Programa de Doctorado
Ingeniería de Sistemas Electrónicos para Entornos Inteligentes

Asignatura
Ingeniería Software para Sistemas Empotrados

Metodología de Desarrollo (2): Programación Extrema

C. López Barrio

29/11/05

Índice

Procesos ligeros y pesados

Pesadas
Ej.: V-Process

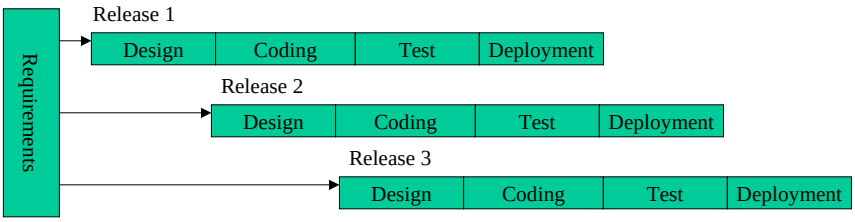
Entorno a la Medida
Ej.: Rational Unified Process (RUP)

Ágiles (Ligeras)
Ej.: eXtreme Programming (XP)

Basado en documentos
Elaborar definiciones Flujo Trabajo
Muchos roles diferentes
Muchos puntos de control
Alto sobrecoste de gestión
Mucha burocracia

Foco en el código sobre el que se trabaja más que en la documentación.
Foco en la comunicación directa (entre desarrolladores y entre desarrolladores y el cliente)
Bajo sobrecoste de gestión.

Incremental



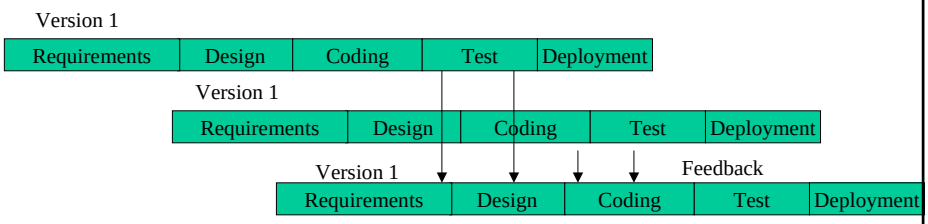
Cada lanzamiento añade más funcionalidad, es decir, un nuevo incremento

(Algunos lo denominan iterativo)

Modelo Incremental (cont.)

- En los modelos en cascada y prototipado rápido
  - Calidad operativa completa con el producto final
- En el modelo incremental
  - Calidad operativa de una parte del producto en un plazo de unas semanas
- Menos traumático
- Menor inversión, rápido retorno de la misma

Evolutivo



Nuevas versiones implementan requisitos *nuevos* y evolutivos

(Algunos lo denominan iterativo)

Modelo evolutivo (cont.)

- Ventajas
  - Involucración y validación constante del cliente
  - Permite una buena gestión del riesgo
- Desventajas
  - Peligro de “Construir-y-arreglar”

Otros tipos de metodologías

- Metodologías orientadas a procesos (basadas en mejores prácticas)
  - ***SPICE, CMMI, ITIL, etc.*: metodologías de desarrollo en la que se definen un conjunto de buenas prácticas para la mejora gradual de los procesos de ingeniería.**
  - ***Rational Unified Process (RUP):* define un ciclo de vida iterativo priorizando el uso de lenguajes de modelado, casos de uso y centrado en la arquitectura.**
  - ***Microsoft Solution Framework (MSF)*: metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso. Se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.**
- Metodologías Ágiles
  - ***eXtreme Programming (XP)*: utilizada para proyectos de corto plazo. Consisten desarrollos rápidos e iterativos, cuya particularidad es tener como parte del equipo al usuario final.**
  - ***SCRUM*: mismas características que XP. Define desarrollos incrementales de 30 días y seguimiento diario del proyecto en reuniones de 15 minutos.**
  - ***Crystal*: centrada en las personas que componen el equipo. Reducción al máximo del número de artefactos**

Programación Extrema
("eXtreme Programming")

Una definición simple

- *Ligero*/adj.: que pesa poco; ágil, veloz, pronto; que muda con facilidad de pensamientos (RAE).
- *Ágil*/adj.: Que se mueve o utiliza sus miembros con facilidad y soltura (RAE)
- El término "Ágil" es ahora el sustituto operativo de "Ligero"
- Proceso SW: El conjunto de actividades utilizadas por una organización o proyecto para planificar, gestionar, ejecutar, supervisar, controlar y mejorar sus actividades SW [SPICE00].
- *Ligero* normalmente significa *no pesado*
- El "Gang of 17" que constituyó la "Agile Alliance" ha acuñado el término "Agile Software Development"
  - Su manifiesto puede verse en http://www.agilealliance.org/

Un proceso ligero puede ser definido como …

El mínimo conjunto de actividades y elementos que deben ser incluidos en el proceso SW para asegurar un buen resultado (éxito) para todas las partes interesadas.

