations in the adoption of XP.

5. INTRODUCCIÓN

Al menos tres importantes estudios del estado de la ingeniería del software realizados a mediados de los noventa, indican que la mayoría de los equipos que desarrollan software no cuentan con metodologías o procesos que permitan cumplir con las expectativas de calidad, tiempo y alcance a costos razonables. Menos del 10% de los proyectos de software son entregados cumpliendo con los recursos inicialmente asignados y en la fecha previamente establecida **/ROY98/**.

Entre 1997 y 1999, la Government Accounting Office de los EEUU evaluó 210 proyectos y obtuvo los siguientes resultados:

- El 40% de los proyectos se extendieron el 67% del tiempo inicialmente establecido.

- El 30% de los proyectos se extendieron un 127% del presupuesto inicialmente establecido.

Por este motivo, los autores de este análisis comentan que al utilizar metodologías tradicionales se puede esperar que la mitad de los proyectos se atrasen y un tercio de ellos se excedan en presupuesto **/COL00/**.

Esto se debe en parte a la utilización de metodologías tradicionales de desarrollo que no se ajustan a la velocidad y constancia de los cambios de la actualidad. Los mercados de hoy son altamente dinámicos y competitivos, lo que hace que las reglas de negocio, tengan que cambiar continuamente para mantener a las organizaciones saludables y competentes. A partir de esta realidad, los equipos de desarrollo deben entregar funcionalidad rápidamente y de manera continua, brindando valor para el presente pero sabiendo que este valor se puede perder a corto plazo.

La tecnología, a menos que sea el fin de una organización en sí misma, se utiliza fundamentalmente para disminuir costos o aumentar la productividad. Hasta la década de los noventa, las organizaciones competían principalmente en sus habilidades de ofrecer productos y servicios, y aquellas que utilizaban una tecnología de soporte[1] cara y compleja de mantener en ese entonces, tenían una ventaja competitiva. Pero el mundo está cambiando; existen soluciones tecnológicas cada vez más económicas y sencillas, accesibles para cualquier organización. Las organizaciones hoy deben competir también en agilidad, o velocidad para ejecutar cambios. La tecnología de soporte, por consiguiente, deberá ser ágil, aceptando al cambio como algo constante. En lo que se refiere específicamente al software, deberá ser cada vez más maleable para poder acompañar la velocidad de cambio. Steve McConnell menciona que los proyectos de software exitosos en la actualidad son aquellos que tienen la mayor cantidad de cambios en etapas avanzadas del ciclo de desarrollo **/MCC96/**.

Uno de los supuestos universales de la ingeniería de software es que los costos de cambiar un programa crecen exponencialmente a lo largo del tiempo **/BOE88/**. Arreglar un problema cuesta un dólar durante la etapa de análisis de requerimientos pero corregirlo en etapas posteriores puede llegar a costar miles de dólares. Las *Metodologías de desarrollo Tradicionales* (MT), tales como el modelo en cascada **/ROY98/** y en espiral **/MCC96/**, están basadas en la validez de este supuesto, y por consiguiente, están diseñadas para determinar todos los requerimientos en las primeras etapas y no están preparadas para afrontar los cambios constantes. Como el costo del cambio crece exponencialmente, las MT intentan "congelar" los requerimientos y diseños en las primeras etapas del ciclo de desarrollo. Esto también se conoce como *Big Design Up-Front* (BDUF) y dificulta los cambios en etapas posteriores del ciclo. Por otro lado, las metodologías tradicionales consideran que ciertas actividades, como la documentación extensiva de los sistemas, son muy importantes para el futuro, pero en la práctica rápidamente pierden su valor ya que no logran acompañar la velocidad del cambio del software. Las MT se consideran

[1] Compuesta por el hardware y software de base.

"pesadas" ya que están basadas en complejos procedimientos y reglas que hacen que los integrantes del grupo tengan que dedicar gran parte de su tiempo en el cumplimiento del proceso, corriendo riesgo de perder el foco de atención en el producto a desarrollar.

Según Kent Beck[2], el problema básico del desarrollo de software es el riesgo **/BEC99/**. Como ejemplos de los riesgos a los que se está expuesto, se puede mencionar:

- Proyectos que se prolongan en tiempo retrasando las entregas.
- Proyectos que se cancelan por atraso o por falta de recursos.
- Sistemas que se entregan en fecha y forma pero con el correr del tiempo el costo del cambio es tan elevado que deben ser sustituidos por uno nuevo.
- Calidad insuficiente (demasiados defectos).
- En el momento en que el sistema se pone en producción ya no resuelve el problema de negocio para el que fue diseñado originalmente.
- Programadores que abandonan la empresa junto con el conocimiento que sólo ellos han adquirido.

Si bien las MT intentan minimizar el riesgo, a veces fallan por la velocidad de los cambios de la actualidad. Por otra parte, los creadores de estas metodologías se basaron en la idea que la construcción del software es análoga a construcciones tangibles como puede ser un puente o un edificio **/COL00/**. A raíz de esto, se adoptaron prácticas de varias disciplinas de la ingeniería que, si bien son aplicables al desarrollo de productos tangibles, no lo son para productos como el software. Cuando se construye un puente, los requerimientos y los diseños están claramente definidos antes de la etapa de construcción. En el caso del software, es posible conocer los requerimientos al comenzar el desarrollo pero los requerimientos al final pueden ser diferentes. Por otro lado, previo a la construcción de un puente se conoce

[2] Kent Beck y Ward Cunningham son los creadores y principales impulsores de Extreme Programming.

a fondo la naturaleza de los materiales, junto con leyes físicas y químicas que los gobiernan. Esto permite realizar estimaciones de construcción con mayor precisión. Pero en el caso del software, la estimación es difícil, ya que las "piezas de construcción" existentes, como la tecnología de componentes, no se han generalizado lo suficiente como para ser aplicables a cualquier desarrollo.

En estos últimos tiempos han aparecido una serie de metodologías "livianas" que intentan disminuir los problemas y riesgos mencionados anteriormente. Esto incluye, entre otras, la metodología Crystal Clear **/CRY01/**, Feature-Driven Development (FDD) **/FDD00/** y Extreme Programming (XP). Si bien cada una de estas metodologías sostiene sus propios principios, en esencia comparten algunos aspectos fundamentales **/COL00/**:

- *El desarrollo de software es una actividad humana*. El software en la actualidad sigue siendo construido por grupos de humanos y para humanos. Por este motivo, los procesos deben maximizar las fortalezas y complementar las debilidades de las personas involucradas, con la finalidad de crear ambientes colaborativos con comunicación honesta, fluida y abierta.

- *La única constante es el cambio*. Como fue explicado anteriormente, las reglas de negocio cambian a velocidades e intensidades cada vez mayores por razones competitivas. Los procesos de desarrollo deben estar diseñados para coexistir con el cambio.

Debido a las repercusiones del éxito de la utilización de XP en el sistema de pago de sueldos de la Daimler Chrysler denominado *Chrysler Comprehensive Compensation* (C3) **/CHR98/**, XP fue el tema de moda en 1999 en lo que respecta a metodologías de desarrollo. Cientos de artículos se han escrito desde entonces en publicaciones del *Institute of Electrical and Electronics Engineers* (IEEE), sitios en Internet como ComputerWorld o C/Net y ha sido tema de debate y de mayor repercusión en los últimos eventos del *Object Oriented Programming Systems, Languages, and*

Applications (OOPSLA). Según Tom Demarco[3], "XP es el movimiento de hoy más importante en el campo del desarrollo de software. Será algo tan esencial a la generación actual como lo fue el *Software Engineering Institute* (SEI) y el *Capability Maturity Model* (CMM) a la generación anterior" **/FOW00/**.

Los primeros libros y artículos publicados sobre XP, tratan principalmente de explicar en qué consiste la metodología y cómo funcionan cada una de sus técnicas. No existía una publicación que pueda servir como guía a los futuros equipos de desarrollo que implementarán XP, donde puedan encontrar los problemas que otros equipos han tenido y las formas en que los han solucionado. Crear una guía de implementación ahora es posible gracias a que XP ya tiene dos años de maduración y ha sido adoptada por gran cantidad de equipos de desarrollo en todo el mundo **/CUN01/ /XPO01/**.

Desde la formalización de XP como metodología en 1999, los equipos de desarrollo que la han adoptado, se han enfrentado con dificultades en su implementación. Esto se debe a que las 12 prácticas de dicha metodología, como ser la programación en pareja, el "testing" continuo, la integración continua, etc., no sólo tienen sus problemáticas individuales, sino también la problemática que surge debido a sus interdependencias.

Por lo tanto, es necesario saber cuáles son los problemas que los equipos de desarrollo se han enfrentado en sus primeros años de aplicación, para poder ayudar a implantar XP correctamente.

El objetivo de este trabajo ha sido crear una guía que muestre los problemas más comunes que los equipos de desarrollo han enfrentado al implementar XP, y una serie de alternativas que puedan solucionarlos. Los problemas y soluciones han sido extraídas a partir de los foros de discusión de Internet que tratan sobre el tema. También han sido utilizadas las experiencias de terceros, recopiladas de libros y

[3] Tom Demarco es autor de *Peopleware* y otros libros relacionados con la ingeniería del software.

artículos existentes. Estas alternativas fueron comentadas y analizadas en los foros de discusión donde participan los creadores de XP y muchos de los escritores de los artículos.

El formato de esta guía es similar a los utilizados en las guías de patrones y está descrito en el capítulo de la *Metodología*. De esta forma, los futuros equipos de programadores que utilicen XP, podrán identificar rápida y claramente el problema y aplicar las soluciones recomendadas.

Se pretende con este trabajo que los equipos de desarrollo que opten por utilizar XP tengan una guía de problemas a los que pueden enfrentarse, junto a sus posibles soluciones. De esta forma, se ayudará a lograr un éxito más rápido en los grupos que decidan adoptar XP.

6. INTRODUCCIÓN A EXTREME PROGRAMMING

Kent Beck, uno de los creadores de XP, define a Extreme Programming como "*una forma de desarrollar software: liviana, eficiente, de bajo riesgo, flexible, predecible, científica y divertida*". **/BEC99/**

XP está pensado para equipos pequeños, de hasta 10 personas, encargados de desarrollar software en proyectos cuyos requerimientos son ambiguos o muy volátiles. Su aplicabilidad en un equipo con mayor cantidad de integrantes o bajo otras condiciones aún está en discusión.

La metodología XP está compuesta por 12 prácticas fundamentales:

1. El **Juego de la Planificación** es la práctica que define la forma general de trabajar. Está compuesta por la *Planificación del "Release"* y por la *Planificación de la Iteración*.

   En la *Planificación del "Release"* se define qué es lo que se pretende tener como producto en un período de 4 o 6 meses. El cliente[4] define una gran cantidad de requerimientos, llamadas Historias, que son los que le dan mayor valor al negocio y que deben ser implementados dentro de ese período. Una vez definido el conjunto de Historias, éstas son analizadas y estimadas por el grupo de programadores para que finalmente el cliente las ordene en función de su valor. Cuando se tienen ordenadas las Historias, se procede a elegir aquellas cuya suma del tiempo de desarrollo no supere el período del "release".

[4] Cliente es uno de los roles definidos por XP y es explicado en detalle más adelante en esta sección.

En la *Planificación de la Iteración*, se definen las actividades para las siguientes 3 o 4 semanas. Teniendo en cuenta la capacidad productiva del grupo de desarrollo para la iteración, denominada *Velocidad*, el cliente elige el conjunto de Historias de mayor valor para que sean implementadas en la iteración planeada. A continuación, los programadores dividen las Historias en tareas más pequeñas, denominadas Tareas de Ingeniería. Luego, cada programador elige las tareas que desea implementar, las analiza en mayor detalle y realiza una estimación de su tiempo de desarrollo. Finalmente, el cliente ordena en función de sus necesidades las Historias estimadas, dejando para iteraciones posteriores aquellas que sobrepasen la capacidad productiva de la iteración.

El Juego de la Planificación del *"Release"* y de la *Iteración* se puede resumir en la siguiente figura:

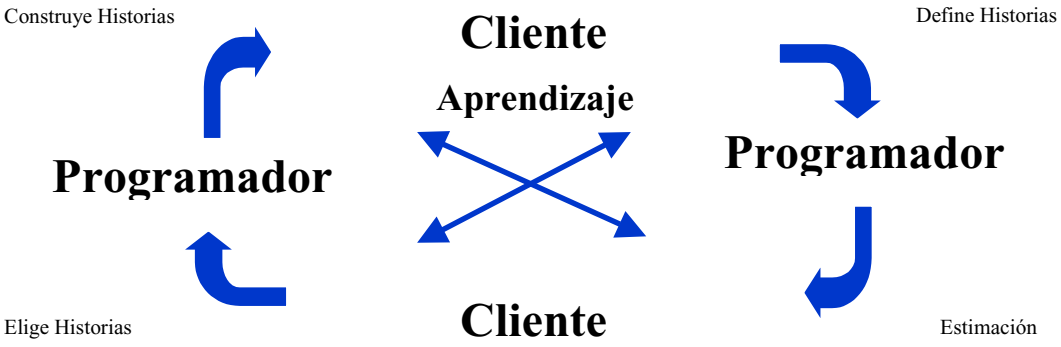

Figura 1 – El juego de la planificación[5]

Además de la mecánica del proceso explicada anteriormente, es importante hacer notar el aprendizaje que tienen los programadores en la realización de estimaciones al tener que repetirlas en períodos cortos de tiempo. También es importante resaltar el aprendizaje que tiene el cliente en la definición de las Historias, para que sean más claras para los programadores.

