
Tema 5 – eXtreme Programming (XP)





eXtreme Programming

- ✗ Metodología ágil desarrollada por Kent Beck. Aplicada por primera vez a un proyecto (C3 de Chrysler) de 1993-1997.

“Un método ligero, para equipos de desarrollo software de tamaño pequeño a mediano, que se enfrentan a requisitos vagos y que cambian rápidamente”

[Kent Beck]

- ✗ El desarrollo de software es un proceso adaptativo y orientado a las personas.

<http://www.extremeprogramming.org>



Características XP

- ✖ Diseñada para **entornos dinámicos** con requisitos cambiantes.
- ✖ Pensada para **equipos pequeños** (hasta 10 programadores).
- ✖ Orientado fuertemente hacia la **codificación y las pruebas**.
- ✖ Énfasis en la **comunicación informal, verbal**.
- ✖ Proceso **evolutivo, adaptativo pero disciplinado**.



¿Por qué extrema?

- ✗ Si las **revisiones de código** son buenas, vamos a revisar el código en todo momento (programación en parejas).
- ✗ Si las **pruebas** son buenas, ponemos a todo el mundo a probar todo el tiempo (prueba de unidad), también a los clientes (pruebas de aceptación).
- ✗ Si el **diseño** es bueno, vamos a hacer que sea parte del trabajo diario de todo el mundo (refactoring).
- ✗ Si la **simplicidad** es buena, vamos a dejar el sistema con el diseño más simple que de soporte a su funcionalidad actual (la cosa más simple que podría funcionar).
- ✗ Si la **arquitectura** es importante, todo el mundo trabajará en la definición y en el perfeccionamiento de la arquitectura durante todo el tiempo (arquitectura emergente).
- ✗ Si las **pruebas de integración** son importantes, vamos a integrar y probar varias veces al día (integración continua).
- ✗ Si las **iteraciones cortas** son buenas, vamos a hacer las iteraciones muy, muy cortas (construcción diaria, liberación semanal).



Las pruebas son básicas

- ✖ XP promueve un énfasis en las pruebas durante todo el proyecto.
- ✖ Todos los programadores escriben pruebas a la vez que codifican.
- ✖ Las pruebas se integran en un proceso de integración y construcción continuo (automático). Se genera una plataforma estable.
- ✖ El énfasis en las pruebas se ha exportado a otras metodologías (TDD, jUnit).

- ✖ ¡¡¡No hay requisitos !!!
- ✖ Renunciamos a los requisitos completos y estables. Adaptabilidad.
- ✖ Se describen usando las **Historias de Usuario**.
 - Trozos de funcionalidad que aportan valor al negocio.
 - Se les asignan **tareas de programación**.
 - Las establece el cliente (proviene de **conversaciones**).
 - Son la base para las **pruebas funcionales y de aceptación** por parte del cliente.

[Ver seminario 3...](#)



- ✗ En XP se plantea la planificación como un **diálogo continuo** entre todos los involucrados.
- ✗ Planificamos al inicio del proyecto, al inicio de cada iteración y todos los días durante el desarrollo.
- ✗ Diferente a la planificación tradicional:
 - Simplicidad del plan. (Es más importante el proceso de planificación que el plan creado).
 - Plan realizado por las personas que realizarán el trabajo.
 - No es una predicción de futuro sino la mejor estimación de cómo saldrán las cosas.

- Se planifican una serie de iteraciones y liberaciones del producto, a alto nivel, sin ser afectado por detalles que no tenemos claros al principio del proyecto.
- Mantenemos la agilidad al permitir cambios en los requisitos y en el plan durante todo el proyecto.
- Sirve como una guía, una línea base y esperamos que se actualice con la colaboración y la información obtenida del producto emergente.

- ✖ **Programador (Programmer)**
 - Responsable de decisiones técnicas.
 - Estimación de historias de usuario y tareas.
 - Sin distinción entre analistas, diseñadores o codificadores.
 - Diseñan, codifican y realizan las pruebas de unidad.
- ✖ **Cliente (Customer)**
 - Es parte del equipo.
 - Escribe las Historias de Usuario (HU) y las prioriza.
 - Escribe las pruebas funcionales.
- ✖ **Entrenador (Coach)**
 - El líder del equipo.
 - Responsable del proceso.
 - Tiende a estar en un segundo plano a medida que el equipo madura.

✗ Consultor

- Miembro externo al equipo con conocimiento específico en un tema necesario.

✗ Rastreador (Tracker)

- Seguimiento del proceso en cada iteración. Cálculo de las métricas del proyecto. Observa el proyecto sin interferir. Guarda el historial del proyecto.

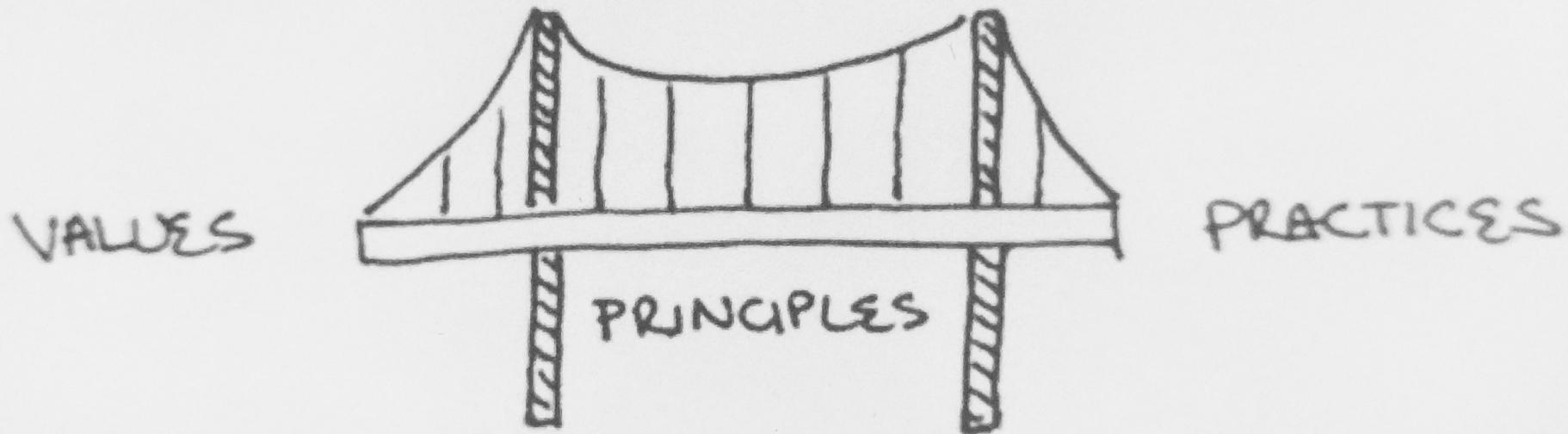
✗ Gestor (Big boss)

- Coordinador entre clientes y programadores.
- Ayuda a que el equipo trabaje eficientemente.

✗ Probador (Tester)

- Ayuda al cliente con las pruebas funcionales.
- Se asegura de que los tests funcionales se ejecutan y gestiona las herramientas usadas en las pruebas.

Valores, Principios y Prácticas





Valores: Comunicación

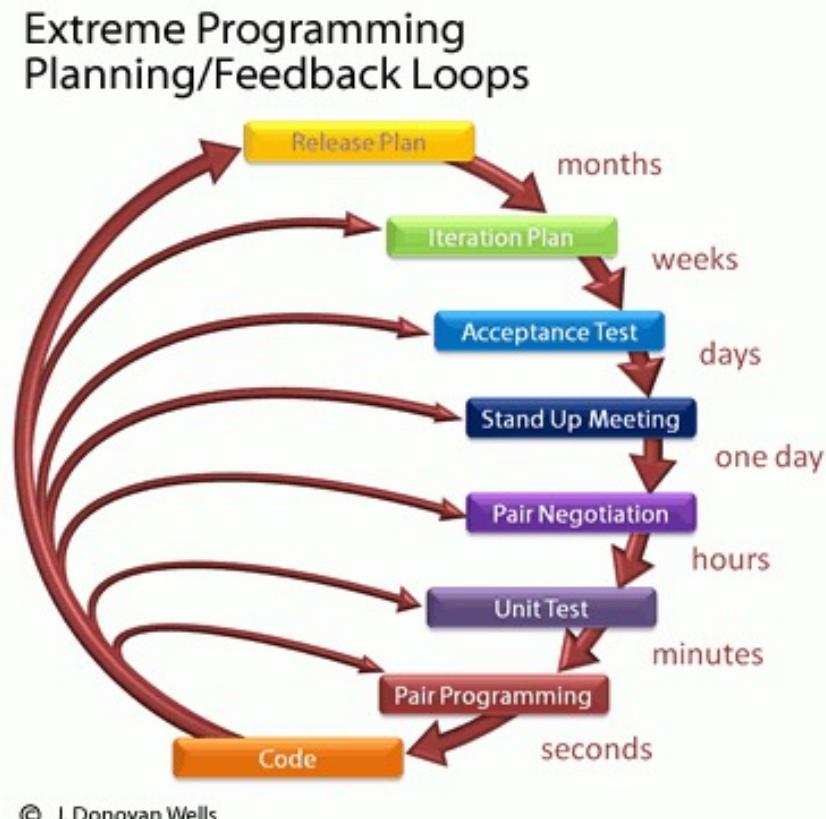
- ✗ La falta de comunicación impide que los conocimientos fluyan dentro de un equipo.
- ✗ Comunicación entre los miembros del equipo de desarrollo y con el cliente.
- ✗ XP ayuda a mantener el flujo de comunicación apropiado empleando muchas prácticas que no se pueden hacer sin comunicación.
- ✗ La comunicación es importante para crear un sentido de equipo y cooperación efectiva.

