CASO PRÁCTICO DE LA METODOLOGÍA ÁGIL XP AL DESARROLLO DE SOFTWARE

LUIS MIGUEL ECHEVERRY TOBÓN
LUZ ELENA DELGADO CARMONA

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍA: ELÉCTRICA, ELECTRÓNICA, FÍSICA Y CIENCIAS DE LA COMPUTACIÓN
INGENIERÍA DE SISTEMAS
PEREIRA
2007

CASO PRÁCTICO DE LA METODOLOGÍA ÁGIL XP AL DESARROLLO DE SOFTWARE

LUIS MIGUEL ECHEVERRY TOBÓN
LUZ ELENA DELGADO CARMONA

Proyecto de Grado presentado como requisito para optar al título de INGENIERO EN SISTEMAS Y COMPUTACIÓN

Director del proyecto
Eliecer Herrera Uribe
Ingeniero de Sistemas

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍA: ELÉCTRICA, ELECTRÓNICA, FÍSICA Y CIENCIAS DE LA COMPUTACIÓN
INGENIERÍA DE SISTEMAS
PEREIRA
2007

Nota de Aceptación:

_____
_____
_____
_____
_____

_____
Firma del presidente del jurado

_____
Firma del Jurado

_____
Firma del Jurado

Pereira, 1 de Octubre de 2007

DEDICATORIA

A mis padres por el esfuerzo que significó patrocinarme este proyecto, a mi familia que siempre me apoyó y en especial a una tía muy especial. Finalmente a Dios que lo hizo posible.

Luis Miguel

A mis padres, a mi hermano y a mis amigos por todo el apoyo y confianza que han depositado en mí.

Luz Elena

AGRADECIMIENTOS

Todos los éxitos conseguidos hasta ahora y los que se obtendrán en el futuro tendrán como primer responsable a la Universidad quién nos proporciono las herramientas para desempeñarnos adecuadamente en nuestra profesión, para ella son los primeros agradecimientos.

Nuestras familias fueron quienes patrocinaron el proyecto universitario en todos los sentidos. Sin su apoyo no habría sido posible el logro de una meta tan importante como lo es terminar una carrera profesional, a ellas un sincero agradecimiento por su esfuerzo y dedicación.

Finalmente a nuestro director que nos impulsó en el último paso.

CONTENIDO

pág.

LISTA DE TABLAS

pág.

LISTA DE ILUSTRACIONES

pág.

LISTA DE ANEXOS

pág.

OBJETIVOS

OBJETIVO GENERAL.

- Realizar una aplicación práctica de la metodología ágil XP a través del desarrollo de un software para tenderos.

OBJETIVOS ESPECÌFICOS

- Consultar acerca de la metodología XP para el desarrollo del software.
- Diseñar en compañía del cliente las iteraciones.
- Desarrollar las tareas planteadas en cada una de las iteraciones.
- Realizar la documentación del proyecto según la guía XP.
- Concluir sobre la conveniencia de la utilización de la metodología XP en este proyecto basándose en los resultados obtenidos.

JUSTIFICACIÓN

Al ser recientes las metodologías ágiles (años 90's), no han sido estudiadas suficientemente por la comunidad académica, generando desconocimiento de sus bondades. La Universidad Tecnológica no ha sido ajena a este fenómeno en el sentido que no ha realizado muchos estudios sobre el tema y menos aplicaciones prácticas que involucren el paradigma de las metodologías ágiles. Por tal motivo se plantea la necesidad de explorar a mayor profundidad dicho campo a través de ejercicios prácticos debidamente documentados. En este orden de ideas es necesario analizar aspectos puntuales tales como las características de los clientes, detalles técnicos de las soluciones y prestar especial atención a la naturaleza cambiante de los requerimientos.

La coyuntura en la cual se desarrolla el software en Colombia está determinada por los bajos presupuestos y necesidad de resultados rápidos, lo que obliga a los equipos de desarrollo a buscar nuevas herramientas y metodologías que les permitan optimizar el proceso de desarrollo. Entre estas nuevas ideas están las metodologías ágiles las cuales plantean un nuevo paradigma a través del cual un proyecto de software se realiza de una forma diferente al sugerido por las metodologías pesadas. Señalan la posibilidad de centrar los esfuerzos en la implementación para lograr resultados rápidos sin sacrificar la calidad de los mismos. Entre las diferentes metodologías ágiles existentes se ha escogido XP ya que plantea, de forma más clara, el proceso metodológico a seguir para la construcción de software, siendo además la mejor documentada y de mayor uso.

RESUMEN

En el proyecto se plantea realizar una experiencia real en la aplicación de XP al desarrollo de software con el fin de determinar, para unas circunstancias específicas, que tan bien se ajusta la metodología. El proceso inició en una revisión bibliográfica de la metodología y otros aspectos relacionados a esta. A continuación se contactó con un posible cliente al cual se hizo una presentación de los objetivos iniciales del proyecto. Una vez acorados todos los detalles se procedió a su ejecución al final del cual se redactó el informe final documentando la experiencia.

El documento cuenta con siete capítulos distribuidos de la siguiente forma. En el primer capítulo se hace un recorrido teórico por XP y se esbozan elementos importantes de las metodologías ágiles. En el segundo capítulo se hace una presentación del proyecto en términos del entorno que lo rodea, una breve descripción del cliente y el tipo de negocio para el cual se desarrolló. En los capítulos tercero a sexto se realizó una comparación entre los enunciados teóricos expuestos en el capítulo primero y la aplicación que los autores hicieron en la ejecución del proyecto. Finalmente en el último capítulo se hace una serie de comentarios relevantes acerca de situaciones especiales que surgieron durante la ejecución del proyecto.

En el recorrido teórico se realiza una introducción breve de las metodologías ágiles resaltando el manifiesto ágil como su punto de partida, seguido de una exposición de los principios sobre los cuales se basa XP.

La parte central del documento consta de los capítulos tercero al sexto correspondiendo a cada una de las fases de desarrollo en XP: planeación, diseño, codificación y pruebas. En cada uno de estos capítulos se discute como se aplicaron los aspectos de la correspondiente etapa al proyecto, así como el resultado obtenido.

Al final del documento el lector encontrará conclusiones, recomendaciones y aportes sobre la experiencia que no pretenden calificar a la metodología, solamente a la experiencia en particular.

INTRODUCCIÓN

Las metodologías ágiles tienen un origen reciente en el entorno de la ingeniería de software comparada con las metodologías pesadas. Su origen está ligado a los constantes inconvenientes que se presentaban en proyectos con algunas características, en los cuales la utilización de las metodologías pesadas era motivo de fracaso.

En la década de los 90, surge eXtreme Programming, mejor conocida como XP, una nueva metodología catalogada entre las ágiles por sus aportes al manifiesto ágil. Su creador, Kent Beck se convirtió en el padre de la programación extrema.

En Colombia, la programación extrema no se ha profundizado debido a su reciente aparición. También cabe señalar la escasez de documentación referente a la misma y de los trabajos realizados empleándola.

El presente documento es la exposición de una experiencia práctica en la cual se empleó Extreme Programming. Debido al entorno académicas que rodearon al proyecto se presentaron circunstancias especiales las cuales son detalladas en el cuerpo del documento, por lo cual la metodología debió ser ajustada.

Es importante aclarar que en ningún momento se pretende hacer una evaluación o juicio de la metodología empleada debido que no es posible tener conclusiones generales a partir de un solo caso de estudio. Solo se hacen comentarios sobre la experiencia en particular y la forma como fue aplicada la metodología.

1. MARCO TEÓRICO

En esta parte del documento se hace una presentación teórica de XP como metodología de desarrollo y de los principios teóricos que inspiraron esta metodología.

Se inicia con una descripción del manifiesto ágil, documento que sirve como punto de partida para las metodologías que reciben el mismo nombre. Prosiguiendo con una conceptualización importante sobre XP como metodología de desarrollo, la cual consta de los valores, principios y el alcance de la misma.

Finalmente se entra en detalle con cada una de las etapas de desarrollo (planeación, diseño, codificación y pruebas) describiendo cada uno de los aspectos que la componen.

1.1. MANIFIESTO ÁGIL

En febrero del 2001 se realizó una reunión en una casa de campo en las montañas de Wasatch en Utah, a la cual asistieron diecisiete personas entre las cuales se encontraban Kent Beck, Mike Beedle, Alistair Cockburn, entre otros[1]. El objetivo de este encuentro era discutir sobre procesos y técnicas para el desarrollo de software, como resultado se utilizó el término "ágil" para referirse a los métodos alternativos a las metodologías ya establecidas en ese momento. Es así como nace el documento del Manifiesto Ágil, en el cual se resumían los ideales de estos métodos alternativos.

En dicha reunión se afirmó que en la labor de desarrollar software debía valorarse:

- A los individuos y su interacción, por encima de los procesos y las herramientas.
- El software que funciona, por encima de la documentación exhaustiva.
- La colaboración con el cliente, por encima de la negociación contractual.
- La respuesta al cambio, por encima del seguimiento de un plan.

Como resultado de los anteriores valores, se derivan una serie de reglas o prácticas:

- Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.
- Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
- Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los periodos breves.
- Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.

[1] Manifesto for Agile Software Development

- Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
- La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software que funciona es la principal medida del progreso.
- Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica enaltece la agilidad.
- La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.
- En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

1.2. INTRODUCCIÓN A XP

XP resalta una serie de valores y principios que deben tenerse en cuenta y practicarlos durante el tiempo de desarrollo que dure el proyecto. Al final de este apartado se enuncian algunas de las características que deben tener los proyectos que se realicen con XP.

1.2.1. Valores

Más que una metodología, XP se considera una disciplina, la cual está sostenida por valores y principios propios de las metodologías ágiles. Existen cuatro valores que cumplen su papel como pilares en el desarrollo de las metodologías livianas:

- La comunicación**.** En la metodología XP es muy importante que exista un ambiente de colaboración y comunicación al interior del equipo de desarrollo, así como en la interacción de éste con el cliente. En XP la interacción con el cliente es tan estrecha, que es considerado parte del equipo de desarrollo.

- La simplicidad. Este valor se aplica en todos los aspectos de la programación extrema. Desde diseños muy sencillos donde lo más relevante es la funcionalidad necesaria que requiere el cliente, hasta la simplificación del código mediante la refactorización del mismo. La programación XP no utiliza sus recursos para la realización de actividades complejas, sólo se desarrolla lo que el cliente demanda, de la forma más sencilla.

- La retroalimentación. Se presenta desde el comienzo del proyecto, ayuda a encaminarlo y darle forma. Ésta se presenta en los dos sentidos, por parte del equipo de trabajo hacia el cliente, con el fin de brindarle información sobre la evolución del sistema, y desde el cliente hacia el equipo en los aportes a la construcción del proyecto.

- El coraje. El equipo de desarrollo debe estar preparado para enfrentarse a los continuos cambios que se presentarán en el transcurso de la actividad. Cada integrante debe tener el valor de exponer los problemas o dudas que halle en la realización del proyecto. Aún con estas variaciones, las jornadas de trabajo deben proporcionar el máximo rendimiento.

1.2.2. Prácticas

A partir de los valores se plantea una serie de prácticas que sirven de guía para los desarrolladores en esta metodología. Una de los aspectos más importantes para XP son las doce reglas que se plantean, las cuales se caracterizan por su grado de simplicidad y por su enfoque en la practicidad, además de que cada regla se complementa con las demás. A continuación se realizará una breve descripción de cada una de ellas.

- El desarrollo está dirigido por pruebas. Antes de realizar una unidad de código, es necesario contar con su respectiva unidad de pruebas. El programador realiza pruebas dirigidas al funcionamiento de nuevas adiciones o módulos al sistema. El cliente con ayuda del tester se encarga de diseñar las pruebas de aceptación, cuyo propósito es verificar que las historias de usuario se hayan implementado correctamente.

- El juego de la planificación. Desde el comienzo del desarrollo se requiere que el grupo y el cliente tengan una visión general y clara del proyecto, es decir, deben entender y estar de acuerdo con lo que el "otro" plantee. En el transcurso del proyecto se realizan diferentes reuniones, con el fin de organizar las tareas e ideas que surgen tanto por parte del cliente como por el equipo.

- Cliente in-situ. El cliente, o un representante del mismo, deben estar en el sitio de desarrollo para solucionar las preguntas o dudas que se puedan presentar a medida que se realice el proyecto.

- Programación en parejas. XP propone que exista una pareja de programadores por monitor y teclado, como medida para aumentar la calidad del código. Esta práctica busca reducir los errores de codificación, mientras uno de los programadores busca una forma de dar funcionalidad a un módulo, el otro programador aprueba dicho código y busca la forma de simplificarlo.

- Entregas pequeñas. En la programación extrema se realizan entregas constantes de módulos funcionales completos, de tal forma que en todo momento el cliente tiene una parte de aplicación funcionando. En XP no existe el desarrollo incompleto de una tarea, ésta se ejecuta en su totalidad o no se hace.

- Refactorización sin piedad. El código se revisa de forma permanente para depurarlo y simplificarlo, buscando la forma de mejorarlo. La refactorización se realiza durante todo el proceso de desarrollo.

- Integración continúa del código. El código de los módulos debe ser integrado a cortos plazos de tiempo, preferiblemente no mayores a un día. Esto facilita la búsqueda y la corrección de errores de codificación e integración que se presenten en el proceso.

- Diseño simple. Sólo se realiza lo necesario para que la aplicación cumpla con la funcionalidad requerida por el cliente. No es conveniente realizar diseños complejos que posiblemente no aporten soluciones claras al proyecto, y que a la hora de cambiar los requerimientos se conviertan en una gran barrera de tiempo.

- Utilización de metáforas del sistema. Para el mejor entendimiento de los elementos del sistema por parte del equipo de desarrollo se acude a la utilización de metáforas, como una forma de universalizar el lenguaje del sistema.

- Propiedad colectiva del código. El código no es conocido por una sola persona del grupo de trabajo, esto facilita implementar cambios al programa por parte de otros integrantes del equipo.

- Convenciones de código. La aplicación de estándares de programación al código fuente de la aplicación, permite que todas las personas que conforman el grupo de trabajo puedan entender y realizar modificaciones al código del sistema.

- No trabajar horas extras. Es preferible volver a estimar los tiempos de entrega. Con esta práctica se busca utilizar al máximo el rendimiento y energía del programador.

1.2.3. Alcance de XP

La programación extrema es conveniente en ciertas situaciones, pero también es necesario saber que presenta controversia en otras. Esta metodología es aplicable con resultados positivos a proyectos de mediana y pequeña envergadura, donde los grupos de trabajo no superan 20 personas.

Otro aspecto importante en la selección de esta metodología radica en el ambiente cambiante que se presenta en los requerimientos de la aplicación. La metodología XP está encaminada hacia los desarrollos que requieren de cambios continuos en el transcurso de un proyecto. La metodología es recomendada para proyectos en los cuales el costo de cambio no se incremente a medida que transcurre vida del mismo.

Los proyectos realizados bajo esta metodología cumplen con lo estrictamente necesario en su funcionalidad en el momento necesario: hacer lo que se necesita cuando se necesita. En XP no es conveniente precipitarse o adelantarse a las tareas que se han establecido previamente sin el consentimiento del cliente, estos hechos conllevan a inyectar complejidad al sistema, alejándolo del concepto de simplicidad.

1.3. PLANEACIÓN

La planeación es la etapa inicial de todo proyecto en XP. En este punto se comienza a interactuar con el cliente y el resto del grupo de desarrollo para descubrir los requerimientos del sistema. En este punto se identifican el número y tamaño de las iteraciones al igual que se plantean ajustes necesarios a la metodología según las características del proyecto.

En este apartado se tendrán en cuenta ocho elementos, los cuales son los siguientes. Historias de usuario, velocidad del proyecto, iteraciones, entregas pequeñas, reuniones, roles en XP, traslado del personal y ajuste a XP.

1.3.1. Historias de usuario

El sistema es desarrollado para el cliente, por lo tanto, el usuario es quien decide que tareas realizará la aplicación. Este planteamiento se desarrolla a lo largo del proyecto: el cliente es quien decide que hacer. Como primer paso, se debe proporcionar una idea clara de lo que será el proyecto en sí.

Las historias de usuario son utilizadas como herramienta para dar a conocer los requerimientos del sistema al equipo de desarrollo. Son pequeños textos en los que el cliente describe una actividad que realizará el sistema; la redacción de los mismos se realiza bajo la terminología del cliente, no del desarrollador, de forma que sea clara y sencilla, sin profundizar en detalles.

Se puede considerar que las historias de usuario en XP juegan un papel similar a los casos de uso en otras metodologías, pero en realidad son muy diferentes. Las historias de usuario sólo muestran la silueta de una tarea a realizarse. Por esta razón es fundamental que el usuario o un representante del mismo se encuentren disponibles en todo momento para solucionar dudas, estas no proporcionan información detallada acerca de una actividad específica.

Las historias de usuario también son utilizadas para estimar el tiempo que el equipo de desarrollo tomará para realizar las entregas. En una entrega se puede desarrollar una o varias historias de usuario, esto depende del tiempo que demore la implementación de cada una de las mismas.

1.3.2. Velocidad del proyecto

Es una medida de la capacidad que tiene el equipo de desarrollo para evacuar las historias de usuario en una determinada iteración. Esta medida se calcula totalizando el número de historias de usuario realizadas en una iteración. Para la iteración siguiente se podrá

(teóricamente) implementar el mismo número de historias de usuario que en la iteración anterior

Cabe recordar que la velocidad del proyecto ayuda a determinar la cantidad de historias que se pueden implementar en las siguientes iteraciones, aunque no de manera exacta. La revisión continua de esta métrica en el transcurso del proyecto se hace necesaria, ya que las historias varían según su grado de dificultad, haciendo inestable la velocidad de la realización del sistema.

1.3.3. Iteraciones

En la metodología XP, la creación del sistema se divide en etapas para facilitar su realización. Por lo general, los proyectos constan de más de tres etapas, las cuales toman el nombre de iteraciones, de allí se obtiene el concepto de metodología iterativa. La duración ideal de una iteración es de una a tres semanas.

Para cada iteración se define un módulo o conjunto de historias que se van a implementar. Al final de la iteración se obtiene como resultado la entrega del módulo correspondiente, el cual debe haber superado las pruebas de aceptación que establece el cliente para la verificar el cumplimiento de los requisitos. Las tareas que no se realicen en una iteración son tomadas en cuenta para la próxima iteración, donde se define, junto al cliente, si se deben realizar o si deben ser removidas de la planeación del sistema.