[5] Ilustración adquirida de /**JEF00/.**

2. Las **Entregas Frecuentes** permiten que el sistema empiece con algo simple y se ponga en producción rápidamente, para luego evolucionar a través de actualizaciones e incorporación de funcionalidad frecuente. Estas actualizaciones son realizadas en base a las prioridades establecidas por el cliente durante la *Planificación de la Iteración*. Las entregas frecuentes se dividen en períodos denominados iteraciones. Se recomienda que las iteraciones sean cortas y que no duren más de 3 o 4 semanas.

3. La **Metáfora** es una descripción general del sistema, que se establece al comenzar el proyecto, que fortifica la integridad conceptual, ayuda a guiar el proceso de desarrollo y mantiene una visión unificada entre los actores. La Metáfora determina un estándar en el vocabulario que será utilizado por los programadores y el cliente, que luego ayudará a establecer las clases y métodos del sistema.

4. Los **Diseños Simples** hacen que los sistemas desarrollados con XP sean creados de la manera más sencilla, pero cumpliendo con la funcionalidad que el cliente especificó en el Juego de la Planificación. XP le resta importancia a las necesidades desconocidas y especulativas del futuro y sólo atiende las necesidades actuales del cliente. Cabe aclarar que esto no quiere decir que los diseños sean de baja calidad, sino que se empieza por lo más sencillo que funcione[6] y luego se transforma en algo más complejo si el diseño demuestra insuficiencias. La complejidad innecesaria debe ser eliminada ni bien se descubra.

5. El **Testing Continuo** exige que los equipos XP validen el funcionamiento del software en todo momento. XP define dos tipos de test. Por un lado, los programadores diseñan y ejecutan los Test de Unidad previo a la implementación, mientras que el cliente diseña y ejecuta los Test de

[6] En la literature de XP este concepto se define como “Do the simplest thing that could possibly work”.

Aceptación. Los Test de Aceptación le permiten al cliente asegurarse que se ha desarrollado la funcionalidad negociada durante el Juego de la Planificación. Cada funcionalidad del sistema (Historia) debe tener por lo menos un Test de Aceptación asociado.

6. El **Refactoring** se define como "el proceso de alterar un sistema computacional de tal forma de mejorar su estructura interna sin alterar el comportamiento externo" **/FOW00/**. La incorporación de esta práctica, permite que los diseños del sistema se vayan perfeccionando continuamente durante todo el proceso de desarrollo, sin atarse a un diseño preliminar rígido como en el caso de las metodologías tradicionales. A diferencia de otras metodologías, XP acepta que en realidad lo único constante es el cambio y se adapta a coexistir junto a él. Aplicando esta práctica de forma continua, XP apunta a que el software se pueda mejorar y modificar con facilidad.

7. La **Programación en Pareja** exige que toda la programación y los test se realicen de a dos programadores por computadora. Hay experimentos **/WIL00/** que demuestran que la programación en pareja produce mejor software a un costo igual o menor que la programación individual **/CUT00/**.

8. La **Propiedad Colectiva del Código** hace que ninguna porción del código tenga programadores "dueños". Esto aumenta la velocidad de desarrollo ya que cuando se necesita algún cambio, cualquier programador lo puede hacer sin depender de los otros.

9. La **Integración Continua** indica que los equipos XP deben integrar el software construido diariamente. Esto minimiza el riesgo de enfrentar severos problemas de integración, vistos en proyectos que no integran con frecuencia.

10. Para mantener al equipo saludable, descansado y aumentar la productividad y la efectividad, XP propone **Semanas de 40 Horas de Trabajo**.

11. La **Presencia del cliente On-Site** permite que el proyecto sea guiado por un individuo dedicado, con el poder de decisión necesario para determinar los requerimientos y las prioridades de entrega. El efecto de la presencia continua en el lugar de desarrollo, hace que la comunicación sea fluida, con menos necesidades de documentación por escrito y permite resolver rápidamente las dudas y decisiones que puedan aparecer.

12. Para que un equipo pueda trabajar de forma efectiva y pueda compartir el código de forma colectiva, los programadores deben ponerse de acuerdo en establecer un estilo en común mediante una serie de reglas que permitan **Estandarizar el Estilo de Programación**. Esto incluye la estandarización de nomenclaturas de variables, formato común para comentarios dentro del código, etc.

Según Kent Beck, todas estas prácticas interactúan y se refuerzan mutuamente, de manera que el éxito del proceso estará dado por la puesta en marcha de todas las prácticas **/BEC99/**.

A diferencia de otras metodologías, XP no requiere que se genere ningún tipo de documentación formal a lo largo del proceso de desarrollo. Se menciona que la documentación del sistema es el propio código fuente junto a los test que validan su funcionamiento.

La flexibilidad de esta metodología permite adaptarla a las necesidades de la organización. De esta forma, se pueden incorporar las prácticas que den resultado y modificar aquellas cuya especificación original no funcione de forma adecuada.

En XP se pueden realizar estimaciones cada vez más acertadas, debido al proceso iterativo utilizado. Esto permite predecir con mayor exactitud que otras metodologías las fechas de entrega, ayudando a los programadores y al cliente manejar mejor los tiempos del desarrollo.

La siguiente figura compara a XP con otras metodologías de desarrollo de software, destacando que en XP, el análisis, el diseño, la implementación y el "testing" se realizan de forma continua:

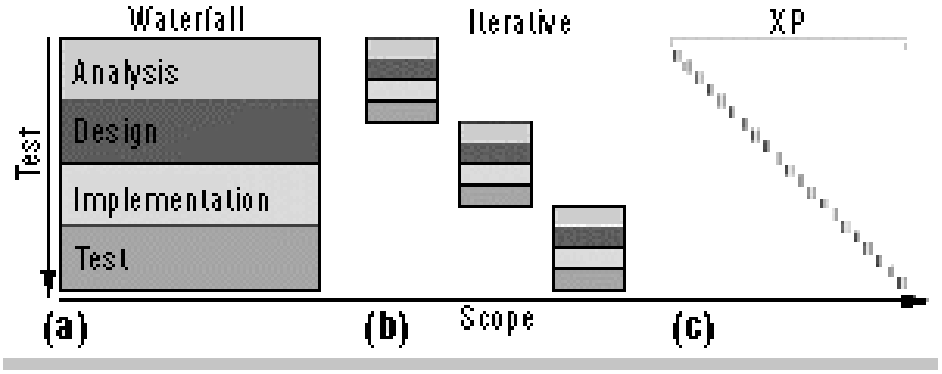

Figura 2 – XP y las metodologías tradicionales[7]

Las 12 prácticas mencionadas, ayudan a reforzar los 4 valores fundamentales de XP:

Simplicidad

XP le pide a cada miembro del equipo que construya lo más sencillo que funcione hoy. XP se basa en hacer algo simple hoy y crear un ambiente donde el costo del cambio sea lo más bajo posible.

Comunicación

XP se enfoca en el entendimiento persona-a-persona de los problemas que se van presentando en el ciclo de desarrollo, minimizando la documentación formal y maximizando la interacción cara-a-cara. A modo de ejemplo, se puede mencionar que a través de la programación en pareja se aumenta la comunicación entre los programadores, mientras que la presencia del cliente On-site, facilita el intercambio de ideas con el equipo de desarrollo.

---

[7] Ilustración adquirida de **/CUT00/.**

Retroalimentación

Los programadores obtienen el estado del software minuto a minuto a través de los Test de Unidad y la Integración Continua. El cliente consigue ver el estado del proyecto a lo largo de la iteración a través de los Test de Aceptación.

Coraje

Todo cambio requiere coraje. Algunos definen al coraje como la capacidad de hacer lo correcto aún con la presión de hacer algo distinto. En muchos casos, los integrantes de un grupo toman las decisiones equivocadas no por falta de coraje sino por falta de convicción. Los valores del equipo deben estar fuertemente alineados. XP intenta crear un soporte liviano con mucha atención a los valores de comunicación, sencillez y retroalimentación de forma que la confianza y el coraje entre los actores fluyan naturalmente.

En XP se definen los siguientes roles:

Cliente

El Cliente determina la funcionalidad que se pretende en cada iteración y define las prioridades de implementación según en el valor de negocio que aporta cada Historia. El Cliente también es responsable de diseñar y ejecutar los Test de Aceptación.

Programador

El Programador es responsable de implementar las Historias solicitadas por el cliente. Además, estima el tiempo de desarrollo de cada Historia para que el cliente pueda asignarle prioridad dentro de alguna iteración. Cada iteración incorpora nueva funcionalidad de acuerdo a las prioridades establecidas por el cliente. El

Programador también es responsable de diseñar y ejecutar los Test de Unidad del código que ha implementado o modificado.

Tracker

Una de las tareas más importantes del Tracker, consiste en seguir la evolución de las estimaciones realizadas por los programadores y compararlas con el tiempo real de desarrollo. De esta forma, puede brindar información estadística en lo que refiere a la calidad de las estimaciones para que puedan ser mejoradas.

Otra de las tareas que merece ser señalada, consiste en visitar a todos los programadores durante la iteración y analizar cuánto tiempo de trabajo le falta para implementar sus Historias y cuánto es que se había estimado para ellas. Con esta información, pueden tener una idea global del progreso de la iteración y evaluar las acciones que se deben tomar.

Con la información mencionada y muchos otros datos que el Tracker colecta, se generan variados reportes estadísticos con el objetivo de mejorar el proceso.

Coach

El Coach es responsable del proceso en general. Se encarga de iniciar y de guiar a las personas del equipo en poner en marcha las 12 prácticas. Es usualmente una persona con mucha experiencia en el desarrollo utilizando XP.

Manager

El Manager se encarga de organizar las reuniones (Ejemplo: Planificación de la Iteración, Planificación del "Release"), se asegura que el proceso de desarrollo se esté cumpliendo y registra los resultados de las reuniones para ser analizados en el futuro. Es de alguna forma el que responde al inversionista en lo que respecta a la evolución del desarrollo.

Los conceptos presentados son un breve resumen de Extreme Programming. Para obtener mayor información se recomienda consultar los libros y los artículos citados en la *Bibliografía.*

7. METODOLOGÍA DE TRABAJO

El método de trabajo utilizado se divide en 6 etapas:

7.1 Detección de fuentes de información

La primera etapa del trabajo consistió en detectar las fuentes con información general sobre Extreme Programming. Al mismo tiempo, se detectaron fuentes que revelaron los problemas de implementación de organizaciones que ya habían adoptado esta metodología.

Las fuentes de información encontradas se dividen en 2 grupos: literatura e Internet.

Literatura

Los libros publicados hasta la fecha sobre Extreme Programming son los siguientes:

- Extreme Programming Explained **/BEC99/**
- Extreme Programming Installed **/JEF00/**
- Planning Extreme Programming **/FOW00/**
- Extreme Programming Explored (Draft) **/WAK01/**
- Refactoring – Improving the design of existing code **/FOW99/**

Internet

En Internet se detectaron las siguientes fuentes:

- Artículos y "papers": Se recopilaron decenas de artículos y "papers" sobre XP. La lista completa de documentos utilizados se encuentra en la *Bibliografía* de este trabajo.

- Foros de discusión: Se detectaron 2 foros en Internet dedicados exclusivamente a discutir temas relacionados con Extreme Programming: el sitio de XP en Yahoo Groups **/YAH01/** y ExtremeProgrammingRoadmap en *Cunningham and Cunningham Inc.* **/WIK01/**.

7.2 Proceso de investigación

Lectura de libros, artículos y "papers"

La lectura de libros permitió comprender el marco teórico detrás de XP. Esto incluye el entendimiento a fondo de las 12 prácticas junto con los valores y roles de XP. Por otro lado, los "papers" y artículos sirvieron para reforzar ideas y también como material de referencia.

Participación en los foros de discusión – Registro de problemas y soluciones

La participación en los foros permitió extraer los problemas de implementación presentes en la comunidad, junto a la diversidad de opiniones y soluciones planteadas. Es importante destacar que tanto el sitio de XP en Yahoo Groups como el sitio ExtremeProgrammingRoadmap en *Cunningham and Cunningham Inc.,* cuentan con la participación activa de las figuras claves de XP, tales como Kent Beck, Ron Jeffries y Ward Cunningham[8]. Esto permitió recolectar soluciones de los propios creadores a los problemas planteados.

El registro de los problemas y soluciones se realizó entre Octubre de 2000 y Marzo de 2001. La comunidad de desarrollo continúa participando en los foros diariamente, presentando nuevos problemas y proponiendo soluciones, pero por restricciones de

[8] En este trabajo, se denominan como figuras claves de XP a Kent Beck, Ward Cunningham, Ron Jeffries, Martín Fowler y a los otros autores de libros sobre XP.

tiempo sólo se analizaron los problemas y soluciones comprendidos entre estas fechas.

Durante la etapa de investigación se leyeron más de 100 e-mails diarios y se analizaron más de 1000 páginas en el sitio ExtremeProgrammingRoadmap de *Cunningham and Cunningham Inc*. En este análisis, se detectaron más de 50 problemas que estaban relacionados con la implementación de XP. Después de un análisis más detallado, se encontró que algunos de ellos eran por errores conceptuales y fueron descartados ya que la solución se encuentra en la propia literatura de XP. Fueron de particular interés aquellos problemas que los individuos o grupos se enfrentaron al implementar una o más de las 12 prácticas de la metodología. Con esta consideración, la cantidad de problemas de implementación se redujo a 32.

La lista completa de problemas encontrados se encuentra en la sección *Resultados*.

Problemas enfrentados durante el proceso de investigación

El proceso de investigación tuvo mayores dificultades en la extracción de problemas y soluciones, principalmente por la estructura de información de los foros de discusión.

Yahoo Groups

En el caso particular del sitio de XP en Yahoo Groups, los mensajes se introducen al foro a través del correo electrónico y cada uno posee un identificador único. Debido a la cantidad de e-mails que pueden llegar a ser enviados en este tipo de foros, existe una opción en la configuración que permite recibir grupos de 25 mensajes por e-mail. Sin embargo, este modo de recepción no incluye el identificador único del mensaje, dificultando su localización en el sitio en Internet para referenciarlo.

