Metodologías de Desarrollo Java EE

Sesión 2: Programación eXtrema

Puntos a tratar

- Introducción
- Valores
- Resultados
- Prácticas
  - Juego de Planificación
- Proceso
- Roles
- Para Saber Más

Introducción

- XP (*eXtreme Programming*) es una metodología ágil, fundada por *Kent Beck*, centrada en:
  - potenciar las *relaciones interpersonales*
  - promoviendo el *trabajo en equipo*
  - preocupada por el *aprendizaje de los desarrolladores*
  - propiciar un *buen clima de trabajo*.
- Trata un cambio social, centrado en el individuo
  - *"El desarrollo de software es demasiado duro para malgastar el tiempo en cosas que no son importantes. ¿Y qué importa realmente? Escuchar, probar, codificar y diseñar"* (Kent Beck)
- XP se basa en un conjunto de :
  - valores
  - principios
  - prácticas

Valores

- Conjunto de pilares que todo integrante debe tomar como propios, y que forman la base de la metodología
- Están más relacionados con el "querer hacer" que con el "saber hacer"
- Los valores son:
  - Comunicación
  - Simplicidad
  - Retroalimentación
  - Coraje

Comunicación

- Se requiere una comunicación fluida entre todos los participantes
  - El software se desarrolla tan rápido como lo permiten los canales de comunicación del proyecto
- La comunicación en XP es bidireccional
  - y se basa un sistema de continuas iteraciones de retroalimentación
- Todos los canales están abiertos a todas horas
  - cliente, programador, usuario, jefe de proyecto, gestor, etc...

Simplicidad

- La mejor solución es la más simple
- Cuanto más sencillas son las soluciones, más fáciles son de entender → KISS
- La solución más simple no quiere decir que sea la más fácil ni la más sencilla de implementar
- Guías:
  - Haz la cosa más simple que pueda funcionar
  - Representa los conceptos una sola vez (OAOO)
  - No vas a necesitarlo (YAGNI)
  - Elimina funcionalidad (o métodos) que ha dejado de utilizarse

Retroalimentación

- Ayuda a identificar los problemas desde el principio, tratar con aspectos desconocidos y clarificar aspectos
  - evita que los problemas se pospongan al final del proyecto
- Los proyectos deben retroalimentarse
  - desde el principio
  - de forma frecuente
  - y por parte del cliente, del equipo de desarrollo, de los usuarios finales, de terceras personas, etc...

Coraje

- Para afrontar los cambios se necesita ser valiente.
  - Para adoptar XP necesitarás coraje.
  - Coraje para refactorizar, programar para hoy y no para mañana, para programar en parejas, escribir las pruebas antes que el código
- ¿Estoy motivado para tener coraje?
  - La motivación se consigue mediante el interés, el miedo o la confianza.
  - XP ayuda a que crezca la confianza exponiendo a los integrantes a pequeños pero continuos éxitos.
  - Supera el miedo mediante los otros 3 valores

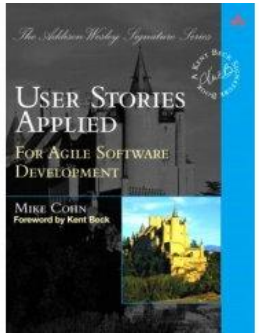
Puntos a tratar

- Introducción
- Valores
- Resultados
- Prácticas
  - Juego de Planificación
- Proceso
- Roles
- Para Saber Más

Resultados XP

Requisitos	Diseño
Historias de usuario	**Tarjetas CRC** **Esbozos** diagramas (UML) en borrador sobre pizarras blancas
Implementación	**Pruebas**
Gestión de Proyectos	**Gestión y Configuración del Cambio**
Listado de tareas ya sea tarjetas de papel o un listado en una pizarra belleda en la cual se escriben las tareas de las historias Su granularidad es de 1 a 2 días. **Gráfico visible** en la pared, para una mejor comunicación. El contenido depende de los equipos (por ejemplo, número de pruebas definidas respecto a pasadas) **Historias de usuario**	

Historias de Usuario (*story cards*) I



- Técnica utilizada en XP para especificar los requisitos del software.
- **Tarjetas de papel** en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales, incluso para crear documentación.
- El tratamiento de las historias de usuario es muy dinámico y flexible
  - en cualquier momento pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas.
- Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Historias de Usuario (*story cards*) II

- Existen varias plantillas sugeridas pero no existe un consenso al respecto.
  - En muchos casos sólo se propone utilizar un nombre y una descripción, más una estimación.

Historia de Usuario	
Número:	Nombre Historia:
Usuario:	Iteración asignada:
Prioridad en negocio: **(Alta / Media / Baja)**	Puntos estimados:
Riesgo en desarrollo: **(Alto / Medio / Bajo)**	Puntos reales:
Descripción:	
Observaciones:	

Tarjetas CRC I

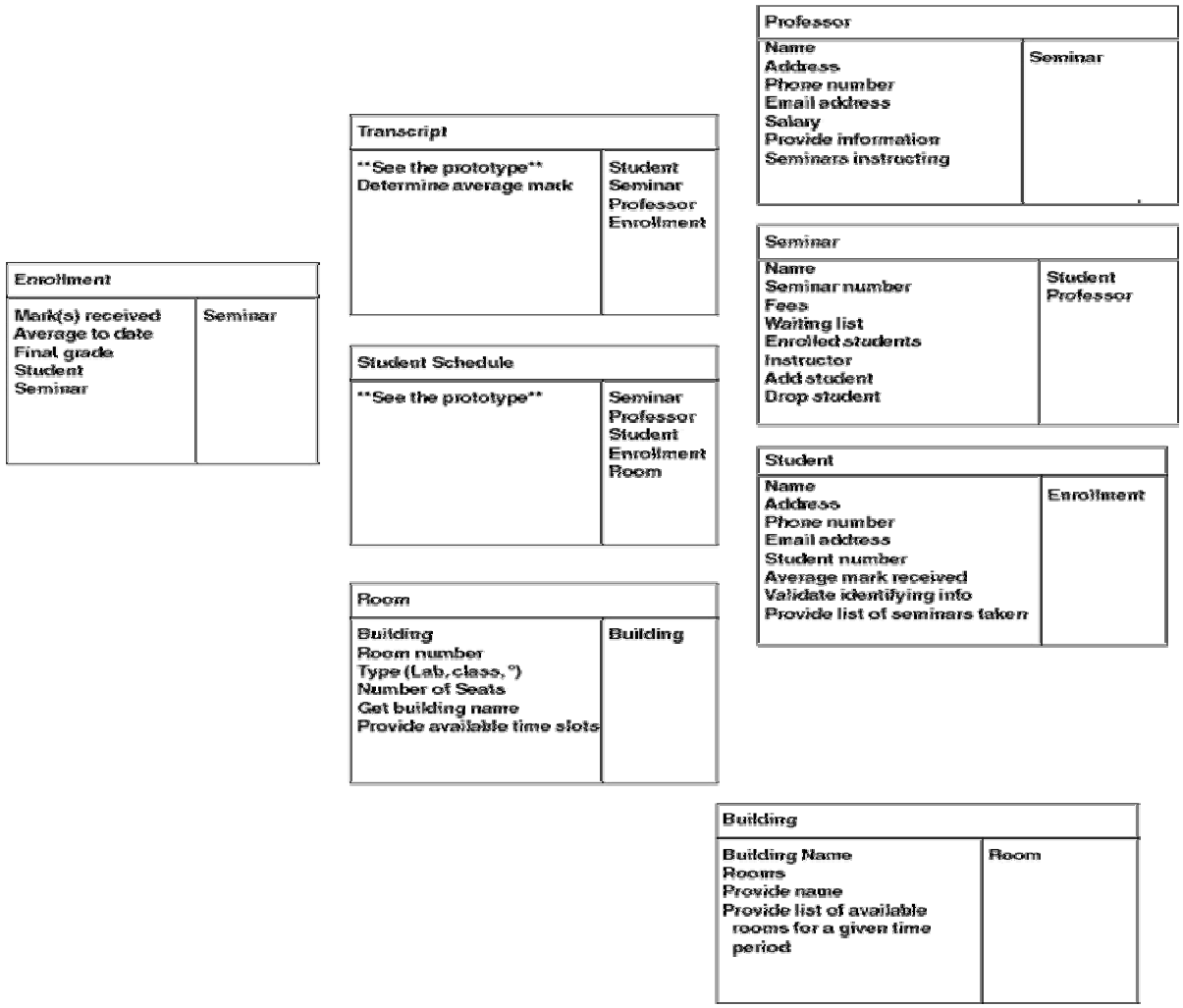
- CRC = **Clase + Responsabilidad + Colaboración**
- Mecanismo sencillo pero efectivo para enfocar un diseño que involucra a varios participantes de un equipo.
  - Se comienza la sesión de diseño con un montón de tarjetas en blanco (cuartillas de cartulina, folio doblado en 2 veces, ...)
  - Cada cartulina representa un objeto del sistema, y se escribe en ella el nombre del objeto en la cabecera, las responsabilidad en la parte inferior izquierda y las clases de colaboración a la derecha.