1.3.4. Entregas Pequeñas

La duración de una iteración varía entre una y tres semanas, al final de la cual habrá una entrega de los avances del producto, los cuales deberán ser completamente funcionales. Estas entregas deben caracterizarse por ser frecuentes.

1.3.5. Reuniones

El planeamiento es esencial para cualquier tipo de metodología, es por ello que XP requiere de una revisión continua del plan de trabajo. A pesar de ser una metodología que evita la documentación exagerada, es muy estricta en la organización del trabajo.

- Plan de entregas

Al comenzar el proyecto se realiza una reunión entre el equipo de trabajo y los clientes. En dicha reunión se define el marco temporal de la realización del sistema. El cliente expone las historias de usuario a los integrantes de grupo, quienes estimarán el grado de dificultad de la implementación de cada historia.

Las historias de usuario son asignadas a las diferentes iteraciones según su orden de relevancia para el proyecto. En el proceso de selección de las historias de usuario para cada

iteración, se tiene en cuenta que la suma de las estimaciones sea aproximada a la velocidad del proyecto de la iteración pasada.

En esta reunión se predicen los tiempos que se utilizaran en la realización de las diferentes etapas del proyecto, los cuales no son datos exactos pero proporcionan una base del cronograma.

Finalmente a partir de las historias de usuario, el cliente plantea las pruebas de aceptación con las cuales se comprueba que cada una de éstas ha sido correctamente implementada.

- Inicial de Iteración.

Al comenzar una iteración se realiza una reunión de la misma, donde se organizan las actividades de programación a realizar. Las historias de usuario son traducidas a tareas y asignadas a los desarrolladores.

Los desarrolladores estiman los tiempos para la realización de las tareas. Cada tarea se estima de uno a tres de días de programación ideales o sin distracciones. Estas estimaciones son más exactas que las realizadas en la planeación de entregas, por lo tanto no deben exceder la velocidad de proyecto de la iteración anterior. De ser así, se consulta con el cliente para determinar que historias de usuario se pospondrán para iteraciones futuras.

- Diarias o "stand-up meeting"

Estas reuniones se realizan al comenzar la jornada laboral. Todo el equipo de desarrollo se reúne para exponer los problemas e ideas que se estén presentando, esto con el fin que el equipo en conjunto construya una mejor solución.

Es de vital importancia evitar las discusiones largas, ya que se está utilizando tiempo laboral que puede ser destinado a la construcción del sistema. También debe evitarse las conversaciones separadas, las dudas que se presenten serán solucionadas por el equipo en conjunto.

1.3.6. Roles XP

En esta metodología se utiliza el concepto de roles para organizar quienes se encargaran de cada una de las actividades que deben realizarse en el transcurso del proyecto. Cada uno de estos papeles son desempeñados por uno o varios integrantes del grupo, sin descartar la posibilidad de rotar los roles entre el equipo durante la realización del sistema.

El jefe de proyecto tiene como responsabilidad la dirección y organización de las reuniones que se realizan durante el proyecto. Es erróneo afirmar que entre sus tareas se encuentra decir que hacer, cuando hacer y de revisar cómo se desarrolla el sistema, para ello se cuenta con el apoyo del cliente, el tracker y los demás miembros del grupo.

El usuario o cliente determina qué se va a construir en el sistema, además de decidir el orden en que se entregarán cada segmento del proyecto. Es parte fundamental del equipo XP (se menciona su importancia como una de las prácticas), en todo proyecto debe existir un cliente. Además, tiene como tarea establecer las pruebas de aceptación, las cuales determinan si el sistema cumple con los requerimientos del usuario.

En el grupo de los programadores se encuentran además los diseñadores y los analistas. Los programadores son quienes construyen el sistema y realizan las pruebas correspondientes a cada módulo o unidad de código. Cuando surgen dudas o preguntas que afectan decisiones sobre la funcionalidad del sistema (las decisiones técnicas son solucionadas gracias a las habilidades de los programadores), el programador no debe hacer suposiciones acerca de lo que el cliente quiere; en este caso, debe dirigirse al mismo y aclarar la situación.
El entrenador (coach) es el responsable de que el proceso se realice de forma correcta. Se asegura de que los conceptos de la metodología se apliquen al proyecto, además de brindar ayuda continua a los demás integrantes del equipo.

El tester o quien realiza las pruebas, colabora en la realización de las pruebas de aceptación y es quien muestra los resultados de las mismas. En este proceso, ayuda al cliente a diseñar tales pruebas y a verificar que las pruebas sean aprobadas.

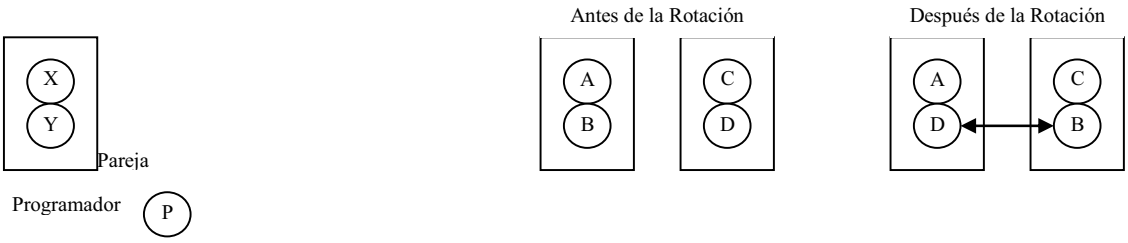
El rastreador (tracker) tiene como tarea observar la realización del sistema. Varias veces por semana cuestiona a los integrantes del equipo para anotar sus logros y avances. Mantiene datos históricos del proyecto.

1.3.7. Traslado de personal

Al mover el personal se evitan problemas relacionados con la pérdida de conocimiento y cuellos de botella. Todos los miembros del grupo deben tener suficiente conocimiento de la estructura del código de modo tal que se eviten las islas de conocimiento las cuales son susceptibles de generar pérdidas de información importante.

En la medida que todos los programadores entienden todas las partes del programa se evita que unos tengan una carga de trabajo muy alta mientras que otros no tengan mucho trabajo por hacer.
La programación en parejas se convierte en una herramienta muy importante para lograr el objetivo del traslado de personal sin que se pierda el rendimiento. Esto se logra haciendo que un miembro de la pareja se traslade mientas que el otro continúe el desarrollo con un nuevo compañero.

Ilustración 1: Rotación de personal



1.3.8. Ajustar XP

Todos los proyectos tienen características específicas por lo cual XP puede ser modificado para ajustarse bien al proyecto en cuestión. Al iniciar el proyecto se debe aplicar XP tal como es, sin embargo no se debe dudar en modificar aquellos aspectos en que no funcione. Eso no quiere decir que los desarrolladores pueden hacer lo que se les antoje. Antes de implementarse un cambio, este debe ser discutido y aprobado por el grupo.

1.4. DISEÑO

En XP solo se diseñan aquellas historias de usuario que el cliente ha seleccionado para la iteración actual por dos motivos: por un lado se considera que no es posible tener un diseño completo del sistema y sin errores desde el principio. El segundo motivo es que dada la naturaleza cambiante del proyecto, el hacer un diseño muy extenso en las fases iniciales del proyecto para luego modificarlo, se considera un desperdicio de tiempo

Es importante resaltar que esta tarea es permanente durante la vida del proyecto partiendo de un diseño inicial que va siendo corregido y mejorado en el transcurso del proyecto.

Los aspectos que se tratarán a continuación son: simplicidad en el diseño, metáfora del sistema, tarjetas CRC, spike solution, no solucionar antes de tiempo y refactoring.

1.4.1. Simplicidad en el diseño

Una de las partes más importantes de la filosofía XP es la simplicidad en todos los aspectos. Se considera que un diseño sencillo se logra más rápido y se implementa en

menos tiempo, por lo cual esto es lo que se busca. La idea es que se haga el diseño más sencillo que cumpla con los requerimientos de las historias de usuario.

Sobre los diagramas, se es muy claro que se pueden usar siempre que no tome mucho tiempo en realizarlos, que sean de verdadera utilidad y que se esté dispuesto a "tirarlos a la basura". En XP se prefiere tener una descripción del sistema o parte de él, en lugar de una serie de complejos diagramas que probablemente tomen más tiempo y sean menos instructivos

1.4.2. Metáfora del sistema

Se trata de plasmar la arquitectura de sistema en una "historia" con la cual se le dé al grupo de desarrollo una misma visión sobre el proyecto además de brindarles un primer vistazo muy completo a los nuevos integrantes del grupo para hacer su adaptación más rápida.

Es muy importante dentro del desarrollo de la metáfora darle nombres adecuados a todos los elementos del sistema constantemente, y que estos correspondan a un sistema de nombres consistente. Esto será de mucha utilidad en fases posteriores del desarrollo para identificar aspectos importantes del sistema.

1.4.3. Tarjetas de clase, responsabilidad, colaboración (CRC cards)

La principal funcionalidad que tienen estas, es ayudar a dejar el pensamiento procedimental para incorporarse al enfoque orientado a objetos. Cada tarjeta representa una clase con su nombre en la parte superior, en la sección inferior izquierda están descritas las responsabilidades y a la derecha las clases que le sirven de soporte.

En el proceso de diseñar el sistema por medio de las tarjetas CRC como máximo dos personas se ponen de pie adicionando o modificando las tarjetas, prestando atención a los mensajes que éstas se transmiten mientras los demás miembros del grupo que permanecen sentados, participan en la discusión obteniendo así lo que puede considerarse un diagrama de clases preliminar.

1.4.4. Soluciones puntuales (Spike Solution)

En muchas ocasiones los equipos de desarrollo se enfrentan a requerimientos de los clientes (en este caso historias de usuario) los cuales generan problemas desde el punto de vista del diseño o la implementación. Spike Solution, es una herramienta de XP para abordar este inconveniente.
Se trata de una pequeña aplicación completamente desconectada del proyecto con la cual se intenta explorar el problema y propone una solución potencial. Puede ser burda y simple, siempre que brinde la información suficiente para enfrentar el problema encontrado.

1.4.5. No solucionar antes de tiempo

Los desarrolladores tienden a predecir las necesidades futuras e implementarlas antes. Según mediciones, esta es una práctica ineficiente, concluyendo que tan solo el 10% de las soluciones para el futuro son utilizadas, desperdiciando tiempo de desarrollo y complicando el diseño innecesariamente.

En XP sólo se analiza lo que se desarrollará en la iteración actual, olvidando por completo cualquier necesidad que se pueda presentar en el futuro, lo que supone uno de los preceptos más radicales de la programación extrema.

1.4.6. Refactorización (Refactoring)

Como se trató al principio de este apartado, el diseño es una tarea permanente durante toda la vida del proyecto y la refactorización concreta este concepto. Como en cualquier metodología tradicional en XP se inicia el proceso de desarrollo con un diseño inicial. La diferencia es que en las metodologías tradicionales este diseño es tan global y completo como se es posible tomando generalmente mucho tiempo en lograrse y con la creencia de que si se ven forzados a modificarlo será un fracaso para el grupo de desarrollo. El caso de XP es el opuesto. Se parte de un diseño muy general y simple que no debe tardar en conseguirse, al cual se le hacen adiciones y correcciones a medida que el proyecto avanza, con el fin de mantenerlo tanto correcto como simple.

La refactorización en el código pretende conservarlo tan sencillo y fácil de mantener como sea posible. En cada inspección que se encuentre alguna redundancia, funcionalidad no necesaria o aspecto en general por corregir, se debe rehacer esa sección de código con el fin de lograr las metas de sencillez tanto en el código en sí mismo como en la lectura y mantenimiento.

Estas prácticas son difíciles de llevar a cabo cuando se está iniciando en XP por varios motivos. En primer lugar debido el temor que genera en los equipos de desarrollo cambiar algo que ya funciona bien sea a nivel de diseño o implementación. Sin embargo si se cuenta con un esquema de pruebas completo y un sistema de automatización para las mismas se tendrá éxito en el proceso. El otro motivo es la creencia que es más el tiempo que se pierde en refactoring que el ganado en sencillez y mantenimiento. Según XP la ganancia obtenida en refactoring es tan relevante que justifica suficientemente el esfuerzo extra en corrección de redundancias y funcionalidades innecesarias.

1.5. CODIFICACIÓN

La codificación es un proceso que se realiza en forma paralela con el diseño y la cual está sujeta a varias observaciones por parte de XP consideradas controversiales por algunos expertos tales como la rotación de los programadores o la programación en parejas.

Además de los mencionados temas, el lector encontrará a continuación una descripción de los siguientes temas: cliente siempre presente, codificar primero la prueba, integración secuencial e integraciones frecuentes.

1.5.1. Cliente siempre presente.

Uno de los requerimientos de XP es que el cliente esté siempre disponible. No solamente para solucionar las dudas del grupo de desarrollo, debería ser parte de éste. En este sentido se convierte en gran ayuda al solucionar todas las dudas que puedan surgir, especialmente cara a cara, para garantizar que lo implementado cubre con las necesidades planteadas en las historias de usuario.

1.5.2. Codificar primero la prueba

Cuando se crea primero una prueba, se ahorra mucho tiempo elaborando el código que la haga pasar, siendo menor el tiempo de hacer ambos procesos que crear el código solamente.

Una de las ventajas de crear una prueba antes que el código es que permite identificar los requerimientos de dicho código. En otras palabras, al escribir primero las pruebas se encuentran de una forma más sencilla y con mayor claridad todos los casos especiales que debe considerar el código a implementar. De esta forma el desarrollador sabrá con completa certeza en qué momento ha terminado, ya que habrán pasado todas las pruebas.

1.5.3. Programación en parejas

Todo el código debe ser creado por parejas de programadores sentados ambos frente a un único computador lo que en principio representa una reducción de un 50% en productividad, sin embargo, según XP no es tal la pérdida. Se entiende que no hay mucha diferencia, en lo que a la cantidad se refiere, entre el código producido por una pareja bajo estas condiciones que el creado por los mismos miembros trabajando en forma separada, con la excepción que uno o ambos programadores sean muy expertos en la herramienta en cuestión.

Cuando se trabaja en parejas se obtiene un diseño de mejor calidad y un código más organizado y con menores errores que si se trabajase solo, además de la ventaja que representa contar con un compañero que ayude a solucionar inconvenientes en tiempo de codificación, los cuales se presentan con mucha frecuencia.

Se recomienda que mientras un miembro de la pareja se preocupa del método que se está escribiendo el otro se ocupe de cómo encaja éste en el resto de la clase.

1.5.4. Integración secuencial

Uno de los mayores inconvenientes presentados en proyectos de software tiene que ver con la integración, sobre todo si todos los programadores son dueños de todo el código. Para saldar este problema han surgido muchos mecanismos, como darle propiedad de

determinadas clases a algunos desarrolladores, los cuales son los responsables de mantenerlas actualizadas y consistentes. Sin embargo, sumado al hecho que esto va en contra de la propiedad colectiva del código no se solucionan los problemas presentados por la comunicación entre clases.
XP propone que se emplee un esquema de turnos con el cual solo una pareja de programadores integre a vez. De esta forma se tiene plena seguridad de cuál es la última versión liberada y se le podrán hacer todas las pruebas para garantizar que funcione correctamente. A esto se le conoce como integración secuencial.

1.5.5. Integraciones frecuentes.

Se deben hacer integraciones cada pocas horas y siempre que sea posible no debe transcurrir más un día entre una integración y otra. De esta forma se garantiza surjan problemas como que un programador trabaje sobre versiones obsoletas de alguna clase.

Es evidente que entre más se tarde en encontrar un problema más costoso será resolverlo y con la integración frecuente se garantiza que dichos problemas se encuentre más rápido o aún mejor, sean evitados por completo.

1.5.6. Estándares y propiedad colectiva del código

Así como se recomienda que la programación se haga siempre en parejas ubicadas en un único computador, también se aconseja que estas se vayan rotando no solo de compañero sino de partes del proyecto a implementar, con el fin de que se logre tener una propiedad colectiva del código. Todos y cada uno de los programadores tienen suficiente conocimiento del código de los demás de modo tal que en cualquier momento puedan continuar la codificación que alguien más empezó sin que represente un traumatismo para nadie.
Uno de los principales motivos por los que se promueve esta práctica dentro de la programación extrema es la posibilidad que brinda de evitar los cuellos de botella. Si una pareja de programadores se retrasa debido a inconvenientes no estimados pueden ser ayudados o reemplazados por otra pareja que al conocer el código no tendrá que familiarizarse con él.

Para lograr lo anterior se recomienda el establecimiento de estándares en la codificación, de modo tal que todo el código escrito por el grupo de desarrollo parezca hecho por una sola persona. No se establecen los aspectos específicos a tener en cuenta dentro de estos estándares, sin embargo se aconseja que sean de total aceptación por parte del equipo.

Si bien en la actualidad existen herramientas de soporte en la integración tales como CVS[2] las cuales ayudan a sobrellevar algunos de los inconvenientes del trabajo en paralelo, es recomendable prestar atención al mecanismo de integración, para evitar problemas en el

[2] Concurrent Versions System

proyecto que reduzcan bien sea la calidad del proyecto o el rendimiento del equipo de desarrollo.

1.6. PRUEBAS

XP enfatiza mucho los aspectos relacionados con las pruebas, clasificándolas en diferentes tipos y funcionalidades específicas, indicando quién, cuándo y cómo deben ser implementadas y ejecutadas.

Del buen uso de las pruebas depende el éxito de otras prácticas, tales como la propiedad colectiva del código y la refactorización. Cuando se tienen bien implementadas las pruebas no habrá temor de modificar el código del otro programador en el sentido que si se daña alguna sección, las pruebas mostrarán el error y permitirán encontrarlo. El mismo criterio se aplica a la refactorización. Uno de los elementos que podría obstaculizar que un programador cambie una sección de código funcional es precisamente hacer que esta deje de funcionar. Si se tiene un grupo de pruebas que garantice su buen funcionamiento, este temor se mitiga en gran medida.

Según XP se debe ser muy estricto con las pruebas. Sólo se deberá liberar una nueva versión si esta ha pasado con el cien por ciento de la totalidad de las pruebas. En caso contrario se empleará el resultado de estas para identificar el error y solucionarlo con mecanismos ya definidos.

1.6.1. Pruebas unitarias

Estas pruebas se aplican a todos los métodos no triviales de todas las clases del proyecto con la condición que no se liberará ninguna clase que no tenga asociada su correspondiente paquete de pruebas. Uno de los elementos más importantes en estas es que idealmente deben ser construidas antes que los métodos mismos, permitiéndole al programador tener máxima claridad sobre lo que va a programar antes de hacerlo, así como conocer cada uno de los casos de prueba que deberá pasar, lo que optimizará su trabajo y su código será de mejor calidad.
Deben ser construidas por los programadores con el empleo de algún mecanismo que permita automatizarlas de modo tal que tanto su implementación y ejecución consuman el menor tiempo posible permitiendo sacarles el mejor provecho
EL empleo de pruebas unitarias completas facilitan la liberación continua de versiones por cuanto al implementar algo nuevo y actualizar la última versión, solo es cuestión de ejecutar de forma automática las pruebas unitarias ya creadas para saber que la nueva versión no contiene errores.