La información se encuentra en un formato no estructurado, lo que dificulta enormemente la búsqueda. Esto obligó a que se hicieran lecturas detalladas de cada mensaje para poder detectar potenciales problemas y soluciones. Por otro lado, la herramienta de búsqueda del sitio en Internet no acepta la utilización de filtros lógicos, por ejemplo AND y OR, y esto dificulta enormemente localizar información por palabras claves.

`ExtremeProgrammingRoadmap en Cunningham and Cunningham Inc.`

El sitio ExtremeProgrammingRoadmap en *Cunningham and Cunningham Inc.* no tiene una estructura fija debido a la propia naturaleza de la herramienta utilizada (Wiki). Además, la información no se encuentra "estática" sino que crece y evoluciona diariamente. El sitio no tiene un modelo de seguridad, permitiéndole a cualquier persona alterar la información en cualquier momento y de forma anónima. Si bien este modelo abierto promueve la investigación y el crecimiento del conocimiento sin barreras, dificulta enormemente buscar información pertinente a un determinado tema.

7.3 Definición de un criterio de ponderación

Durante el proceso de investigación, surgió la necesidad de crear una función matemática que permitiese cuantificar la popularidad de los 32 problemas encontrados. De esta forma se logró determinar cuáles eran los problemas más comunes de la comunidad.

Para poder desarrollar la función de ponderación, primero se determinaron las variables que pudieran incidir en el grado de popularidad de los problemas planteados.

Las variables establecidas fueron las siguientes:

- *Número de personas que participan en el problema (PP)*: Cantidad de personas distintas que participan en el planteo del problema.
- *Número de personas que participan en la solución (PS):* Cantidad de personas distintas que participan en una o más soluciones.
- *Número de prácticas afectadas (PA)*: Esta variable indica la cantidad de prácticas de XP que son afectadas por el problema.
- *Cantidad de soluciones propuestas (CS)*: Número de soluciones diferentes para resolver el problema.
- *Figuras claves en la solución (PC)*: Indica si las figuras claves de XP participan en alguna de las soluciones propuestas. El número "1" indica que participaron figuras claves, mientras que el "0" indica que no participaron.
- *Cantidad de soluciones probadas experimentalmente (CSP)*: De todas las soluciones propuestas, algunas son especulativas y otras están basadas en experiencias prácticas anteriores. Esta variable indica la cantidad de soluciones que han sido probadas experimentalmente.

Una vez detectadas las variables, se le asignaron pesos a cada una de ellas. Como el objetivo era crear una función que cuantifique la popularidad de cada problema, se le asignó mayor peso a aquellos problemas que tuvieron mayor actividad en los foros. La actividad puede ser tanto en el aporte de un nuevo problema como en el planteo de una solución. Por lo tanto, se le asignó más peso a las variables que midieron la cantidad de problemas y soluciones planteadas, así como también a aquellos problemas que resultaron con mayor diversidad de soluciones.

Los pesos asignados a cada una de las variables se indican en la siguiente tabla:

Variable	Peso asignado
PP	25%
PS	25%
PA	10%
CS	25%
PC	5%
CSP	10%

Tabla 1 – Ponderación de las variables

Cuando se finalizó la lista de problemas con mayor actividad en la comunidad, como parte de este trabajo, los autores crearon una página **/XPI01/** en el sitio ExtremeProgrammingRoadmap de *Cunningham and Cunningham Inc.* El objetivo de esta página fue listar los problemas más populares para que la comunidad participe y opine. Se agregó una página de discusión en ExtremeProgrammingRoadmap para aquellos problemas que no la poseían. Además, se incluyeron las referencias a los mensajes en los foros de discusión del sitio de XP en Yahoo Groups y a las páginas existentes en ExtremeProgrammingRoadmap de *Cunningham and Cunningham Inc.,* donde estos problemas fueron detectados, para poder ser consultados. Como resultado de esta publicación, se logró obtener nuevos comentarios y nueva información en los problemas a ser analizados, que complementaron de muy buena forma la información que ya se poseía.

7.4 Definición de formato de presentación

Una vez determinados los problemas de implementación más comunes, se definió el formato de presentación para la guía de problemas y soluciones. Si bien estaba definido desde un principio que se iba a utilizar una estructura similar a la de los patrones, existen varias propuestas en lo que se refiere al formato de los mismos.

La estructura elegida fue basada en el artículo *Patterns in a Nutshell* escrito por Brad Appleton **/APP00/**. Los patrones fueron estructurados con los siguientes elementos:

- *Nombre*: Nombre asignado al patrón. Permite identificarlo unívocamente.
- *Problema*: Narración del problema. Explica el "qué" del problema.
- *Contexto*: Esta sección explica "cómo" y "cuándo" ocurre el problema. Dicho de otra manera, limita el problema en el espacio y en el tiempo.
- *Fuerzas*: Fuerzas que motivan la resolución del problema.
- *Solución:* Esta sección explica cómo generar la solución, así como la propia estructura de la solución, sus participantes y colaboraciones.
- *Contexto resultante (opcional):* En esta parte se describe el resultado final que se logra luego de aplicar la solución, junto a sus beneficios y consecuencias.
- *Explicación (opcional):* Contiene los principios subyacentes o heurísticas que justifican la solución y por qué ella funciona.

Cabe aclarar que la guía desarrollada estará compuesta por proto-patrones[9] ya que ninguna de las soluciones propuestas en los foros tienen suficientes pruebas experimentales para comprobar su validez.

7.5 Desarrollo de ejemplo

Con la finalidad de poner a prueba el formato de presentación elegido, se desarrolló el proto-patrón de "Adoptando XP en sistemas legados" de forma completa. El ejemplo fue presentado en el Seminario de Memoria de Grado en Marzo de 2001.

[9] Por más información sobre los proto-patrones consultar **/APP00/**.

7.6 Desarrollo del trabajo completo

En esta etapa se desarrollaron los 5 patrones restantes según el formato establecido. Además, se generó un CD con los artículos, "papers", secciones de ExtremeProgrammingRoadmap de *Cunningham and Cunningham Inc.*, mensajes del sitio de XP en Yahoo Groups y planillas de cálculo utilizadas, para acompañar este trabajo.

8. RESULTADOS

8.1 Lista de problemas detectados en la implementación de XP

A continuación se presentan los 32 problemas detectados en la comunidad en formato pregunta-explicación, ordenados según la función de ponderación descrita en la sección anterior. No se pretende dar solución a todos los problemas ya que esto excede los límites de este trabajo. Se incluyen las referencias encontradas en los foros de discusión en caso de que el lector quiera profundizar en algún problema específico.

No se incluyen comentarios adicionales sobre los primeros 6 problemas ya que serán analizados detenidamente más adelante en este trabajo.

1. **¿Cómo se debe adoptar XP en sistemas legados?**

2. **¿Qué se debe hacer para que los Test de Unidad puedan acceder a los métodos privados?**

3. **¿Cuál debe ser el orden de implementación de las 12 prácticas de XP?**

4. **¿Cómo se deben testear las entradas y salidas de los GUI? ¿Se pueden automatizar estas pruebas?**

5. **¿Qué tipo de actividades pueden realizar aquellos programadores que han finalizado el trabajo planificado para la iteración?**

6. **¿Qué tipo de estrategia Test-Code-Refactor (TCR) mejor se ajusta a la metodología Extreme Programming?**

7. **¿Los defectos encontrados deben ser ingresados como Historias en la iteración? ¿Cómo se debe definir la prioridad entre la corrección de defectos y el desarrollo de nueva funcionalidad?**

   La metodología XP no explica una forma de administrar los defectos que no hayan sido descubiertos en los Test de Unidad o en los Test de Aceptación, ni la forma en que entran en el Juego de la Planificación.

   Referencias:

   http://c2.com/cgi/wiki?ExtremeProgrammingChallengeTwenty

8. **¿La Metáfora de XP es un sustituto válido a lo que tradicionalmente se conoce como la arquitectura de un producto de software?**

   Los requerimientos no funcionales, tales como la seguridad, la escalabilidad y el desempeño, deben ser soportados estructuralmente y se definen en las etapas de arquitectura de las metodologías tradicionales. Como XP no tiene una etapa de aquitectura, no queda claro cómo ni cuándo entran estos requerimientos en el ciclo de desarrollo.

   Referencias:

   http://c2.com/cgi/wiki?ExtremeProgrammingMetaphorVsArchitecture

9. **¿Cómo debe ser interpretado el concepto de simplicidad en XP?**

   En la comunidad existe confusión en lo que se refiere al concepto de simplicidad desde la perspectiva de XP. ¿Cuándo es un diseño más simple que otro? Algunos opinan que la simplicidad está dada por la menor cantidad de LC (líneas de código) posible, mientras otros opinan que las dependencias entre clases se deben minimizar para lograr mayor simplicidad.

   Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingSimplicityDefinition
http://c2.com/cgi/wiki?WhatIsSimplest
http://c2.com/cgi/wiki?DoTheSimplestThingThatCouldPossiblyWork
http://c2.com/cgi/wiki?ExtremeProgramming
http://groups.yahoo.com/group/extremeprogramming/message/15572
http://groups.yahoo.com/group/extremeprogramming/message/15585
http://groups.yahoo.com/group/extremeprogramming/message/15903

10. ¿Es aceptable desarrollar la documentación una vez terminado con el desarrollo o debe ser desarrollada en paralelo?

Cualquier sistema de mediana complejidad requiere manuales para los usuarios. Por este motivo, en muchos casos el cliente solicita que además del producto de software sean entregados manuales de usuario y otros tipos de documentación.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingChallengeTwentyOne

11. ¿Es viable utilizar a XP en la construcción de cualquier tipo de software?

Hay ciertos tipos de desarrollos que si bien tienen algún usuario final, son muy difíciles de especificar. Esto incluye el desarrollo de compiladores, aplicaciones de “proceso por lotes” y sistemas embebidos. Por lo tanto, hay desarrollos que no pueden estar centrados en el usuario, pero por otro lado XP requiere la presencia del cliente On-Site.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingChallengeOne
http://c2.com/cgi/wiki?WhenXPisUnpopular

12. ¿Hay que administrar las dependencias entre los componentes?

En todo producto de software existen dependencias entre sus componentes. Las dependencias pueden ser físicas o lógicas. Algunos participantes de los foros de discusión opinan que la implementación continua de soluciones rápidas y sencillas resuelven los problemas a corto plazo, pero aumentan las dependencias entre sus componentes. Esto sucede hasta un punto en que el producto pierde su integridad conceptual y se dificultan las modificaciones posteriores.

Referencias:

http://c2.com/cgi/wiki?DoTheSimplestThingThatCouldPossiblyWork
http://c2.com/cgi/wiki?ContinuousIntegration
http://c2.com/cgi/wiki?YouArentGonnaNeedItDiscussion
http://c2.com/cgi/wiki?YouArentGonnaNeedItOriginalPage

13. ¿Se puede utilizar XP para desarrollar software a gran escala?

El tema de la escalabilidad de XP ha desatado mucho debate en los foros de discusión. Si bien los límites en lo que refiere a cantidad de personas en el proyecto o a su complejidad fueros esbozados por Kent Beck en su primer libro de la serie, no está probado que XP funcione o deje de funcionar en desarrollos a gran escala.

Referencias:

http://c2.com/cgi/wiki?HundredPersonProject
http://groups.yahoo.com/group/extremeprogramming/message/15815
http://groups.yahoo.com/group/extremeprogramming/message/15832

14. ¿Es recomendable utilizar XP en proyectos que exigen documentación formal?

Existen casos en donde el cliente exige documentación formal adicional. Esto puede incluir documentos de especificación de requerimientos, diagramas UML, Pert, Gantt, etc.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingChallengeNine

15. ¿Cómo se debe aplicar XP en organizaciones en las que los requerimientos son especificados por más de una persona?

En un proyecto de software se puede requerir de personas de distintas áreas, tales como Marketing, Operaciones, etc. De acuerdo a los creadores de XP, la voz del cliente debe ser una sola pero hay casos en donde esto no es posible.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingMultipleVoices
http://c2.com/cgi/wiki?HundredPersonProject
http://industriallogic.com/wp/exposures.html
http://c2.com/cgi/wiki?OnSiteCustomer
http://groups.yahoo.com/group/extremeprogramming/message/17146

16. ¿Cómo se pueden evitar las colisiones causadas por la edición simultánea del código fuente y cómo debe ser tratada en caso de que ocurra?

Una de las 12 prácticas de XP es la propiedad colectiva del código. Esto habilita a que los programadores puedan realizar cambios en cualquier parte del código cuando lo vean necesario. Los creadores de XP defienden que la adopción de esta práctica aumenta la velocidad de desarrollo, ya que cualquiera puede

realizar cambios sin impedimentos. Sin embargo, esto tiene la desventaja de aumentar la probabilidad de colisión entre los programadores.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingSourceCodeCollision
http://c2.com/cgi/wiki?ContinuousIntegration
http://c2.com/cgi/wiki?FrequentReleases
http://c2.com/cgi/wiki?CollectiveCodeOwnership
http://groups.yahoo.com/group/extremeprogramming/message/15211

17. ¿Se deben administrar las dependencias entre las Historias?

El cliente solicita nueva funcionalidad a través de las Historias. En algunos casos se presentan dependencias entre las Historias, desconocidas por el usuario ya que éstas se pueden perder de vista bajo su óptica.

En otra situación en donde la dependencia entre las Historias influye, es en organizaciones donde los gerentes solicitan diagramas Pert o Gantt para hacer seguimiento del proyecto de desarrollo. Pero para poder crear estos diagramas se deberán conocer y administrar las dependencias entre las Historias. El problema radica en que Extreme Programming considera que las Historias son entidades independientes.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingStoryDependencyManagement
http://c2.com/cgi/wiki?BeyondExtremeProgramming
http://c2.com/cgi/wiki?PlanningGame
Planning Extreme Programming, Capítulos 12 y 13

18. ¿Se puede utilizar XP cuando el equipo de desarrollo se encuentra distribuido geográficamente?