Clase	
Responsabilidad	Colaboración

Sesión CRC

- La sesión comienza con una historia de usuario, y cada miembro del equipo con un número de tarjetas.
- El equipo conversa sobre el escenario, y el miembro responsable de la tarjeta se responsabiliza de guiar dicha historia.
- El detalle de las tarjetas cambia como resultado de la discusión, hasta que se identifican un conjunto de clases, responsabilidad y las colaboraciones asociadas que permitirán ejecutar el escenario requerido.
- En una sesión CRC de XP, no se produce un diseño formal, pero el grupo se lleva una idea del como el sistema implementará la funcionalidad requerida.

Ejemplo CRC

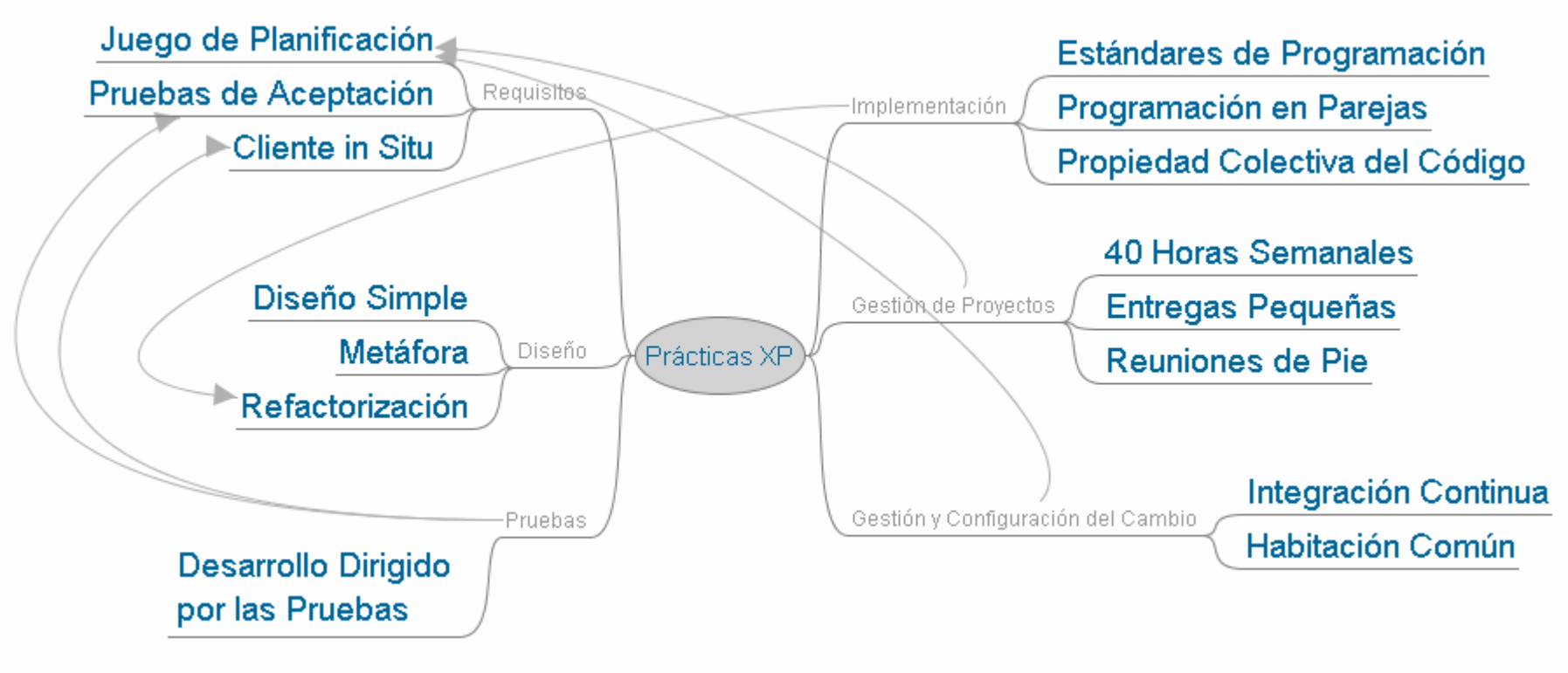

Professor
Name
Address
Phone number
Email address
Salary
Provide information
Seminars instructing
Seminar
Transcript
See the prototype
Determine average mark
Student
Seminar
Professor
Enrollment
Seminar
Name
Seminar number
Fees
Waiting list
Enrolled students
Instructor
Add student
Drop student
Student
Professor
Enrollment
Mark(s) received
Average to date
Final grade
Student
Seminar
Seminar
Student Schedule
See the prototype
Seminar
Professor
Student
Enrollment
Room
Student
Name
Address
Phone number
Email address
Student number
Average mark received
Validate identifying info
Provide list of seminars taken
Enrollment
Room
Building
Room number
Type (Lab, class, *)
Number of Seats
Get building name
Provide available time slots
Building
Building
Building Name
Rooms
Provide name
Provide list of available rooms for a given time period
Room

Puntos a tratar

- Introducción
- Valores
- Resultados
- Prácticas
  - Juego de Planificación
- Proceso
- Roles
- Para Saber Más

Prácticas XP

- Las prácticas XP que pueden apoyarse en varias disciplinas
- Se muestran agrupadas por fases de desarrollo:



Juego de Planificación I

- Espacio frecuente de comunicación entre el cliente y los programadores.
- Se plantea como un juego, donde existen dos tipos de jugadores:
  - El **Cliente** establece la prioridad de cada historia de usuario, de acuerdo con el valor que aporta para el negocio
    - decide el ámbito y tiempos de la entregas
  - Los ***Desarrolladores*** estiman el esfuerzo asociado a cada historia de usuario. Se ordenan las historias según prioridad y esfuerzo, y se define el contenido de la entrega y/o iteración, apostando por enfrentar lo de más valor y riesgo cuanto antes.
    - realiza la estimación de las historias y las implementa
- Este juego se realiza durante:
  - la planificación de la entrega
  - en la planificación de cada iteración
  - cuando sea necesario reconducir el proyecto.