1.6.2. Pruebas de aceptación

Las pruebas de aceptación, también llamadas pruebas funcionales son supervisadas por el cliente basándose en los requerimientos tomados de las historias de usuario. En todas las iteraciones, cada una de las historias de usuario seleccionadas por el cliente deberá tener una o más pruebas de aceptación, de las cuales deberán determinar los casos de prueba e identificar los errores que serán corregidos.

Las pruebas de aceptación son pruebas de caja negra, que representan un resultado esperado de determinada transacción con el sistema. Para que una historia de usuario se considere aprobada, deberá pasar todas las pruebas de aceptación elaboradas para dicha historia.

Es importante resaltar la diferencia entre las pruebas de aceptación y las unitarias en lo que al papel del usuario se refiere. Mientras que en las pruebas de aceptación juega un papel muy importante seleccionando los casos de prueba para cada historia de usuario e identificando los resultados esperados, en las segundas no tiene ninguna intervención por ser de competencia del equipo de programadores.

1.6.3. Cuando se encuentra un error

Al momento de encontrar un error debe escribirse una prueba antes de intentar corregirlo. De esta forma tanto el cliente logrará tener completamente claro cuál fue y dónde se encontraba el mismo como el equipo de desarrollo podrá enfocar mejor sus esfuerzos para solucionarlo. Por otro lado se logrará evitar volver a cometerlo.

Si el error fue reportado por el cliente y este creó la correspondiente prueba de aceptación junto al equipo de desarrollo, el programador encargado podrá a su vez producir nuevas pruebas unitarias que le permita ubicar la sección específica donde el error se encuentra.

1.7. PROCESO DE DESARROLLO EN XP

Todo proyecto de software en XP inicia con una o varias reuniones con el cliente, en las cuales se da claridad a la necesidad puntual del mismo a través de las historias de usuario. Estas también sirven de base para crear una metáfora del sistema con el cual todo el equipo de trabajo tendrá una idea general de la aplicación a implementar. Con base en las historias de usuario se crean las pruebas de aceptación las cuales deben ser diseñadas antes de iniciar la codificación.

Concluida esta etapa, se debe acordar un plan de entregas con el cliente del cual surge el número inicial de iteraciones y duración de las mismas. Esta reunión de entregas puede repetirse en el transcurso del proyecto, siempre que la velocidad del mismo cambie lo suficiente para tener que replantear el plan de entregas o que surjan nuevas historias de usuario que justifiquen la alteración de dicho plan. Dentro de esta(s) reunión(es) de planeación de entregas debe considerarse la realización de algunos Spike Solution para

tener claridad sobre la dificultad y tiempo necesario para implementar determinada historia de usuario.

Toda iteración debe iniciar con una reunión en la que se da claridad a las tareas a desarrollar, basándose en el plan de entregas, la velocidad del proyecto y las historias de usuario sin concluir de la iteración anterior. De esta reunión se obtiene un plan que sirve de hoja de ruta en el transcurso de la iteración.

Todos los días debe hacerse un reunión corta en la cual se discute el avance de la iteración basándose en el plan obtenido de la reunión de inicio de iteración y las tareas concluidas con el cual se acuerda el trabajo del día.

2. PRESENTACIÓN

En este capítulo se hace una introducción a las condiciones del entorno que rodearon al proyecto entre las cuales se resaltan aspectos relacionados con el cliente y el negocio para el cual se desarrolló el proyecto. Además se hace una descripción de las herramientas que se emplearon en el desarrollo del proyecto y el motivo por el que fueron elegidas.

2.1. HERRAMIENTAS EMPLEADAS

Se optó por seleccionar herramientas libres para el desarrollo de la aplicación. Por un lado se empleó JAVA como herramienta de desarrollo mientras que como motor de base de datos se decidió por PostreSQL. A continuación se detalla cada una de éstas planteando los motivos por los cuales fueron seleccionadas

2.1.1. JAVA:

Se trata de un poderoso y flexible lenguaje de programación con el cual se puede desarrollar desde aplicaciones para celulares hasta páginas web. Obviamente se convierte en una herramienta óptima para desarrollar aplicativos de escritorio.

El primer motivo por el que se seleccionó a JAVA como herramienta de desarrollo, es el amplio conocimiento que ambos programadores tienen del lenguaje. En segundo lugar existe una API de pruebas desarrollada para trabajar con JAVA especialmente diseñada para proyectos desarrollados con XP.

2.1.2. NetBeans:

Es un IDE de licenciado libre para desarrollar aplicaciones en JAVA. Se optó por la versión 5.5 la cual al momento del desarrollo del programa era la más reciente.

Si bien no es el único IDE disponible para JAVA, a juicio de los programadores involucrados en el proyecto, es el más adecuado, por lo cual se convirtió en la mejor elección. Por otro lado cuenta con soporte para JUnit, herramienta seleccionada para realizar las pruebas.

2.1.3. JUnit:

Es un API para realizar pruebas que fue diseñado para ser empleado en JAVA. Un aspecto importante es que cumple con la mayoría de las recomendaciones realizadas por XP en lo que a pruebas se refiere, de las cuales se destaca el permitir hacer pruebas autónomas. Por otro lado, algunos autores lo recomiendan para desarrollar aplicaciones en JAVA empleando XP.

2.1.4. JasperReport e IReport:

Es una combinación de librerías de JAVA y software libre que permiten el diseño e implementación de reportes impresos. Entre las ventajas más importantes se resalta flexibilidad y facilidad de empleo.

2.1.5. PostgreSQL:

Es el motor de base de datos empleado para el proyecto. Se caracteriza por estar entre los motores de base de datos más estable y robustos, razones que motivaron su selección.

2.2. DESCRIPCIÓN DEL NEGOCIO

Se trata de un mini mercado en formato de autoservicio con un capital de aproximadamente quince millones de pesos el cual atiende a una población alrededor de 550 familias ubicado en la zona de Altos de Llano Grande cerca al Parque Industrial en la ciudad de Pereira.

Al momento de iniciar el proyecto, el negocio contaba con una caja registradora convencional la cual no ofrecía las funcionalidades que requería el cliente, por lo cual se acordó desarrollar un software que desempeñara las funcionalidades de un sistema POS (Point Of Sale) con elementos de administración de inventario que cumpliera las necesidades específicas del cliente. Sin embargo después de tener avanzado el proyecto, el mini mercado debió cerrar lo que no impidió terminar el proyecto debido que existía un compromiso con el cliente de terminarlo. Producto del cierre, el programa no pudo ser puesto a funcionar lo que hizo perder varios aspectos de implantación y depuración, que habría sido interesante documentar.

2.3. DESCRIPCIÓN DE CLIENTE Y USUARIO

Para este proyecto se contó con dos usuarios de los cuales uno era el cliente.

El primer usuario que además era el cliente se trataba de una persona mayor de cincuenta años el cual tenía conocimientos muy básicos sobre informática que limitaba de forma

importante su capacidad para plantear las funcionalidades que requería el programa. En muchas ocasiones se abstenía de manifestar necesidades por creer que estas no podrían ser incluidas dentro de la aplicación dificultando la comunicación con el equipo de desarrollo.
El segundo usuario fue una ex empleada del cliente de edad entre veinte y veinticinco años, la cual no solo contaba con conocimientos suficientes de informática, también tenía pleno conocimiento del funcionamiento del negocio ya que lo había administrado hacía poco. La participación de este segundo usuario fue muy importante para el proyecto debido que facilitó la comunicación y planteo numerosos aspectos para tener en cuenta en el proyecto que el primer usuario desconoció.

Posterior al ya mencionado cierre del negocio se presentó una calamidad doméstica en el entorno de ambos usuarios que significó un impacto importante en el desarrollo obligando al equipo de trabajo a desarrollar planes de contingencia para cumplir con el cronograma inicial.

3. PLANEACIÓN

A partir de este capítulo se describe la experiencia obtenida en la realización del proyecto. Inicialmente se comenta sobre cada uno de los aspectos que XP propone para etapa de planeación. Para cada uno de los elementos se enuncia lo que la teoría sobre XP recomienda contrastándola con la experiencia real en la realización del proyecto. Entre los elementos a discutir para este capítulo se encuentran las historias de usuario, el plan de entregas, lo relacionado con las iteraciones como las modificaciones que se aplicaron a XP para hacerla más adecuada para el proyecto.

En este capítulo se encontrará la misma estructura de la sección de planeación del marco teórico. Para cada apartado se verá una serie de ideas que resumen la teoría contrastada con la interpretación, aplicación y resultados en la práctica.

3.1. HISTORIAS DE USUARIO

3.1.1. Lo que dice XP

- Escritas por el usuario.
- Terminología del cliente.
- Bajo nivel de detalle
- Sirve de base para estimar los tiempos de implementación
- No deben ser menos de 20 ni más de 80

3.1.2. Experiencia en POSitron

Si bien el cliente no fue quien escribió personalmente las historias de usuario, fue él quien diseñó su contenido y dirigió la redacción de las mismas, debido a que no tenía los conocimientos necesarios en formato para elaborarlas. A pesar de lo anterior, el propósito de las mismas no se vio alterado de alguna forma, manteniendo no solamente la terminología del cliente al punto en que este fuera autosuficiente en la comprensión de su contenido, sino también su oficio como punto de partida en la planificación del proyecto.

Desde el punto de vista del nivel de detalle, se siguió la directiva en el sentido de no profundizar ni en descripciones ni en procesos, manteniéndolas de esta forma breves y claras. Sin embargo se logró abstraer la información suficiente de ellas para realizar su implementación sin requerir demasiadas aclaraciones por parte del cliente, siendo factor

fundamental para no ocasionar retrasos motivados por falta de claridad en los requerimientos.

Por otro lado es muy importante resaltar el papel fundamental que jugaron las historias de usuario en la estimación de los tiempos requeridos para el desarrollo del proyecto. Una vez recolectadas todas las historias de usuario, se hizo una reunión del equipo de trabajo donde se plantearon los tiempos necesarios para su implementación, los cuales resultaron en estimaciones inusualmente aproximadas de los tiempos de desarrollo en comparación con los realmente requeridos. Esto es importante resaltarlo debido al poco nivel de detalle que las historias de usuario tenían, significando la poca información sobre las implicaciones técnicas de su implementación.

Finalmente desde el punto de vista del número de historias de usuario, se obtuvo un total de veintiuno. Considerando por un lado la recomendación de que no sean menos de 20 ni más de 80, y por el otro que el tamaño del proyecto fue pequeño en comparación a otros sistemas POS similares; se deduce que en número es muy adecuado y por consiguiente el nivel de agrupación de tareas que cada historia de usuario tenía.

3.2. VELOCIDAD DEL PROYECTO

3.2.1. Lo que dice XP

- Número de historias de usuario o tareas de programación realizadas por iteración.
- Sirve de ayuda para estimar la cantidad de historias de usuario a implementar en una determinada iteración

3.2.2. Experiencia en POSitron

El número de historias de usuario realizadas por iteración no fue una buena medida de la velocidad del proyecto debido que no todas tenían el mismo nivel de dificultad y por tanto el mismo requerimiento de horas de desarrollo. Por esto se encontró que mientras en la segunda iteración se trabajaron menos horas semanales en comparación con las demás, también fue donde más historias de usuario se evacuaron, lo que supondría un nivel de rendimiento muy superior, lo que no es cierto. El motivo de este resultado fue que el nivel de dificultad y por lo tanto, el número de horas requeridas para las historias de usuario de la segunda iteración fue el más bajo de todo el proyecto.

Tabla 1: Velocidad del Proyecto

	Iteración 1	Iteración 2	Iteración 3	Iteración 4
Horas	46,00	41	42	30
Semanas	2	2	2	**1***
Horas semanales	23	20,5	21	30
Historias de usuario (Velocidad del proyecto)	**5**	**6**	**4**	**6**

Si bien esta medida de velocidad del proyecto fue tenida en cuenta para el análisis de tiempos, resultó de mayor utilidad estimar el número de horas que tomaría implementar cada historia de usuario y planificar las entregas acorde con esta medida. De esta forma, al tener la disponibilidad de cada desarrollador en horas por semana, se pudo estimar con mucha precisión cuántas historias de usuario podían ser asignadas en iteración. Esta medida de la velocidad del proyecto resultó tan acertada que permitió realizar un plan de entregas preciso, y lo más importante, cumplirlo. De esta forma se planearon cuatro entregas las cuales se realizaron sin necesidad de solicitud de aplazamientos o trabajar horas extras.
Una dificultad inesperada que se presentó con la velocidad del proyecto fue el refactoring, En la tercera iteración surgieron varias recomendaciones por parte del cliente que se convirtieron en refactoring, el cual no se había considerado dentro de ninguna de las dos medidas de velocidad. Producto de esta omisión el grupo de desarrollo debió re estimar el tiempo dedicado a la iteración para no tener que remover historias de usuario de esta. Finalmente la contingencia fue bien administrada por el grupo y se pudo cumplir con las metas de la iteración sin necesidad de trabajar más horas de las planeadas.

3.3. DIVISIÓN EN ITERACIONES.

3.3.1. Lo que dice XP

- El proyecto se divide en varias iteraciones
- La duración de una iteración varía entre una y tres semanas

* Semana Santa

3.3.2. Experiencia en POSitron

El proyecto fue dividido en cuatro iteraciones, por consiguiente se obtuvo un total de cuatro entregas para las cuales se desarrollaron partes de la aplicación completamente funcionales. La primera iteración se refirió al módulo de POST mientras las demás iteraciones se relacionaron con la manipulación de inventarios. Este orden se eligió debido a la naturaleza del negocio del cliente y la importancia que tiene para él la prestación del servicio a la comunidad.

En la planeación de iteraciones se tomaron dos semanas como período, excepto en la última, la cual sólo se fijó para una semana, ya que se redujo la carga de obligaciones externas al proyecto. La estimación de las dos semanas para una iteración fue correcta, se cumplieron a cabalidad dichos plazos y se realizaron entregas completas, es decir, sin posponer historias para posteriores iteraciones. En la última iteración también se demostró que la decisión de trabajar en una semana fue la más adecuada, debido que se incrementó el ritmo de trabajo en dicha semana.

Aunque las entregas fueron planeadas con fechas para su realización y la mayoría se cumplieron para dichas fechas, la reunión para la última entrega no se pudo realizar el día que se tenía planeado, no por razones de retraso en la implementación de la aplicación sino debido a la disponibilidad del cliente. El distanciamiento del cliente con el proyecto ocasionó una paralización irremediable del mismo mientras podía volver a participar, lo cual evidenció una peligrosa dependencia con el cliente.

3.4. ENTREGAS PEQUEÑAS

3.4.1. Lo que dice XP

- Entregas funcionales del proyecto frecuentemente

3.4.2. Experiencia en POSitron

Debido a que las iteraciones tenían una duración de 15 días, fue al término de este plazo que se realizaron entregas, las cuales siempre fueron funcionales, lo que quiere decir que al momento de la entrega estaban en condiciones de ser puestas en funcionamiento en las instalaciones del cliente. Esto representó un éxito en el desarrollo del proyecto ya que mantenía el interés del cliente en continuarlo debido a que estaba viendo resultados en el corto plazo.

Para las entregas se fijaron las siguientes fechas

Tabla 2: Fecha y duración de cada reunión de entrega

Iteración	Fecha	Duración
1ra	28 de enero	01:55:00
2da	17 de marzo	01:07:00
3ra	1ro de abril	01:34:00
4ta	20 de abril	02:00:00

Cada una de las reuniones se dividió en dos partes. Por un lado se hizo la entrega y familiarización de las funcionalidades acordadas, con la respectiva aprobación del cliente o sus recomendaciones para refactoring. Al final de la reunión, en un periodo no superior a 20 minutos, se confirmó cuáles serían las historias de usuario a implementar para la siguiente iteración y la fecha exacta de la próxima reunión.

3.5. PLAN DE ENTREGAS

3.5.1. Lo que dice XP

- Reunión al inicio del proyecto
- Cuales historias de usuario serán implementadas para cada entrega
- Grado de relevancia para cada historia de usuario
- Se aproxima el tiempo para la realización de cada iteración

3.5.2. Experiencia en POSitron

Se realizaron tres reuniones iniciales. En la primera reunión, se grabó en audio las historias de usuario que el primer representante del cliente expuso. Para la segunda reunión se contactó al segundo representante, el cual complementó las historias que el primero redactó, y agregó algunas más. La tercera reunión se realizó para que el primer cliente confirmara que las historias de usuario que se crearon en la primera y segunda reunión cubrían todos los requerimientos, además de profundizar en las definiciones de las historias.

La realización de dichas reuniones retrasó en varios días el desarrollo del proyecto, al no ser necesarias más de dos reuniones. Con el trabajo realizado en la primera y segunda reunión habría sido suficiente para continuar con el proyecto, ya que en estas reuniones los

clientes expusieron la totalidad de las historias de usuario. Debido a la claridad de las historias de usuario, la última reunión fue considerada inútil además de no ser recomendada por XP.

Aunque XP propone que el cliente sea quien decida cuáles historias se implementarán y cuál es el grado de importancia de cada una en la correspondiente iteración, la tarea de escoger las historias fue realizada por el grupo en conjunto, incluyendo al cliente, lo cual no generó problemas en las entregas de los módulos funcionales.

La clasificación de las historias no fue realizada estrictamente por su grado de importancia en el proyecto. Sólo se optó por desarrollar el módulo de servicio al cliente en la primera iteración, por tratarse de la actividad más importante en el negocio. En las demás iteraciones se priorizó la dependencia con los módulos ya implementados.

Para aproximar el tiempo que demoraría cada iteración, se tomó como medida dos semanas. Cada semana constaba de cuatro días (lunes, martes, jueves y viernes) en los que se trabajaban cuatro horas sin distracciones. Esta decisión fue acogida por el equipo debido a obligaciones externas al proyecto.

3.6. REUNIÓN INICIAL DE ITERACIÓN.

3.6.1. Lo que dice XP

- Las historias son traducidas a tareas al comienzo de cada iteración.
- Se estiman los tiempos para la realización de tareas en días ideales.