Hay situaciones en las que el equipo de desarrollo de software no se encuentra en un mismo espacio geográfico. Teniendo en cuenta que una de las prácticas que se utiliza en Extreme Programming es la Programación en Pareja, surge la interrogante en lo que respecta a cómo desarrollar en conjunto cuando las personas no comparten el mismo espacio físico.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingInDistributedEnvironments
http://c2.com/cgi/wiki?ExtremeProgrammingChallengeSix
Extreme Programming Explored (draft), Capítulo 4
http://groups.yahoo.com/group/extremeprogramming/message/16739
http://groups.yahoo.com/group/extremeprogramming/message/16758
http://groups.yahoo.com/group/extremeprogramming/message/16972
http://groups.yahoo.com/group/extremeprogramming/message/17231

19. ¿Cómo se debe calcular la Velocidad del Proyecto (Project Velocity) para la primera iteración?

Durante el Juego de la Planificación, se debe tener en consideración la velocidad de desarrollo de los programadores, para poder determinar la cantidad de Historias que se pueden implementar en cada iteración. En la comunidad existe confusión acerca de cómo calcular este coeficiente en la primera iteración de un proyecto.

Referencias:

http://c2.com/cgi/wiki?ExtremePlanning
http://groups.yahoo.com/group/extremeprogramming/message/16617
http://groups.yahoo.com/group/extremeprogramming/message/16890
Extreme Programming Installed, Capítulo 8

20. ¿Cómo es posible lograr la integración continua si el "testing" o la compilación del producto demora mucho tiempo?

El tiempo de compilación y linkedición puede ser demasiado extenso como para poder realizar varias integraciones diarias. Lo mismo puede ocurrir con los Test de Unidad o los Test de Aceptación.

Referencias:

http://c2.com/cgi/wiki?ContinuousIntegration

http://c2.com/cgi/wiki?YouArentGonnaNeedItDiscussion

21. ¿Qué alternativas se proponen cuando hay programadores que no logran concentración en áreas abiertas?

Los creadores de XP recomiendan que todos los programadores trabajen en un área abierta ya que esto incentiva la comunicación. Sin embargo, hay programadores que prefieren trabajar en oficinas privadas para lograr mayor concentración. Para estos individuos, las áreas abiertas de trabajo disminuyen enormemente su productividad.

Referencias:

http://c2.com/cgi/wiki?XpConceptsInAcceptedMethodologies

22. ¿Qué estrategia se debe utilizar para formar las parejas en XP?

Por lo general, en los equipos de desarrollo existen diferencias en cuanto a la experiencia y nivel de conocimiento técnico de sus integrantes. La programación en pareja genera la disyuntiva de juntar a dos personas con experiencia para resolver las Historias con mayor velocidad, o que los principiantes formen parejas con individuos que tienen más experiencia para aprender de ellos. Por otro lado, en el equipo pueden haber pocos individuos que dominen la tecnología necesaria para implementar ciertas Historias, por ejemplo C++, ASP, etc., generando una disyuntiva similar.

Referencias:

http://c2.com/cgi/wiki?PlanningGame

23. ¿Cómo se debe administrar el código de los Test de Unidad?

En la medida en que avanza un proyecto de desarrollo de software, los Test de Unidad evolucionan y se hacen cada vez más complejos, a tal punto que su mantenimiento puede exigir enormes cantidades de tiempo y esfuerzo. Si el código de test se hace complejo y no se simplifica a través del Refactoring, la velocidad de desarrollo decrece ya que los programadores deben dedicar parte del tiempo a su mantenimiento.

Referencias:

http://groups.yahoo.com/group/extremeprogramming/message/15913
http://groups.yahoo.com/group/extremeprogramming/message/16463
http://groups.yahoo.com/group/extremeprogramming/message/18375

24. ¿Cómo se deben distribuir los roles de XP en equipos pequeños?

En equipos de desarrollo pequeños se deben asignar los roles minimizando los problemas de compatibilidad. Si bien existen roles que pueden ser ocupados por el mismo individuo, como ser el de Tracker y Manager, existen otras combinaciones que no son compatibles.

Referencias:

http://c2.com/cgi/wiki?ExtremeRoles

25. ¿Es posible mantener el modelo de datos normalizado si la aplicación se desarrolla con XP?

En muchos casos las aplicaciones dependen de una base de datos para almacenar información. En la comunidad comentan que aquellas bases de datos que crecen evolutivamente según lo que propone la metodología XP sufren problemas de

normalización y redundancia, ya que el modelo de datos nunca fue analizado con detenimiento.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingSystem

http://c2.com/cgi/wiki?ExtremeProgrammingChallengeThirteen

http://c2.com/cgi/wiki?RefactoringWithRelationalDatabases

26. ¿Cómo se deben tratar los problemas de incompatibilidad de personalidades en la Programación en Pareja?

En la mayoría de los casos, los equipos de desarrollo están formados por personas con distintos tipos de personalidades. Esta variedad es positiva ya que las debilidades de algunos son contrarrestadas con fortalezas de otros. Pero por otro lado, esto trae consigo incompatibilidades que reduce la cantidad de parejas que se pueden formar para la Programación en Pareja.

Referencias:

http://c2.com/cgi/wiki?WhenXPIsUnpopular

http://groups.yahoo.com/group/extremeprogramming/message/16023

http://groups.yahoo.com/group/extremeprogramming/message/16631

27. ¿Cómo se deben testear las clases que son difíciles de aislar?

Hay clases que son muy difíciles de testear en aislamiento. Si bien en la mayoría de los casos es posible simular el entorno de la clase para validar su funcionamiento, hay otros casos en donde esto no es posible.

Referencias:

http://c2.com/cgi/wiki?UnitTests

28. ¿Cómo se debe determinar la frecuencia de integración?

La integración continua es una de las prácticas fundamentales de Extreme Programming. Si se integra regularmente, se pueden evitar problemas que ocurren en ambientes en donde la integración se hace con poca frecuencia. Pero por otro lado, la integración con excesiva frecuencia puede disminuir la velocidad de desarrollo.

Referencias:

http://c2.com/cgi/wiki?ContinuousIntegration
http://c2.com/cgi/wiki?CollectiveCodeOwnership
http://groups.yahoo.com/group/extremeprogramming/message/15191
http://groups.yahoo.com/group/extremeprogramming/message/15819
http://groups.yahoo.com/group/extremeprogramming/message/15936

29. ¿Qué se debe hacer cuando el cliente no está capacitado para diseñar los Test de Aceptación?

En XP el cliente es responsable de diseñar y ejecutar los Test de Aceptación. Pero en algunos casos, si bien el cliente cuenta con la experiencia de dominio para guiar el proceso de desarrollo, no cuenta con la capacitación o formación necesaria para diseñar estos test.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingChallengeTwentyTwo
http://groups.yahoo.com/group/extremeprogramming/message/16651

30. ¿Cómo se deben testear las aplicaciones "multi-threaded"?

En el desarrollo de aplicaciones "multi-threaded" surgen defectos propios de la programación concurrente. Estos defectos son difíciles de reproducir y corregir.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingChallengeFourteen

31. ¿Cómo se puede evitar el "Data Base (DB) Ownership"?

En el desarrollo y mantenimiento de sistemas grandes hay una tendencia a que existan figuras dueños de datos, responsables de diseñar y de mantener consistente al modelo de datos. Esto tiene la enorme desventaja de que el conocimiento deja de ser colectivo y se concentra en pocos individuos. Esto se conoce como "DB ownership" y se opone a los principios de XP.

Referencias:

http://c2.com/cgi/wiki?ExtremeProgrammingChallengeThirteen

32. ¿Es válida la utilización de sistemas informáticos para gestionar el Juego de la Planificación?

Los creadores de Extreme Programming recomiendan el uso de tarjetas para la especificación de las Historias y Tareas de Ingeniería. Pero al utilizar tarjetas físicas se presenta el riesgo de perderlas y se dificulta enormemente compartir información.

Referencias:

http://c2.com/cgi/wiki?DesignToolsForXP

http://groups.yahoo.com/group/extremeprogramming/message/15845

8.2 Lista de problemas analizados

Con la función ponderación descrita en la sección de la *Metodología,* se generó la lista de los 6 problemas de implementación más comunes:

#	Nombre del problema	PP	PS	PA	CS	PC	CSP	Valor
1	Adoptando XP en sistemas legados	5	9	3	4	1	2	**0.78**
2	Test de Unidad de métodos privados	5	9	1	3	0	2	**0.68**
3	Orden de implementación de las 12 prácticas de XP	1	2	12	7	1	1	**0.54**
4	Test de Unidad de interfaces de usuario gráficas (GUIs)	5	4	2	2	0	2	**0.52**
5	Actividades para el tiempo de holgura de los programadores	1	5	1	4	1	3	**0.49**
6	Estrategias Test-Code-Refactor (TCR)	2	6	1	2	1	2	**0.46**

Tabla 2 – Problemas de implementación de XP más comunes

A continuación se describen estos problemas y las soluciones propuestas.

8.3 Adoptando XP en Sistemas Legados

Nombre

Adoptando XP en Sistemas Legados

Problema

¿Cómo se debe adoptar XP en sistemas donde el código no está en su estado más simple, no hay Test de Unidad, sus programadores no se encuentran más en la organización y no se tiene documentación sobre su estructura?

Contexto

Organizaciones que han decidido adoptar XP y poseen sistemas legados. En estos sistemas hay que corregir fallas de funcionamiento e ir efectuando modificaciones para cubrir los nuevos requerimientos del negocio. Usualmente fueron desarrollados con lenguajes donde la modificación del código puede afectar negativamente las dependencias de contexto implícitas. Además, su estructura interna se fue deteriorando a través del tiempo, dejando código duplicado y código que no se usa.

Fuerzas

Al ser un sistema legado, el programador que lo modifica no tiene la confianza ni la certeza de qué es lo que realmente se puede ver afectado al realizar su modificación. Muchas veces, las personas que lo desarrollaron ya no trabajan para la empresa o el sistema pudo haber sido diseñado por terceros por lo que conocimiento interno es mínimo.

La mayoría de los sistemas actuales son legados.

Existen y se usan herramientas de ingeniería en reversa para facilitar la comprensión del código legado.

Existen y se usan herramientas de control de versión para facilitar la administración del cambio.

Solución

Es posible considerar realizar un gran Refactoring desde el comienzo, pero no es recomendable ya que la probabilidad de introducir errores aumenta. Por esto es aconsejable ir paso a paso y realizar pequeñas modificaciones cada vez. Por este motivo se recomienda seguir los siguientes pasos:

1. *Obtener una visión general del sistema*: Para comprender mejor el sistema legado, es muy recomendable la utilización de herramientas de ingeniería en reversa. Si existen dichas herramientas para el lenguaje del sistema legado, se puede obtener automáticamente documentación donde se muestran todas las dependencias internas al código. Muchas de estar herramientas, pueden encontrar automáticamente el código duplicado y extraerlo. Estas herramientas también pueden encontrar secciones del código que no son utilizadas, disminuyendo en gran medida el código que se debe comprender.

2. *Ponerse en contacto con el cliente*: Solicitar al cliente que genere las Historias correspondientes a los nuevos requerimientos, cambios en la funcionalidad existente y modificaciones para mejorar la estabilidad del sistema.

3. *Estimar las Historias:* Para poder estimar las Historias generadas por el cliente, es necesario identificar el código afectado. Como se menciona en el paso 1, las herramientas de ingeniería en reversa son de gran ayuda en estos casos. Sin un mínimo conocimiento del código, las estimaciones no serían correctas.

4. *Iteraciones:* El cliente ordena las Historias según su valor en el negocio y se distribuyen en las iteraciones. En cada iteración:

    - Escribir y correr los test necesarios para el código que se va a modificar.

    - Realizar un primer Refactoring sobre dicho código, utilizando en lo posible las herramientas de ingeniería en reversa mencionadas en el paso 1. De esta forma se remueve el código duplicado e información innecesaria. A continuación se corren los test para verificar que los cambios no han afectado otras secciones.

    - Una vez que el código está en una condición aceptable para ser modificado, se modifican los test para contemplar los nuevos requerimientos y se realizan los cambios correspondientes al código. Al finalizar, se corren los test nuevamente.

    - Para finalizar, se realiza el Refactoring, se testea nuevamente y se genera, en caso de ser posible, la nueva documentación del sistema con la herramienta de ingeniería en reversa.

5. *Versionado:* Los programas versionadores de código, pueden ser de mucha ayuda en caso de querer recuperar el estado anterior del sistema legado. Cuando se llega a una versión estable, hay que realizar el "check in" en la herramienta versionadora y etiquetarlo para tener una referencia. Cuando se hayan realizado varias pequeñas modificaciones, se debe generar una versión de "release" y se comienza de nuevo con el proceso.

Contexto Resultante

La productividad con XP sobre sistemas legados es menor que sobre sistemas desarrollados con XP desde el comienzo. Esto se debe a que (1) no hay test previos

para realizar modificaciones y Refactoring; (2) el código no ha sido sometido al Refactoring previamente; (3) puede haberse utilizado lenguajes que no favorecen las modificaciones debido a que no hacen énfasis en la modularidad y el encapsulamiento; (4) el código no se encuentra en su estado más simple por lo que el tiempo y las dificultades de entendimiento son mayores que lo usual.

Las mejoras de pequeñas partes del código hace que el sistema en general funcione mejor. Esto usualmente ocurre porque en sistemas legados el código está altamente acoplado y los cambios locales tienen efectos globales.

El efecto del Refactoring en el sistema legado, se va acumulado iteración tras iteración. El código es llevado en pequeñas partes a su estado más simple y se facilitan los cambios futuros.