Valores: Simplicidad

- ✗ ¿Qué es lo más simple que funcionará?
- ✗ La simplicidad dice que siempre hay que hacer lo **más sencillo que funcione.**
- ✗ La sencillez no es fácil.
- ✗ Es el valor más intelectual y es contextual.
- ✗ La sencillez y la comunicación se apoyan mutuamente.

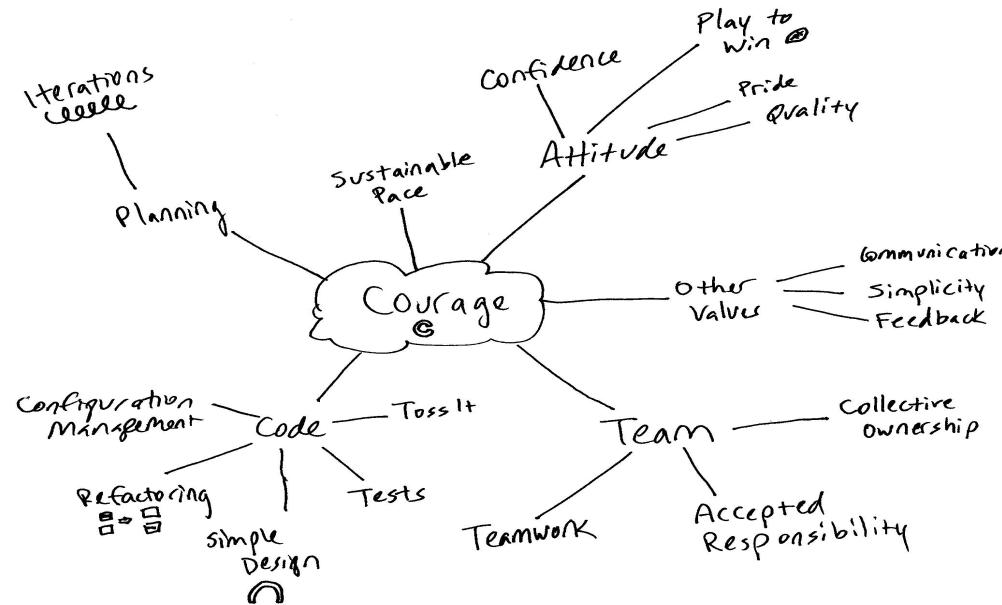
Valores: Realimentación

- × La realimentación continua permite un diseño simple.
- × La realimentación se produce a diferentes escalas de tiempo.



Valores: Valentía

- “Una acción efectiva frente al miedo”



W. Wake, 9-18-02

Valores: Respeto

- ✗ “El respeto es un valor que está bajo la superficie de los otros cuatro”.
- ✗ Los miembros de un equipo deben respetarse mutuamente, comunicarse entre sí, proporcionar y aceptar comentarios que beneficien su relación, y trabajar juntos para identificar diseños y soluciones simples.

Principios

Humanidad	Economía	Beneficio Mutuo	Auto-similitud
Mejora	Diversidad	Reflexión	Flujo
Oportunidad	Redundancia	Fallo	Calidad
Pasos de bebe	Responsabilidad		

- × **Humanidad**: Los factores humanos son la clave para crear un software de calidad.
- × **Economía**: El producto que se cree debe ser rentable y producir beneficios.
- × **Beneficio mutuo**: Pensar siempre en el beneficio de todas las partes implicadas en un desarrollo.
- × **Auto-similitud**: Soluciones similares en diferentes contextos.
- × **Mejora**: Perfeccionar en el desarrollo.
- × **Diversidad**: Distintas ideas, perspectivas, habilidades y actitudes presentan oportunidades más que conflictos.

- × **Reflexión:** Los buenos equipos reflexionan cómo y por qué hacen el trabajo.
- × **Flujo:** Entrega continua de valor.
- × **Oportunidad:** Cada problema es una oportunidad de mejora.
- × **Redundancia:** Los problemas críticos se tienen que resolver de diferentes formas.
- × **Fallo:** Un fracaso no es un desperdicio si me sirve para aprender algo.

- ✗ **Calidad:** Rebajar la calidad no implica un desarrollo más rápido.
- ✗ **Pasos de bebe:** Realizar los cambios en pequeños pasos, a todos los niveles.
- ✗ **Responsabilidad:** La responsabilidad debe ser aceptada, nunca asignada.

✗ Valores (5).

Comunicación, Simplicidad, Realimentación, Valentía (Coraje), Respeto.

✗ Principios (14).

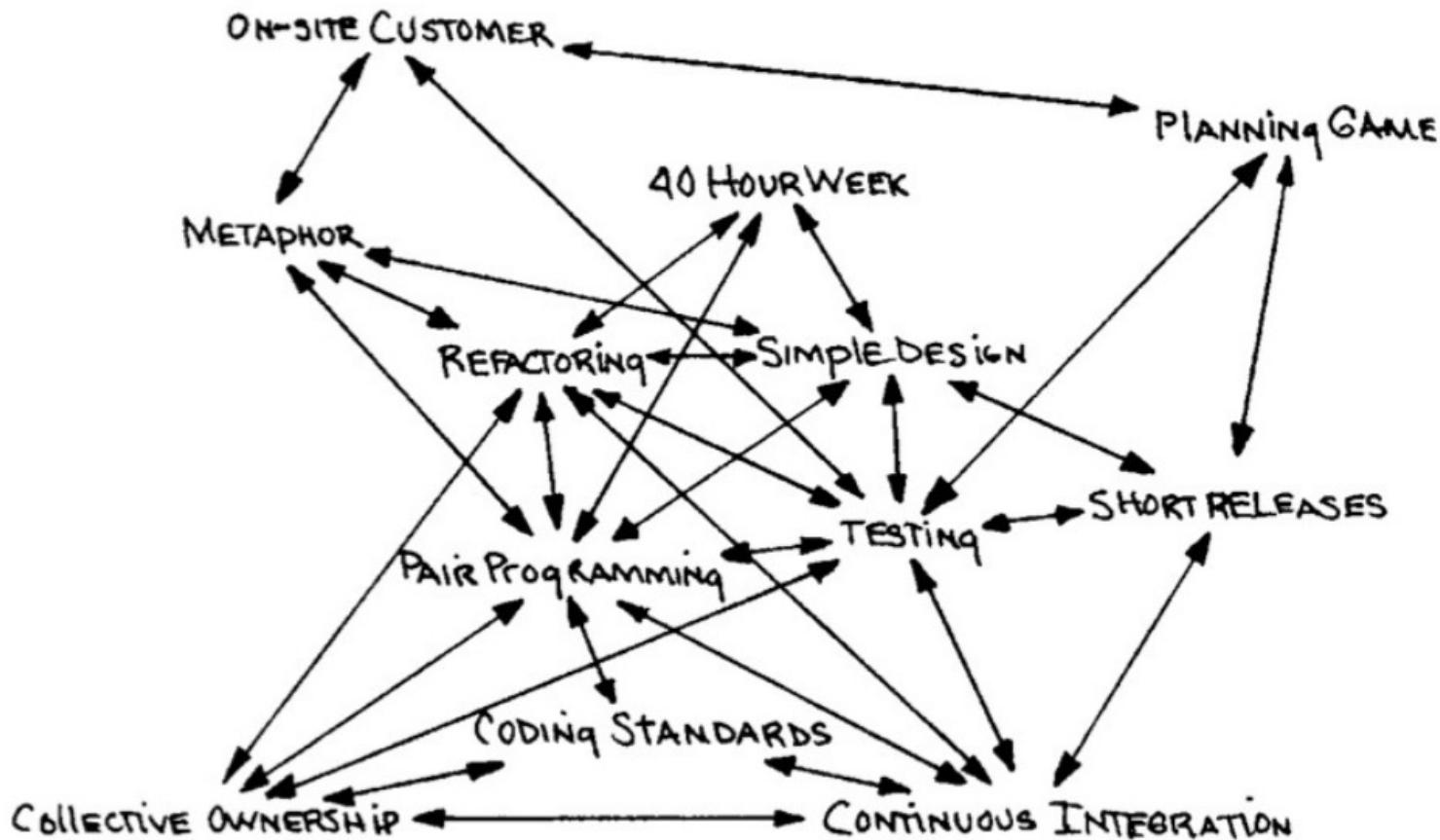
Humanidad, Economía, Beneficio Mutuo, Auto-similitud, Mejora, Diversidad, Reflexión, Flujo, Oportunidad, Redundancia, Fallo, Calidad, Pasos de bebe, Responsabilidad.