3.6.2. Experiencia en POSitron

En la reunión que realizaba el equipo de trabajo, se transformaba el contenido de las historias de usuario en responsabilidades que eran plasmadas en las CRC, para luego proceder a la asignación de dichas tareas a los programadores. Esta traducción facilitó la creación de clases y métodos iniciales de las mismas, ya que fue la etapa de diseño del proyecto. Esta etapa siempre fue realizada en conjunto por ambos miembros del equipo de desarrollo, lo que supone una pequeña modificación a XP, en el sentido que no plantea el diseño en esta reunión.

Las tareas fueron cuidadosamente estimadas en horas, no en días, lo cual aportó más precisión al momento de calcular las historias a implementar (lo que se plantea en la velocidad de proyecto). En dichas estimaciones no se tomó en cuenta el tiempo que se necesita para el refactoring, lo que se considera una omisión, sin embargo cuando se requirió, se les pudo hacer gestión sin afectar al proyecto. Al elegir las historias de usuario,

se trató de igualar el número de horas trabajadas en la anterior iteración, sin embargo en la primera se realizó un estimativo que resultó muy conveniente, lo que evitó el retraso en la realización de las historias a implementarse.

3.7. REUNIÓN MATINAL.

3.7.1. Lo que dice XP

- La realización de una reunión al comenzar el día laboral.
- Esta reunión se realizará en el sitio de trabajo del equipo.
- Evitar discusiones largas.

3.7.2. Experiencia en POSitron.

El beneficio que se desea obtener con esta práctica es la facilitación del proceso de comunicación del equipo. Debido al entorno de trabajo, en el cual no se contaba con un sitio físico, donde reunir al equipo para el desarrollo del proyecto, este aspecto no fue implementado a cabalidad.

Se recurrió a la utilización del Internet para apoyar la comunicación del equipo, lo cual resultó ser una excelente estrategia como solución al alejamiento de los puestos de trabajo y reemplazo de la reunión matinal. La comunicación de problemas y soluciones se realizó a lo largo de la jornada de trabajo, ya que se contaba con la permanencia en la red durante la mayor parte del día, además de las consultas telefónicas, y las que se realizaba antes o después de las clases de la universidad.

Las discusiones que surgieron no fueron muy largas. Los problemas que se plantearon no demandaron mucho tiempo para encontrar su solución, debido a que eran dudas relacionadas con la codificación.

3.8. MOVER PERSONAL.

3.8.1. Lo que dice XP

- Cada persona en un equipo debe conocer mucho del código de cada sección del proyecto.
- Rotar los programadores en partes del desarrollo.

3.8.2. Experiencia en POSitron

El objetivo que ambos miembros del grupo conocieran cada aspecto de la implementación fue fácil de lograr para el proyecto, debido a que por un lado solo se tenían dos desarrolladores y por el otro que el tamaño del proyecto así lo permitió.

Para el proyecto se presentaron dos temas que debieron ser tratados de forma especial para evitar islas de conocimiento. El primero fue la librería JUnit, con la cual se implementaron las pruebas y el segundo se trató de IReport, el cual fue empleado para la generación de reportes impresos. En ambos casos se encargó la realización de Spike Solution3 a uno de los miembros del grupo, el cual al finalizar éste, compartió el conocimiento adquirido con el otro compañero, de modo que ambos estuvieran capacitados para hacer implementaciones de éstos cuando fuera necesario.

Al no trabajar en parejas, la posibilidad de hacer una rotación de los programadores alrededor del proyecto, en el más estricto sentido de la palabra, era imposible, sin embargo cada decisión de diseño fue tomada por ambos miembros del grupo y los detalles de la implementación fueron permanentemente compartidos al otro compañero lográndose así el objetivo de evitar las islas de conocimiento y los cuellos de botella.

La rotación de programadores no es un fin en XP, es un medio para lograr la propiedad colectiva del código, la que se obtuvo gracias a que solo fueron dos programadores, que la comunicación fue muy buena, y que se hizo el diseño en conjunto.

3.9. MODIFICAR XP CUANDO SEA NECESARIO.

3.9.1. Lo que dice XP

- Corrija las reglas de XP cuando estas fallen.
- Las reglas solo se cambian cuando el equipo lo apruebe en conjunto

3.9.2. Experiencia en POSitron.

La principal fuente de modificaciones al proyecto, surgió del hecho que este fue desarrollado en un ambiente académico y sólo se involucraron dos programadores, modificando las siguientes normas.

- Jornada laboral: se adaptó a las obligaciones de los dos integrantes en el sentido de flexibilizar el número de horas dedicadas al proyecto semanalmente de acuerdo con las obligaciones académicas.

- Programación en parejas: ambos miembros no trabajaron en un mismo computador ni en el mismo sitio físico. Sin embargo todo el tiempo estuvieron conectados por medio del Internet.

[3] Spike Solution: Solución rápida. Ver marco teórico

- Cliente en sitio: se ajustó la participación del cliente en el proyecto permitiéndole no estar permanentemente en el sitio de trabajo, debido a su disponibilidad, compensando esto con una comunicación telefónica permanente.

- Mover personal: al ser solo dos programadores, no se hizo rotación de personal lo cual no afectó el objetivo de la propiedad colectiva del código.

Todos los cambios mencionados en los párrafos anteriores fueron discutidos por el grupo de desarrollo y aprobados por éste, antes de ser implementados, sin embargo en ocasiones y para detalles muy particulares se hicieron cambios unilaterales por alguno de los miembros del equipo de desarrollo, antes de ser discutidos para aprobarse, lo cual era inevitable ya que, si cada decisión referente a la aplicación de XP en el proyecto debía ser discutida con el compañero por irrelevante que éste fuera, el proyecto enfrentaría demasiados tropiezos; por tal motivo, se autorizó hacer algunos ajustes a la metodología en forma unilateral, siempre y cuando éstos fueran irrelevantes a juicio de quién los tomó y posteriormente los comunicara al resto del equipo.

En general, los cambios que se hicieron a XP fueron acertados y facilitaron la culminación exitosa del mismo; sin embargo lo más importante no fue tomar la decisión de hacerlos, si no la formalización de los mismos. De no haberse tenido un procedimiento para ajustar a XP en el transcurso del proyecto, se habría diluido la filosofía de la programación extrema y al final no se podría haber dicho que se aplicó XP al proyecto.

4. DISEÑO

A diferencia de las metodologías pesadas, el diseño se realiza durante todo el tiempo de vida del proyecto, siendo constantemente revisado y muy probablemente modificado debido a cambios presentados durante el desarrollo.

Tal como se presentó en el capitulo anterior, este capítulo presenta una estructura similar a la sección de diseño del marco teórico donde se observará para cada uno de los elementos constitutivos de dicha etapa una serie de ideas que describen la teoría contrastada con lo vivenciado durante la ejecución del proyecto.

Entre los elementos más importantes que menciona XP referentes al diseño está la simplicidad, las tarjetas CRC, el refactoring y Spike Solution. A continuación se detalla la experiencia vivida con cada uno de ellos.

4.1. SIMPLICIDAD

4.1.1. Lo que dice XP

- El diseño debe ser sencillo.
- Sólo se crearán diagramas útiles.

4.1.2. Experiencia en POSitron

En lo que respecta a la sencillez del diseño, se acogió la recomendación de XP, sólo invirtiendo el tiempo exclusivamente necesario en elaboración de diagramas y diseño de interfaz grafica. A consecuencia de esta decisión se debieron hacer algunos sacrificios. Al no haber hecho muchos diagramas, la orientación a objetos no fue tan completa, sacrificando de esta forma escalabilidad, versatilidad y elegancia del diseño, lo que fue considerado un precio justo a cambio del cumplimiento de los plazos. Desde el punto de vista de las interfaces, tampoco se invirtió mucho tiempo en su diseño, sin embargo se prestó mucha atención a ubicar los elementos tal y como el cliente las había solicitado y presentándolos en una forma elegante pero sencilla. A consecuencia de esto se notó una reacción muy positiva del cliente, manifestando conformidad con la apariencia visual de la

aplicación. Es importante aclarar que estos sacrificios en ningún momento representaron una baja en la calidad de la aplicación en cuanto a la funcionalidad se refiere.

En lo que se refiere a los diagramas, se crearon las tarjetas CRC, algunos diagramas de secuencia y el modelo Entidad Relación, del cual surgieron varias versiones en la medida que se incorporaban funcionalidades a la aplicación. Si bien no fueron muchos diagramas, si fueron muy útiles y se convirtieron en la columna vertebral del desarrollo. Todos estos diagramas fueron elaborados a mano y sin prestar mucha atención a la estética de los mismos tal y como lo plantea XP. La única excepción fueron los diagramas Entidad Relación los cuales se construyeron en una herramienta CASE con la cual se elaboraban mucho más fácil que a mano y se podían exportar directamente a la base de datos.

4.2. METÁFORA DEL SISTEMA

4.2.1. Lo que dice XP

- Plasmar la arquitectura del sistema en una "historia".
- Convención de nombres para los objetos del sistema

4.2.2. Experiencia en POSitron

Debido que el programa es una aplicación sencilla y de fácil comprensión tanto para los desarrolladores como para el cliente, no se requirió del empleo de una metáfora, manteniendo todos los nombres en contexto con una aplicación POS.
Para ver con detalle los estándares adoptados, remítase a anexo ESTÁNDARES.
El poseer una metáfora que incluya una convención de nombres clara facilita enormemente el logro del objetivo de la propiedad colectiva del código ya que con solo ver el nombre de un objeto o de un método se puede tener una claridad bastante amplia sobre la función que este desempeña y el lugar que ocupa dentro de todo el sistema.

4.3. TARJETAS CRC

4.3.1. Lo que dice XP

- Su principal utilidad es dejar el enfoque procedimental y entrar al modelo orientado a objetos.
- Todo el grupo participa en su elaboración.

4.3.2. Experiencia en POSitron

Una de las principales piezas de diseño empleada en el proyecto fueron las tarjetas CRC que no sólo sirvieron como columna vertebral de este, sino que también fueron la base del modelo Entidad Relación, elaborado para modelar la base de de datos. Cada Tarjeta CRC se convirtió en un objeto, sus responsabilidades en métodos públicos y sus colaboradores en llamados a otras clases.

En el proceso de elaboración de las tarjetas CRC los dos miembros del equipo estuvieron presentes manipulándolas, de modo tal que tanto el diseño fue producto de la participación de los dos desarrolladores, como el resultado del mismo fue ampliamente asimilado por ambos, favoreciendo la propiedad colectiva del código.

En XP el proceso de diseño es iterativo, por lo cual las tarjeas CRC no fueron creadas todas en la primera iteración. Al inicio de cada iteración se les fueron agregando responsabilidades, llamados, o fueron creadas otras CRC nuevas de modo tal que el diseño se convirtió en un proceso dinámico que se adaptaba a las necesidades planteadas para el momento. Sin embargo su utilidad no fue la misma durante todo el proceso de desarrollo. En las primeras iteraciones fueron supremamente útiles dando una idea clara de la arquitectura del sistema, distribución de clases, paquetes y la ubicación de las diferentes responsabilidades sobre la lógica del negocio, pero en las últimas iteraciones donde ya se tenía claridad sobre todos estos elementos, las tarjetas CRC fueron menos empleadas.

XP no propone una estrategia para afrontar la implementación de las tarjetas CRC, por lo cual se creó una con la cual se garantizó el poder correr las pruebas desde el mismo momento que inició la implementación. Primero fueron implementadas las clases más sencillas, aquellas que no hacían llamados a ninguna otra, para seguir con las que hacían llamados a las ya implementadas y así sucesivamente. Aunque XP no plantea una metodología para implementar un modelo de CRC, fue importante adoptar esta metodología debido a que era la forma más cómoda de poder aplicar las pruebas en todo momento. Cuando se empieza por codificar las clases que sólo hacen llamados a clases de JDK*, las pruebas pueden correrse desde el mismo momento que inicia el proceso de codificación, y al ir avanzando en el proceso de implementación en la forma ya indicada, las pruebas irán corriendo todo el tiempo de modo tal que se mantiene absoluto control sobre el desarrollo en lugar de tener que esperar un largo tiempo antes de ejecutar las pruebas.

4.4. SPIKE SOLUTION (SOLUCIÓN RÁPIDA)

4.4.1. Lo que dice XP

* JDK: Java Development Kit: Conjunto de clases, librerías y paquetes para desarrollar aplicaciones en JAVA

- Se trata de una prueba que se hace para resolver un problema técnico o de diseño.
- Es un programa muy simple que explora una solución potencial.

4.4.2. Experiencia en POSitron.

Se presentaron dos situaciones en las cuales se debió implementar Spike Solution. Cada uno de los cuales se le entregó a un desarrollador diferente.

Para implementar las pruebas tal como XP las recomienda se debió recurrir a una librería especialmente diseñada para tal fin, JUnit. El estudio de esta fue encargado a uno de los desarrolladores, el cual destinó aproximadamente ocho horas a su estudio, al término de las cuales en un periodo no mayor a dos horas capacitó en el empleo de la librería al otro desarrollador. Por otro lado, se requirió de una herramienta que permitiera crear reportes impresos de calidad de una forma sencilla. La mejor solución encontrada fue JasperReport, la cual fue estudiada en el transcurso de la primera iteración por un desarrollador, al término del cual se compartió la información al otro desarrollador tal como en el caso anterior, sin que retrasara sus demás responsabilidades con el proyecto.

XP recomienda asignar estos Spike Solution por parejas, sin embargo tal como se explicó en el capítulo de planeación no fue hecho de esta forma, lo que no representó un problema. El objetivo de lograr una comprensión rápida de cada uno de esto asuntos fue logrado, asegurando el cumplimiento de los plazos del proyecto.

Un aspecto importante relacionado con los Spike Solution es que el conocimiento adquirido con la elaboración de estos debe ser compartido con el resto del grupo, debido que es uno de los puntos más sensibles de convertirse en islas de conocimiento y posteriormente cuellos de botella, lo que definitivamente procura evitar XP. En este sentido se fue muy cuidadoso garantizando que ambos miembros pudieran hacer implementaciones de ambas librerías si así lo necesitaban, lo cual no fue necesario en ambos casos. Si bien ambos desarrolladores participaron activamente en el empleo de JUnit, no fue el mismo caso para JasperReport que por comodidad solo fue empleada por quién la estudio con esporádicas intervenciones del otro desarrollador.

Uno de los aportes más importantes del concepto de Spike Solution fue la posibilidad de desligarse completamente del proyecto mientras se trabaja en él, lo que facilita la concentración del esfuerzo personal en adquirir un conocimiento en lugar de disiparlo intentando solucionar un problema sin saber cómo enfrentarlo.

4.5. NO ADICIONE FUNCIONALIDAD ANTES DE TIEMPO

4.5.1. Lo que dice XP

- El adicionar funcionalidades que no se han acordado para la iteración conlleva una pérdida de tiempo que es inaceptable.

4.5.2. Experiencia en POSitron

La idea de no adicionar funcionalidad antes de tiempo es uno de los conceptos más polémicos que tiene XP, por lo tanto posee lados positivos como negativos. En el desarrollo del proyecto se fue muy estricto con el cumplimiento de esta recomendación, lo que produjo beneficios e inconvenientes a la vez.

Fue evidente el rendimiento en términos de funcionalidades implementadas versus el tiempo. Este fue logrado gracias a la claridad que se tenía en términos de los objetivos que se debían cumplir durante cada una de las iteraciones, claridad obtenida en gran medida a la idea de no adicionar funcionalidad antes de tiempo.

Es importante resaltar que en muchas ocasiones durante la realización del proyecto uno o ambos desarrolladores se vieron tentados de crear alguna funcionalidad extra que se presumía no consumiría mucho tiempo y posiblemente ofrecería una utilidad importante, intensión que fue "reprimida" al recordar la premisa de "no adicionar nada que el cliente no haya pedido explícitamente". Producto de este concepto no solo se logró optimizar al máximo el tiempo, también se vio reflejado en la experiencia que el cliente tuvo con la aplicación en cuanto tenía contacto con ella. Al no tener funcionalidades que no se había solicitado explícitamente, el programa estaba libre de botones o menús que presentaran confusión, encontrándose solamente con elementos que resultaron familiares, lo que facilitó el proceso de aprendizaje con la herramienta. Este resultado positivo no es mencionado por XP, sin embargo es importante resaltarlo.

Probablemente la idea de no adicionar funcionalidad antes de tiempo fue llevada al extremo y por tanto surgieron algunos pequeños inconvenientes. En las reuniones iniciales donde se diseñó el plan de entregas quedaron algunos aspectos claros sobre el proyecto, sin embargo al inicio de cada iteración sólo se discutió la parte del proyecto concerniente a dicha iteración por cuanto cualquier detalle de diseño o implementación que concerniera a una iteración posterior fue omitido, aún si se tenía completamente claro que tarde o temprano debía ser considerado. Por ejemplo, en la primera iteración se construyó en el modelo E-R* una tabla llamada producto de la cual se sabía que tenía llaves foráneas a dos tablas más, proveedor y departamento, las cuales no fueron implementadas en dicha iteración debido a que correspondían a la siguiente. El costo de incluirlas en la segunda iteración fue considerablemente superior al requerido si se hubieran implementado de una vez en la primera, sobre todo en lo que tiene que ver con las pruebas. Es importante aclarar que cuando se hace una modificación en el diseño de la base de datos, también hay que modificar las pruebas y el código de la aplicación, por lo que su impacto es el más alto de todos. En otros elementos como las interfaces, o la codificación misma, el impacto es menos importante.

La idea de no implementar antes de tiempo es interesante y se convierte en una herramienta muy importante para optimizar el tiempo de desarrollo, sin embargo no debe abusarse y

* Modelo Entidad Relación. Para diseño de Bases de Datos

administrarse con sentido crítico. Si se sabe que algo con toda seguridad va a requerir ser implementado, sobre todo si se trata de la base de datos, y se sabe también que el costo de implementarlo después será superior al requerido ahora, probablemente no sea tan mala idea considerar adelantarlo.

4.6. REFACTORIZACIÓN

4.6.1. Lo que dice XP

- Diseño como tarea permanente
- Se conserva el código sencillo
- Rehacer secciones de código si es necesario.

4.6.2. Experiencia en POSitron

Al transcurrir el desarrollo de la aplicación, se revisó constantemente el diseño de la misma surgiendo situaciones que no fueron tomadas en cuenta al comienzo del proyecto en el diseño general. Como salida a estos problemas se optó por la refactorización de las partes afectadas, buscando las soluciones más convenientes y sencillas, conservando la simplicidad del código. Aunque estos cambios fueron extensos, en ningún momento se convirtieron en cuellos de botella.