La realización de Test de Unidad no sólo evita modificar secciones de código no deseadas sino que también le hace aumentar al programador el conocimiento sobre el sistema y la confianza para realizar cambios.

Explicación

Con los Test de Unidad se asegura que no se ha cambiado ningún comportamiento existente. Por otro lado, comprueba que el código modificado ha implementado la nueva funcionalidad correctamente.

Las herramientas automatizadas para la ingeniería en reversa han madurado lo suficiente en los últimos años, para soportar la planificación de modificación del código existente y la ejecución del cambio utilizando XP.

Referencias

- *Legacy to the extreme*
  Arie van Deursen – Tobias Kuipers – Leon Moonen
  http://www.cwi.nl/~{arie,kupers,leon}/

- *Cunningham and Cunningham Inc. http://www.c2.com*
  - http://c2.com/cgi/wiki?ExtremeProgrammingWithLegacyCode
  - http://c2.com/cgi/wiki?AddingFeaturesToLegacyCode
  - http://c2.com/cgi/wiki?UnitTestsForLegacyCode
  - http://c2.com/cgi/wiki?UnitTests
  - http://c2.com/cgi/wiki?LegacyCode
  - http://c2.com/cgi/wiki?RefactoringLegacyCode

- *Extreme Programming Installed – páginas 32, 51 y 196*
  Ron Jeffries – Ann Anderson – Chet Hendrickson

- *Yahoo Groups*
  - Thread: RelentlessTesting ContinuousIntegration & legacy C++ code
    http://groups.yahoo.com/group/extremeprogramming/message/15109
  - Thread: Adding Features To Legacy Code
    http://groups.yahoo.com/group/extremeprogramming/message/15560
  - Thread: What would you do?
    http://groups.yahoo.com/group/extremeprogramming/message/18187
  - Thread: Test-first even works for legacy code
    http://groups.yahoo.com/group/extremeprogramming/message/18358

8.4 Test de Unidad en métodos privados

Nombre

Test de Unidad en métodos privados

Problema

El Test de Unidad previo a la codificación es una tarea requerida en XP. Por este motivo, se han desarrollado "frameworks" para diferentes lenguajes para facilitar y automatizar estas pruebas. Durante los primeros años de utilización de estos "frameworks", en la comunidad de XP ha surgido la siguiente pregunta: ¿Qué se debe hacer para que los Test de Unidad puedan acceder a los métodos privados?

Contexto

Este proto-patrón está orientado principalmente a programadores que utilicen algún "framework" para Test de Unidad, en cualquier lenguaje de programación que permita la utilización de métodos privados. Las ideas aquí presentadas, también pueden ser de mucho interés para aquellos que desean lograr mayor conocimiento sobre el "testing" de los métodos privados.

En los foros de discusión de Internet, se encontraron muchas opiniones que se oponen a testear los métodos privados[10], por lo que el problema aquí planteado se aplica a programadores que consideran que los métodos privados deben ser testeados de igual manera que los métodos públicos.

Fuerzas

Muchos sugieren que el "testing" sea una de las primeras prácticas en ser implementadas en la adopción de XP[11].

---

[10] Por más información ver **Anexo** a continuación de esta sección.
[11] Ver proto-patrón de Orden de Implementación de Prácticas de XP en esta guía.

Un sistema diseñado utilizando buenas prácticas de diseño, tendrá clases con métodos públicos y privados.

Los que justifican el test de los métodos privados aseveran que muchas veces existen objetos con complejos métodos no públicos y que el test sobre los métodos públicos solamente, es impráctico.

Solución

- *Sin encapsulamiento y fácil de implementar*

  Como primera alternativa se puede considerar utilizar solamente métodos públicos. Como una mejora a este planteo, se considera marcar los métodos privados como “friend”, en los lenguajes donde esto es posible.

- *Con encapsulamiento y complejidad adicional*

  Como solución al problema de la violación del encapsulamiento, la alternativa más mencionada consiste en la utilización del patrón **PrivateInterface**. A través de esta solución probada y ampliamente utilizada, que consiste en la creación de clases abstractas para acceder a las partes internas de la clase, es posible aislar completamente las clases testeadas. Por más información sobre el uso de **PrivateInterface**, ver la referencia *Private Interface Pattern.*

- *Dependiente del lenguaje y complejidad adicional*

  La última alternativa sugerida consiste en extraer los métodos privados de las clases a métodos públicos en nuevas clases o subclases. Estas nuevas clases pueden ser accesibles desde cualquier otra clase o pueden ser internas (inner classes) en los lenguajes que lo permiten.

Contexto Resultante

Todas las alternativas propuestas solucionan el problema del test de los métodos privados. Algunas lo logran perdiendo el encapsulamiento y otras a través de agregar mayor complejidad al código.

Las alternativas denominadas *Sin encapsulamiento y fáciles de implementar,* si bien solucionan el problema planteado, permiten que cualquier otra clase tenga acceso a secciones de implementación que no fueron pensadas para el exterior. Si se puede acceder a las partes de la implementación, se estaría creando mayor acoplamiento y agregando mayor dificultad para realizar el Refactoring. Además, si todos los métodos de la clase son públicos, los programadores no pueden saber cuáles son partes reales de la interfase y cuáles son parte de la implementación. Esta alternativa es la menos recomendable ya que va contra las buenas prácticas de diseño. Con la utilización de la otra solución que implica el uso de métodos tipo "friend", también se estaría violando el encapsulamiento pero sólo a nivel del "package" que contiene las clases a testear y las clases del "framework" de "testing".

La solución denominada *Con encapsulamiento y con complejidad adicional* tiene la sobrecarga de tener que crear varias clases más, con el propósito de "testing" solamente. Además, estas clases quedan en el código de producción y removerlas es bastante complejo. Esta solución es válida para aquellos lenguajes que permiten las alternativas mencionadas.

Hay que tener en cuenta que en la solución *Dependiente del lenguaje y complejidad adicional*, la utilización de clases internas es usualmente temporal, hasta que se detecte una recurrencia de dichas clases para ser extraídas. Muchas veces los métodos privados que deben ser testeados posiblemente deban pertenecer a otra clase. Sin embargo, la utilización de esta solución, implica la creación de nuevas clases que, de no existir el test, no existirían.

Referencias

- *Private Interface Pattern*

  Fames Newkirk - PLoP97

  http://www.objectmentor.com/publications/privateInterface.pdf

- *Cunningham and Cunningham Inc. http://www.c2.com*
  - http://www.c2.com/cgi/wiki?PrivateInterface
  - http://www.c2.com/cgi/wiki?PackageDesign
  - http://www.c2.com/cgi/wiki?UnitTestingNonPublicMember

- *Yahoo Groups*
  - Thread: Why test public methods?

    http://groups.yahoo.com/group/extremeprogramming/message/13263
  - Thread: Should protected/privated methods also be tested?

    http://groups.yahoo.com/group/extremeprogramming/message/16412
  - Thread: Test vs. Diagnostics

    http://groups.yahoo.com/group/extremeprogramming/message/13157
  - Thread: Why do I test private methods

    http://groups.yahoo.com/group/extremeprogramming/message/13084
  - Thread: Difficult testing

    http://groups.yahoo.com/group/extremeprogramming/message/16098
  - Thread: Unit testing

    http://groups.yahoo.com/group/extremeprogramming/message/16411
  - Thread: Question on XP in VB

    http://groups.yahoo.com/group/extremeprogramming/message/16567
  - Thread: Unit testing ad Accessibility

    http://groups.yahoo.com/group/extremeprogramming/message/16714
  - Thread: Test-first and private code

    http://groups.yahoo.com/group/extremeprogramming/message/16944

- Frameworks para Test de Unidad
  http://www.xprogramming.com/software.htm

Anexo - ¿Se deben incluir los métodos privados en el Test de Unidad?

Muchos opinan que sólo los métodos públicos de las clases son los que deben ser testeados. Ellos argumentan que los métodos privados son indirectamente testeados a través de los métodos públicos. Además, si el método privado necesita ser testeado, puede indicar que hay baja cohesión en dicha clase y que debe ser extraído a una nueva clase para ser testeado. Sometiendo al test sólo a los métodos públicos de un "package" que contiene las clases, se asegura su funcionamiento con relación al exterior y si esto no es posible, entonces puede existir algún error en el diseño. También se menciona que testear un método privado es una forma de seguir un "bug" revelado por un método público y que la utilización del "debugger" es la alternativa más rápida. Otro punto a tener en cuenta es que una clase que tiene demasiados métodos privados puede ser síntoma de que debe ser dividida. Como último argumento, se menciona que testear los métodos privados puede violar el encapsulamiento dependiendo de la forma en que sea realizado. Si se realizan en forma incorrecta, luego no se puede cambiarlos ni removerlos sin romper los test, incluso si el comportamiento externo de la clase permanece correcto.

Como ventajas de no testear los métodos privados se menciona que se evita tener que cambiar los test cada vez que se cambia la estructura interna con Refactoring. Si fuesen testeados los métodos privados, con cada cambio habría que cambiar los test. Otra ventaja radica en que se puede decir que los test también sirven como documentación de la interfase y que incluir los métodos privados oscurecen esta documentación.

8.5 Orden de implementación de las 12 prácticas de XP

Nombre

Orden de implementación de las 12 prácticas de XP

Problema

De acuerdo a los creadores de XP, es recomendable adoptar las 12 prácticas ya que las fortalezas de unas contrarrestan a las debilidades de las otras. Pero en algunos casos la adopción de XP en organizaciones puede ser lenta y gradual. Por este motivo, es de particular interés encontrar un orden de adopción de las prácticas que facilite el camino para la adopción completa de XP. De aquí surge la siguiente pregunta: ¿Cuál debe ser el orden de implementación de las 12 prácticas de XP?

Contexto

Organizaciones que han decidido adoptar XP de forma gradual.

Fuerzas

Hay millones de combinaciones posibles en el orden de adopción de las 12 prácticas de Extreme Programming y no todas son efectivas.

Existen organizaciones con intención de adoptar XP pero de forma lenta y gradual, en cuyo caso se adoptarán algunas prácticas antes que otras.

Cada práctica presenta debilidades que pueden ser contrarrestadas con las fortalezas de otra(s).

Solución

- *Soluciones propuestas por las figuras claves de XP*

  Kent Beck considera que se debe empezar por adoptar el Juego de la Planificación. Sin tener Historias, o al menos el principio de Historias, se arriesga perder tiempo en desarrollar funcionalidad que no le agrega valor al

cliente. Por otro lado, el Juego de la Planificación es de suma importancia ya que establece un flujo de trabajo (workflow) para todos los actores de XP, lo que le permite al equipo trabajar de forma organizada y coordinada.

Ron Jeffries sugiere implementar las prácticas en el siguiente orden: Test Continuo (lo que comprende Test de Unidad y Test de Aceptación), y luego el Juego de la Planificación. Comenta que los test son de suma importancia ya que esta práctica ayuda enormemente a determinar cuando una Historia ha terminado, y al mismo tiempo les permite a los programadores hacer cambios con confianza, sin miedo a romper funcionalidad existente.

- *Soluciones propuestas en los foros de discusión*

De acuerdo a Alistair Cockburn[12], la Programación en Pareja y los Test de Unidad deben ser las primeras técnicas a adoptar. Referente a esto, Ron Jeffries considera que Programación en Pareja debería ser puesto a prueba durante una semana y luego evaluar los resultados, ya que esto no es esencial para XP, pero sí de mucho valor. Por otro lado, los Test de Unidad sí los considera como esenciales.

Hay otros que consideran que los problemas de implementación de XP son más bien de aceptación cultural, y que las prácticas más problemáticas son la Programación en Pareja y los Test de Unidad. Sugiere pedirle al equipo un período de 3 semanas de plena colaboración en donde serán observados y registrados los problemas enfrentados en la ejecución de estas prácticas.

Por otro lado, otros sugieren buscar una "esquina" del software que se pueda construir de cero con Test de Unidad y asignarle este trabajo a un equipo de 2 programadores. Los que mantienen esta postura piensan que los buenos

[12] Inventor de la metodología Crystal Clear y autor de varios libros sobre la ingeniería del software.

resultados obtenidos a partir de la puesta en marcha de esta práctica será "contagiada" al resto del equipo.

- *Solución orientada a minimizar el riesgo*

Hay algunos que sugieren enfrentar las prácticas más problemáticas primero. De acuerdo a una encuesta[13] realizada a 274 personas vía el XP Mailing List y presentada en ExtremeProgrammingRoadmap de *Cunningham and Cunningham Inc.*, las prácticas más problemáticas son las siguientes:

Práctica XP	#Personas	%
Cliente On-Site	56	20.44
Programación en Pareja	45	16.42
Juego de la planificación	32	11.68
Metáfora	31	11.31
Test continuo	23	8.39
Semanas de Cuarenta Horas	18	6.57
Propiedad colectiva	17	6.2
Integración continua	15	5.47
Diseño simple	15	5.47
Refactoring	12	4.38
Estandares de codificación	5	1.82
Iteraciones cortas	5	1.82
Total	274	

Tabla 3 – Encuesta de prácticas problemáticas

- *Solución dependiente del grado de escepticismo*

Por otro lado, se proponen diferentes órdenes de adopción según el grado de escepticismo y conocimiento que tenga la empresa con respecto a Extreme Programming. Si hay escepticismo y/o desconocimiento de XP, la Programación

[13] *Fuente: http://c2.com/cgi/wiki?AdoptingXpPatternLanguage*

en Pareja y el cliente On-Site deben ser de las últimas prácticas a adoptar. Pero por otro lado, si la organización se encuentra abierta a la idea de adoptar XP, la Programación en Pareja y el cliente On-Site deben ser de las primeras prácticas a adoptar.

Contexto Resultante

Cabe aclarar que las soluciones resuelven solamente la determinación del orden de implementación de las prácticas, y no problemas puntuales de implementación de cada práctica.