Juego de Planificación II

- El juego se divide en tres fases:
  1. **Exploración**: determina nuevas historias de usuario del sistema
  2. **Compromiso**: determina que funcionalidades se desarrollaran en una determinada entrega
  3. **Ajuste (*Steering*)**: actualiza la planificación conforme el desarrollo progresa.
- Estas 3 fases son iterativas y suelen interactuar entre si.
  - Por ejemplo, si en la fase de *Ajuste* nos damos cuenta de que necesitamos nuevas historias de usuario, entonces se vuelve a jugar en la fase de *Exploración* para las nuevas historias necesarias, tras lo cual se realiza el *Compromiso*, y posteriormente se vuelve a revisar el *Ajuste* para producir una planificación revisada.

Juego de Planificación III - Exploración

- Durante la fase de Exploración, el juego intenta ayudar al equipo a identificar que debería hacer el sistema, mediante 3 pasos:
  1. **Escribir**.
     - El *Cliente* empieza a hablar sobre lo que el sistema debe realizar. Estas descripciones se escriben como Historias de Usuario (en cuartillas).
       - Las ideas iniciales se discuten mejor sobre pizarras antes de pasarlas a papel.
  2. **Estimar**
     - Los *Desarrolladores* estiman cuanto tardarán en implementar la historia.
     - Si los *Desarrolladores* no pueden estimar la historia, entonces pueden pedir aclaraciones o solicitar que la historia se divida (para facilitar su comprensión).
       - En el juego de planificación inicial se intenta obtener una cifra aproximada.
       - Durante la planificación de la entrega se requerirá mayor nivel de detalle.
  3. **Dividir**
     - Las historias de usuario pueden variar en tamaño y complejidad.
       - Los usuarios normalmente no son conscientes de cuanto trabajo implica una cierta funcionalidad del sistema.
       - Los desarrolladores normalmente no entregan lo que el cliente quiere.
     - Las historias necesitan refinarse: las historias grandes (en términos de complejidad o duración) necesitan partirse en historias más pequeñas (y menos complejas).
     - Ninguna historia de usuario puede ser tan grande que ninguna pareja de programadores sea capaz de completarla dentro de una iteración.

Juego de Planificación IV – Compromiso I

- Esta etapa se plantea de forma secuencial:
  1. El *Cliente* ordena la importancia relativa de las diferentes historias de usuario
  2. Los *Desarrolladores* determinan sobre las historias importantes que riesgo conllevan
  3. El *Cliente* decidirá qué historias formarán parte de una entrega particular
  4. Los *Desarrolladores* llegan a un compromiso de aceptar la duración y el contenido de la entrega
     - Si esta situación no se puede alcanzar, entonces o bien se modifica la fecha de la entrega, o se altera el contenido de la misma.

- Los pasos detallados que se realiza para alcanzar el acuerdo son:
  1. **Ordenar por Valor**.
     - El *Cliente* ordena las historias de usuario en tres montones.
       - Representan las historias con **prioridades alta** (obligatorias/*must have*), **media** (deberían hacerse/*should have*) y **baja** (pueden aportar valor/*nice to have*).
     - Las historias del montón de prioridad alta son las más importantes.
     - El equipo se centrará primero en este montón, ya que serán las que aporten mayor valor al negocio.

Juego de Planificación V – Compromiso II

2. **Ordenar por Riesgo**.
   - Las historias de usuarios se reordenan en otros 3 montones
     - Representan un **riesgo alto** (no se pueden estimar), **medio** (estimación razonable) y **bajo** (estimación fiable).
   - Permiten averiguar las historias de usuario que el *Cliente* considera que tienen una prioridad alta y que se han estimado con fundamentos.
   - Como resultado quizás debamos volver a la etapa de Exploración para intentar aclarar algunos aspectos relativos a la estimación de las historias de usuario.
   - Algunas historias pueden dejarse para investigar posteriormente una vez haya terminado el juego.
3. **Elegir Alcance**.
   - El Cliente debe elegir el conjunto final de historias de usuario que formaran parte de la siguiente iteración/entrega.
     - La primera entrega debería estar completa en términos de uso (aunque con una funcionalidad muy limitada)
     - Y cada entrega posterior debe añadir algo que aporte valor al *Cliente* (para que se considere una entrega).

¡ No todas las iteraciones tienen como resultado una entrega !

Juego de Planificación V – Compromiso II

4. **Fijar la Velocidad del Proyecto**.
   - Este paso mapea la Unidad Ideal de Ingeniería (IDU) con la realidad y toma en cuenta la cantidad de tiempo que los desarrolladores realmente son productivos, su experiencia, etc...
   - Ofrece un modo de comparar periodos estimados idealmente con tiempo real restante.

- La realidad es que se trata de un proceso iterativo, donde el *Cliente* revisa las historias conforme el juego avanza, influenciado por
  - los *Desarrolladores*
  - la necesidad de dividir las historias
  - y descubrir nuevas historias.

Juego de Planificación VI - Ajuste

- A lo largo de la vida de un proyecto, una planificación requiere revisiones frecuentes y extensas.

- El objetivo de esta etapa es ajustar el proyecto y retomar la dirección correcta. Los pasos son:
  1. **Planificación de Iteración**.
     - Sólo debemos planificar en detalle la iteración actual.
     - Al inicio de cada iteración (cada 1-3 semanas), el *Cliente* planifica las Historias de Usuario a implementar, y los *Desarrolladores* planifican las tareas necesarias para implementar dichas historias.
  2. **Recuperación de Proyecto**.
     - Conforme progresa la iteración, si los Desarrolladores consideran que van retrasados o adelantados respecto al calendario, pueden pedirle ayuda al Cliente para re-priorizar las historias de usuario a implementar.

Juego de Planificación VI - Ajuste

3. **Identificar una nueva Historia**.
   - Si se identificar una nueva historia y se determina que es necesaria para la entrega actual, entonces se puede escribir, estimar y añadida a la iteración.
   - Las restantes historias necesitan revisarse y algunas serán descartadas para lograr cumplir la entrega.
4. **Reestimación de Proyecto**.
   - Si los desarrolladores consideran que la planificación se desvía de la realidad, entonces se puede:
     - replanificar la iteración completa
     - reestimar las historias de usuario
     - poner a cero la velocidad del proyecto
     - considerar las implicaciones del calendario del proyecto.