Una de estas situaciones se refirió a la decisión que se tomó de no crear la relación entre pago a proveedores y sesión en el momento en que se realizaban las correspondientes clases. Las medidas tomadas para subsanar este error no tomaron más de 1 hora para su solución. En una metodología pesada, este tipo de situaciones podría generar costos extras, mientras en XP no tuvo la menor trascendencia, sólo demandó la implementación de una estrategia de solución en el menor tiempo posible.

Otra situación a la que se aplicó refactoring fue la conveniencia de cambiar el tipo de datos de las llaves primarias de las tablas proveedor y departamento, casi al finalizar el proyecto. Dicha modificación afectaba aproximadamente la mitad de las clases y métodos de la aplicación, entre las cuales se encontraban las relacionadas con Producto. Aunque este inconveniente se presentó a última hora, los programadores cambiaron los tipos de datos de cadena a entero no sólo en la base de datos sino también en los métodos donde existían estas variables, en un tiempo aproximado de dos horas. Este refactoring fue logrado con éxito gracias a la estructura del programa que permitió la adición del código faltante, y al plan de organización que se ideó antes de dicha modificación.

5. CODIFICACION

En metodologías pesadas, la codificación es un proceso al cual solo se llega después de largas fases de análisis y diseño de las que queda una gran cantidad de documentación a partir de la cual el proceso de codificación es relativamente sencillo. En XP el proceso es muy diferente. Prácticamente desde un principio se inicia con la codificación, favoreciendo el logro del objetivo de estar haciendo entregas frecuentemente al cliente.

Algunos de los elementos más importantes en cuanto a la codificación son que, el cliente siempre debe estar presente en esta, se debe trabajar en parejas y debe haber una propiedad colectiva del código. Todos estos elementos representan paradigmas nuevos en lo que a la ingeniería de software se refiere, planteando entornos de discusión sobre la conveniencia de adoptarlas.

A continuación encontrará para cada uno de los aspectos que conforman la etapa de codificación una comparación entre las ideas teóricas de la codificación y lo aplicado en la ejecución del ejercicio práctico.

5.1. CLIENTE SIEMPRE PRESENTE

5.1.1. Lo que dice XP

- El cliente debe estar disponible en el sitio de trabajo
- El cliente es fundamental para solucionar dudas cara a cara

5.1.2. Experiencia en POSitron

La idea de tener al cliente, un representante de éste o a un usuario no es fácil de asimilar si se consideran los costos que esto representa. En el caso de este proyecto, el cliente no podía desplazarse a ninguno de los lugares de trabajo de los desarrolladores dado que debía estar al frente de su negocio. Pretender tener al otro usuario implicaría pagarle el tiempo que estuviera acompañando al grupo, gasto que ni el grupo de desarrollo podía asumir ni el cliente le interesaba pagar. Por tal motivo se debió implementar una estrategia de comunicación distinta en la cual los programadores podían llamar vía telefónica al cliente

en el momento que requirieran solucionar cualquier duda en el proceso de implementación. Si bien esta estrategia no fue igual de efectiva que haber tenido al usuario acompañando el desarrollo, fue suficiente para lograr una buena comunicación con el cliente.

Es importante tener en cuenta que contar con un representante del cliente en las instalaciones del equipo de desarrollo demandará una inversión económica representada en la remuneración económica para el representante. En muchos proyectos susceptibles de ser desarrollados por medio de XP, este gasto se vuelve inaceptable por elevar de forma importante el costo del proyecto. Sin embargo se puede plantear una solución intermedia donde el representante del cliente solo esté presente durante un periodo de tiempo acordado con el grupo de desarrollo por día, que deberá ser aprovechado para disipar cualquier duda que haya surgido durante el resto del tiempo de desarrollo en que no estuvo. Esta es una modificación importante a XP, ya que el "Cliente In Situ" es uno de los elementos más relevantes de esta metodología y la hace mucho más aplicable en la práctica.

5.2. EL CÓDIGO SE ESCRIBE SIGUIENDO ESTÁNDARES

5.2.1. Lo que dice XP

- Al escribir el código del programa se deben seguir estándares de programación.

5.2.2. Experiencia en POSitron

La estandarización del código fue asumida desde el mismo momento en que se inició la codificación. Debido que el grupo de desarrollo había estado trabajando unido por largo tiempo, ya tenían un esquema de estándares acordados de forma tácita, sin embargo por seguir una disciplina se formalizaron estos en un documento.*

Se debe resaltar que el programar empleando estándares es una práctica que no solo se recomienda en XP, es una buena práctica que debe ser seguida en cualquier metodología de desarrollo lo que no implica algo muy novedoso en XP.

Tal como se ha presentado en varias oportunidades en el transcurso de este documento, el programar siguiendo estándares no es un fin en sí mismo. Se trata de un medio con el cual se pretende facilitar la propiedad colectiva del código. En el caso de este proyecto se aplicaron todos los estándares con gran éxito debido a dos motivos. En primer lugar, la

* Ver anexo ESTÁNDARES

herramienta empleada para desarrollar facilitaba el aplicar dichos estándares y en segundo, que los mismos venían siendo empleados desde antes por el equipo de desarrollo.

5.3. CODIFICAR PRIMERO LA PRUEBA

5.3.1. Lo que dice XP

- Escribir primero la prueba que el código.
- El tiempo de escribir una prueba y luego el código del programa para dicha prueba es menor que solo escribir el código.
- Escribir pruebas primero permite identificar los casos especiales que deberá pasar el código.

5.3.2. Experiencia en POSitron

No es fácil seguir este planteamiento de XP por varios motivos. En primer lugar, el framework escogido para implementar las pruebas si bien es el más adecuado y recomendado por Kent Beck para XP, tiene algunas limitaciones, especialmente cuando se trata de hacer pruebas a elementos gráficos tales como Ventanas y Botones. En tal sentido se optó por no hacer pruebas empleando JUnit a estos objetos.

Una de las características de JUnit es que las pruebas corran en forma autónoma, lo cual obviamente es positivo pero desde el punto de vista del diseño de las pruebas propone algunos retos, como por ejemplo garantizar que después de cualquier manipulación de la base de datos, esta quede en mismo estado en que estaba antes de la prueba. Sin embargo, la idea de ser autónomas se convierte en un obstáculo importante en algunos tipos de pruebas. ¿Cómo probar por ejemplo que un reporte en papel quede bien hecho sin la intervención del tester? O cómo saber si el programa obtuvo la hora del sistema de forma correcta si no es comparándola con la propia hora del sistema?

El carácter privado de muchos métodos representa un obstáculo insalvable para hacerle pruebas, ya que como solo puede ser accedido desde el interior de la clase no puede ser probado independientemente. La solución que se tomó en este sentido fue probarlo a través del método público que hace uso de él. Si el método público pasa determinada prueba, se asume que el método privado también la ha pasado.

Todos los elementos anteriores representaron obstáculos en el desarrollo de las pruebas, y plantearon una inquietud importante sobre el alcance del concepto “codificar primero las

pruebas". ¿Se trata de codificar SIEMPRE una prueba antes que el código, o solo aquellas clases encargadas de realizar la lógica del negocio? Debido que XP no tiene una respuesta clara a esta inquietud el grupo de desarrollo optó por probar solo aquellas clases que ejecutan la lógica del negocio, que en definitiva son las más importantes y de las cuales se debe tener garantía de estar muy bien construidas.

Es importante resaltar las ventajas que representa hacer las pruebas antes que el código de la aplicación. En primer lugar el tiempo que toma escribir determinado código después de haber implementado la prueba es considerablemente menor que si no se hubiera escrito la prueba antes. En segundo lugar al hacer la prueba se identifican de manera precisa cuales son los casos especiales y rutas alternas que deben ser consideradas dentro del código haciendo de este un producto más robusto y tolerante a fallos. Finalmente una ventaja no expuesta de forma vehemente por XP es la estética del código de la aplicación. Al tener tal claridad sobre cómo debe ser escrito determinado código, la tarea de realizarlo es más sencilla y el mismo queda organizado de forma más estética.

5.4. TODA LA PRODUCCIÓN DE CÓDIGO DEBE SER HECHA EN PAREJAS

5.4.1. Lo que dice XP

- Toda la producción de código debe ser hecha en parejas sentadas frente a un único computador.
- Al trabajar en parejas se tiene un diseño de mejor calidad y un código más organizado.
- Al trabajar en parejas se solucionan los problemas más fácilmente.

5.4.2. Experiencia en POSitron

El no contar con una sede permanente complicó seriamente el cumplimiento del objetivo de programar en parejas. Por otro lado al solo haber una pareja de programadores se hacía completamente imposible cumplir con el objetivo de tener varias parejas de programadores trabajando uniformemente. Sin embargo en la primera iteración se procuró trabajar en pareja tal como lo plantea XP en un solo computador, ensayo que fracasó rotundamente. En lugar de ser un apoyo, ambos programadores se estorbaron mutuamente disminuyendo el rendimiento por debajo del cincuenta por ciento. Al hacer un balance del rendimiento de la primera semana bajo esta metodología se decidió trabajar la segunda semana de la primera iteración cada uno en su propio computador con la salvedad de mantener el mayor nivel de comunicación posible. En tal sentido se tomó como estándar que ambos programadores si

bien estarían trabajando en lugares geográficos diferentes, siempre trabajarían en el mismo horario y conectados a internet con un servicio de mensajería abierto*. De esta forma se logró generar la sensación de cercanía y siendo realmente un apoyo muy importante el uno para el otro.

En XP se tiene como salvedad para trabajar en parejas que uno o ambos de los programadores sean expertos en la herramienta que se está empleando. En el caso de ambos desarrolladores, tenían conocimientos elevados sobre todas las herramientas que se emplearon, por lo cual el nivel de autonomía fue superior.

En lo relacionado a la mejoría en el diseño al trabajar en parejas, se observó que el primero no es precisamente una consecuencia directa de lo segundo. La explicación de tal fenómeno se debe a que al chocar los puntos de vista sobre un determinado problema de programación, el resultado no es una versión complementaria de ambos panoramas si no una mezcla heterogénea de los dos diseños que dista de ser "elegante".

Es importante resaltar que el concepto de programación en parejas ni es la gran panacea, ni debe ser descartado de plano. Es muy probable que dos programadores que no dominan la herramienta en la cual estén desarrollando sean mucho más productivos trabajando bajo un mismo computador que estando solos. También es de resaltar la empatía que se requiere en la pareja para que el proceso de resultados exitosos. Se requiere que ambos programadores tengan concepciones similares en términos de cómo enfrentar un problema de programación para que sean productivos.

En la última iteración, que coincidió con Semana Santa, se decidió trabajar de tiempo completo en el proyecto, empleando la casa de uno de los programadores como sede para trabajar en un mismo sitio geográfico. El resultado fue un incremento del rendimiento de ambos programadores. Si bien no estaban trabajando en un mismo computador, el estar uno junto a otro les permitió ser de más apoyo que si lo hicieran en sitios geográficos diferentes. Aunque el rendimiento empleando esta metodología no fue muy superior al descrito un par de párrafos atrás, fue lo suficientemente importante como para recomendarlo al emplear XP. Según lo anterior, es de resaltar que aunque hay una mejora en el rendimiento si se trabaja en un mismo sitio geográfico y debe procurarse que así sea, se puede distribuir el equipo de trabajo en diferentes lugares con la condición que se compense esto con

* Windows Messenger y/o GoogleTalk.

herramientas y tecnologías que permitan una comunicación PERMANENTE a muy bajos costos.

5.5. SOLO UNA PAREJA HACE INTEGRACIÓN A LA VEZ (INTEGRACIÓN SECUENCIAL).

5.5.1. Lo que dice XP

- Antes de integrar nuevo código a un proyecto se debe garantizar que la última versión halla pasado todas las pruebas.
- Solo se debe hacer una integración a la vez
- Se debe tener claro cuál es la última versión funcional.

5.5.2. Experiencia en POSitron

A diferencia de otras metodologías donde se definía propiedad sobre algunas clases, en esta ocasión se le dio la propiedad de todo el proyecto a alguno de los dos programadores, alternando ésta posesión durante todo el proceso de desarrollo según como la necesidad de implementación lo iba requiriendo.

Durante todo el tiempo se tenía completa claridad de quién tenía la última versión de modo que la otra persona pasaba permanentemente el código a la poseedora de la mencionada última versión, haciéndola responsable de hacer la integración y garantizar que la nueva última versión no tuviera errores. Esto no va en contra de la propiedad colectiva del código debido a que cualquier desarrollador podía modificar cualquier clase, siempre y cuando coordinara esto con el dueño del proyecto para tenerla en cuenta en la siguiente integración.

Esta metodología resultó no solo efectiva si no también flexible. Se garantizó que solo una persona hiciera integración a la vez, que siempre se supiera no solo cuál era sino también donde estaba ubicada esta y que se pudiera traspasar la responsabilidad de integrar según como fuera necesario.

Se siguió la directiva de XP en el sentido de no dar posesión de clases o elementos de código a nadie, que solo se hiciera una integración a la vez y siempre se tuviera claro cuál era la última versión. Al cumplir a cabalidad con estas tres instrucciones se evitaron muchos problemas haciendo que el proceso de integración fuera fluido y sin inconvenientes.

5.6. INTEGRACIONES FRECUENTES

5.6.1. Lo que dice XP

- Se deben hacer integraciones cada pocas horas o en lo posible no tardar más de un día entre una y otra integración.
- Entre más se tarde en encontrar un problema, resultará más costoso resolverlo.
- Integrar frecuentemente evita problemas como el trabajar sobre una clase obsoleta.

5.6.2. Experiencia en POSitron

Generalmente se hacían una o dos integraciones diarias y en el peor de los casos no transcurrían más de dos días de programación sin que se realizara una integración, garantizando de esta forma que en todo momento se estuviera trabajando sobre la última versión del proyecto.

Al no emplear software de versionado por no contar con un servidor conectado a Internet permanentemente se recurrió a un estándar para conocer cuál era la última versión. Cada versión funcional se comprimía en un archivo .zip y era ubicado en una carpeta conocida por ambos desarrolladores en ambos computadores de desarrollo. Este archivo tenía la siguiente estructura de nombres: Tesis<Día><Mes><Hora>. De esta forma se garantizaba que cualquier desarrollador en forma autónoma encontrara la última versión del proyecto.

La persona encargada de la integración no era siempre quién había construido la clase que iba a ser añadida o reemplazada. Esta responsabilidad era de quién en el momento fuera el dueño del proyecto tal como se planteó en el apartado de integración secuencial. Las tareas que tenía dicha persona eran la de integrar el código y las pruebas, realizar y supervisar dichas pruebas y garantizar la funcionalidad del programa antes de liberarlo. Al momento de realizar una integración se le enviaba el resultado de esta al otro programador para que de ese momento en adelante trabajara sobre ella.

Al realizar en forma cuidadosa este procedimiento, permanentemente se evitaron problemas relacionados con versiones obsoletas de elementos del sistema y se permitió encontrar problemas que de no haberse hecho integraciones frecuentes habían sido mucho más costosos.

Durante buena parte del proyecto se presentaron problemas de conexión con la base de datos al ejecutar las pruebas. Al detectarse estos problemas se verificó que sólo se presentaran en las pruebas y no en la ejecución de la aplicación, sin embargo en algunas ocasiones y debido a estos problemas las pruebas no pasaron, sin que esto indicara que habían errores en el código de la aplicación. Por tal motivo se debió ser mucho más cuidadoso al ejecutar las pruebas antes de realizar la integración, prestando especial atención a aquellas que fallaban, en relación a si dicho fallo era por errores de conexión con la base de datos o por fallas en la codificación de la aplicación.

5.7. Propiedad Colectiva del Código

5.7.1. Lo que dice XP

- Se debe procurar rotar a los programadores no solo de compañero, también de partes del proyecto a desarrollar.
- Cualquier programador debería poder continuar la codificación que alguien más empezó sin muchas dificultades.

5.7.2. Experiencia en POSitron

Sobre la propiedad colectiva del código se ha discutido bastante durante el documento, pero solo en este momento se afronta el tema de forma directa. XP propone muchas estrategias destinadas a facilitar la propiedad colectiva del código debido a la dificultad de lograrse en proyectos pequeños, y se complica aún más en la medida que el proyecto crece.

Estrategias como la rotación del personal, el empleo de estándares y la programación en parejas van destinadas a la consecución de la propiedad colectiva del código, de modo tal que solo se logrará este objetivo en la medida que las estrategias planteadas sean ejecutadas cuidadosamente.

Analizando el proyecto del cual se presenta este documento, el rotar a los programadores por diferentes partes de la aplicación no fue fácil de lograr, principalmente porque no se pudo desarrollar la estrategia de la programación en parejas, sin embargo gracias a la aplicación de las estrategias ya discutidas anteriormente fue posible rotar a los

programadores en diferentes partes del proyecto según como las necesidades de este así lo requirieran.

En la medida que transcurrían las iteraciones se requería dedicar más horas de trabajo en determinadas partes de la aplicación, pudiéndose enfocar el trabajo de alguno de los programadores en partes del programa que habitualmente no trabajaba. Esta flexibilidad en la distribución de tareas fue fundamental para evitar la sobrecarga de trabajo en uno de los programadores agilizando aún más el desarrollo. De no ser porque se había obtenido muy buenos resultados en la propiedad colectiva del código no habría sido posible esto. Sin embargo sería demasiado optimista pretender lograr la propiedad colectiva del código al cien por ciento, además de un desgaste excesivo para el equipo de desarrollo. Debería procurarse lograr esta meta entre un setenta y noventa por ciento para que no se convierta en un lastre para el desarrollo.

El objetivo que cualquier programador pudiera continuar el desarrollo de que alguien más, se logró, sin embargo se prefería que quién había creado una clase continuara trabajando en ella, debido a que si bien el otro programador entendía el código y podría haber continuado su desarrollo no tendría el mismo rendimiento que el creador. El motivo es muy simple, por más que se sigan estándares y se haga un diseño muy cuidadoso, la labor de programar es creativa y cada programador enfrenta un mismo problema de diferentes formas para facilitarse el trabajo, lo que probablemente podría confundir a otro.

Una disciplina que se adoptó para lograr la propiedad colectiva del código fue notificar al otro desarrollador cuando se había modificado una clase y enviar dichos cambios. La importancia de esta medida era evitar que un desarrollador creyera conocer un código cuándo la última versión de este difería de la que conocía.

Probablemente uno de los grandes aportes de XP a la Ingeniería de Software es la propiedad colectiva del código, pero no como concepto, el cual es deseable desde el inicio de programación. El aporte de XP radica en los métodos que se deben emplear para alcanzarlo con algún éxito. Sin embargo es importante aclarar que solo es posible lograr la propiedad colectiva del código en proyectos pequeños. En la medida que estos crecen, el intentar alcanzarla deja de ser productivo para el proyecto y empieza a convertirse un lastre para el equipo de desarrollo.