✗ Prácticas

– XP1 (12) y XP2 (13+11).

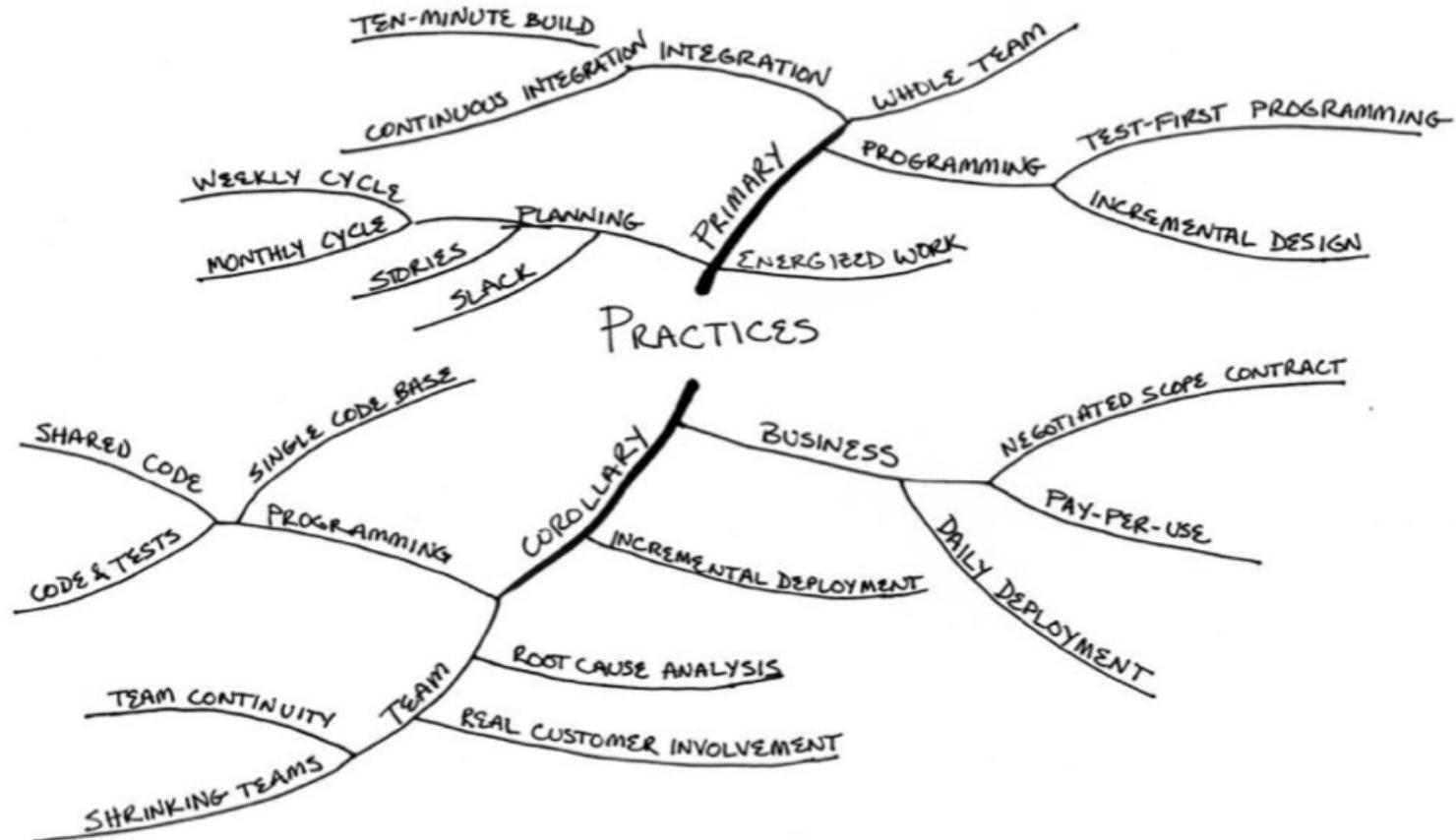


Prácticas XP1(2000)



Kent Beck, Extreme Programming Explained, Primera edición 2000

Prácticas XP2 (2005)



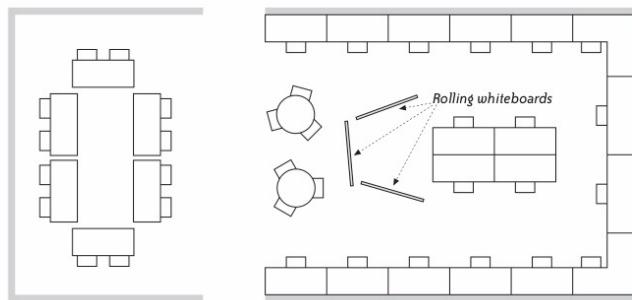
Kent Beck, *Extreme Programming Explained*, Segunda edición, 2005

- 2 Sentarse juntos
- 2 Sensación de equipo
- 2 Espacios de trabajo informativos
- 2 Trabajo a pleno rendimiento (40 horas por semana)
- 1,2 Programación en parejas
- 2 Historias de usuario
- 2 Ciclo semanal (Juego de planificación)
- 2 Ciclo trimestral (Lanzamientos cortos)
- 2 Aflojar
- 2 Construcción en 10 minutos
- 1,2 Integración continua
- 1,2 Pruebas antes de implementar
- 2 Diseño incremental (refactoring)
- 1 Metáfora
- 1 Propiedad colectiva del código
- 1 Cliente in situ
- 1 Estándares de programación

1. Sit Together (Sentarse juntos)

XP2

- ✗ Forma del lugar de trabajo. Espacio abierto donde trabajan todos.
- ✗ Espacio privado limitado.
- ✗ El espacio favorece la comunicación.
- ✗ Cuantas más reuniones cara a cara mejor.





2. Whole Team (Sensación de equipo)

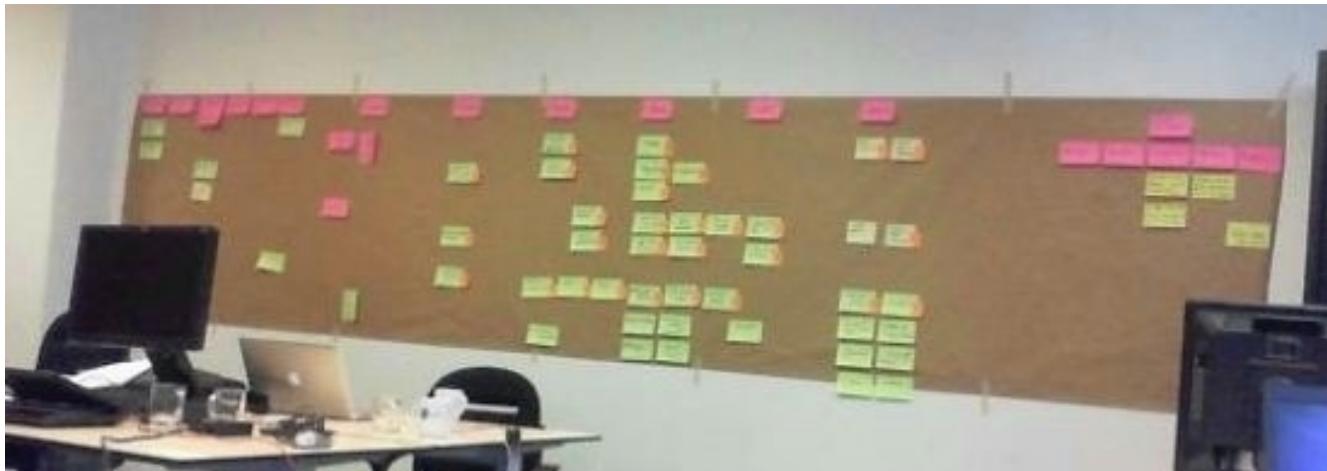
XP2

- ✖ Todos somos el equipo, estamos juntos, apoyamos el crecimiento de los demás.
- ✖ Equipos multifuncionales.
- ✖ Equipos dinámicos (Si el diseñador de IU o el administrador de las bases de datos terminan su trabajo pueden ayudar a otra actividad del equipo).
- ✖ Disponemos de todos los recursos a nivel de personal para tener éxito.