Explicación

Los órdenes de implementación planteados anteriormente están basados en experiencias de terceros. Inclusive algunos de los creadores de XP han participado en la solución de este problema, aumentando la confianza de que la solución al problema funcione.

Referencias

- *Cunningham and Cunningham Inc. http://www.c2.com*
  - http://c2.com/cgi/wiki?StartingWithExtremeProgramming
  - http://c2.com/cgi/wiki?DryExtremeProgramming
  - http://c2.com/cgi/wiki?MigrationToXp
  - http://c2.com/cgi/wiki?AdoptingXpPatternLanguage
  - http://c2.com/cgi/wiki?ExtremeProgrammingPracticeAdoptionOrder
  - http://c2.com/cgi/wiki?AlistairCockburn
  - http://c2.com/cgi/wiki?MichaelHill
  - http://c2.com/cgi/wiki?KielHodges

- *Extreme Programming Explained – página 63*
  Kent Beck

8.6 Test de Unidad de interfaces de usuario gráficas (GUIs)

Nombre

Test de Unidad de interfaces de usuario gráficas (GUIs)

Problema

La evolución de la informática trata constantemente de acortar la brecha de comunicación existente entre el hombre y la máquina. Actualmente se utilizan GUIs, o interfaces de usuario gráficas, para interactuar con los usuarios.

Las GUIs reciben entradas del usuario a través de ingresos por teclado, movimientos de "mouse" y distintos tipos de "clicks". Por otro lado, generan salidas que por lo general se traducen en una actualización del área visible por el usuario. Surge el problema de cómo testear este tipo de aplicación, ya que la participación activa del usuario es fundamental para validar su funcionamiento. A partir de esto se generan las siguientes preguntas:

¿Cómo se deben testear las entradas y salidas de las GUIs?
¿Se pueden automatizar estas pruebas?

Contexto

Organizaciones que han decidido adoptar XP.

Organizaciones que desarrollan aplicaciones con interfaces de usuario gráficas (GUIs).

Fuerzas

La situación tecnológica actual se caracteriza por aplicaciones que interactúan con el usuario a través de interfaces gráficas.

La automatización del "testing" puede disminuir los costos operativos ya que los recursos humanos son un componente costoso en el proceso de desarrollo de software. Cabe aclarar que la organización debe realizar un análisis costo-beneficio para determinar si la automatización se traduce efectivamente a una disminución en los costos.

Solución

Se proponen las siguientes soluciones:

1. *Test manual*: Se realiza el test de las GUIs de forma manual, ya sea por los propios programadores o por un grupo independiente.

2. *Técnica "Screenscraper"*: Similar a una caja negra de avión, en donde se registra todo el "input" del usuario como movimientos de "mouse", "clicks", teclado, etc., para luego ser reproducido de forma automática.

3. *Utilización de productos enlatados*: Algunos sugieren la utilización de productos enlatados (Ej. Rational Visual Test) para efectuar las pruebas de regresión sobre los componentes GUI.

4. *Utilización del "framework" Model View Controller (MVC)*: Este "framework" fue concebido por los inventores de Smalltalk. Consiste en separar una aplicación en sus tres partes fundamentales:

   Model: un objeto que representa datos o una actividad, por ejemplo la lógica de negocio.
   View: alguna forma de visualización del estado del modelo.
   Controller: ofrece servicios de alteración de estado del modelo.

En su artículo *The Test/Code Cycle in XP: Part2, GUI* **/WAK00/**, William Wake sugiere dividir el proceso de test, desarrollo y diseño de un GUI en las siguientes etapas:

a. *Modelo*: William Wake recomienda que cuando se está creando un GUI, lo primero que se debe desarrollar y testear es el modelo (de acuerdo al "framework" *MVC*). En lo posible recomienda hacer el test sobre las interfaces del modelo y no sobre la implementación de las clases.

b. *"Widgets"* [14] *(View):* Una vez que se tiene el diseño de la pantalla, lo primero que se puede testear es, por un lado, la presencia de los "widgets" mediante afirmaciones (asserts). También se puede implementar código que verifique la correcta inicialización de los "widgets".

c. *Interconexión (Controller):* Esta etapa consiste en testear la interconexión entre los "widgets" (parte del GUI) y el modelo. Los "clicks" de los botones y las entradas del usuario pueden ser simulados a través de métodos tales como setText(), doClick(), etc.

5. *Transformación de "render" en secuencia de texto*: Esta solución consiste en transformar el dibujo en una secuencia de texto y luego compararla con la secuencia esperada (llamada string de referencia, o "referenceString"). El proceso consta de crear una subclase de la clase responsable de generar los gráficos para que, además de generar los gráficos, genere una representación textual de los pasos necesarios para generar la imagen. De esta manera, se puede grabar la representación textual correcta, y utilizarla en ejecuciones

[14] Botones, etiquetas, etc.

posteriores para validar la generación correcta del componente GUI, por ejemplo mediante una afirmación (assert).

Contexto Resultante

La aplicación del *Test Manual* tiene la gran desventaja de ser caro y propenso a errores.

La *Técnica Screencraper* tiene el inconveniente que cualquier alteración en la geometría de la imagen, estrategia de dibujo, resolución de pantalla, tipos de letra, etc. invalida la reproducción de la secuencia grabada y por lo tanto requerirá un cambio en el caso de prueba.

La utilización de *Productos enlatados* usualmente requieren de mucho trabajo de adaptación y costos de capacitación, dependiendo del producto y del lenguaje de programación. Además, estos productos muchas veces no permiten el test de componentes desarrollados por terceros que pueden formar parte de la interfaz gráfica.

Con la utilización del "framework" *MVC* se permite crear componentes GUI livianos y sencillos que pueden ser testeados con mayor facilidad. Hay lenguajes que permiten hacer esta separación con mayor o menor facilidad. En el libro *Extreme Programming Installed* se habla de separar el código *modelo* del GUI, ya que el código *modelo* es mucho más fácil de testear. Los autores recomiendan simplificar el código GUI al máximo y testear el *modelo* con más detalle.

Referencias

- *Cunningham and Cunningham Inc. http://www.c2.com*
  - http://www.c2.com/cgi/wiki?ExtremeProgrammingTestingGuiIntensiveApps
  - http://www.c2.com/cgi/wiki?GuiTesting
  - http://www.c2.com/cgi/wiki?ThoughtfulReactionsToXp
  - http://www.c2.com/cgi/wiki?ModelViewController

- *"The Test/Code Cycle in XP: Part2, GUI", William Wake*
  http://users.vnet.net/wwake/xp/xp0001/index.shtml

- http://www.stqemagazine.com/webinfo_detail.asp?id=102

- *Extreme Programming Installed - página 103*
  Ron Jeffries – Ann Anderson – Chet Hendrickson

8.7 Actividades para el tiempo de holgura de los programadores

Nombre

Actividades para el tiempo de holgura de los programadores

Problema

Hacia el fin de una iteración, uno se puede encontrar con la situación en que hay algunos programadores que han terminado todas las Tareas de Ingeniería que han elegido, mientras que otros no. ¿Qué tipo de actividades pueden realizar aquellos programadores que han finalizado el trabajo planificado para la iteración?

Contexto

Este problema puede darse en cualquier equipo de desarrollo que utiliza XP, porque la estimación del tiempo de implementación de las tareas que puede realizar un programador, con frecuencia es inexacta.

Fuerzas

El tiempo libre al final de las iteraciones es un problema recurrente en equipos de desarrollo que recién comienzan a implementar XP o en equipos con experiencia en XP donde se integran nuevos programadores. Esto se debe principalmente a la falta de experiencia en la realización de estimaciones por parte de los programadores.

Es muy difícil que a la última hora del último día de la iteración, todos los programadores concluyan sus Tareas de Ingeniería simultáneamente.

Tener alternativas bien definidas para los programadores ociosos, permite aumentar la productividad de la organización.

Solución

Hay que mencionar que algo del tiempo libre del que poseen los programadores que terminaros sus tareas antes de finalizar la iteración, desaparece cuando estos programadores trabajan en pareja con otros que aún no han terminado.

- *Tiempo libre superior a una jornada laboral*
  - *Soluciones propuestas por las figuras claves de XP*

    Martín Fowler hace referencia a que si el tiempo libre es del orden de los días, entonces debe ser dedicado a la implementación de nuevas Historias.

  - *Soluciones propuestas en los foros de discusión*

    Así como algunos programadores tienen tiempo libre, otros pudieron haberse equivocado en la estimación y haber elegido demasiadas Tareas para la iteración. En esta situación, los programadores con tiempo libre pasarían a hacerse cargo de estas Tareas.

    En la planificación de una iteración, también se puede planear la siguiente. Por lo tanto, si hay programadores que terminaros sus Tareas, pueden continuar con las Tareas de la próxima iteración. En caso de no haber planificado la siguiente iteración, simplemente debe organizarse una reunión con el cliente y pedirle más Historias.

- *Tiempo libre inferior a una jornada laboral*
  - *Soluciones propuestas por las figuras claves de XP*

    Martín Fowler hace mucho énfasis en que el tiempo libre de los programadores, si es del orden de las horas, debe ser dedicado al Refactoring. Menciona que no hay nada mejor que tener el código en buen estado.

- *Soluciones propuestas en los foros de discusión*

  Otra alternativa consiste en agregar a la iteración, un pequeño grupo de Historias opcionales elegidas por el cliente, para tratar que sean implementadas. En caso de no poder ser incluidas, quedarán para la siguiente iteración.

  Otras tareas pueden incluir el mejoramiento del proceso de construcción o los "scripts" de puesta en producción.

  Puede destinarse tiempo para investigar algún tipo de problema específico o para investigar nuevas tecnologías.

  El "testing" es fundamental en XP, por lo que nunca está de más ocupar el tiempo en verificar si es posible crear nuevos Test de Unidad en las clases, que no fueron detectados en la implementación inicial. De esta forma, se estaría llevando al test a un mayor nivel de detalle aumentando la confianza del programador y la calidad del software.

  Una de las herramientas con la que la mayoría de las veces no se cuenta, es una herramienta para automatizar los Test de Aceptación. Por lo tanto, este tiempo libre puede ser dedicado a su investigación y desarrollo.

  Un número considerable de personas hacen énfasis en que este tiempo libre sea aprovechado por los programadores para distenderse y divertirse.

Contexto Resultante

Con esta guía de actividades propuestas para el tiempo de holgura de los programadores, se logra tener bien definidas las tareas posibles a realizar para mantener la productividad de la organización.

Referencias

- *Yahoo Groups*
  - Thread: What happens at the end of an iteration?
    http://groups.yahoo.com/group/extremeprogramming/message/15818
  - Thread: Trivial question about iteration planning
    http://groups.yahoo.com/group/extremeprogramming/message/16404
  - Thread: What would you do?
    http://groups.yahoo.com/group/extremeprogramming/message/18187

8.8 Estrategias “Test-Code-Refactor” (TCR)

Nombre

Estrategias “Test-Code-Refactor” (TCR)

Problema

El ciclo “Test-Code-Refactor” es una técnica fundamental de Extreme Programming. Junto con los Test de Aceptación realizados por el cliente, es lo que permite asegurar un producto de la alta calidad interna y externa durante todo el proceso de desarrollo. El Test de Unidad consiste en diseñar e implementar pruebas previas a la implementación de los métodos de las clases. Pero en muchos casos, los programadores descubren que los métodos a ser testeados dependen de métodos de otras clases, o incluso nuevas clases que ni siquiera han sido implementadas. Existen diversas estrategias que determinan el orden en el que se debe realizar el “testing”, la implementación y el Refactoring de clases con dependencias. A raíz de la diversidad de combinaciones posibles, surge la siguiente pregunta: ¿Qué tipo de estrategia de TCR mejor se ajusta a la metodología Extreme Programming?

Contexto

Organizaciones que utilizan Extreme Programming para desarrollar software.

Fuerzas

El proceso de TCR es fundamental para los practicantes de XP, ya que la construcción, validación y mantenimiento continuo del código son vitales al usar esta metodología de desarrollo.

Un sistema diseñado utilizando buenas prácticas de diseño, estará formado por más de una clase.

Solución

- *Soluciones propuestas en los foros de discusión*

A modo de ejemplo, supongamos que un programador debe implementar la clase A pero ya supone de antemano que precisará las clases B, C, D y E. Se proponen las siguientes estrategias TCR donde "Test X" implica testear la clase X y lograr que pase el 100% de los Test de Unidad:

- *Serializada*

```
Escribir Test E, Escribir E, Test E
Escribir Test D, Escribir D, Test D
Escribir Test C, Escribir C, Test C
Escribir Test B, Escribir B, Test B
Escribir Test A, Escribir A, Test A
```

- *Anidación (Nesting):*

```
Escribir Test A, Escribir A
     Escribir Test B, Escribir B
           Escribir Test C, Escribir C
                Escribir Test D, Escribir D
                    Escribir Test E, Escribir E. Test E
               Test D
          Test C
     Test B
Test A
```

- *Anidación con implementación parcial:*

```
Escribir Test A, Escribir A (parcial), Test A
      Escribir Test B, Escribir B (parcial), Test B
            Escribir Test C, Escribir C (parcial)
                 Escribir Test D, Escribir D (parcial)
                       Escribir Test E, Escribir E (total), Test
                 Escribir D (total), Test D
            Escribir C (total), Test C
      Escribir B (total), Test B
Escribir A (total), Test A
```

La estrategia de *Anidación con implementación parcial* consiste en implementar los métodos en dos fases. Siguiendo el ejemplo, primero se realiza una implementación parcial de la clase A, que contiene solamente lo

suficiente para pasar el test previamente implementado. Una vez que pasa el test, se desarrolla parcialmente y se testea la próxima clase B y así sucesivamente hasta llegar a la clase E, que se desarrolla en su totalidad. Luego se desarrollan las clases restantes (D, C, B y A) en orden descendente.