- Esta etapa suele realizarse en cualquier momento de la iteración (o al final).
- Sin embargo, las 2 primeras etapas ocurren al inicio de la iteración.
- Por lo tanto, el juego de la planificación como práctica XP se parte en trozos que se ejecutan en diferentes momentos den del ciclo de vida del proyecto.

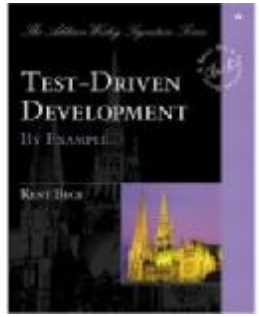
Entregas Pequeñas

- La idea es producir rápidamente versiones del sistema que sean operativas
  - aunque obviamente no cuenten con toda la funcionalidad pretendida para el sistema
  - pero si que constituyan un resultado de valor para el negocio.
- Cada entrega es lo más corta posible:
  - Contenga requisitos más valiosos del sistema (básicos)
  - Reducen el riesgo → mayor retroalimentación desde el cliente, y más frecuente
- Minimizar el nº de historias de usuario que componen una entrega → No realizar historias de usuario a medias
- Una entrega no debería tardar más 3 meses.

Diseño Simple

- Se debe diseñar la solución más simple que pueda funcionar
  - Debe ser implementada en un momento determinado del proyecto.
- ¡No confundir el diseño simple con el diseño fácil o trivial!
- La complejidad innecesaria y el código extra deben ser eliminados inmediatamente
- Aunque pueda parecer que XP minoriza el diseño, realmente se integra dentro de la implementación
- El mejor diseño para el software es aquel que:
  - supera con éxito todas las pruebas
  - no tiene lógica duplicada
  - refleja claramente la intención de implementación de los programadores, por lo cual es fácilmente comprensible por un programador externo
  - tiene el menor número posible de clases y métodos

Desarrollo Dirigido por las Pruebas I



- En XP las pruebas son la actividad dominante.
- Las pruebas se producen antes de escribir cualquier código (*Test Driven Development - TDD*)
  - las pruebas de aceptación se generan tan pronto como se escriben los requisitos
  - las pruebas unitarias se producen antes de implementar el código
  - todo el código debe pasar todas las pruebas durante el desarrollo
  - las pruebas se ejecutan tras cada modificación
- **¡ No puede existir código sin su prueba asociada !**
- Debemos ser capaces de escribir las pruebas para un módulo antes de implementarlo.
  - si no puedes escribir las pruebas antes de implementar el módulo, es que todavía no sabemos lo suficiente sobre él.
- Dentro de este contexto de desarrollo evolutivo y de énfasis en pruebas, la automatización para apoyar esta actividad es crucial.

Desarrollo Dirigido por las Pruebas II

- Resultados
  - Un conjunto completo de pruebas para todo el código producido.
  - El código más sencillo que pasa las pruebas.
  - Una visión clara de lo que el código debería y no debería hacer.
  - Una forma sencilla de comprobar que el código refactorizado no ha modificado la funcionalidad.
  - Un modo excelente de "documentación" explicando que debería o no debería hacer un modulo.

- Tipos de Pruebas
  - Pruebas Unitarias
  - Pruebas de Aceptación

Desarrollo Dirigido por las Pruebas III

- Pruebas Unitarias
  - Conforman la columna vertebral del enfoque TDD, siendo **responsabilidad del desarrollador** su creación.
  - Consisten en un conjunto de pruebas de granularidad fina que normalmente comprueban la funcionalidad de un componente a través de sus métodos y atributos públicos.
  - Las pruebas unitarias se pueden considerar en una combinación de pruebas de:
    - *caja negra* (la prueba unitaria valida la clase a través de sus entradas y sus salida, confirmando que se cumplen los requisitos especificados)
    - *caja blanca* (la prueba se basa en el interior de la clase, en el código implementado).
  - Dentro de XP, siempre se utiliza algún tipo de framework, y en concreto, dentro de la plataforma Java tenemos
    - **JUnit** (junit.org)
    - **Cactus** (jakarta.apache.org/cactus)
    - **HttpUnit** (sourceforge.net/projects/httpunit)

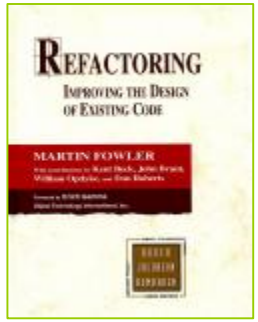
Desarrollo Dirigido por las Pruebas IV

- Pruebas de Aceptación
  - Un prueba de aceptación confirma que el sistema cumple los criterios de aceptación de acuerdo con los clientes finales.
  - Comúnmente, este tipo de pruebas lo **realiza el propio equipo** de QA **del cliente**, o un equipo formado por el cliente y el usuario final.
  - Dependiendo del tipo de aplicación, podemos utilizar algún framework específico como puede ser **Fit** (fit.c2.com y www.fitnesse.org).
    - Este tipo de pruebas, al estar relacionadas con la perspectiva del usuario final son más difíciles de probar.

- Independientemente del tipo de prueba, todo el código debe pasar las pruebas antes de darlo por bueno.

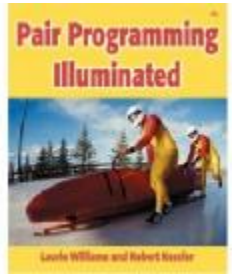
Refactorización

- Arte de mejorar la estructura interna del código sin alterar su comportamiento externo
- Objetivos
  - remover duplicación de código, redundancia
  - mejorar la legibilidad, homogeneidad
  - simplificar el código → KISS
  - hacerlo más flexible para facilitar los posteriores cambios.



- Para mantener un diseño apropiado, es necesario realizar actividades constantes de cuidado continuo durante todo el ciclo de vida del proyecto.
  - Este cuidado continuo sobre el diseño es más importante que el diseño inicial.
  - Un concepto pobre al inicio puede ser corregido mediante refactorización, pero sin ella, un buen diseño inicial se degradará.
- El 25% del esfuerzo de un proyecto XP se consume en técnicas de refactorización
  - Razón por la cual no se realiza un modelado intenso.