3. Informative Workspace (Espacios de trabajo informativos)

XP2

- ✗ El espacio de trabajo mantiene información sobre el proyecto y su estado.
- ✗ Colocar las HU en las paredes y agrupadas por su estado.



4. Energized Work (Trabajo a pleno rendimiento)

XP2

- ✖ Trabajar a pleno rendimiento pero solo el tiempo en el que seamos productivos.
- ✖ Es posible ser más productivo gestionando mejor el tiempo.
- ✖ Desarrolladores frescos tienen mejores ideas.

40 Horas por Semana

XP1

- ✖ Paso sostenible: Encontrar un ritmo de trabajo para el equipo de desarrollo donde todos los miembros se sientan cómodos.
- ✖ Si tenemos que romper esta regla es que algo no funciona, y la solución no tiene porque ser trabajar más tiempo.

5. Programación en Parejas

XP1 XP2

- ✗ La programación en parejas es un **diálogo continuo** entre dos personas que intentan simultáneamente programar, además de analizar, diseñar, probar y comprender juntos como programar mejor.
- ✗ Las parejas de desarrolladores se intercambian con cierta frecuencia.

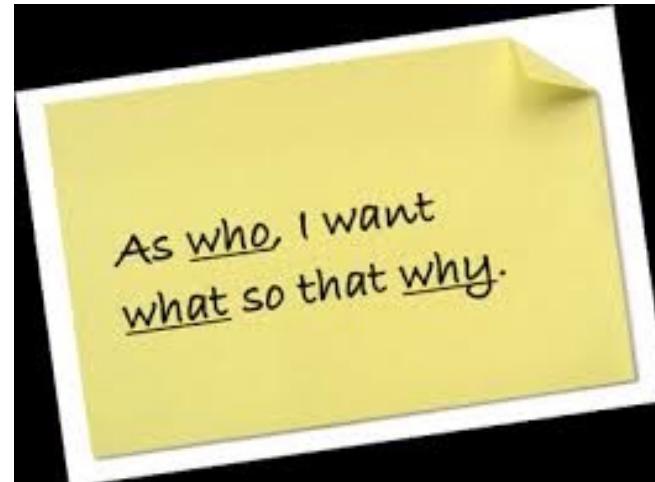
5. Programación en Parejas (2)

- ✖ **Economía:** Al agregar una persona al equipo se reflejará un pequeño aumento en el costo de desarrollo. Sin embargo, el código resultante debe tener menos defectos. Los ahorros al eliminar los defectos deberían ser mayores que el aumento en el costo de desarrollo.
- ✖ **Satisfacción.** La experiencia de programar es más agradable que el trabajar de forma individual. Disfrutan del trabajo.
- ✖ **Diseño de calidad.** Producen programas más cortos y con diseños mejores.
- ✖ **Revisiones continuas.** La técnica del codo a codo de la programación por pares sirve como un continuo diseño y revisión del código, haciendo más eficaz la búsqueda de defectos.
- ✖ **Aprendiendo.** Se aprende más del sistema y de la programación en general.
- ✖ **Comunicación del equipo.** Las personas aprenden a discutir y a trabajar juntos. Esto mejora la comunicación del equipo y por consiguiente su efectividad.
- ✖ **Personal y Dirección del Proyecto.** Puesto que múltiples personas tienen familiaridad con cada trozo de código, la programación por pares reduce el riesgo de pérdida de personal.

6. Historias de Usuario

XP2

- ✗ Es más adecuado hablar de historias que de requisitos.
- ✗ Colocarlas en un lugar visible y usarlas para comunicarnos.





7. Weekly Cycle (Ciclo Semanal)

XP1 XP2

Trabajo para cada semana:

Juego de la planificación

1. Revisar el progreso hasta la fecha, analizando como puede afectar al progreso de esta semana.
 2. Hacer que los clientes escogen que HU vamos a poner en valor esta semana y van a ser implementadas.
 3. Dividir cada HU en tareas. Los miembros del equipo asignarán y estimarán su esfuerzo.
- ✗ Comenzar la semana escribiendo pruebas para las HU.
 - ✗ Al final de la semana las HU están disponibles para ser lanzadas.
 - ✗ La semana es un ciclo corto y permite experimentar.

8. Quarterly Cycle (Ciclo trimestral)

XP1 XP2

Lanzamientos Cortos

- ✗ Hacer reuniones con un mayor ciclo para ver el estado del proyecto y objetivos grandes.

Trabajo para cada ciclo trimestral:

1. Identificar cuellos de botella.
 2. Identificar tema o temas para el ciclo.
 3. Seleccionar HU dentro de los temas escogidos.
 4. Usar una visión general de cómo el proyecto encaja en la organización y del valor que añade.
- ✗ Podemos reflexionar sobre el equipo y plantearnos experimentos más grandes.

- ✖ Reducir los compromisos imposibles.
- ✖ Introducir un colchón que reduzca tensiones.
- ✖ Trabajar con plazos realistas.
- ✖ Confianza Cliente/Equipo. Los compromisos se cumplen y hay una comunicación clara y honesta.

10. Ten minute build

XP2

- ✖ 10 minutos para construir todo el sistema y ejecutar todos los test (... automatizar el proceso).
- ✖ Comprobar lo añadido y lo anterior.
- ✖ Podemos automatizar el despliegue y el lanzamiento de las nuevas funcionalidades en producción.

11. Integración Continua

XP1 XP2

- ✖ Éxito de un proyecto de software = velocidad con la cuál el sistema se adapta a los requerimientos cambiantes + velocidad con la cuál se le puede agregar una nueva funcionalidad.
- ✖ Repositorios de código donde los desarrolladores deben integrar y enviar el código cada pocas horas, procurando no postergar esta tarea más de un día (proceso disciplinado y automatizado).

11. Integración Continua (2)

- ✖ Los elementos de la práctica de Integración Continua:
 - Lo mejor es más frecuente.
 - Construcción exitosa.
 - Un único repositorio de código fuente.
 - Scripts de automatización de construcciones.
 - Código de prueba automatizada.
 - La construcción maestra.
 - Revisión.



12. Pruebas antes de implementar

XP1 XP2

- ✖ Escribir las pruebas antes que el código.
- ✖ Pruebas incrementales en el desarrollo partiendo de los escenarios de uso.
- ✖ El usuario forma parte de las pruebas de desarrollo y en la validación.
- ✖ Automatizar las pruebas y realizar todas las pruebas cada vez que se genera una nueva versión del producto.

13. Diseño Incremental

XP1 XP2

Refactoring

- ✖ Diseño incremental y con paso firme.
- ✖ No solo se hace al principio (intentamos mejorar el diseño todo el tiempo).
- ✖ Pensar en el diseño de todo el sistema.
- ✖ Objetivo. El coste de cambiar algo debe ser bajo.
- ✖ Diseñar la cosa más **simple** que pueda funcionar.



Refactoring

XP1

“El refactoring es realizar modificaciones en el código con el objetivo de mejorar su estructura interna, sin alterar su comportamiento externo”

- ✗ ¿Existe una forma de cambiar el programa existente para añadir la característica de una forma más simple?
- ✗ Para aplicar la técnica del refactoring es imprescindible que existan códigos de prueba, que permita al equipo saber en cualquier momento si el desarrollo sigue cumpliendo con los requisitos que implementaba.

Refactoring (2)

Indicio	Síntoma	Refactoring mas común
Código Duplicado	Es el principal y más común de los errores. Si se encuentra con la misma estructura de código en más de un lugar será mejor si se encuentra una manera de unificarlos.	Extract Method.
	Si se encuentra la misma expresión en dos subclases, debe ser eliminado con el método previo y posteriormente aplicar el nuevo.	Pull Up Method
Métodos Largos	Las aplicaciones con métodos más cortos tienen una mayor vida. Los comentarios muchas veces son indicadores de una distancia semántica, se puede reemplazar el comentario por un método cuyo nombre tenga el mismo significado.	Extraction Method, Replace temp. with Query, Descompose Conditional.