- *Anidación parcial*

```
Escribir Test A, Escribir A
      Escribir Test B, Escribir B, Test B
      Escribir Test C, Escribir C, Test C
      Escribir Test D, Escribir D, Test D
      Escribir Test E, Escribir E, Test D
 Test A
```

- *Implementación parcial*

Aclaración: Ixy se utiliza para abreviar “Implementar lo suficiente de la clase “x” para saber como llamar a la clase “y”

```
Escribir Test A
IAB
      Escribir Test B
      IBC
            Escribir Test C
            ICC
                  Escribir Test D
                  ICD
                        Escribir Test E
                        Escribir E
                        Test E
                  Escribir resto de D
                  Test D
            Escribir resto de C
            Test C
      Escribir resto de B
      Test B
Escribir  resto de A
Test A
```

- *Implementación con Refactoring posterior*

```
Escribir Test A, Escribir A, Test A
Refactoring de A en A, B, C, D, E
```

La solución propuesta consiste en implementar toda la funcionalidad que se necesita en una clase sola, la clase A. Solamente después de que la clase esté implementada y testeada se debe hacer Refactoring, lo que puede significar una descomposición en varias clases.

- *Programación intencional*

```
Mientras A no esté terminada
     Escribir Test de Intención de A
     Escribir lo suficiente de A para pasar Test de Intención
          Escribir A
          Si la clase A se hace difícil de mantener
               Refactor A
           Sino
                Test A
               Clase A Terminada
           Fin si
Fin Hacer Mientras
```

La idea de programar por intención se menciona en el libro *Extreme Programming Installed*. Los autores comentan que sólo se debe implementar nueva funcionalidad en una clase cuando falla su prueba correspondiente. Primero, las pruebas deben testear la intención del método, y una vez que pasa la prueba, se procede a codificar y testear la verdadera implementación.

Contexto Resultante

Aplicando una estrategia TCR, se obtendrán clases implementadas y testeadas, lo cual le permitirá a los programadores seguir avanzando en el proyecto con confianza y dejando el código en su estado más simple para futuras modificaciones.

Algunos comentan que la estrategia *Serializada* no se ajusta a XP ya que viola el principio "*You ain't gonna need it*" (YAGNI). El principio YAGNI dice que se debe descartar completamente las necesidades especulativas del futuro. Dicho de otra manera, no se deben implementar métodos o clases que no se precisen en ese momento. Por lo tanto, implementar las clases en orden inverso (E, D, C, B, A) viola este principio.

En cuanto a la estrategia de *Anidación*, surgen tres observaciones: 1) El tiempo entre la implementación de la clase X y su "testing" puede ser demasiado extenso, en particular si hay muchos niveles de anidación. 2) Viola YAGNI por los mismos motivos que la estrategia *Serializada* 3) La *Anidación* no es aconsejable ya que requiere un almacenamiento de información para usar en el futuro, lo que también viola el principio YAGNI.

Los participantes de los foros comentan, por un lado, que la estrategia con *Anidación con implementación parcial* también viola a YAGNI por razones obvias, y que la *Anidación* no es aconsejable por los motivos explicados anteriormente.

Por otro lado, la estrategia de *Anidación parcial* tiene, según comentarios, deficiencias en aquellos casos en que existan dependencias entre las clases B, C, D y E. Como el TCR de cada clase se hace en serie, no se toma en cuenta las posibles dependencias entre las clases. Por otro lado, el tiempo entre la implementación y el test de la clase A puede llegar a ser muy extenso, lo que no es recomendable.

Como ventaja de la estrategia de *Implementación parcial* se puede mencionar que todas las clases (A-E) se implementan "test-first" y los test pasan en un 100% antes

de seguir. Como desventaja se observa que el tiempo entre la implementación de la clase X y su "testing" puede ser demasiado extenso.

Algunos recomiendan que independientemente de la solución se deben cumplir dos condiciones:

1) Sólo debe fallar a lo sumo un test a la vez.
2) Se debe diseñar sólo para la tarea inmediata, evitando el diseño especulativo. Dicho de otra manera, tener al principio YAGNI presente en todo momento.

Según este criterio, la única solución que satisface ambas condiciones es la solución de *Programación intencional*. En esto caso se considera de suma importancia hacer una distinción en lo que son los test de intención, realizados previo a la implementación de la clase, con los Test de Unidad hechos posterior a la implementación.

Referencias

- *Yahoo Groups*
  - Thread: Unit Testing
    http://groups.yahoo.com/group/extremeprogramming/message/15342
    http://groups.yahoo.com/group/extremeprogramming/message/15370
  - Thread: Top-Down Versus Bottom-Up
    http://groups.yahoo.com/group/extremeprogramming/message/16139
    http://groups.yahoo.com/group/extremeprogramming/message/16171
    http://groups.yahoo.com/group/extremeprogramming/message/16181
  - Thread: more on learning "test-first"
    http://groups.yahoo.com/group/extremeprogramming/message/16164

- *Cunningham and Cunningham Inc. http://www.c2.com*
  http://c2.com/cgi/wiki?ExtremeProgrammingUnitTestingApproach

- *Extreme Programming Installed - página 107*

  Ron Jeffries, Ann Anderson, Chet Hendrickson

9. POSIBLES TRABAJOS COMPLEMENTARIOS

El propósito de esta sección es dejar una puerta abierta a la realización de nuevos trabajos de investigación relacionados con Extreme Programming. La siguiente lista de trabajos propuestos, ha sido confeccionada basándose en las discusiones que han tenido mayor interés en la XP mailing list y en el sitio de ExtremeProgrammingRoadmap en www.c2.com.

1. Continuación de la guía de problemas y soluciones presentada en este trabajo

En este trabajo fueron presentados los problemas de implementación de XP que han tenido mayor interés y participación en el final del año 2000 y comienzos del año 2001. A medida que transcurra el tiempo, se publicará más material sobre los temas que no se encuentran en esta guía y sobre nuevos problemas. El objetivo de este trabajo consiste en ampliar la guía actualmente presentada extendiéndola con algunos de los 32 problemas mencionados aquí o con nuevos problemas que generen interés. También sería de mucho interés validar las soluciones planteadas para los problemas presentados y estudiar su aceptación como patrones.

2. Certificación con Extreme Programming

En XP no se requiere mantener ningún tipo de documentación formal en todo el proceso. Se dice que el código y los test son toda la documentación necesaria. Sin embargo, hoy en día muchas instituciones, tanto gubernamentales como no gubernamentales, requieren que se genere como parte del proceso, una cantidad de documentación que de alguna forma certifique que el desarrollo del software se generó con cierto nivel de calidad. Además de dicha documentación, algunas

certificaciones incluyen determinados procedimientos que deben ser seguidos. El objetivo de este trabajo consiste en comparar a XP con los niveles de CMM, los requerimientos de la FDA de los EE.UU., la ISO9000, o la norma SPICE ISO15504 y determinar su compatibilidad.

3. Comparación de Extreme Programming con otras metodologías livianas de desarrollo

En los últimos años han surgido en la comunidad de programadores nuevas formas de desarrollar software. Estas nuevas metodologías surgieron para contrarrestar los problemas y riesgos que usualmente son detectados al aplicar las metodologías actuales. La idea de estas nuevas metodologías de desarrollo, denominadas livianas, es hacer hincapié en el producto final, que es el software y no en el proceso de desarrollo. Como ejemplo de algunas de las nuevas metodologías se puede mencionar XP, Crystal Clear **/CRY01/**, Aspect Oriented Programming **/AOP01/** o Scrum **/SCR01/**. El objetivo de este trabajo consiste en comparar estas nuevas formas de desarrollar software, mostrando las ventajas, desventajas y criterios de utilización de cada una de ellas.

4. VNC – Virtual Networking Computing aplicado a la programación en pareja

Cada vez más, los proyectos de software se desarrollan en espacios geográficos distribuidos. Muchos llegan a ser desarrollados incluso por diferentes compañías en varias partes del mundo. Teniendo en cuenta que una de las prácticas que se utiliza en Extreme Programming es la programación en pareja, surge la interrogante en lo que respecta a cómo desarrollar en conjunto cuando las parejas no comparten el mismo lugar físico. El objetivo de este trabajo es analizar las ventajas y desventajas de la programación en pareja en el mismo espacio geográfico en relación a la programación remota. Además, se deberá analizar los

requerimientos de una herramienta para la programación en pareja remota e investigar las herramientas existentes.

5. Enseñanza de Extreme Programming

Como una nueva forma de desarrollar software, XP debe ser presentada en el mundo universitario. En prácticas como la Programación en Pareja, "Test-First-Programming" o el Juego de la Planificación, la idea teórica no es suficiente para la total comprensión. Es necesario experimentar de forma práctica para lograr comprender su funcionamiento. El objetivo de este trabajo consiste en presentar formas de introducir las diferentes prácticas de XP a una clase universitaria, poniendo especial énfasis en experimentos y juegos prácticos.

6. Software para el soporte del Juego de la Planificación

Actualmente, existe una variedad de software para el control de versiones del código, para el control de las convenciones de código y para el Test de Unidad. Sin embargo, no existen muchas herramientas que soporten y puedan llevar el flujo de trabajo planteado en el Juego de la Planificación. El objetivo de trabajo consiste primeramente en determinar los requerimientos necesarios que debe cumplir una herramienta para soportar el Juego de la Planificación. En una segunda etapa, se deberá realizar un análisis de las herramientas existentes para esta práctica, presentando las ventajas y desventajas de cada una de ellas en relación a los requerimientos planteados.

7. Escalabilidad de Extreme Programming

El tema de la escalabilidad de XP ha desatado mucho debate en los foros de discusión. Si bien los límites en lo que refiere a cantidad de individuos en el proyecto o a su complejidad fueros esbozados por Kent Beck en su primer libro de la serie **/BEC99/**, no está probado que XP funcione o deje de funcionar a gran

escala. El objetivo de este trabajo es recabar experiencias y datos en lo que refiere al tamaño del grupo de trabajo, la complejidad del proyecto y su duración, para encontrar algún tipo de patrón de éxito o de fracaso.

8. Aplicabilidad de XP con diversos lenguajes de programación y entornos de desarrollo

¿Se puede usar XP con cualquier lenguaje de programación? Cada lenguaje presenta problemas particulares al ser utilizados con XP. Para cada lenguaje, existe una variedad de compiladores, entornos de trabajo y herramientas que pueden facilitar o dificultar la adopción de XP. El objetivo de este trabajo consiste en analizar varios lenguajes de programación, con el fin de detectar las fortalezas y debilidades de utilizarlos con XP. Por otro lado, estudiar la posibilidad de utilizar herramientas, entornos de desarrollo, etc. que puedan contrarrestar las debilidades encontradas.

9. Software para el soporte del “testing”

Según los creadores de Extreme Programming, la ejecución de los test (Unidad y Aceptación) debe ser aislada y automática. Existe una gran variedad de herramientas en el mercado que permiten diseñar y administrar los Test de Unidad, Aceptación, carga, GUI “testing”, etc. Inclusive, algunas herramientas posibilitan su ejecución automática. El trabajo consiste en evaluar las herramientas existentes en el mercado y determinar cuáles de ellas se ajustan mejor a XP.

10. CONCLUSIONES

El objetivo de este trabajo ha sido crear una guía que muestre los problemas más comunes que los equipos de desarrollo han enfrentado al implementar XP, y una serie de alternativas que puedan solucionarlos. Fueron presentados y analizados los seis problemas más populares con el propósito de ayudar a futuras organizaciones en la adopción de XP.

En la investigación realizada entre octubre de 2000 y marzo de 2001, fueron detectados 32 problemas. De ellos, 19 fueron publicados en *http://www.c2.com/cgi/wiki?ExtremeProgrammingImplementationIssues*, para obtener información más actualizada, obtener nuevos comentarios y compartir nuestros hallazgos con la comunidad.

Los seis problemas más populares, junto a las soluciones propuestas por la comunidad, fueron incluidos en está guía en el formato de patrones propuesto.

Haber obtenido como problema más popular la adopción de XP en sistemas legados es un resultado bastante lógico, teniendo en cuenta que la amplia mayoría de las organizaciones ya posee sistemas legados. Por lo tanto, una vez que la organización decide adoptar XP como metodología de desarrollo, comienzan a surgir una serie de problemas que, de haber adoptado XP para desarrollar sistemas nuevos, no se tendrían. Para esta adopción, se presentó una serie ordenada de pasos para implementar XP en sistemas legados, haciendo énfasis en lugares donde estos sistemas lo requieren.

Los problemas en el “testing” son también de los problemas más comunes en la implementación de XP ya que el Test de Unidad y el Test de Aceptación, según las figuras claves de XP, deberían ser de las primeras prácticas en ser adoptadas. En particular, los problemas relacionados con el “testing” de los métodos privados todavía tienen a la comunidad un poco desorientada. Algunos argumentan que se

deben testear los métodos privados mientras que otros opinan lo opuesto. En este trabajo se presentaron diferentes alternativas para el "testing" de los métodos privados, así como la opinión de aquellos que no están de acuerdo con esta idea.

El tercer problema planteado está relacionado con el orden de adopción de las prácticas. En la bibliografía sobre XP no se define un orden específico de adopción, ni se delimitan criterios claros para generar alguno. Por este motivo, la comunidad se encuentra bastante confundida con este tema y fue recopilada bastante información al respecto. Se mencionaron diferentes propuestas proporcionadas por las figuras claves de XP y por otros participantes de los foros de discusión. De esta forma se logró diferentes órdenes posibles, en los cuales se da énfasis a los propuestos por las figuras claves, mientras que en otros, se tienen en cuenta las características de la organización y su actitud frente a XP.