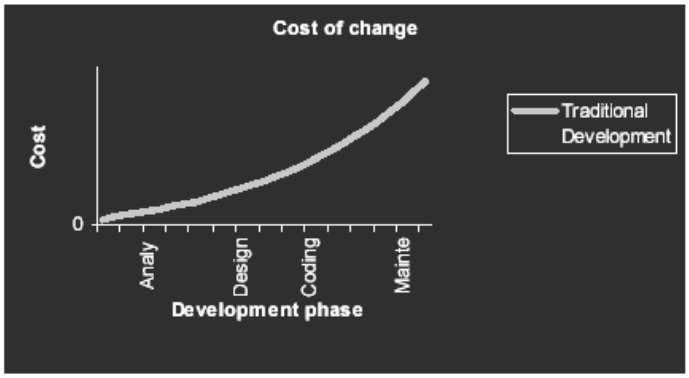
Ágil: ¿por qué bueno?

- Identificación y reducción del riesgo (desarrollo iterativo)
- "Diseñar un poco, construir un poco, probar un poco, aprender mucho".
- Con capacidad de respuesta ante cambios, permitiendo la adaptación a requisitos de negocio cambiantes.
- Inversión inicial menor.
- Ver resultados tangibles antes; el sistema crece a lo largo del tiempo.
- Equipos reducidos permiten una comunicación verbal más efectiva, reduciendo (sin eliminar) la necesidad de documentación escrita.

> Standish Group CHAOS 2000 Report: Reduce requirements to the bare minimum, provide constant communication systems coupled with a standard infrastructure, add good stakeholders, an iterative development process, management tools, adherence to key roles (PM, etc.) and success is almost guaranteed

Los procesos actuales son pesados

- Las metodologías tradicionales buscan reducir los costes conforme a la curva de coste de cambio de Boehm.



Curva de Boehm

- Para conseguir esto:
  - Se necesita mucha planificación por adelantado, lo que se traduce en metodologías pesadas.
  - Cada fallo localizado de forma temprana ahorra dinero, ya que los modelos son más fáciles de modificar que el código.
  - Se realizan grandes inversiones por adelantado en modelos de análisis y diseño, debido al coste de detección tardía de errores.
  - Esto lleva a una mentalidad de desarrollo en cascada con BDUF (Big Design Up Front)
- Pero esta lógica está basada en desarrollos de los 70 y 80's

Programación eXtrema: Definición

- Problemas con el SW de hoy:
  - El desarrollo SW presenta riesgos y es difícil de gestionar
  - Los clientes a menudo están insatisfechos con el desarrollo
  - Los programadores también están insatisfechos
- Programación Extrema (Extreme Programming -XP-) es una metodología basada en los valores de simplicidad, comunicación, realimentación y ánimo.
- Se basa en poner junto a todo el equipo y emplear prácticas sencillas, con suficiente realimentación para facilitar que el equipo vea donde está y ajuste las prácticas a su situación concreta.

Programación Extrema

- Idea relativamente nueva:
  - El proyecto C3 (Chrysler, 1996)
  - Kent Beck, pionero de XP (Libro: "*eXtreme Programming eXplained*", 2000)
- Adecuada para proyectos pequeños (2-20 personas)
- Utilización de tecnología orientada a objetos
  - Uso de lenguajes modernos
- Relaja la pesada gestión en favor de mayor informalidad
- El proceso de desarrollo SW se divide en cuatro tipos de actividades
  - Planificación
  - Diseño
  - Codificación
  - Pruebas

¿Por qué ayuda XP?

- Programación Extrema es un proceso "ligero" que crea y a continuación explota una curva de coste aplanada.
- XP está orientado a las personas y no a los procesos, y trata de trabajar de forma explícita con la naturaleza humana en lugar de ir contra ella.
- Las prácticas de XP aplana la curva de coste de cambio.

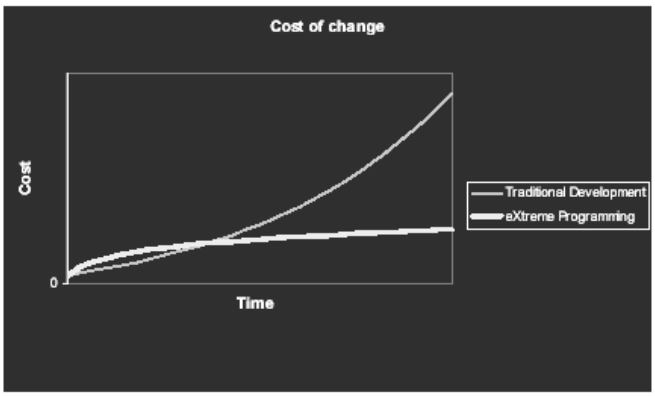
- "Software development is too hard to spend time on things that don't matter. So, what really matters? Listening, Testing, Coding, and Designing."