Programación en Parejas I



- Toda la producción de código debe realizarse en parejas de programadores
  - dos programadores en un ordenador (un teclado y un ratón)
- Los 2 desarrolladores juegan roles diferentes:
  - Uno se centra en el método, clase o interfaz a implementar
    - este es el que tiene el teclado y el ratón
  - El otro se centra en temas más estratégicos
    - la solución elegida se integra bien con el resto del sistema: implicaciones y dependencias
    - ¿ existe una solución mejor ?
    - ¿ pruebas cubren todas las posibilidades ?
- Estas parejas no deberían durar más de 2 o 3 días (incluso 2 o 3 horas), dependiendo de la historia a implementar.
  - Durante este periodo, ambos programadores deben intercambiar los mandos

Programación en Parejas II

- Las parejas cambian conforme cambian las tareas y las historias .
  - si un desarrollador esta trabajando en un área del código que no domina, debería formar pareja con un desarrollador experto en dicha área
- ¡ Nunca se deben unir 2 programadores jóvenes o inexpertos !
  - Lo ideal es juntar a un experto con un programador júnior

Programación en Parejas III - Ventajas

- 2 cerebros son mejor que uno.
- El conocimiento se extiende → **aprendizaje cruzado**
  - posibilita la transferencia de conocimientos.
- Inspecciones de código continuas (detección de errores en caliente)
  - la tasa de errores del producto final es más baja
  - los diseños son mejores
  - el tamaño del código menor (continua discusión de ideas de los programadores).
- Mayor cobertura de las pruebas,
  - 2 personas ofrecerán 2 perspectivas distintas.
- La experiencia del equipo se extiende
  - mediante la fusión de expertos y novatos, programadores senior y júnior.
- Los programadores conversan mejorando así el flujo de información y la dinámica del equipo.
- Los programadores disfrutan más su trabajo.

Propiedad Colectiva del Código I

- *¿Quien debe cambiar un objeto que necesita modificarse?*

- Cualquier programador puede mejorar cualquier parte del código en cualquier momento, ya que el equipo entero es responsable de todo el código.
- Cada integrante tiene el derecho y el deber de realizar modificaciones sobre cualquier parte del sistema cada vez que identifica una oportunidad de mejorarlo.

- No confundir con “cada desarrollador es responsable de su código”, ignorando el resto
  - fomenta una cultura de culpar e increpar a los compañeros

Propiedad Colectiva del Código II

- Objetivo → evitar la exclusividad y dependencia del código respecto a un desarrollador, lo que puede dar pie a opiniones del estilo de "es su código, es su problema".
  - Todos conocen algo sobre todas las partes y conocen muy bien aquellas en las que trabajan
- Motiva a todos a contribuir con nuevas ideas en todos los segmentos del sistema,
  - evita que algún programador sea imprescindible para realizar cambios en alguna porción de código
  - fomenta la agilidad en los cambios.
- Un equipo que no sea XP puede encontrar esta práctica problemática: *“mí codigo” vs “vuestro código”*

Integraciones Continuas

- *¿Cómo y cuando un equipo debe comprobar que el código de cada uno de los integrantes funciona de forma correcta cuando se integran todos los elementos?*
- Cada pieza de código es integrada en el sistema una vez que esté lista.
  - Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
  - Se ejecutan todas las pruebas y tienen que ser validadas para que el nuevo código sea incorporado definitivamente al sistema de control de versiones.
  - Para tener un mayor control sobre el proyecto, las integraciones de los componentes y la calidad del mismo, es esencial el desarrollo de un **proceso disciplinado y automatizado**.
- Solución → Máquina separada que corra un proceso 24/7 que se dedique a construir y probar el sistema de forma ininterrumpida.

Cliente in-situ I

- *Cuantas veces hemos leído un documento de requisitos y nos han aparecido dudas del tipo ¿esto que significa? ¿que pasa si cancela en este punto?*
- El cliente tiene que estar presente y disponible todo el tiempo para el equipo.
- Éxito XP → es el cliente quien conduce el trabajo hacia lo que aportará mayor valor de negocio
  - los programadores pueden resolver de manera inmediata cualquier duda asociada.
- La comunicación oral es más efectiva que la escrita
  - la escrita toma mucho tiempo en generarse y puede tener más riesgo de ser mal interpretada.
- También se propone la práctica de *Habitación Común*
  - programadores y cliente trabajan juntos en la misma habitación.

Cliente in-situ II

- Uno o más clientes ("el cliente" ≠ "un cliente") deben permanecer, a ser posible, todo el tiempo con el equipo.
- De ellos se espera que:
  - sean expertos en la materia
  - tengan peso suficiente en el proyecto para tomar decisiones relativas a los requisitos y sus prioridades.
- Si no disponemos del cliente en la Habitación Común:
  - intentar conseguir un representante que pueda estar siempre disponible y que actúe como interlocutor del cliente
  - contar con el cliente al menos en los juegos de planificación
  - establecer visitas frecuentes de los programadores al cliente para validar el sistema (demos de entregas)
  - anticiparse a los problemas asociados estableciendo llamadas telefónicas frecuentes y conferencias (también IM)
  - reforzar el compromiso de trabajo en equipo.

Estándares de Programación I

- XP enfatiza la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación
  - del equipo
  - de la organización
  - u otros estándares reconocidos para los lenguajes de programación utilizados
- Deben promover la simplicidad, y deben ser aceptadas por todos los integrantes del equipo.
  - las imposiciones pocas veces funcionan → llegar a un consenso
- Debido a la propiedad colectiva del código, la refactorización y la programación por parejas, el código debe cumplir un estándar.

Estándares de Programación II - Ventajas

- Mantienen el código homogéneo → mismo estilo
  - siendo legible para los miembros del equipo, facilitando los cambios.
- Se evitan discusiones tontas
  - de si las llaves deben ir en la misma línea o en la posterior de una sentencia.
- Cada integrante del equipo se siente cómodo con el estilo de codificación adoptado
  - lo que facilita su lectura y comprensión.
- Simples guías como "*Todas las constantes en mayúsculas*" significan que tan pronto como se visualiza la convención, se sabe el significado.
- Ayudan a crear un entorno donde el código se despersonaliza
  - sin propietarios ni creadores
- Facilitan la refactorización y programación en parejas.