Cada vez más, las aplicaciones tienen componentes visuales en la interacción con el usuario. Por este motivo y debido a las exigencias en el "testing" que tiene XP, se deben encontrar formas fáciles y automáticas para poner en práctica el "testing" de las GUI. Como realizar este tipo de pruebas nunca fue simple, tampoco lo será utilizando XP. Diferentes propuestas son mencionadas para orientar a aquellos que enfrenten este problema.

Si bien XP mejora el proceso de estimación de las tareas a realizar durante las iteraciones, este proceso no es exacto. Esto implica que los programadores pueden tener iteraciones en las que su carga de trabajo no cubre totalmente su tiempo productivo. En estos casos, varias actividades posibles son presentadas en esta guía, de forma de aumentar la productividad al máximo en cada iteración.

El "Test-Code-Refactor" es uno de los procesos más importantes de XP. Existen diferentes estrategias para llevarlo a cabo pero algunas de ellas no se ajustan a la filosofía de esta metodología. Se presentaron las diferentes alternativas propuestas en los foros de discusión y bibliografía existente, mencionando sus ventajas y

desventajas. Las estrategias más aceptadas en la comunidad fueron aquellas que resuelven los problemas del presente y no las necesidades especulativas del futuro.

Es importante mencionar que el orden de popularidad de los problemas obtenidos probablemente cambie si se realiza otra investigación en un período de tiempo diferente; ya sea más extenso o en otro intervalo. Incluso, la lista de los problemas podría cambiar si el estudio se lleva a cabo en otra fecha.

Este trabajo se realizó en una etapa de evolución de XP en donde las organizaciones recién están comenzando a implementarla. Recordemos que XP surgió a fines de 1999 y que recién en el 2000 se comenzó con la difusión masiva de esta metodología. Por lo tanto, este estudio debería ser replicado en los años siguientes para poder determinar si estos problemas fueron ya superados o continúan siendo los más comunes.

El formato de presentación elegido tiene la ventaja de ser muy fácil de ampliar, lo que le permitirá a futuros investigadores agregar nuevos problemas y soluciones en la implementación de Extreme Programming. Es importante mencionar la originalidad de haber elegido este formato de presentación, ya que permite identificar rápida y claramente los problemas y soluciones encontrados. Por otra parte, la publicación de la guía facilitará la evolución y revisión dinámica de los problemas y soluciones encontrados hasta la fecha. Todas las soluciones propuestas se encuentran en la categoría de proto-patrón, pero su publicación posibilitará que la comunidad pueda validar o refutar su aceptación como patrón.

Finalmente concluimos que los problemas de implementación detectados surgen por la propia flexibilidad de la metodología. A diferencia de otras metodologías que establecen reglas y procedimientos rigurosos, Extreme Programming sugiere la adopción de 12 prácticas soportadas por 4 valores. Pero estas prácticas son descritas por los creadores de XP de manera muy abierta, lo que trae mucha libertad en su implementación. Pero esta libertad genera duda y confusión en las organizaciones que han intentado implementarla.

Si bien Extreme Programming no es la solución para todos los equipos de desarrollo de software, plantea una nueva alternativa que debe ser considerada en cualquier proyecto cuyos requerimientos son ambiguos o muy volátiles.

En opinión de los autores de este trabajo, Extreme Programming plantea una filosofía de trabajo que crea una cultura organizacional flexible sin perjudicar la creatividad de sus actores. Esta conclusión reafirma la hipótesis inicialmente planteada en la propuesta: "*Además de la resolución de la problemática de implementación de cada práctica, el éxito de XP en una organización dependerá en gran parte de cuan alineados estén los valores y principios de su gente con los pilares que sostienen a XP: comunicación, simplicidad, retroalimentación y coraje.*"

11. BIBLIOGRAFÍA

Libros citados

/BEC99/ - *Extreme Programming Explained*, Kent Beck, 1999.

/BOE88/ - *Understanding and controlling software costs,* Barry Boehm, Philip Papaccio, 1988.

/FOW99/ - *Refactoring – Improving the design of existing code*
Martin Fowler, 2000.

/FOW00/ - *Planning Extreme Programming*, Kent Beck, Martin Fowler, 2000.

/JEF00/ - *Extreme Programming Installed,* Ron Jeffries, Ann Anderson, Chet Hendrickson, 2000.

/MCC96/ - *Rapid Development*, Steve McConnell, 1996.

/ROY98/ - *Software Project Management,* Walker Royce, 1998.

/WAK01/ - *Extreme Programming Explored (Draft)*, William Wake, 2001.

Artículos y sitios en Internet citados

/AOP01/ - *Aspect-Oriented Programming*
http://www.parc.xerox.com/csl/projects/aop/

/APP00/ - *Patterns in a Nutshell – Brad Appleton*
http://www.enteract.com/~bradapp/docs/patterns-nutshell.html
http://www.enteract.com/~bradapp/docs/patterns-intro.html#AntiPatterns

/CHR98/ - *Chrysler Goes to Extremes by the C3 Team*
www.distributedcomputing.com

/COL00/ - *XP Distilled*
http://www.rolemodelsoft.com/articles/xpCorner/xpDistilled.htm
Chris Collins y Roy Miller, 2000

/CRY01/ - *Crystal Clear*
http://members.aol.com/humansandt/crystal/clear/

/CUN01/ - *Cunningham and Cunningham Inc. - "Companies doing XP"*
http://www.c2.com/cgi/wiki?CompaniesDoingXp

/CUT00/ - *Cutter Consortium e-business application delivery*
http://cutter.com/ead/ead0002.html

/FDD00/ - *Feature-driven development*
http://www.togethercommunity.com/coad-letter/Coad-Letter-0070.html

/SCR01/ - *Scrum*
http://www.controlchaos.com/

/WAK00/ - *Sitios de William Wake*
http://users.vnet.net/wwake/
http://xp123.com

/WIK01/ - *ExtremeProgrammingRoadmap en Cunningham and Cunningham Inc. www.c2.com*
http://www.c2.com/cgi/wiki?ExtremeProgrammingRoadmap

/WIL00/ - *Sitio de Laurie Williams*
http://collaboration.csc.ncsu.edu/laurie/main.htm

/XPO01/ - *People doing XP*
http://www.extremeprogramming.org/people.html

/YAH01/ - *Sitio de XP en Yahoo Groups*
www.egroups.com/extremeprogramming

/XPI01/ - *Sitio creado por los autores de esta memoria de grado para recabar nuevas opiniones y comentarios actualizados sobre los problemas y soluciones encontradas.*
http://www.c2.com/cgi/wiki?ExtremeProgrammingImplementationIssues

Libros y trabajos consultados

- *Metodología de la Investigación*
  Roberto Hernández Sampieri, Carlos Fernández Collado, Pilar Baptista Lucio, 1991

- *Writing Up Research – Experimental research report writing for students of English*
  Robert Weissberg, Suzanne Buker
- *Microsoft Visual Basic Design Patterns*
  William Stamatakis, 2000
- *Patrones de software*
  Loza, UCUDAL, 2000

Artículos y sitios en Internet consultados

- *Sito oficial de Ron Jeffries*
  http://www.xprogramming.com
  http://www.xprogramming.com/xpmag/cost_of_change.htm
  http://www.xprogramming.com/xpmag/test_first_intro.htm
  http://www.xprogramming.com/xpmag/c3space.htm
  http://www.xprogramming.com/xpmag/Curves.htm
  http://www.xprogramming.com/xpmag/NotXP.htm
  http://www.xprogramming.com/xpmag/hurricane.htm
  http://www.xprogramming.com/xpmag/incremental_req1.htm
  http://www.xprogramming.com/xpmag/kings_dinner.htm
  http://www.xprogramming.com/xpmag/BizAnalysis.htm
  http://www.xprogramming.com/xpmag/houseAnalogy.htm
  http://www.xprogramming.com/xpmag/an_open_approach.htm
- *Sitio oficial de Don Wells*
  www.extremeprogramming.org
- *Sitio oficial de Martin Fowler*
  http://www.martinfowler.com
  http://www.martinfowler.com/articles/planningXpIteration.html
  http://www.martinfowler.com/articles/XpVariation.html

Sitios relacionados con el "testing"

- www.junit.org
- *Extreme Testing*
  www.stqemagazine.com - Marzo/Abril 1999
- *Borland Community – Testing at the speed of thought*
  http://community.borland.com/article/0,1410,20645,00.html

Sitios relacionados con la programación en pareja

- http://www.pairprogramming.com
- http://www.inquiry.com/pubs/infoworld/vol22/issue30/x22-30.asp
- *Costos y Beneficios de la programación en pareja* – Allistar Cockburn y Laurie Williams
  http://members.aol.com/humansandt/papers/pairprogrammingcostbene/pairprogrammingcostbene.htm
- *Support for Distributed Teams in eXtreme Programming* - Till Schummer, Jan Schummer
  http://citeseer.nj.nec.com/schuemmer00support.html

Sitios relacionados con la estandarización del código

- *C++ Coding Conventions*
  http://www.cryst.bbk.ac.uk/~classlib/bioinf/standards.html
- *Coding Conventions for C++ and Java applications - Macadamian Technologies Inc*
  http://www.macadamian.com/codingconventions.htm
- *Draft Java Coding Standards*
  http://gee.cs.oswego.edu/dl/html/javaCodingStd.html
- *VBScript Coding Conventions*
  http://msdn.microsoft.com/scripting/vbscript/doc/vbscodingconventions.htm
- *Visual Basic Coding Conventions*

http://msdn.microsoft.com/library/devprods/vs6/vbasic/vbcon98/vbconcodingconventionsoverview.htm

Sitios relacionados con la integración continua

- *IEEE Software, Vol.13, No. 4, July 1996*
  http://www.construx.com/stevemcc/bp04.htm

Sitios de Organizaciones que utilizan o brindan consultoría en XP

- *Accudaq*
  http://www.accudaq.com/xprogramming.htm
- *Euclid Technologies Inc.*
  http://www.euclidtech.com/solutions/xp.asp
- *ObjectMentor*
  www.objectmentor.com
- *ThoughtWorks*
  http://www.thoughtworks.com
- *Industrial Logic Inc.*
  http://industriallogic.com/index.html
- *Net Objectives*
  http://www.xprefined.com/
- *ObjectWind Software Ltd.*
  http://www.objectwind.com/xp/
- *RoleModelSoft*
  http://www.rolemodelsoft.com
- *Tireme International Ltd.*
  http://www.trireme.com/xp.htm
- *Integrity First Consultants Inc.*
  http://integrity-first-inc.com/
- *Sierra Computer Group*
  http://www.sierragrp.com/XP.htm

Sitios con información general sobre XP

- *IEEE Computer Society Dynabook*
  http://computer.org/seweb/dynabook
- *XP en North Carolina State University*
  http://www.csc.ncsu.edu/eos/users/e/efg/517/f99/course_locker/www/syllabus/lectures/XPOverview.html
- *XP en Walla Walla College*
  http://cs.wwc.edu/~aabyan/XP/solution.html
- *Oregon Graduate Institute of Science and Technology*
  http://www.cse.ogi.edu/courses/CSE924/
- *IBM DeveloperWorks Java Poll results*
  http://www-4.ibm.com/software/developer/library/java-ollresults/x.html
- *DevX - Going to extremes*
  http://www.devx.com/free/articles/2000/abbott01/abbott01-1.asp
- *C/net – News.com*
  http://news.cnet.com/news/0-1006-200-5424853.html?tag=ch_mh
- *Taking programming to the extreme edge*
  www.infoworld.com/articles/mt/xml/00/07/24/000724mtextreme.xml
- *InformationWeek.com*
  http://www.informationweek.com/831/appdev.htm
- *ComputerWorld*
  http://www.computerworld.com/cwi/stories/0,1199,NAV47-81_STO59388,00.html
- *Visual Basic Developer*
  http://www.pinnaclepublishing.com/VB/VBmag.nsf/37fd7b62c1dfc5d88525689600515109/52cfb37b974b9ffe8525690b006301a7!OpenDocument
- *Xml.com*
  http://www.xml.com/pub/a/2001/04/04/xp.html
- *C++ _Report OnLine*
  http://www.creport.com/archive/9905/html/from_pages/features.shtml
- http://www.jera.com/techinfo/xpfaq.html

- http://www.xpdeveloper.com/
- http://ootips.org/xp.html
- http://www.cch.co.za/sd/knowledge/methodology/xp.htm
- http://www.larkfarm.com/books/xp.htm
- http://www.mindspring.com/~happyjac/site/info/process/pp/XP_List.html
- http://members.home.net/wmallen/articles/xp/
- http://www.ee.ualberta.ca/~yip/611/XPpresentation.html
- http://www.jera.com/techinfo/xpfaq.html
- http://www.bobjectsinc.com/cstug/xpslides/
- http://clabs.org/caseforxp.htm
- http://www.ddj.com/articles/2000/0050/0050k/0050k.htm

Artículos relacionados con el diseño del software

- *Is Design Dead? – Martin Fowler – ThoughtWorks*
  http://www.martinfowler.com/articles/designDead.html

Artículos relacionados con los tipos de contratos de desarrollo de software

- *Optional Scope Contracts, Kent Beck-Dave Cleal, First Class Software*
  http://groups.yahoo.com/group/extremeprogramming/files/Optional%20scope%20contracts.pdf

Sitios de Seminarios o Congresos

- *XP 2000 Conference*
  http://www.martinfowler.com/articles/xp2000.html
- *XP 2001 Conference*
  http://www.xp2001.org/
- *XP Universe*
  http://www.xpuniverse.com/

Artículos sobre otras metodologías

- *The Rational Edge – www.therationaledge.com*
  http://www.therationaledge.com/content/jan_01/f_rup_pk.html
  http://www.therationaledge.com/content/mar_01/f_xp_gp.html

- *Rational Unified Process (RUP)*
  http://www.rational.com/products/rup

Artículos sobre metodologías ágiles

- *Constantine & Lockwood*
  http://www.foruse.com/Files/Papers/agiledesign.pdf