  (Kent Beck, "padre" de la Programación Extrema)

- XP se centra en lo que "realmente importa" en el desarrollo SW.

¿Por qué ayuda XP¿ (Cont.)

- La curva de coste de cambio con XP



Economía de XP

- Promueve desarrollo incremental con mínimo diseño por adelantado.
- Resultados se consiguen en un proceso "pagar conforme se avanza", en lugar de requerir una alta inversión por adelantado.
- Entrega primero aquello de mayor valor para el negocio
- Permite la opción de "corta y corre" a través de frecuentes entregas que son probadas a fondo.
- XP tiende a emplear equipos pequeños, reduciendo, por tanto, los costes de comunicación.
- XP pone a los clientes y programadores en un mismo lugar físico.
- XP prefiere tarjetas con índice a caros entornos de diagramas UML* de ida y vuelta.
- Las prácticas de XP trabajan de forma conjunta y sinérgica, para conseguir un equipo que se mueve tan rápido como es posible para entregar el valor que quiere el cliente.

(*) El Unified Modeling Language™ - UML - es la especificación más utilizada del OMG (Object Management Group), y la forma en la que el mundo modela no sólo la estructura, comportamiento y arquitectura de las aplicaciones, sino también los procesos de negocio y la estructura de datos.

Interacción entre las 4 variables de gestión de proyectos

Variable	Si aumenta en exceso ...	Si se reduce ...
Alcance		Permite mejorar la calidad, siempre que resuelva el problema básico del cliente. También permite reducir plazo y coste. La herramienta más potente de gestión (*)
Tiempo	Más puede mejorar calidad y alcance, pero en exceso puede dañar, pues la mejor realimentación viene del sistema en producción.	Si poco, sufrirá la calidad e inmediatamente detrás el alcance, el tiempo y el coste.
Coste	Más dinero puede engrasar el sistema, pero en exceso puede crear más problemas que los que resuelve.	Con poco dinero será imposible resolver los problemas del cliente.
Calidad	Insistir en mayor calidad permite conseguir plazos menores o hacer más en un tiempo dado. Efecto humano: se trabaja mejor si se siente que se hace un buen trabajo.	Variable terrible de control. Se puede sacrificar para obtener ganancias a corto, pero los costes posteriores son enormes (humanos, de negocio y técnicos).

(*) "Los clientes no saben decirnos lo que quieren … y no les gusta lo que les entregamos."

Programación eXtrema: Base

Fuerzas externas (clientes, gestores) eligen tres de las cuatro variables. El equipo de desarrollo obtiene el resultado de la cuarta.

No es posible controlar las cuatro a la vez.

Variable	Programación tradicional	Programación eXtrema
Alcance	Fijo	Variable
Tiempo	Fijo	Fijo
Coste	Fijo	Fijo
Calidad	Variable	Fijo

Axiomas de XP

Axioma de XP	Consecuencias
Equipos de pequeñas dimensiones	**Equipos de 4 a 8 representan proyectos que son subconjuntos de un proyecto de desarrollo.**
Requisitos vagos	**En el mundo de los negocios, los requisitos vagos son un indicativo de problemas sistemáticos mayores. Si los resultados del negocio son vagos, el proceso de medida del valor puede no ser estable.**
Requisitos cambiantes	**Los proyectos cuyos requisitos cambian a menudo tienen también otros problemas. Las partes implicadas no pueden decidir cuál debe ser el resultado, y por tanto la medida del éxito del negocio posiblemente sea también inestable.**

Declaración de derechos del Cliente

- Como Cliente, Vd. tiene derecho a:
  - Un plan global, para saber qué puede obtenerse, cuando y a qué coste;
  - Obtener el mayor valor posible de cada semana de programación;
  - Ver el progreso en un sistema que corre, demostrando que funciona pasando las pruebas repetidas que Vd. especifique (de regresión);
  - Cambiar de opinión, para sustituir funcionalidad y cambiar prioridades sin pagar costes desorbitados;
  - Estar informado de los cambios de planificación, a tiempo para escoger cómo reducir el alcance y mantener la fecha inicial o incluso cancelar en cualquier momento y quedarse con un sistema útil que funcione y sea reflejo de la inversión realizada hasta la fecha.