40 Horas Semanales

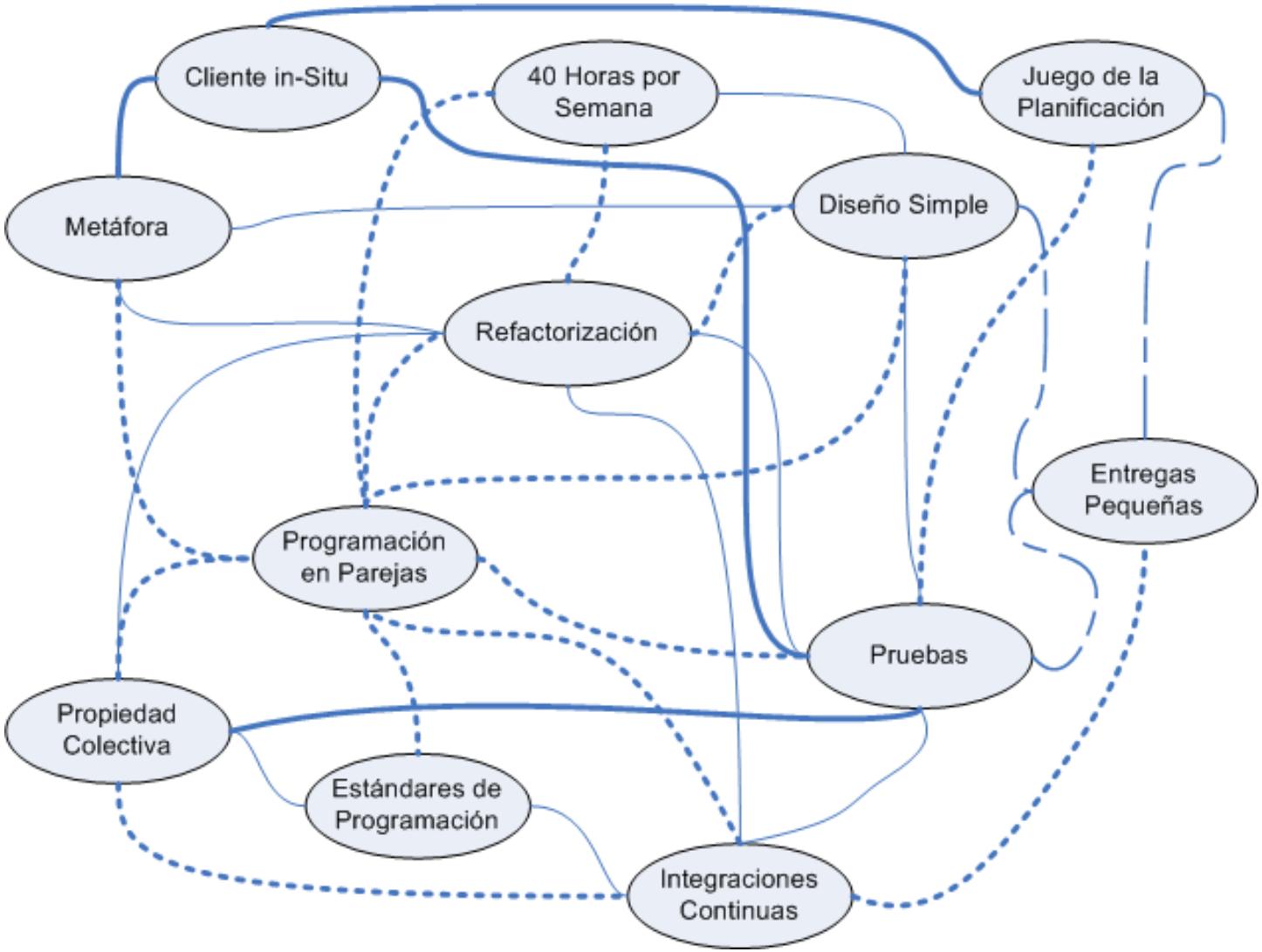
- *¿Qué pasa cuando un equipo de desarrollo trabaja más de la cuenta porque no se llega a una fecha de entrega?*
  - El trabajo extra desmotiva al equipo.
  - Los programadores que descansan son más productivos
- Se debe trabajar un máximo de 40 horas por semana (de 35 a 45), con un ritmo de trabajo adecuado, así como no trabajar horas extras en dos semanas seguidas.
- Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse.
  - Los proyectos que requieren trabajo extra para intentar cumplir con los plazos suelen al final ser entregados con retraso.
  - Se debe realizar el juego de planificación para cambiar el ámbito del proyecto.

Metáfora

- Historia compartida que describe cómo debería funcionar el sistema, formando un conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema.
  - Este conjunto de nombres ayuda a la nomenclatura de clases y métodos del sistema.

- XP no enfatiza la definición temprana de una arquitectura estable para el sistema.
  - La arquitectura se asume evolutiva
  - Los posibles inconvenientes que se generarían por no contar con ella explícitamente en el comienzo del proyecto se solventan con la existencia de una metáfora.
  - El sistema se define mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo.

Interacción entre Prácticas



Aplicación de las Prácticas

- La mayoría de las prácticas propuestas por XP no son novedosas
  - De alguna forma ya habían sido propuestas en Ingeniería del Software e incluso demostrado su valor en la práctica.
- El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.
- Es un error aplicar un subconjunto de prácticas no compensadas, ya que unas se apoyan en otras.
  - Primero hemos de probarlas todas y luego, dependiendo de nuestras circunstancias particulares, "customizar" aquellas prácticas que nos sean más útiles.

¿Qué hay de eXtremo en XP?

- XP es muy ligero, ya que realmente solo se centra en la parte de programación de un sistema software.
- XP toma un conjunto de prácticas que son exitosas y las lleva al extremo:
  - Si las revisiones de código son buenas, entonces revisaremos el código continuamente (**programación en parejas**).
  - Si las pruebas son buenas, todo el mundo realiza pruebas continuamente (**pruebas unitarias**), incluso los clientes (**pruebas de aceptación**).
  - Si diseñar es bueno, entonces debe formar parte de lo que todos hacen cada día (**refactorizar**).
  - Si la simplicidad es buena, entonces siempre lucharemos por la solución más simple (la **solución más simple** que funcione).
  - Si las pruebas de integración son buenas, entonces la integración y las pruebas tienen que ser una todo (**integraciones continuas**) continuo (diarias o por horas).
  - Si las iteraciones cortas son buenas, entonces acortémoslas al máximo; por ejemplo, horas o días, no semanas o meses (**juego de planificación**).

Puntos a tratar

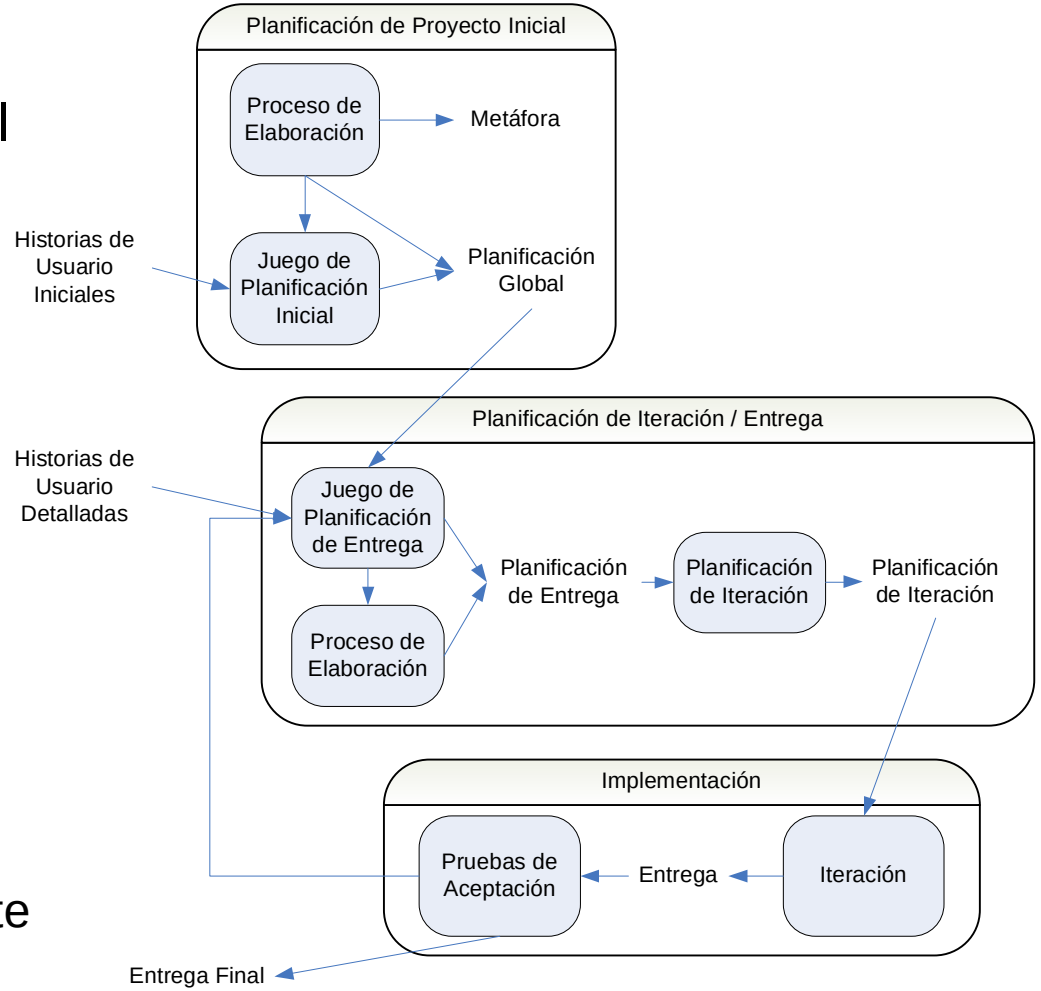
- Introducción
- Valores
- Resultados
- Prácticas
  - Juego de Planificación
- Proceso
- Roles
- Para Saber Más