Refactoring (3)

Indicío	Síntoma	Refactoring mas común
Clase Grande	Se encuentra en aquellas aplicaciones donde una clase intenta hacer demasiado, lo que como indicio presenta un alto número de variables. EL método utilizado intenta disminuir este número.	Extract Class.
Lista de Parámetros Larga	Una larga lista de parámetros es difícil de entender, corregir o alterar. No siempre hay que pasar toda la información al método, a veces se puede preguntar a otro objeto.	Replace Parameter with Method, Preserve Whole, <u>Introduce Parameter Object</u> .

- ✗ Una metáfora es algo que todos entienden, sin necesidad de mayores explicaciones.
- ✗ Utilizar una metáfora como una forma sencilla de explicar el propósito del proyecto, y guiar la estructura y la arquitectura del mismo.
- ✗ Debe ser comprensible para el cliente y tener información para que sirva de guía a la arquitectura del proyecto.
- ✗ Como metáfora podríamos usar un **modelo conceptual** (UML) con los elementos básicos y sus relaciones.

Propiedad colectiva del código

XP1

- ✗ Cualquier miembro del equipo desarrollador podrá cambiar una línea de código para añadir funcionalidad o eliminar algún tipo de error.
- ✗ El objetivo fundamental es reducir al mínimo las obstrucciones para distribuir e implementar rápidamente las tareas de programación.
- ✗ Problema al mantener bajo control el código (comunicación, pruebas unitarias, integración frecuente).
- ✗ El conocimiento de la arquitectura está distribuido entre los miembros del equipo de desarrollo.
- ✗ El código malo o erróneo será detectado tarde o temprano por alguien y a los programadores les cuesta más trabajo introducirlo de primeras.



Propiedad colectiva del código (2)

Ventajas y beneficios:

- Ayuda a la integración, confiabilidad y creatividad en los equipos de trabajo.
- Requiere y se esfuerza por un estilo y filosofía consistente para todo el sistema.
- Maneja fácilmente el crecimiento y la contracción de los equipos.
- Responde rápidamente a cambios o incrementos en los requerimientos.
- Tiende a prevenir código complejo en la primera parte de construcción del sistema.

Desventajas:

- La responsabilidad y los roles de las tareas pueden ser difícil de establecer.
- El sistema podría perder una arquitectura común, y su propósito inicial podría ser difícil de alcanzar.
- Existen posibilidades de realizar cambios repetidos de acuerdo a diferentes estilos personales de programación.

- ✖ Tener al usuario siempre disponible, no solo para ayudar al equipo de desarrollo, sino formando parte de él.
- ✖ Es el usuario o cliente quien tomá las decisiones que le afecten para alcanzar los objetivos del negocio.
- ✖ Tiene un papel muy importante en los procesos de planificación.

Cliente in-situ (2)

Actividades que desarrolla el Cliente

Actividad	Práctica	Herramienta
Escribir Historias de Usuario	El Juego de la Planificación	Implementación de Historias de Usuario.
Priorizar Historias de Usuario	El Juego de la Planificación	Procedimiento de priorización de Historias de Usuario.
Fijar el alcance de la entrega	El Juego de la Planificación	Guía para Planificación por Alcance o Tiempo.
Agrupar o dividir historias de usuario.	El Juego de la Planificación	Implementación de Historias de Usuario.
Colaborar en el Plan de Entregas	El Juego de la Planificación	Plan de Entregas.
Verificar las pequeñas liberaciones	Pequeñas Liberaciones	Checklist de Percepción.
Definir y ejecutar las pruebas de aceptación.	Testing	Desarrollo e Implementación de las Pruebas de Aceptación.
Verificar que las pruebas son ejecutadas correctamente	Testing	Checklist de Aceptación de Pruebas.
Aceptar la entrega final del proyecto	Cliente siempre Disponible	Checklist de Aceptación de la Entrega del Sistema.

Estándares de programación

XP1

- ✖ El código debe ser desarrollado siguiendo estándares de desarrollo para facilitar su lectura y la modificación por cualquier miembro del equipo de desarrollo.
- ✖ Todo el mundo se sienta cómodo con el código escrito por cualquier otro miembro del equipo.

Equipo

14. Participación de los clientes reales.
15. Continuidad del equipo.
16. Reducir el tamaño del Equipo.

Programación

17. Código Compartido.
18. Código y pruebas.
19. Un sólo código base.

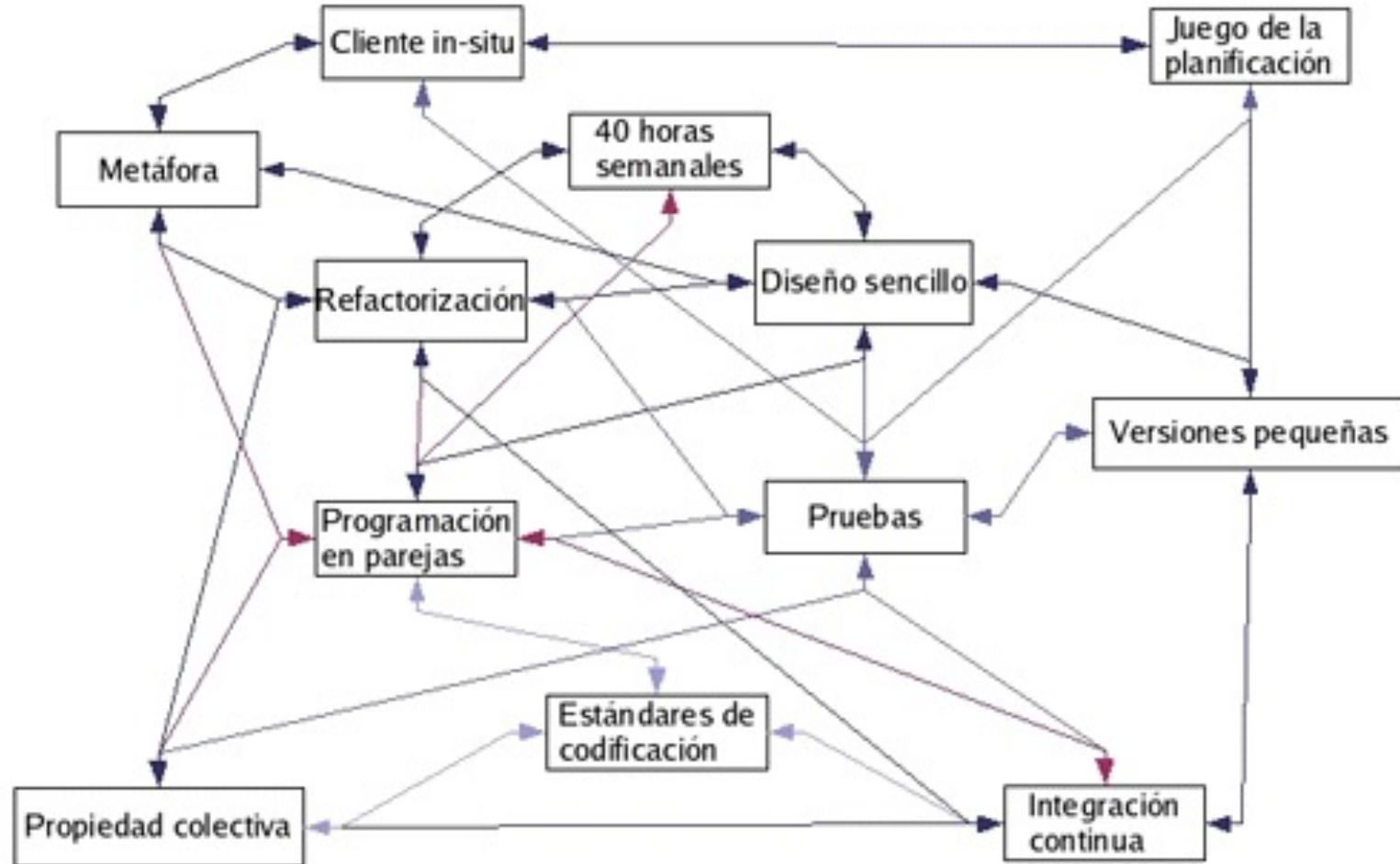
Negocio

20. Eliminar errores y su causa.
21. Negociar contratos por alcance.
22. Conectar pagos y desarrollo.

Entregas

23. Despliegue incremental.
24. Despliegue diario.

Interacción entre prácticas



XP2 Primary Practice	Sustained/New/ XP1 Name
Sit together	New
Whole team	New
Informative workspace	New
Energized work	40-hour week
Pair programming	Sustained
Stories	Planning game
Weekly cycle	Planning game
Quarterly cycle	Small releases
Slack	New
Ten-minute build	New
Continuous integration	Sustained
Test-first Programming	Testing
Incremental Design	Simple Design Refactoring