5.8. No trabajar horas Extras

5.8.1. Lo que dice XP

- No trabajar horas extras.
- El dedicar horas extras a un proyecto retrasado, no lo va a poner al día
- Es preferible replantear los plazos a trabajar horas extras.

5.8.2. Experiencia en POSitron

Debido a que los desarrolladores no se dedicaron exclusivamente al proyecto, el concepto de horas extras se vuelve más subjetivo. Desde un contexto laboral, horas extras se entiende como el tiempo trabajado después de ocho horas diarias, concepto que no es aplicable al proyecto en cuestión.

Para cualquier persona que haya programado es claro que se trata de una tarea de importante esfuerzo mental y como tal requiere no ser desempeñada por muchas horas consecutivas. En tal sentido, el plantearse trabajar horas extras después de una jornada completa de desarrollo sugiere más una pérdida de tiempo que una recuperación de los atrasos del proyecto. En el caso de este proyecto no se trabajaron horas extras debido a que se tuvo la precaución de plantear plazos amplios en las entregas con el fin de considerar cualquier posible inconveniente que surgiera en la implementación. Solo hubo una circunstancia especial que entra a colación. En la tercera iteración surgieron más refactoring que lo previsto inicialmente representando un pequeño inconveniente en la planificación del proyecto*. Lo que se decidió fue aumentar en un par de horas el tiempo dedicado al proyecto, pero no se trató de un aumento de horas por día, solo se programó trabajar un día más. Se hace complejo determinar si fueron horas extras ya que normalmente en ese día se trabajaba, solo que en otra actividad relacionada con las obligaciones académicas. Podría afirmarse más bien que se reprogramaron las actividades del grupo para subsanar el inconveniente presentado por el refactoring.

Uno de los elementos que considera XP para hacer la mayoría de sus planteamientos es el hecho que la labor de programación es muy difícil de estimar y proyectar en el tiempo. La experiencia tanto en este proyecto como en otros anteriores es que el rendimiento en la labor de programación varía mucho según un sinnúmero de circunstancias que afectan no sólo al proyecto en sí mismo sino también a quién lo esté implementando. En tal sentido, se

* Ver Velocidad del proyecto.

debería poder tanto flexibilizar en lo posible los plazos como diseñarlos con suficiente holgura para contemplar estas posibles demoras. Si bien XP plantea que debería ser posible replantear algunos plazos, esta posibilidad debería quedar clara desde el principio con el cliente para evitar malos entendidos.

6. PRUEBAS

XP enfatiza en la realización de un sin número de pruebas a lo largo del proyecto, con el fin de asegurar en todo momento la realización de lo planteado en el diseño. En este proceso no sólo participa el equipo de desarrollo, también es importante los aportes del cliente, sobre todo en las pruebas de aceptación.

Cabe señalar que el diseño de pruebas se realiza para todas las partes del sistema como una práctica para garantizar el buen funcionamiento, independiente de la decisión que se tome sobre implementarlas en un framework como JUnit.

Este capítulo se divide en partes que corresponden a las pruebas unitarias, de aceptación y qué hacer cuando se encuentra un error.

6.1. Pruebas unitarias

6.1.1. Lo que dice XP

- Las pruebas deben ser escritas antes que los métodos.
- Su implementación y ejecución deben consumir el menor tiempo posible.

6.1.2. Experiencia en POSitron

La creación de pruebas fue una experiencia nueva para el equipo de trabajo al ser una de las reglas de la metodología XP que no se había llegado a utilizar en proyectos anteriores. Debido a esto, la realización de pruebas al principio del proyecto fue traumática y demandó más tiempo de lo planeado, lo cual no fue conveniente ya que la metodología intenta disminuir los cuellos de botella, no aumentarlos.

El carácter obligatorio de la escritura de las pruebas antes del desarrollo de los métodos del sistema implica un proceso de diseño previo. Esto se considera una ventaja ya que se destina tiempo en la construcción de la prueba, pero al realizar la codificación del método, éste resultaba de manera casi inmediata. También se destaca la autonomía que deben tener dichas pruebas a la hora de su ejecución, lo que implicaba la manipulación de la base de datos y la recuperación de su estado inicial al finalizar la prueba.

Según XP, se deben crear todas las pruebas de una clase antes de comenzar a desarrollar los métodos. En la experiencia fue conveniente realizar las pruebas individualmente, debido a que se producían errores al ejecutar todas las pruebas en un solo llamado. Se descubrió que este inconveniente estaba relacionado con la base de datos y no con los métodos, lo que al principio del proyecto aplicó dificultad al evaluar si un método había pasado o no la prueba. Una vez que se descubrió este error se empezó a ejecutar las pruebas por grupos en lugar de ejecutarlas todas de una vez. De esta forma no había problemas de comunicación con la BD y se garantizaba que si una prueba fallaba era solo por errores de lógica.

XP propone la utilización de una herramienta que realice de forma automática la implementación y ejecución de las pruebas. En este caso se recurrió a JUnit, una herramienta de Java especializada para la creación de unidades de prueba, que resultó ser una elección apropiada a la hora de probar las clases y métodos que se referían a la lógica del negocio. Los elementos gráficos y de impresión no fueron probados mediante el JUnit, en su reemplazo para verificar resultados se realizó la observación directa de los mismos como es el caso de los reportes. Tampoco fueron implementadas las pruebas de los métodos privados.

6.2. Pruebas de aceptación

6.2.1. Lo que dice XP.

- Se deben diseñar las pruebas de aceptación con base en las historias de usuario

6.2.2. Experiencia en POSitron

Tres elementos permitieron al grupo de desarrollo diseñar las pruebas de aceptación. En primer lugar el tipo de sistema implementado era suficientemente sencillo y conocido por todos los miembros del equipo de desarrollo, principalmente porque uno de ellos laboró como empleado para el cliente. En segundo lugar las reuniones de las cuales se obtuvieron las historias de usuario fueron grabadas en audio y video con lo cual fue posible la reconstrucción del las pruebas de aceptación por parte del equipo de desarrollo sin toda la intervención del cliente. En tercer lugar el cliente aceptó el delegar esta función de diseño de las pruebas debido que su disponibilidad de tiempo, como ya es mencionada en otros apartados del documento, se lo impidió.

6.3. Cuando se encuentra un error

6.3.1. Lo que dice XP

- Al encontrar un error debe escribirse primero la prueba antes que corregirlo.

6.3.2. Experiencia en POSitron

Como se mencionó antes, no se crearon unidades de prueba para los errores de las interfaces gráficas, estos tipos de inconvenientes fueron solucionados mediante pruebas manuales, es decir, sin la ayuda de la herramienta JUnit debido que el API no soporta este tipo de pruebas. Estas pruebas manuales consistían en cajas negras donde se verificaba la solución del problema mediante la ejecución del programa.

Cuando el código de una de las clases principales pasaba la prueba que se había diseñado con anterioridad, pero los programadores encontraban que la funcionalidad no era correcta, se modificaba la prueba y se corregía el código.

La aplicación de esta idea en XP es relativa a dónde se halla encontrado el error. Evidentemente si el error fue encontrado a partir de la ejecución de una prueba unitaria no se hace necesario el escribir una nueva prueba ya que existe una para detectar el error (el caso más común en la realización del proyecto). Por otro lado, si el error se detectó por una inspección manual, efectivamente se debía elaborar una nueva prueba que detectara este error o revisar alguna que ya existiera, relacionada con este elemento de la aplicación, que detectara tal error.

7. COMENTARIOS SOBRE LA EXPERIENCIA

En este capítulo se hacen observaciones sobre algunos aspectos que desde el punto de vista de los autores son relevantes para su consideración. Algunos de estos comentarios son recomendaciones que hace XP, pero en la práctica no fueron tan adecuados. Otros de estos comentarios fueron beneficios adicionales que obtuvieron por aplicar dichas recomendaciones y también se resaltan algunas modificaciones que se asumieron por las características del proyecto. En resumen, son comentarios que no se consideraron como conclusiones, sin embargo su nivel de relevancia les merece un capítulo exclusivo.

Para dar una estructura conceptual al capítulo, se agruparon los comentarios en seis partes correspondientes a las cuatro fases del proceso de desarrollo en XP y dos más correspondientes a la propiedad colectiva del código y a los usuarios.

7.1. PLANEACIÓN

Los comentarios de este apartado tienen que ver con la forma como se elaboran las historias de usuario y con la medida de tiempo para la realización de las tareas.

7.1.1. El cliente no puede redactar las historias de usuario solo.

Si bien XP plantea que las historias de usuario deben ser redactadas por el cliente, en la experiencia se nota que a éste se le dificulta hacer este proceso por sí mismo. Por lo tanto se hace evidente la necesitad de que sean escritas en conjunto con el equipo de desarrollo o que se le dé un proceso de capacitación en dicha labor. En primer lugar el usuario no sabe qué es una "historia de usuario". En segundo lugar, aún explicándole a qué se refiere, para el cliente no es fácil exteriorizar sus necesidades y le es aún más difícil redactarlas, aunque se le permita no utilizar términos técnicos. En tal sentido es mucho más adecuado que las historias de usuario sean el resultado de un diálogo entre el usuario o cliente y los desarrolladores, porque este diálogo canaliza las ideas del cliente en un documento con términos comprensibles tanto para él como para los desarrolladores.

7.1.2. Horas ideales en lugar de días ideales

El término de días de programación es muy amplio para considerarlo como una medida de tiempo porque existen muchos factores que lo afectan. El considerar un día ideal como una jornada de ocho horas sin interrupciones presenta muchas omisiones tales como distracciones, interrupciones y actividades externas. El plantear los estimativos en términos de horas permite considerar con mayor precisión tiempos consumidos como las posibles interrupciones que afecten al proyecto. En tal sentido se hace más interesante y de mayor exactitud plantear los tiempos en las historias de usuario en términos de horas ideales de programación.

7.2. DISEÑO

En este apartado se discute puntos de vista opuestos sobre la adición de funcionalidades antes de tiempo, identificando los casos en los cuales fue conveniente y aquellos en los que no lo fue.

7.2.1. No adicionar funcionalidad antes de tiempo para facilitar la capacitación del usuario.

Cuando se adicionan funcionalidades antes de tiempo o cuando se incorporan elementos no solicitados por el cliente, la interfaz del usuario estará cargada de botones, etiquetas y otros elementos que se convertirán en distractores para el usuario, dificultando el proceso de aprendizaje del mismo, sin mencionar que probablemente no haga uso de ellos en el futuro.

XP afirma que no se debe adicionar funcionalidad antes de tiempo, pero no manifiesta lo anterior como un motivo para hacerlo, siendo una razón importante para justificar esta práctica.

7.2.2. Agregar funcionalidad antes de tiempo.

Sin importar qué metodología de desarrollo se esté empleando, la base de datos es un elemento esencial y por consiguiente su modelo. Cualquier cambio que se realice en éste durante el proyecto tiene un impacto superior que si fuera solamente en el código. Al alterar un modelo de base de datos, se tendrá que modificar tanto la propia base de datos como los métodos relacionados a esos en la aplicación, lo que conlleva mayor consumo de tiempo.
Existen algunos elementos cuya presencia en el proyecto se presenta como obvia y por tal motivo deberán ser considerados en el modelo de datos en algún momento. Cabe aclarar que con lo mencionado en el párrafo anterior, el impacto de incluirlos ya avanzado el proyecto es alto. Por tal motivo se hace conveniente su adición al modelo de datos desde un principio, aunque no sean utilizados en el código hasta fases avanzadas del proyecto. Si

bien, este juicio sólo se presenta para el modelo de datos, pueden existir algunos otros elementos ajenos a éste que requieran la misma consideración.

Esta conclusión se contradice directamente con lo manifestado por XP, la cual sugiere no adicionar funcionalidad antes de tiempo, y solo implementar estrictamente aquello planteado para determinada iteración, con lo cual se sugiere aceptar este planteamiento de XP, con algunas excepciones consideradas por el equipo en las primeras fases de diseño.

7.3. CODIFICACIÓN

XP plantea una metodología de trabajo muy diferente al usado en las metodologías tradicionales presentando puntos positivos y negativos. En este orden de ideas se discuten algunos de ellos resaltando la forma como se experimentaron en la ejecución del proyecto.

7.3.1. Costo del Cliente in Situ

Aunque la presencia de un representante del cliente es lo más recomendable para el proyecto, su viabilidad en términos financieros es compleja. En las instalaciones del equipo de desarrollo podrá estar tanto el cliente como un usuario representante de éste. En ambos casos este tiempo representa dinero. Costo que deberá ser asumido por el cliente o por el equipo de desarrollo, lo que en últimas es un costo adicional al proyecto que se considera elevado.

Estos costos tienen un impacto importante para el proyecto, principalmente porque el tipo de proyectos para los que está recomendado XP son los pequeños y medianos donde el dinero se convierte en una limitante.

7.3.2. Programación en parejas

La programación en parejas es un concepto más complejo que lo planteado por XP. Antes de tomar la decisión de implementarla o rechazarla de plano hay que analizar a profundidad el nivel de conocimientos que tengan en la herramienta, compatibilidad personal y laboral y enfoques de programación que empleen. Además se debe definir una metodología clara de cómo enfrentar los problemas de programación. De lo contrario, más que una ayuda, sería un obstáculo el uno para el otro.

Cabe recordar que XP si plantea una metodología para desarrollar la programación en parejas, pero ésta no es lo suficientemente completa para considerar todos los aspectos, tanto técnicos como humanos, que implica a dos personas que trabajan en un mismo monitor y teclado.

7.3.3. Un solo sitio geográfico

Si bien existen herramientas como el Internet que facilitan la comunicación en aquellos casos en que no es posible la interacción cara a cara, resulta más adecuado lograr que todas las partes involucradas en el proyecto se encuentren ubicadas en el mismo sitio geográfico. El aplicar esta práctica hace que el proyecto tenga mejor rendimiento, pero su omisión no invalida a XP, siempre y cuando se encuentren otras formas que permitan altos grados de comunicación.

7.3.4. Trabajar horas extras

El proceso de programación requiere de un esfuerzo mental alto, por lo cual el desgaste que produce una larga jornada de trabajo disminuye el rendimiento de los desarrolladores al punto en que los avances logrados en horas extras se consideran irrelevantes en comparación con el tiempo que toma lograrlos.

Cuando los proyectos se retrasan, se comete el error de alargar las jornadas de trabajo con el fin de recuperar tiempo, pero al contrario en lugar de lograr el objetivo, se aumenta el nivel de desgaste y estrés de los programadores convirtiéndose en una bola de nieve que en vez de solucionar el problema, lo empeora.

Definitivamente no es recomendable replantear los plazos de entrega con el cliente cuando se presenta una situación de retraso, sin embargo para este tipo de proyectos puede convertirse en la opción más adecuada. Para evitar el riesgo de retrasos en el proyecto es mejor plantear cronogramas más flexibles que consideren los posibles inconvenientes que se presenten en el transcurso del proyecto.

7.4. PRUEBAS

Las pruebas son uno de los elementos más importantes de XP y en tal sentido surgieron algunas consideraciones relacionadas con sus características, las cuales se describen a continuación.

7.4.1. Pruebas autónomas.

Uno de los requisitos de XP para las pruebas unitarias se refiere a que éstas deben correr en forma autónoma, es decir que no requieran la intervención humana para determinar si han sido exitosas. Esta condición es difícil de cumplir en muchas ocasiones, sobre todo si se emplea una librería de pruebas.

Existen una variedad de rutinas que por su forma de funcionar se hace complejo hacerle pruebas. Entre ellas cabe resaltar la obtención de la hora del sistema. En este caso hacer una prueba que evalúe su éxito requerirá bien sea la participación humana para determinar que la hora sea correcta o la ejecución de la misma rutina dentro del código de la prueba, lo que no tiene sentido.

Elementos visuales como ventanas, botones y reportes representan un reto adicional para ser probados de forma autónoma, ya que la intervención humana se hace indispensable para determinar el éxito de la prueba. En estos casos es mucho más práctico hacer las pruebas manualmente.

Finalmente los errores más difíciles de encontrar y de mayor gravedad son aquellos relacionados con la lógica de negocio. Para este tipo de errores resulta especialmente práctico el uso de una librería que permita las pruebas autónomas, por lo que resulta mucho más conveniente dedicar los esfuerzos a probar este tipo de errores empleando una librería de pruebas.

7.4.2. Prueba antes que código.

Realizar la prueba antes que el código hace que éste sea más fácil de construir, se identifiquen con mayor claridad los casos especiales y quede "más elegante".

Al construir una prueba queda más clara la funcionalidad que debe implementarse por lo cual la construcción de la rutina requiera de menos tiempo. Por otro lado y producto de dicha claridad se evidencian los casos especiales que debe considerar dicha rutina lo cual la hace más completa y tolerante a fallos, requisito indispensable de una rutina bien hecha.

Finalmente, gracias a lo anterior, el resultado del proceso de codificación es más organizado, haciéndolo fácil de entender y refactorizar.

7.4.3. Probar gradualmente.

Teniendo en cuenta lo anteriormente dicho, las pruebas deben ser elaboradas antes que el código, pero debe considerarse la conveniencia de la realización de la totalidad de las pruebas de una clase o la realización de cada una de ellas con su respectiva comprobación.

Esta última posición presenta la ventaja de una constante corroboración del código de modo tal que se puede trabajar con la seguridad de que se está implementando sobre un código que funciona.

7.4.4. Probar al encontrar un error.

Aunque XP plantea la realización de pruebas en la medida en que surgen errores, poner en marcha esta práctica consume demasiado tiempo. Resulta más conveniente la revisión de las pruebas existentes para determinar a cuál de éstas le incumbe el error que se presenta.

En caso de no encontrar dicha prueba, se recurrirá a la realización de una prueba especial para el error.

De esta forma se evita la aparición de una gran cantidad de pruebas aisladas, preservando la organización de las mismas.

7.5. PROPIEDAD COLECTIVA DE CÓDIGO

Es muy importante que esta práctica se aplique de forma eficiente en todos los proyectos que son desarrollados empleando XP. En este apartado se exponen tanto las ventajas de lograr esta práctica como los costos que se deben asumir para obtenerla.

7.5.1. La propiedad colectiva a medida que aumenta el proyecto.

En la medida en que un proyecto es más grande se hace mayor el número de desarrolladores y la cantidad de código para distribuir entre los mismos, por cuanto lograr la propiedad colectiva de código se vuelve una tarea más compleja. Existe un punto en el cual el costo que implica poner en marcha dicha práctica supera los beneficios que por éste se obtienen, ralentizando los procesos del proyecto.