Proceso XP

- Un proyecto XP tiene éxito cuando:
  - El cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo
    - para medir la funcionalidad que puede entregar a través del tiempo.
  - El cliente debe de manejar el ámbito de entrega del producto
    - para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.
- El ciclo de desarrollo consiste en los siguientes pasos:
  1. El cliente define el valor de negocio a implementar.
  2. El programador estima el esfuerzo necesario para su implementación.
  3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
  4. El programador construye ese valor de negocio.
  5. Vuelve al paso 1.
- En todas las iteraciones tanto el cliente como el programador aprenden.
- No se debe presionar al programador a realizar más trabajo que el estimado
  - perdida de calidad en el software o no se cumplirán los plazos.

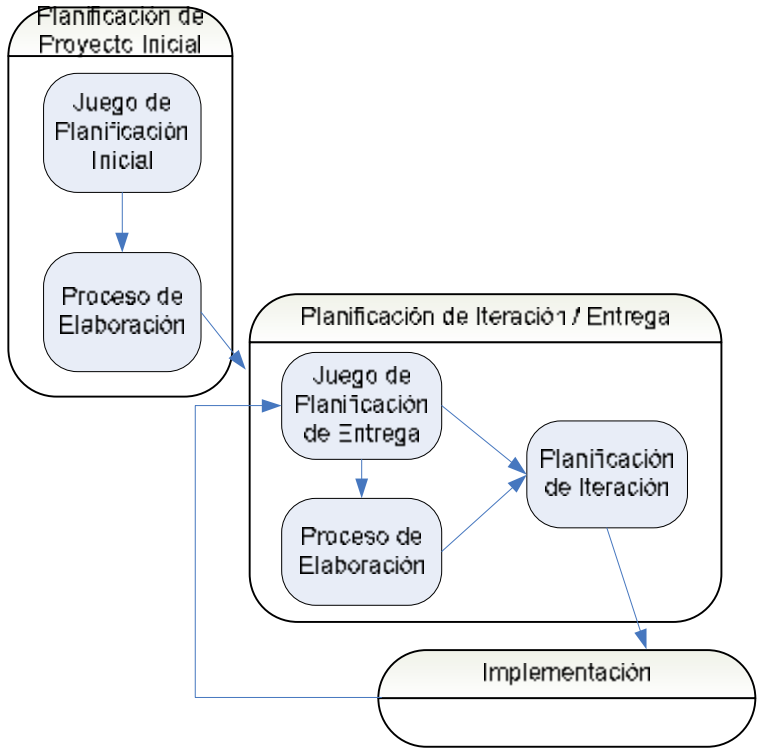
Planificación Dentro del Proceso XP

- Comienza con un proceso de **planificación inicial** del proyecto → esbozo de planificación global del proyecto.
- Le sigue una (o más de una) **planificación de entrega**, donde se planifican los contenidos y las tareas a implementar para la entrega.
- A continuación, se **implementa** la entrega
  - los resultados de la retroalimentación se utilizan como base de la planificación de la siguiente iteración.



Planificación y Juegos

- ¿Cómo utilizamos el juego de la planificación dentro del proyecto XP? ¿Cuando hacemos qué y cómo?
- Existe los juegos de planificación inicial y de entrega, existe otro paso, conocido como la fase/proceso de Elaboración.
  - Permite aclarar las historias de usuarios sin las restricciones del juego.
- Por lo tanto, un proyecto típico XP se puede planificar del siguiente modo:
  1. Un juego de planificación inicial
  2. Proceso de elaboración inicial
  3. Juego de Planificación de la Entrega 1
  4. Proceso de Elaboración de la Entrega 1
  5. Planificación de la iteración 1
  6. ..Iteración/Implementación de la Entrega 1..
  7. ..
  8. Juego de Planificación de la Entrega N
  9. Proceso de Elaboración de la Entrega N
  10. Planificación de la iteración N
  11. ...Iteración/Implementación de la Entrega N



Juego de Planificación Inicial

- Se centra en lo que tiene que realizar el sistema como lo haría un todo, como un producto → visión global
- Considera todas las historias de usuario que están dentro del alcance (y de hecho se define el alcance del proyecto).
- Se realiza al inicio del proyecto y se puede volver a convocar en ciertos momentos del proyecto para
  - revisar el alcance del sistema
  - el conjunto de historias de usuario
  - sus prioridades, etc...

Juego de Planificación de Entrega

- Se centra en los contenidos de una entrega o iteración.
- Se realiza igual que el juego de planificación inicial, pero el nivel de detalle necesario es mucho mayor.
- Para definir los detalles:
  1. Se desarrollan las historias de usuario y quizás necesiten dividirse en historias de usuario más pequeñas (para que puedan implementarse en una iteración).
  2. Se obtienen las estimaciones detalladas y las prioridades de las historias.
  3. Se confirman las historias de usuarios que se implementarán dentro de la entrega. También puede ser necesario la revisión o modificación de las historias que así lo requieran.
  4. Se revisa la velocidad del proyecto.
     - Conforme el equipo XP obtiene mayor experiencia y conocimiento de la aplicación, el equipo cada vez será más rápido.