XP1 Practice	Disposition
Metaphor	Removed
Collective code ownership	Corollary: Shared code
On-site customer	Corollary: Real customer involvement
Coding standard	Removed

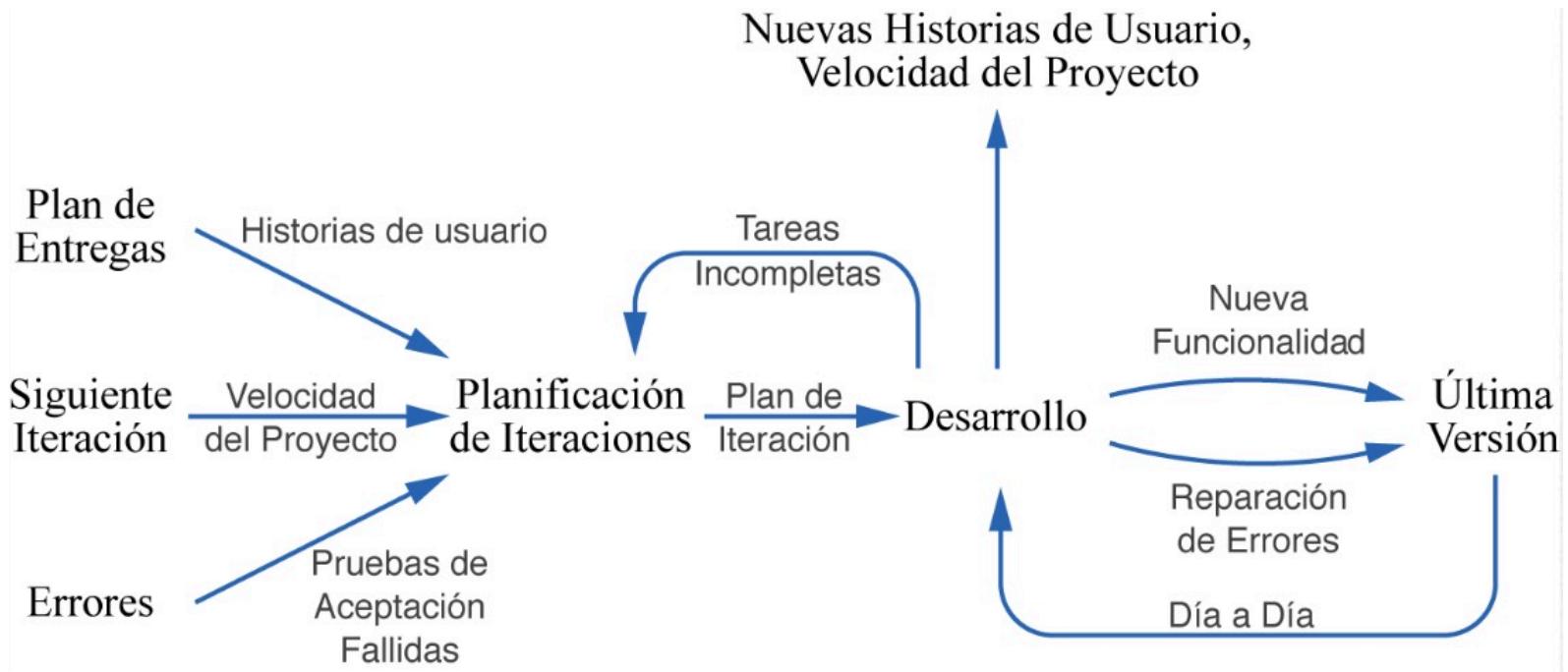


El ciclo de desarrollo

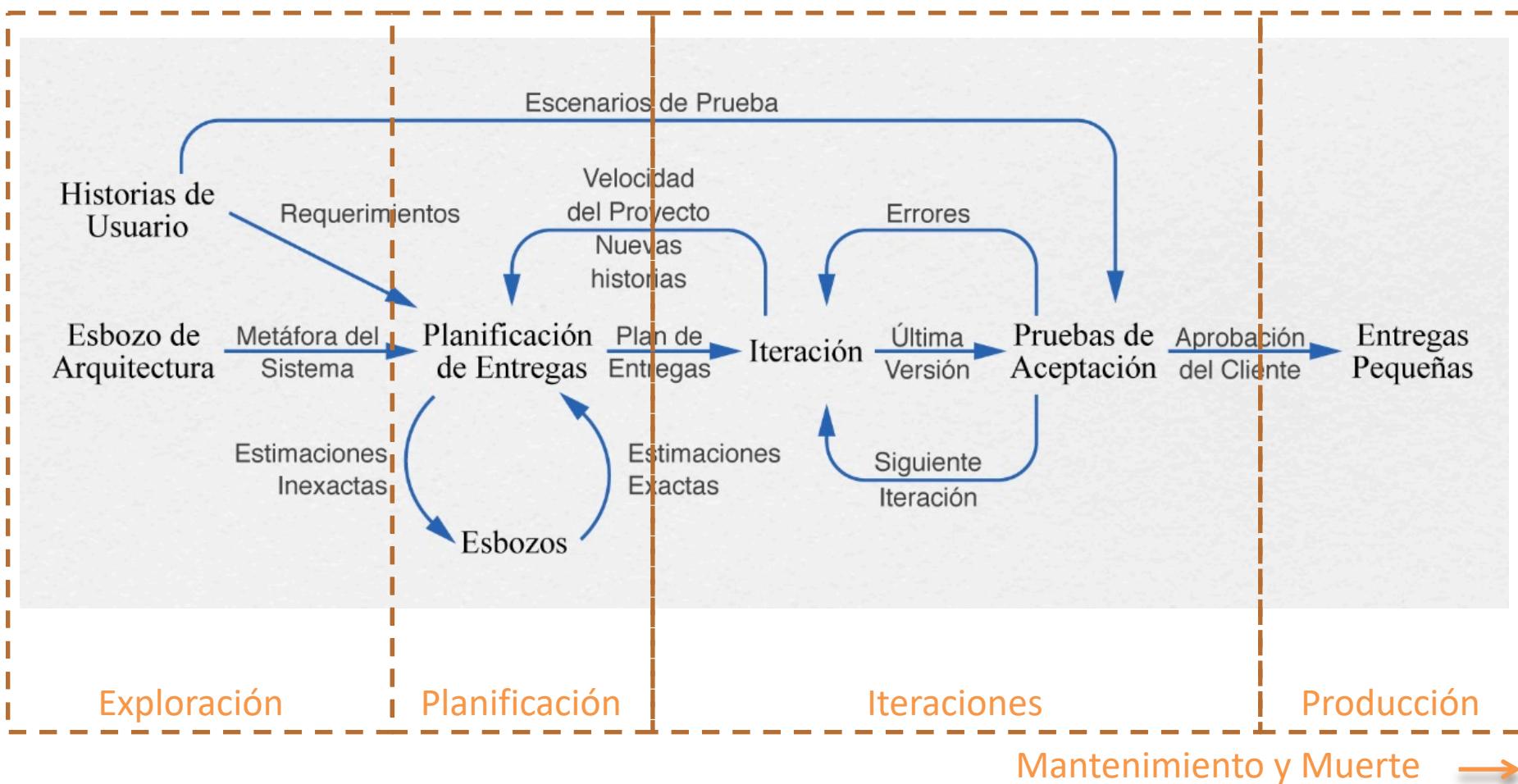
1. El cliente define el valor de negocio a implementar (HU).
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con las prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio descomponiendo las HU en Tareas.
5. Vuelve al paso 1.

El ciclo de desarrollo...