Cuando un proyecto aumenta su tamaño debe buscarse un punto en el cual no se pretenda lograr la propiedad colectiva de todo el código, sino dividir el proyecto en partes de tal forma que dentro de cada una de ellas se busque la propiedad colectiva del código.

7.5.2. La propiedad colectiva del código inicia en el diseño.

El código es la consecuencia de un proceso de diseño por cuanto la propiedad colectiva debe iniciar desde esta etapa. Es importante tener clara ésta idea, ya que si todos los integrantes del equipo no participan en el diseño no sólo se perderán estos valiosos aportes sino que desde ese momento se perderá la propiedad colectiva del código.

7.5.3. Medios para conseguir la propiedad colectiva.

Varias de las recomendaciones de XP tales como la rotación de personal o la programación en parejas no son objetivos que se deban cumplir dentro de un proyecto, ya que por sí solas no brindan ningún beneficio. Estas prácticas son el medio por el cual se logra la propiedad colectiva, que sí es un objetivo en XP.

7.5.4. Propiedad colectiva del código no es anarquía.

La propiedad colectiva no significa anarquía en la codificación. El hecho de que cualquier persona pueda manipular el contenido del proyecto en determinadas circunstancias no

quiere decir que lo pueda hacer siempre que él quiera. En caso de hacerse de esta forma, no existiría ningún control de cuál es la última versión de determinada sección de código generando un desorden difícil de controlar.

Es importante que en todo momento exista un responsable encargado de determinar quién va a manipular determinada parte del código. Así se evita perder el control del proyecto. Tan relevante como lo anterior es que esta responsabilidad se rote dentro de los miembros del equipo, ya que esta persona es la que en dicho momento mayor conocimiento tiene de la estructura del sistema y en caso de no rotarse esta responsabilidad se pierde la propiedad colectiva del código y tener un único punto de falla, que debe ser evitado en XP

7.6. USUARIO

A diferencia de otras metodologías de desarrollo, el usuario es el punto central de XP. A continuación se hacen algunas observaciones sobre la influencia del cliente en el éxito del proyecto.

7.6.1. No solo es importante que el cliente sepa lo que quiere.

Si bien, es importante que el usuario tenga clara la necesidad que va a ser implementada, también lo es que haya tenido interacción con sistemas informáticos. Entre menor experiencia posea un usuario en este tipo de tecnologías mayor será la dificultad para plantear sus necesidades porque no sabrá que esperar del sistema por construir.

7.6.2. El cliente como único punto de falla.

XP ubica al cliente como el eje central para las actividades del proyecto. Esta posición se plantea como una ventaja, pero presenta un riesgo. Al ser el punto central también es un único punto de falla. En la media en que el cliente, por cualquier motivo se ausente del proyecto, los desarrolladores se verán seriamente afectados debido que no pueden trabajar sin la participación constante del cliente.

Dado que al iniciar el proyecto no se tiene completa claridad sobre el contenido del proyecto, todos estos detalles se descubren en la medida que transcurre el tiempo con participación del cliente. Si el cliente no se encuentra para suministrar todos los detalles de implementación, el grupo de desarrollo no contará con otra fuente de información para obtenerlos.

Como se pretende abolir todo tipo de formalismos entre el cliente o usuario con el equipo de desarrollo, la comunicación se convierte en una herramienta fundamental para el

proyecto. En la media que ésta comunicación sea interrumpida, el equipo de desarrollo no tendrá más herramientas para saber qué implementar, con qué orden y cómo hacerlo.

Por todo lo anterior se debe procurar minimizar este riesgo teniendo varios usuarios que puedan relevarse unos a otros en un esquema similar al de rotación de personal planteado en el capítulo de planeación.

8. CONCLUSIONES

La experiencia del desarrollo del proyecto resultó satisfactoria. La elección y aplicación de dicha metodología dadas las características del problema arrojó resultados positivos en términos de satisfacción del cliente, cumplimiento de los plazos y buen ambiente de trabajo. Se encontró que la metodología se ajustó muy bien no solo al tipo de cliente y a las características del problema, también resultó adecuada para el entorno de trabajo y las características de los desarrolladores.

Se encontraron aspectos positivos y negativos en la aplicación de la metodología que no pretenden calificarla, pero sirven como puntos de referencia para proyectos de condiciones similares al expuesto. A continuación se exponen dichos aspectos.

- *Entregas frecuentes como elemento motivador del proyecto.*

En las metodologías pesadas, el tiempo que transcurre entre la firma del contrato y las primeras versiones del proyecto en manos del cliente, es considerablemente largo, sin embargo, durante este tiempo en que no se ven resultados se deben hacer inversiones económicas. Para algunos clientes este tiempo puede resultar demasiado prolongado por lo cual pierda interés en continuarlo para finalmente llevarlo al fracaso.

Debido que en XP se plantean entregas frecuentes, esta situación no se presenta. El cliente constantemente verá redituada su inversión en software con funcionalidades nuevas cada entrega. En este sentido se asegura el interés de éste por continuar el proyecto, al menos hasta que haya cumplido con sus expectativas iníciales, o aún mejor, se plantee incorporar nuevas funcionales no contempladas inicialmente que se convierten en aportes interesantes para el proyecto.

- *La comunicación es fundamental.*

Al no haber contratos rígidos ni mecanismos formales, se requiere que el proceso de comunicación sea muy fluido requiriendo que en todo momento haya un ambiente que la facilite. Es muy importante que el cliente pueda manifestar libremente sus necesidades y que el equipo de trabajo esté abierto tanto al lenguaje del cliente como a sus necesidades.

Desde el interior del equipo de trabajo, se requiere de igual forma una actitud muy abierta y de mucha "camaradería" que evite mecanismos formales de comunicación tales como memos o cartas. La comunicación deberá ser en lo posible directa y al mismo nivel evitando perder tiempo en malos entendidos. Es mucho más útil una explicación directa de una rutina que la documentación de la misma.

- *Internet como facilitador de la comunicación*

XP plantea como requisito fundamental una comunicación permanente entre todas las partes involucradas en el proyecto, enfatizando en un diálogo cara a cara. Sin embargo, no menciona al Internet como una herramienta. Dependiendo a la forma como se le dé uso, se puede convertir en una herramienta muy importante en aquellas ocasiones en que no se puede contar con una comunicación directa. Elementos como videoconferencias, chat, correo electrónico y voz IP, acompañados de una buena conexión a Internet pueden suplir, aunque no totalmente, una interacción cara a cara.

- *Spike Solution en cualquier metodología.*

El aporte más interesante de Spike Solution es la oportunidad que brinda a un desarrollador de solucionar un problema o adquirir un nuevo conocimiento abstrayéndose totalmente de cualquier particularidad del proyecto. El hecho de trabajar sobre un tema en particular permite llegar a un grado de profundización en el tiempo que no se logra de otra forma, dándole al programador la habilidad de inyectar dichos conocimientos en el proyecto.

Las bondades planteadas son aplicables sin importar bajo qué metodología se esté trabajando puesto que se presenta como herramienta fundamental para superar obstáculos de implementación.

- *Cliente Vs Usuario*

Existe una diferencia importante entre el cliente y el usuario. El primero es quién solicita el desarrollo de un software, que generalmente es el dueño o gerente de la empresa. El segundo es una persona que opera el sistema a construirse.

XP no hace tal distinción y constantemente plantea la permanencia del cliente en las instalaciones del equipo de desarrollo. Existen dos situaciones que dificultan esto. En primer lugar, si bien el cliente es quién solicita el desarrollo del producto, no es precisamente quién más conoce del mismo ya que habitualmente no es el operador. En segundo lugar, si el cliente es el gerente o dueño de la empresa, es precisamente la persona con menos tiempo disponible de todos y cuyas funciones son las más difíciles de delegar.

- *Integración frecuente*

En cualquier metodología, la integración es un tema que debe ser tratado con cuidado ya que muchos problemas se relacionan a errores cometidos durante la integración. Al ser frecuentes las integraciones son pocos los cambios que resultan de una versión a otra, por consiguiente son menos los errores que se encuentran a la hora de integrar. Además el equipo de trabajo siempre contará con una versión que posee los elementos más recientes, evitando el uso de código obsoleto que puede ser incompatible con el nuevo.

Al aumentar el número de integraciones también aumentan los riesgos asociados a una integración, lo que requiere la aplicación de normas más estrictas relacionadas al proceso del mismo. Es indispensable la asignación de dicha responsabilidad a una persona, la cual no solo hará la integración sino también debe garantizar que el programa no haya quedado con errores, y además asegurar la distribución de la última versión a todo el grupo de desarrollo.

- *La novedad que presenta la propiedad colectiva.*

Si bien la propiedad colectiva del código es una de las prácticas más importantes para XP no es una idea nueva. Se trata de un concepto deseado en cualquier proyecto de programación, para muchos, idealista. Lo realmente novedoso en XP es que bajo determinadas circunstancias es posible realizarlo e indica la forma en la cual puede ser obtenido.

El lograr la propiedad colectiva del código no es una tarea fácil, y así lo expone XP indicando una serie de disciplinas que se deben seguir como son la rotación de personal y la programación en parejas.

- *Se ajusta al entorno local*

Al XP plantear una metodología donde se omiten largas fases de análisis y diseño que pueden resultar costosas, se ajusta muy bien al entorno local en el cuál se requiere que los proyectos se realicen en forma rápida y economizando al máximos los recursos.
Una de las características que hace ideal a XP para el entorno local se trata de las entregas pequeñas, donde el cliente podrá empezar a ver redituada su inversión en el corto plazo en el sentido que tendrá un software funcional rápidamente.

En el mercado local se encuentran muchas necesidades que implican proyectos de software de pequeña o mediana envergadura los cuales se harían costosos al emplearse metodologías pesadas.

- *Ideal para proyectos donde no se tienen claros los requerimientos*

En proyectos donde los requerimientos no están claramente definidos desde un principio, se prevea la aparición de nuevos requerimientos o donde el alcance del proyecto no se ha delimitado se ajusta muy bien XP como metodología de desarrollo.

En la experiencia se encontró que al inicio del proyecto, el cliente no tenía claras todas las necesidades de su negocio, las cuales fueron surgiendo en la medida que este avanzaba. En caso de haberse empleado una metodología pesada donde los requerimientos solo se obtenían al principio, posiblemente se hubiesen omitido algunas funcionalidades que revestían de importancia para el cliente.

En la medida que se van cubriendo algunas necesidades, surgen o se descubren otras nuevas las cuales al emplearse XP podrán ser consideradas para su solución, lo cual es común cuando la delimitación del proyecto no está completamente definida desde el principio.

- *XP funciona en equipos de desarrollo pequeños*

Considerando los cambios que se realizaron a XP (debido a que el equipo de trabajo fue de solo dos integrantes) y algunos elementos de la programación en parejas, la utilización de la metodología en términos del trabajo en equipo funcionó muy bien. Los mecanismos de comunicación fueron exitosos, la estrategia de integración no presentó problemas, el proceso de designación de tareas fue adecuado y cumplimiento de los plazos a nivel individual fue satisfactorio.

- *La programación en parejas no es un proceso fácil.*

Según la experiencia obtenida, la programación no resultó tan sencilla como se podría pensar al documentarse en XP. No es simplemente sentar a dos programadores frente a un computador. Se requiere de una estrategia clara y de roles específicos a desempeñar por cada uno.

Para que la programación en parejas funcione de forma adecuada se necesita de un proceso de capacitación y entrenamiento, además de cierta "empatía" entre las dos personas que estarán trabajando juntas.

9. RECOMENDACIONES

Debe hacerse lo posible por no realizar modificaciones a XP demasiado drásticas ya que se corre el riesgo de alterar la esencia de la metodología.

Debe plantearse una estrategia para afrontar el diseño de datos en XP.

Se deben fijar una serie de reglas generales en la comunicación con el cliente ya que por el grado de informalidad que la metodología presenta, pueden surgir diferencias que pongan en peligro la culminación exitosa del proyecto.

Debe hacerse una capacitación al cliente sobre XP antes de iniciar el proyecto debido que este hace parte del equipo de desarrollo.

Plantear una estrategia especial de refactoring para las bases de datos.

Tener un buen conocimiento de las herramientas para la implementación antes de iniciar dicha etapa.

Plantear como unidad de tiempo horas en lugar de días para la asignación de tareas.

Considerar el internet y herramientas basadas en él como mecanismos de comunicación válidos dentro de XP y discutir la necesidad de un único sitio geográfico de trabajo.

Hacer una experiencia realizando un mismo proyecto por dos grupos de desarrollo independientes empleando una metodología pesada y XP con el fin de comparar los resultados obtenidos.

Emplear XP en un proyecto de mayor envergadura con el fin de evaluar el desempeño de la metodología en ese tipo de proyectos.

10. APORTES

- Un caso de estudio real en un entorno local con muchos de los ingredientes que se encuentran en dichas circunstancias debidamente documentado.

- Un paquete de conclusiones sobre la experiencia, exponiendo aspectos tanto positivos como negativos de la aplicación de la metodología XP.

- Las Recomendaciones para futuros trabajos sobre XP.

- El Marco Teórico amplio que describe los elementos principales de la metodología XP dando al lector un entendimiento amplio sobre la metodología.

- Se hizo una comparación entre los aspectos teóricos de XP y su aplicación por parte de los autores en el caso de estudio.

- Se aporta en el desarrollo académico que tiene la metodología XP.

-

11. BIBLIOGRAFÌA

BECK, Kent y CLEAL, Dave. Optional scope contracts: 1999.

BURKE, Erín y COYNER, Brian M. Java Extreme Programming CookBook. Beijing: O'Reilly

C3 TEAM. Chrysler Goes to "Extremes" : Case Study. Distribuited Computing: 1993.

CALERO SOLIS, Manuel. Una explicación de la programación extrema (XP) : V Encuentro usuarios x Base 2003. Madrid: 2003.

CASTILLO, Oswardo; FIGUEROA, Daniel y SEVILLA, Hector. Fases de la programación extrema<http://programacionextrema.tripod.com/fases.htm>

DUITAMA CASTELLANOS, Jorge Alexander; HERNÁNDEZ DURÁN, Mauricio y MORALES OLAYA, Juan Pablo. Programación Extrema y J2EE. Bogotá, 2003, 199p. Trabajo de grado (Ingeniero de Sistemas y Computación). Universidad de los Andes. Facultad de Ingeniería. Departamento de Ingeniería de Sistemas y Computación.

Ejemplo de desarrollo software utilizando la metodología XP. 9 de Julio de 2007. <http://www.dsic.upv.es/asignaturas/facultad/lsi/ejemploxp/>

eXtreme Programming: 9 de Julio de 2007. <http://deigote.blogspot.com/2006/03/extreme-programming.html>

Extreme Programming: A gentle introduction. 9 de Julio de 2007. <http://www.extremeprogramming.org/>

Extreme Roles. 9 julio de 2007 <http://c2.com/xp/ExtremeRoles.html>

FERNÁNDEZ ESCRIBANO, Gerardo. Introducción a Extreme Programming. 9 julio de 2007. <http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf>

Fundamentos de Extreme Programming. 10 diciembre de 2006. <http://www.programacion.com/blogs/84_metricas_web/archive/526_fundamentos_de_extreme_programming_parte_ii.html>

GONZÁLEZ CAMPOS, Saúl y FERNÁNDES MARTÍNEZ, Luis Felipe. Programación Extrema: Prácticas, Aceptación y Controversia. 10 diciembre de 2006. <http://www.uacj.mx/IIT/CULCYT/mayo-agosto2006/8ArtProg.pdf>
Programación eXtrema y Software Libre. 9 de Julio de 2007. <http://es.tldp.org/Presentaciones/200211hispalinux/ferrer/robles-ferrer-ponencia-hispalinux-2002.html>

Programación Extrema. 9 de Julio de 2007. <http://es.wikipedia.org/wiki/Programaci%C3%B3n_Extrema>

Programación Extrema. 9 de Julio de 2007. <http://www.chuidiang.com/ood/metodologia/extrema.html>

Programación Extrema: 9 de Julio de 2007. <http://www.programacionextrema.org/>

Una introducción a eXtreme Programming. 10 diciembre de 2006. <http://oness.sourceforge.net/proyecto/html/ch05.html>

XP: Extreme Programming. 9 de Julio de 2007. <http://iie.fing.edu.uy/~nacho/blandos/seminario/XProg1.html>

XP123. XPlorations. 9 de Julio de 2007. <http://xp123.com/xplor/>

ANEXOS

A. ESTÁNDARES

Los siguientes estándares responden a una serie de buenas prácticas recomendadas para desarrollar aplicaciones y promovidas entre otros por los grupos de usuarios de JAVA.

Si bien no se adoptaron todos los estándares recomendados para desarrollar en JAVA y se implementaron otros nuevos, lo que se buscó fue facilitar la comprensión del código para a su vez lograr la propiedad colectiva del código.

- Todo paquete inicia con minúscula.
- Toda clase inicia con mayúscula.
- Todo método inicia con minúscula.
- Toda variable inicia con minúscula.
- Sea clase, método o variable. Si consta de dos o más palabras. Todas van sin espacios y de la segunda en adelante con mayúscula inicial.
- Crear JavaDoc: Antes de todo método, variable y clase debe ir una explicación del mismo empleando el siguiente formato, con excepción de aquellos en que su comprensión resulte intuitiva a juicio del desarrollador, acogiéndose a la simplicidad según lo plantea XP.
    - /** comentario */
    - /**
      * comentario
      */
- Todo el código debe estar correctamente identado.
    - Se abre una llave en la siguiente línea vacía.
    - En la misma columna de la primera letra de la línea anterior.
    - No se escribe nada más en esa línea.
    - Se cierra en la misma columna donde fue abierto.
    - Tampoco se escribe nada en la línea donde se cerró.
    - EXCEPCIONES
        - Cuando no hay cuerpo dentro de las llaves o solo hay una línea.
        - Los try-catch.

          try{
                CUERPO
          }catch(Exception){

CUERPO

}

- Solo se declaran variables al inicio de una clase, método o ciclo.
- Se deben comprimir las versiones en .zip con siguiente formato.
  - dd-mmm-hh-mm.zip.
  - La hora en formato de 24 horas.
- NOMBRES DE VARIABLES
  - Todo elemento de un componente visual debe iniciar con la primera letra del tipo de objeto.
    - Label: lnombre.
    - TextField: tfnombre.
    - Combo: cnombre.
    - List: lnombre.
    - Etc.