Proceso de Elaboración

- Se realiza tras juego de planificación inicial y, a menor escala tras un juego de planificación de entrega.
- Se realiza un estudio de las historias para estimar, aclarar los requisitos, o cualquier aspecto técnico.
- Objetivos:
  - Minimizar los riesgos de una mala estimación.
  - Experimentar/prototipar las diferentes soluciones.
  - Mejorar la compresión del dominio/tecnología
  - Confirmar los procesos y procedimientos necesarios.
- ¿Duración?
  - Entre los juegos de planificación inicial y de entrega, puede durar desde un día, un mes o varios meses, dependiendo del nivel de experiencia de los equipos de desarrollo sobre el dominio a analizar, las tecnologías disponibles y los métodos a utilizar.
  - Entre las entregas, el proceso de elaboración es mucho más corto, normalmente unos pocos días.

Tamaño de una Iteración

- ¿Cómo determinar cual debería ser el tamaño de una iteración?
  - Una iteración tiene que ser suficientemente grande para permitir tanto crear una nueva entrega que añada valor al negocio, como ser capaz de realizar un progreso significativo.
  - Debe suficientemente pequeña para que se desarrolle demasiado sin que se realice una revisión
- La duración clásica varía **de 1 a 3 semanas**.
- Normalmente, los proyectos XP que son pequeños/ medianos involucran **de 2 a 6 desarrolladores**, como mucho 10.
  - Esto limita la cantidad de trabajo que se puede realizar para dentro de 2 o 3 semanas.
    - Si tenemos un equipo de 6 personas y la iteración dura 2 semanas → como mucho tenemos 12 personas/semana con las que contar.

Historias y Tareas

- Durante la planificación de la iteración, las historias de usuario se convierten en **tareas** que resultarán en la implementación de la historia.
- Una historia de usuario se puede implementar por una sola tarea, o por muchas tareas.
- Una tarea a su vez puede agrupar varias historias.
- Normalmente, una historia se implementa mediante una o más tareas.
- Algunas tareas puede que no estén relacionadas directamente con una historia:
  - migrar la aplicación a la última versión de Java.

Fases de la Planificación de la Iteración

1. Evaluación de las últimas iteraciones → retroalimentación
2. Revisar las historias de usuario a incorporar en la iteración, teniendo en cuenta:
   - historias de usuario no abordadas
   - velocidad del proyecto
   - pruebas de aceptación no superadas en la iteración anterior
   - tareas no terminadas en la iteración anterior.
3. Exploración de las tareas escritas para las historias de usuario. Estas tareas se puede dividir en tareas menores para ayudar a planificar y estimar.
   - Para dividir una historia, se necesita al menos a otro desarrollador (estilo programación en parejas) para ayudarle con el análisis.
   - Conforme progresa este paso, puede necesitar la ayuda del cliente o de otros desarrolladores.
4. Compromiso de las tareas durante la estimación de las mismas, carga de trabajo de los desarrolladores, etc...
5. Finalmente, se verifica y valida la planificación de la iteración.

Tareas y Programadores

- Los desarrolladores autoseleccionan las tareas, eligiendo primero aquellas tareas que tienen mayor prioridad y comiencen a desarrollarlas.
- Existen 2 enfoques posibles para esto:
  - **Elegir una tarea cada vez**
    - Un desarrollador elige una única tarea a implementar, y procede a su realización
    - Una vez acabada, seleccionará otra tarea y así hasta que no queden tareas pendientes o termine la iteración.
  - **Llena tu mochila**
    - Se realiza un proceso de selección de tareas de tipo *round robin*: cada desarrollador elige una o más tareas que le gustaría realizar.
      - En general, los desarrolladores eligen primero aquellas tareas que previamente habían identificado, analizado y estimado.
    - Este proceso continúa hasta que no quedan historias o hasta que la duración de las estimación coincide con la duración de la iteración.
    - Cada desarrollador conoce el conjunto de tareas que tiene asignado para la iteración actual.
    - Esto puede cambiar conforme progresa la iteración, ya que algunos desarrolladores pueden quedarse bloqueados con alguna tarea en particular mientras que otros se sienten agraciados y completan sus tareas más rápidamente.

Resumen del Proceso

- Al inicio del proyecto se define el alcance completo del sistema
  - y se esbozan las iteraciones y entregas.
- A continuación, el equipo elabora las historias de usuario
  - centrándose en aquellas que no saben estimar
- Tras esto, se planifica e implementa una iteración
  - se consideran en detalle las tareas que forman parte de cada historia de usuario.

Puntos a tratar

- Introducción
- Valores
- Resultados
- Prácticas
  - Juego de Planificación
- Proceso
- Roles
- Para Saber Más

Roles XP – Cliente

- Cliente
  - Escribe Historias de Usuario y especifica Pruebas de Aceptación.
  - Establece prioridades y explica/detalla las Historias
  - Decide las Historias a implementar dentro de una iteración
  - Puede ser o no un usuario final
  - Realiza funciones de interlocutor cuando representa a varias personas
  - Tiene autoridad para decidir cuestiones relativas a las Historias

- “Propietario de Oro”
  - La persona que paga el proyecto, que puede ser o no la misma que el Cliente

Roles XP – Desarrollo

- Programador
  - Realiza las estimaciones sobre las Historias
  - Identifica las Tareas a partir de las Historias y realiza estimaciones
  - Implementa las Historias y las Pruebas Unitarias
- Encargado de Pruebas (*Tester*)
  - Ayuda al Cliente a escribir las Pruebas de Aceptación
  - Implementa y ejecuta las Pruebas de Aceptación (¡no Pruebas Unitarias!)
  - Presenta gráficas de los resultados y se asegura de que la gente conoce cuándo los resultados empiezan a decaer.
- Consultor
  - Miembro externo al equipo con conocimientos específicos
  - Guía al equipo para resolver un problema específico

Roles XP – Gestión

- Entrenador / Tutor
  - Responsable del proceso
  - Observa todo, identifica señales de peligro, se asegura que el proyecto se mantiene en curso
  - Ayuda en todo
  - Da avisos cuando se necesita
- Encargado de Seguimiento / Perseguidor (*Tracker*)
  - Monitoriza el progreso de los programadores y toma acciones si las cosas tienden a salirse de su senda
    - si fallan las estimaciones, etc...
  - Las acciones incluyen reuniones con el Cliente, solicitar ayuda al Entrenador u otro Programador, ...

Roles XP – Gestión II

- Gestor (*Big Boss*)
  - Vinculo entre Cliente y Programadores
  - Planifica las reuniones, se asegura que el proceso de las reuniones se sigue, anota los resultados de la reunión para futuros informes y los pasa al Encargado de Seguimiento.
  - Posiblemente responsable ante el “Propietario de Oro”
  - Asiste a las reuniones, aporta información útil anterior.

Libros XP (XP Series)

Roadmap → Para Saber Más

- **Bibliografía**
  - ***Extreme Programmed Explained***, Addison-Wesley, de *Kent Beck*
    - ***Una explicación de la programación Extrema,*** Addison-Wesley, de *Kent Beck*

    (muy mala traducción – nada recomendable)
  - ***Agile Software Construction***, Springer, de *John Hunt*
  - XP Series http://www.informit.com/series/series.asp?st=44007&rl=1

- **Enlaces**
  - Wiki original: www.extremeprogramming.org
  - Wiki en castellano: www.programacionextrema.org
  - Portal de *Ron Jeffries*: www.xprogramming.com

¿Preguntas...?