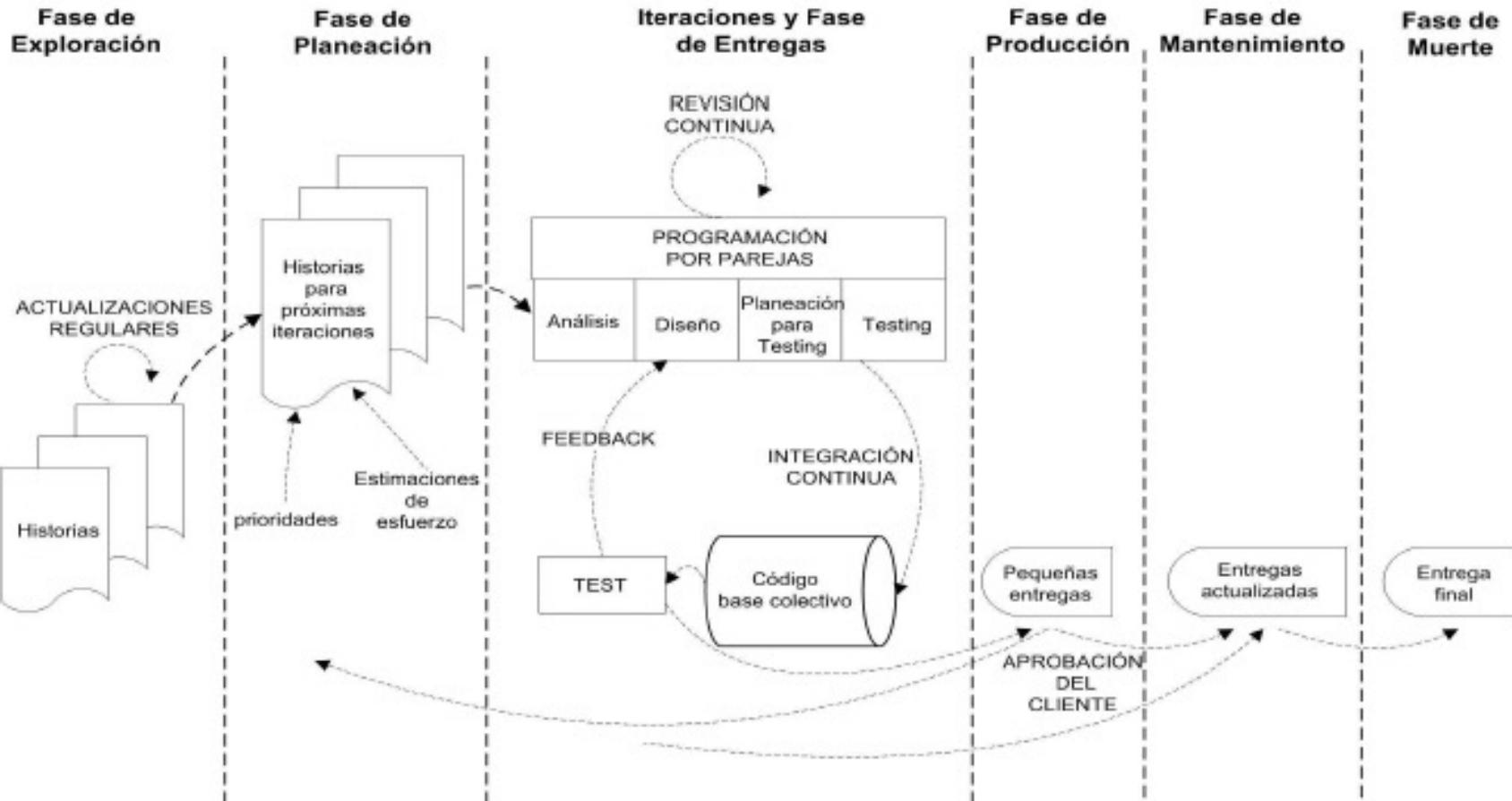
Es un proceso iterativo ...



XP. Ciclo de vida. Fases



XP. Ciclo de vida



FASE 1. Exploración

- Obtener las HU de interés para las primeras entregas. (Brainstorming)
- Probar tecnología y arquitectura del sistema (¿Prototipo?)
- El equipo de desarrollo se familiariza con la tecnología, prácticas y herramientas a usar en el proyecto.
- Duración: Pocas semanas – Pocos meses

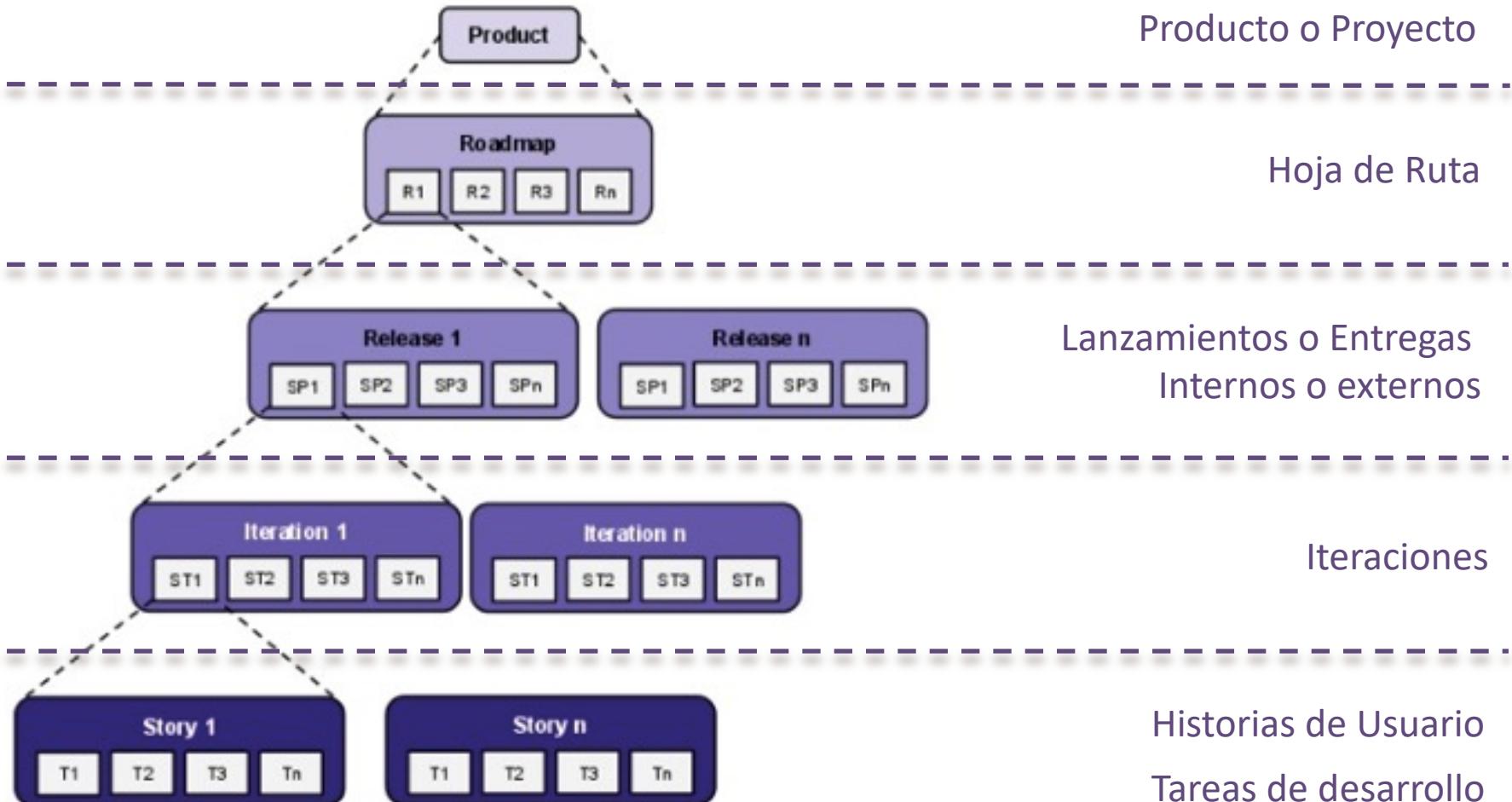
Metáfora del sistema

- Definimos el sistema con un conjunto de metáforas compartidas por el cliente y los desarrolladores.
- Seleccionamos las HU más características del sistema (las que fuerzan a construir la arquitectura inicial del sistema).
- Ayuda a poder hablar sobre el sistema con una nomenclatura común. (Podríamos usar un diagrama de conceptos e incluso un diagrama de clases del diseño inicial).