- El nombre de las tablas es en singular y minúscula.
- Los objetos que administran la lógica del negocio tienen el mismo nombre que la tabla símil en la base de datos respetando los estándares anteriores.
- Las columnas de una tabla se convierten en atributos privados en la respetiva clase manteniendo el tipo de dato.
- Todos los atributos de una clase están encapsulados. Son privados y su acceso se da por medio de los métodos setAtributo() y/o getAtributo().
- Todas las ventanas tienen como convención de nombres ventantaFuncion o dialogoFuncion siendo la palabra función reemplazada por la correspondiente tarea que le permite al usuario desarrollar y usando la palabra ventana o dialogo según el tipo de interfaz gráfica creada en JAVA.*

- Todas las clases que correspondan con una tabla en la base de datos poseen los métodos crearRegistro, modicarCampo(llavePrimaria) y eliminarRegistr(llavePrimaria) según si son o no pertinentes.

* Los dos tipo principales de interfaces gráficas en JAVA son JFrame y JDialog

B. HISTORIAS DE USUARIO

NUMERO HISTORIA: 01
NOMBRE HISTORIA: Pago a proveedores
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 6 HORAS

DESCRIPCIÓN:
Se le cancela el dinero a un proveedor, luego se ingresa al sistema dicho pago del cual se imprime un tiquete y queda registrado para hacer parte de un resumen diario.
En algunos pagos se da un descuento en porcentaje sobre el total de la factura.
Se debe asociar con un código de factura de compra y un proveedor
No se le hace trato alguno al IVA, por ser régimen simplificado

NOTAS:

NUMERO HISTORIA: 02
NOMBRE HISTORIA: Venta normal
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 16 HORAS

DESCRIPCIÓN:
El cliente se acerca con los productos a la caja. El cajero ingresa cada uno de los productos. El sistema muestra el total. El cliente paga, se calcula la devuelta y se le entrega un tiquete de venta

Se ingresan los productos por código pero se acepta la posibilidad de hacerse por nombre
Si es por nombre, debe aparecer información complementaria a este.
Estándar para el código. Cada departamento tiene un rango de códigos

El tiquete de venta tiene la siguiente información
Encabezamiento, razón social, teléfono, fecha, etc.
Nombre del producto, código
Precio de todos los productos agrupado por cantidad (El precio de cinco bolsas de leche)
Nro de artículos
Total venta
Pago
Devuelta
hora al final

NOTAS:

NUMERO HISTORIA: 03
NOMBRE HISTORIA: Retirar un producto de venta en curso
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 2 HORAS

DESCRIPCIÓN:
En medio de una venta en proceso el cliente decide retirar un producto ya registrado de esa venta. El sistema retira el producto de la venta y se recalculan los valores necesarios. Se hace un retorno de mercancía

En el tiquete de venta se indica el producto, la cantidad y precio retornado. Es igual que en registro pero con un menos.

Debe haber dos funciones. Una que retire el último y otra que retire cualquier otro.

NOTA: Debe ser un proceso muy ágil

NUMERO HISTORIA: 04
NOMBRE HISTORIA: Cancelación de venta completa
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 1 HORAS

DESCRIPCIÓN: el cliente en medio de una venta sin terminar decide no comprar nada, por lo cual se debe cancelar la venta en curso presionando una tecla y un mensaje de confirmación.

NOTA: Debe ser un proceso muy ágil

NUMERO HISTORIA: 05
NOMBRE HISTORIA: Ingreso de mercancías a la tienda
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 2 HORAS

DESCRIPCIÓN: El ingreso de mercancía se realiza ítem por ítem, registrando el producto y la cantidad del mismo.

NOTAS:
No hay que asociar el ingreso de mercancía con el pago a un proveedor

---

NUMERO HISTORIA: 06
NOMBRE HISTORIA: Una sola venta partida con diferentes tiquetes
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 2 HORAS

DESCRIPCIÓN: Un único cliente desea partir su compra en varias ventas individuales con sus respectivos tiquetes. Cuando este hace esa solicitud antes de iniciar la venta, se registran los productos como ventas normales separadas, las cuales pueden hacerse con subtotales o con tiquetes independientes. Pero si este hace la solicitud después de haber registrado algunos productos, se debe cancelar la venta e iniciar varias independientes para cuenta

NOTAS:

NUMERO HISTORIA: 07
NOMBRE HISTORIA: Ventas en paralelo
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 1 HORAS

DESCRIPCIÓN: Al cajero se acercan varios clientes a la vez. Hay la posibilidad de iniciar una nueva venta sin haber terminado una en curso.

NOTAS:

NUMERO HISTORIA: 08
NOMBRE HISTORIA: Informe de necesidad de compra a proveedores
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO??): Alirio Ramírez
TIEMPO ESTIMADO: 5 HORAS

DESCRIPCIÓN: el sistema informa sobre los productos en los cuales la cantidad en inventario está llegando al límite mínimo de existencia. En el momento en que esto sucede se hace emisión de una alerta en pantalla y el despliegue de un icono distintivo para hacer el pedido en los siguientes días.

NOTAS:
El sistema debe permitir que haya cantidades negativas en el inventario.

NUMERO HISTORIA: 09
NOMBRE HISTORIA: Reporte Ganancias mensuales por descuentos de proveedores
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 3 HORAS

DESCRIPCIÓN: Mensualmente se debe saber cuánto dinero se ha ahorrado por concepto de descuentos dados por el proveedor. Para esto se cuenta con un historial de dichas compras con sus descuentos.

NOTAS:

NUMERO HISTORIA: 10
NOMBRE HISTORIA: Devolución de mercancía deteriorada o vencida al proveedor
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 10 HORAS

DESCRIPCIÓN: Hay varias formas de hacer cambios de productos deteriorados o vencidos según las políticas del proveedor.
La primera forma es mano a mano, donde el sistema no es notificado.
La segunda forma es por dinero, del cual se puede reconocer el 100% o una fracción del mismo, donde se hace la baja del producto del inventario y se realiza una nota a crédito

En el caso de devolución se saca con el precio de compra

NOTAS:
- SOLUCIONAR PROBLEMAS: QUÉ PASA SI EL PRODUCTO A CAMBIAR HA SUBIDO DE PRECIO?
- Tener control de vencimiento de los productos

NUMERO HISTORIA: 11
NOMBRE HISTORIA: Gestión de la base de datos
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 8 HORAS

DESCRIPCIÓN: Teniendo claro a cual departamento va, se ingresan los datos al sistema sobre proveedor, depto, cantidades, etc.
De c/ítem debemos tener dos precios. El de compra y el venta.

Igual se hace con proveedores, clientes, marcas, etc.

Un producto puede tener varios proveedores. Cada uno de ellos da precios diferentes.

En resumen las funciones son crear, modificar y los eliminar que sean posibles.

NOTAS:
Se debe hacer gestión de esos datos (crear, modificar, eliminar y consultar)

NUMERO HISTORIA: 12
NOMBRE HISTORIA: Inventario actualizado de mercancías
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 2 HORAS

DESCRIPCIÓN: El sistema muestra un informe del inventario del negocio
Nombre, código, existencia, valor compra y valor venta y punto crítico

NOTAS:

NUMERO HISTORIA: 13
NOMBRE HISTORIA: Reporte de históricos de movimiento de mercancías (reporte de ventas mensuales)
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 4 HORAS

DESCRIPCIÓN: En pantalla y en papel se visualizan los históricos del movimiento del negocio en pesos por meses y de un mes, todos sus días

NOTAS:

NUMERO HISTORIA: 14
NOMBRE HISTORIA: listados usuales
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
PRIORIDAD:
RIESGO:
TIEMPO ESTIMADO: 4 HORAS

DESCRIPCIÓN: el sistema suministra listados de proveedores, etc.
Lista de proveedores. No se saca lista de clientes. Reporte de productos que más y menos rotan.

NOTAS:

NUMERO HISTORIA: 15
NOMBRE HISTORIA: Cuadre diario (reporte de venta diario)
FECHA: 6 de Febrero 2007
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 8 HORAS

DESCRIPCIÓN:
Todo lo que se vendió en el día por ítems y dinero. Aparece nombre código cantidad y precio.
Totaliza el dinero y la cantidad. Aparece la hora al final del reporte.
En la caja registradora aparece en otro reporte pero podría estar en e l mismo la siguiente información: Dice cuanto inició la base, dice cuanto se compro a proveedores y nro. de facturas las cuales debería discriminar. Aparece cuanto se hizo en el día y los artículos que fueron retornados de ventas.
Inmediatamente sale el acumulado del mes hasta la fecha actual.

Debe decir cuánto dinero hay en caja.

Que diga los productos que están por debajo del límite

NOTAS:
Este es el X y el Z. solo se diferencian por una opción que reinicie la caja en el caso del Z

NUMERO HISTORIA: 16
NOMBRE HISTORIA: Reporte de mercancías con cantidad por debajo de la mínima
FECHA: 8 de febrero
ENTREVISTADO (USUARIO): Alirio Ramírez
TIEMPO ESTIMADO: 2 HORAS

DESCRIPCIÓN: Debe haber un reporte con nombre, código, cantidad actual y mínima de todos los ítems que estén por debajo de la cantidad mínima

NOTAS:

NUMERO HISTORIA: 17
NOMBRE HISTORIA: Productos en promoción.
FECHA: 10 de febrero
ENTREVISTADO (USUARIO): Diana Patricia Cardona
TIEMPO ESTIMADO: 6 HORAS

DESCRIPCIÓN: Ítem especiales para los productos en promoción. Algo así como ítems temporales que se le diga cuántos días va a existir. Que se elimine del inventario al término de ese plazo informando al cajero.

NOTAS:

NUMERO HISTORIA: 18
NOMBRE HISTORIA: Usuario prueba
FECHA: 10 de febrero
ENTREVISTADO (USUARIO): Diana Patricia Cardona
TIEMPO ESTIMADO: 6 HORAS

DESCRIPCIÓN: Usuario que pueda hacer de todo pero que no afecte las bases de datos para entrenar nuevo personal

NOTAS:

NUMERO HISTORIA: 19
NOMBRE HISTORIA: Privilegios de usuario
FECHA: 10 de febrero
ENTREVISTADO (USUARIO): Diana Patricia Cardona
TIEMPO ESTIMADO: 6 HORAS

DESCRIPCIÓN: Se debe poder establecer diferentes niveles de acceso a las funciones del sistema para cada usuario
NOTAS:

NUMERO HISTORIA: 20
NOMBRE HISTORIA: Control para apertura del cajón monedero
FECHA: 10 de febrero
ENTREVISTADO (USUARIO): Diana Patricia Cardona
TIEMPO ESTIMADO: 1 HORAS

DESCRIPCIÓN: Debe haber control para la apertura del cajón monedero. Por ejemplo, que solo abra cuando se culmine una venta, ingrese pago al proveedor, etc.
NOTAS:

C. TARJETAS CRC

Ilustración 2: Departamento

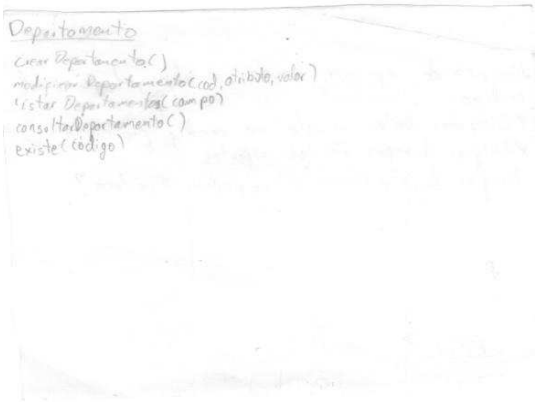


Ilustración 3: Extras

Ilustración 4: Función

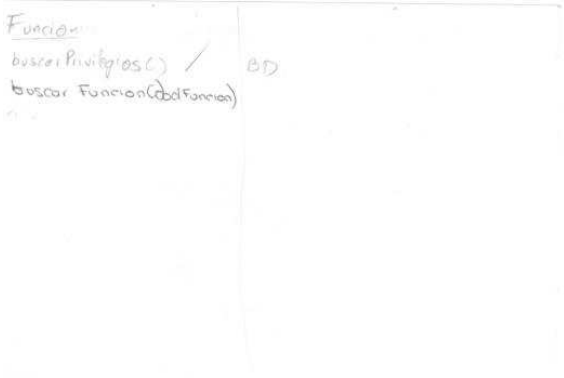


Ilustración 5: Impresión

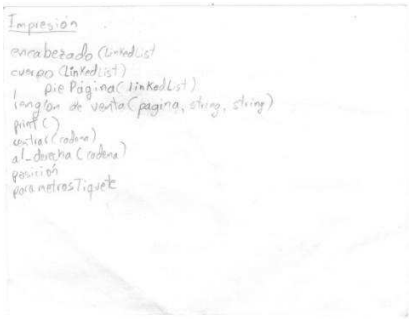


Ilustración 6: Parámetros

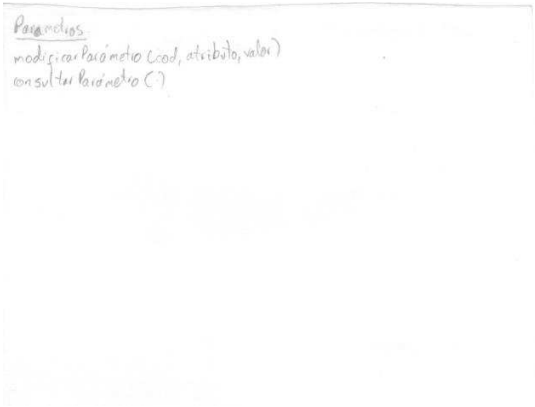


Ilustración 7: Producto

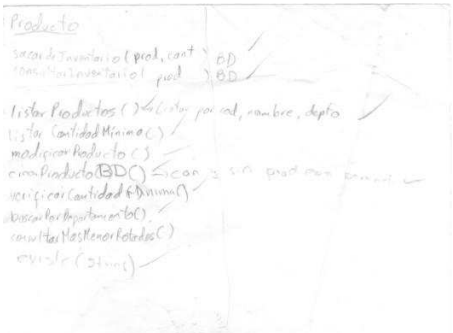


Ilustración 8: Proveedor

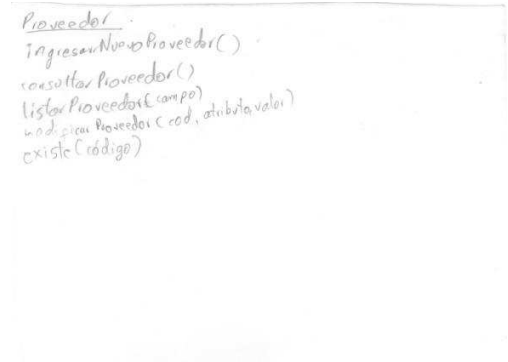


Ilustración 9: Sesión

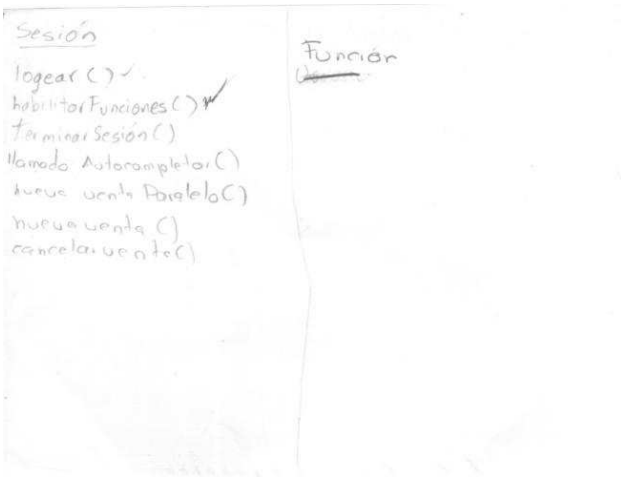


Ilustración 10: Tiquete

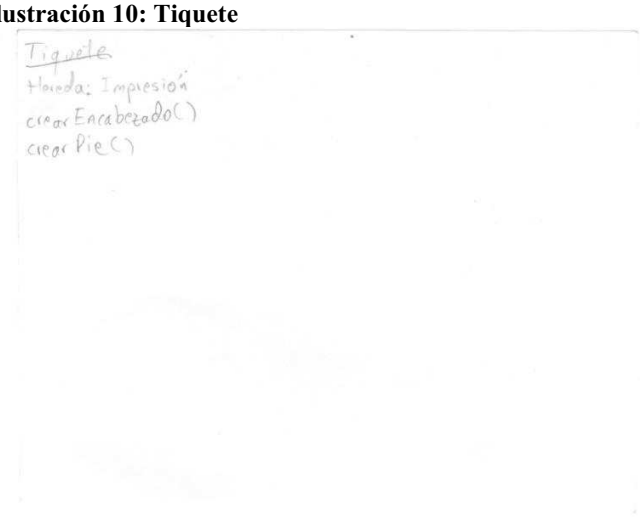


Ilustración 11: Usuario

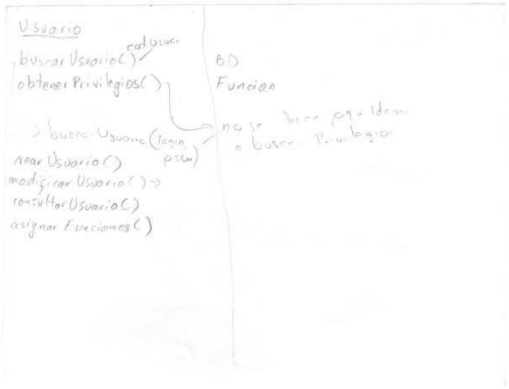
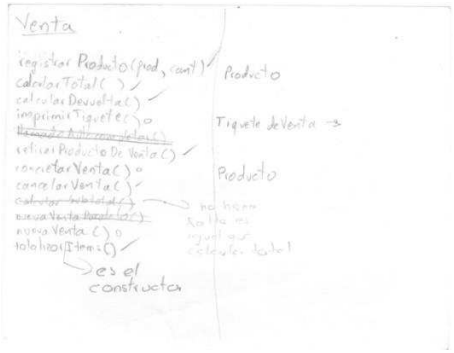


Ilustración 12: Venta


Venta
registrar Producto(prod, cant)
calcularTotal( )
calcularDevuelta( )
imprimirTiquete( )
retirarProductoDeVenta( )
cancelarVenta( )
cantidadVentas( )
nuevaVenta( )
totalItemsItem( )
es el constructor
Producto
Tiquete de Venta
Producto
no hacen falta en siguiente calcular total

D. MODELO ENTIDAD-RELACIÓN FINAL