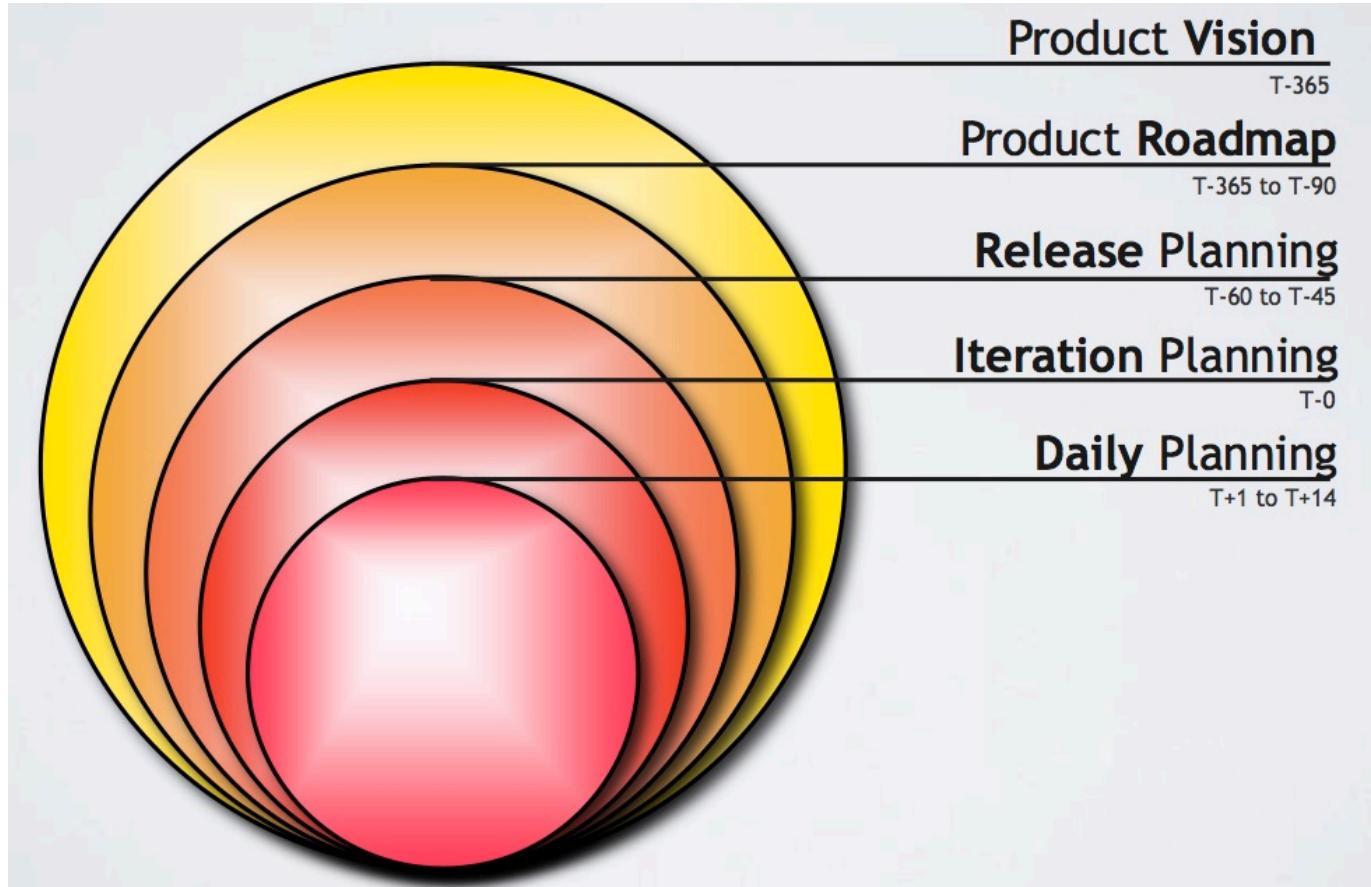
FASE 2. Planificación de Entregas

- El cliente prioriza las HU.
- Los programadores estiman el esfuerzo.
- Se acuerda el contenido de la primera entrega y un cronograma.
- Duración: Unos pocos días.

Estructura de un proyecto



Niveles de planificación



Reuniones de planificación

Release Planning Meeting
Iteration Planning Meeting
Daily Planning Meeting





Reuniones de planificación (2)

Iteration 1

As a frequent flyer, I want to...	3
As a frequent flyer, I want to...	5
As a frequent flyer, I want to...	5
As a frequent flyer, I want to...	2
As a frequent flyer, I want to...	2

Iteration 2

Code the UI	8
Write test fixture	6
Code middle tier	12
Write tests	5
Automate tests	4

“Yesterday I started
on the UI; I should
finish before the end
of today.”



© Mountain Goat Software, LLC

- Podemos describir el futuro

en términos de las características deseadas del producto, los clientes, objetivos y las claves diferenciadoras con otros productos o formas de hacer las cosas.

Planificando las entregas “Release planning meeting”

- ✖ Realizar entregas del producto en **no más de 3 meses**.
- ✖ El equipo toma decisiones técnicas y el cliente de negocio. (Discutir y debatir).
- ✖ El cliente decide que HU son más importantes o prioritarias para el negocio.
- ✖ Se realiza la estimación del esfuerzo de cada una de las HU por parte del equipo de desarrollo.

Evolución del Producto

- ✖ Al final de cada **iteración** tenemos un producto probado, funcionando y listo para entrar en producción que puede ser usado como demostración para el usuario.
- ✖ Después de cada **lanzamiento/entrega** el cliente evalúa el producto y genera una realimentación al equipo de desarrollo.
- ✖ El cliente decidirá si quiere ponerlo en **producción**.

Enfocando la planificación

- ✗ **Planificación por tiempo:** Se calcula el tiempo que queda hasta una fecha límite, y se decide cuántas iteraciones se van a realizar teniendo en cuenta que se recomienda que cada iteración dure alrededor de tres semanas.
- ✗ **Planificación por alcance:** Se determina las HU que deben ser implementadas en una entrega. Se suma el número de semanas ideales para esas historias, y se divide entre la velocidad del proyecto para determinar el número de iteraciones.
- ✗ Se debe procurar que las historias de usuario más importantes se implementen en las primeras entregas del proyecto.

Plan de entregas

- ✗ Documento que describe que entregas se realizarán del producto, sus objetivos, fechas y las HU que serán implementadas.
- ✗ Debe ser negociado y elaborado en forma conjunta entre el cliente y el equipo de desarrollo en las reuniones de planificación de entregas.

[Ver ejemplo y formato del documento ...](#)



Planificando la iteración “Iteration planning meeting”

- ✖ Hacemos iteraciones de 1 a 3 semanas de duración.
- ✖ Al principio de cada iteración hacemos la reunión para planificar las tareas a realizar (planificación “just-in-time”).
- ✖ Si hay problemas de planificación se repite la reunión, se reestima y se eliminan tareas si es necesario.



Planificando la iteración “Iteration planning meeting”

Partimos de las HU priorizadas en el plan de entregas y de las HU que fallaron en las pruebas de aceptación.

El cliente selecciona HU en base a la *velocidad del equipo de desarrollo*.

1. Descomponer la HU en tareas de programación individuales.
2. Asumir la realización de una tarea por parte de un desarrollador y estimar la duración de la tarea.

Pueden aparecer nuevas HU y tareas de programación postergadas a otra entrega.

Reuniones diarias de seguimiento

- ✗ El objetivo es mantener la comunicación y la colaboración entre el equipo y compartir problemas y soluciones.
- ✗ Scrum Daily Meeting, Stand-up Meeting.



FASE 3. Iteraciones

- Se incluyen varias iteraciones antes de cada entrega.
- Primera iteración. Arquitectura del sistema. ¿HU de más valor para el cliente?
- Se planifica cada iteración (“Iteration planning meeting”).
- Duración: No más de 3 semanas por iteración.

Estrategias de diseño

“Hay que diseñar hoy para los problemas de hoy y mañana para los problemas de mañana”

1. Comenzar con una prueba. Se tiene que hacer una cantidad de diseño mínimo para poder escribir la prueba, y señalar cuáles son los objetos y los métodos visibles.

2. Diseñar e implementar justo lo suficiente para conseguir que la prueba funcione.

Repetir los pasos 1 y 2.

✗ Buscar hacer siempre el diseño más simple.

FASE 4. Producción

- Pruebas adicionales y revisiones de rendimiento antes de trasladar el sistema al entorno del cliente.
- Decisión de inclusión de nuevas características.
- Duración: 1 semana por iteración.

FASE 5. Mantenimiento

- La primera versión está en producción, hay que mantenerla en funcionamiento.
- Requiere esfuerzo del equipo para mantener el sistema a la vez que se trabaja en otras entregas.
- Duración: Lo que duren las otras entregas.

FASE 6. Muerte del proyecto

- El cliente ya no tiene más HU para incluir en el sistema.
 - Generar la documentación final del sistema.
 - Puede morir por decisión del cliente.
-
- Duración: Sin duración determinada.

- ✗ Beck, K.. "***Extreme Programming Explained. Embrace Change***", Pearson Education, 1999. Traducido al español como: "***Una explicación de la programación extrema. Aceptar el cambio***", Addison Wesley, 2000.
- ✗ Beck, K., Cynthia Andres, "***Extreme Programming Explained: Embrace Change, 2nd Edition*** (The XP Series) , Addison Wesley, 2004.

Lectura recomendada:

- ✗ M. Marchesi, “The New XP” “[TheNewXP.pdf](#)”