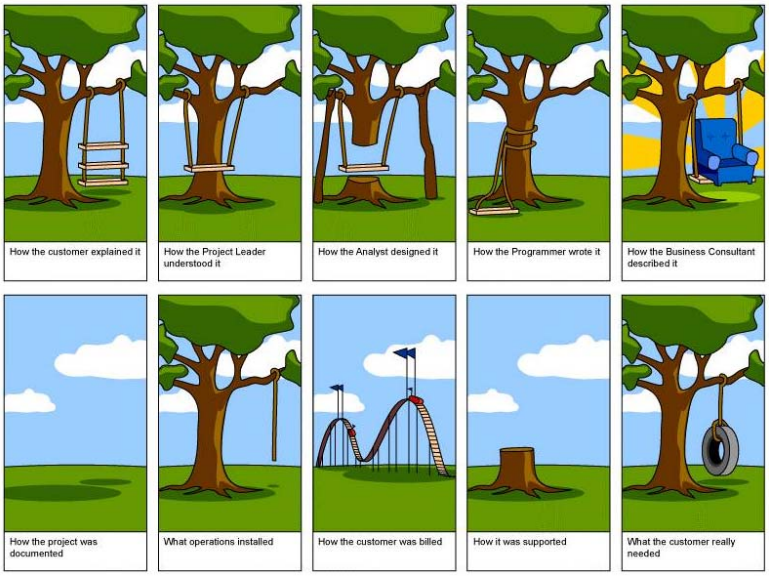
Declaración de derechos del Desarrollador

- Como Desarrollador Vd. tiene derecho a:
  - Conocer lo que se necesita, con definiciones claras de prioridad;
  - Producir siempre trabajo de calidad;
  - Solicitar y recibir ayuda de los compañeros, superiores y clientes;
  - Hacer y actualizar sus propias estimaciones;
  - Aceptar sus responsabilidades en lugar de que le sean asignadas.

Algunos aspectos culturales de XP

- XP es una cultura verbal en lugar de escrita
  - El código comunica el mensaje
  - La documentación no es necesaria excepto cuando el código no puede comunicar el mensaje.
- Se emplean métodos de lectura sin documentación
  - Juego de pruebas unitarias
  - "Simplemente pregunta"
- Se supone que el lector es conversador en la cultura
  - Se basa en un lenguaje y valores compartidos entre los miembros del equipo.
- Son lenguajes y valores diferentes de los de "Programadores de Producto" donde ...
  - La documentación se emplea para comunicar valores y lenguaje.
  - Participantes externos, con diferentes valores.

Diálogo cliente-proveedor



Valores de XP

- XP está basado en 4 valores clave:
  - Comunicación
  - Simplicidad
  - Realimentación
  - Ánimo

Valores de XP (2)

- Comunicación
  - Un representante del Cliente se ubica con los desarrolladores
  - Se prefieren las conversaciones a la documentación formal, aunque se puede emplear esta para tomar nota de las conversaciones
  - Propiedad colectiva del código - cualquier desarrollador tiene el derecho de cambiar cualquier cosa.
  - Programación en pareja - revisión de código en tiempo real.
- Simplicidad
  - ”Construye la cosa más sencilla que pueda funcionar”
  - Codificar sólo lo que se necesite ahora; no codificar para los requisitos de mañana, dado que pueden cambiar.

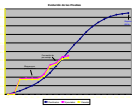
Valores de XP (3)

- Realimentación
  - Minutos a horas: Programadores escriben pruebas unitarias - incluso antes que el código.
  - Horas a días: Programadores estiman el tiempo de desarrollo siempre que el cliente escribe una nueva “historia” de requisito.
  - Semanas a meses: Clientes escriben pruebas de aceptación para cada iteración y lanzamiento.
- Ánimo
  - Se anima a los Programadores para que reescriban el código y refabriquen los diseños.
  - Pruebas unitarias permiten a los Programadores verificar rápidamente que los cambios no han afectado al sistema.
  - Comunicación y simplicidad apoyan el ánimo: el descontento es comunicado rápidamente a todo el equipo, y los diseños son fáciles de modificar.

Roles de XP

- Cliente
  - Escribe “Historias de Usuario” y especifica Pruebas Funcionales.
  - Establece prioridades, explica las Historias
  - Puede ser o no un usuario final
  - Tiene autoridad para decidir cuestiones relativas a las Historias
- Programador
  - Hace estimaciones sobre las Historias
  - Define Tareas a partir de las Historias y hace estimaciones
  - Implementa las Historias y las Pruebas Unitarias
- Tutor
  - Observa todo, identifica señales de peligro, se asegura que el proyecto se mantiene en curso
  - Ayuda en todo
  - Da avisos cuando se necesita

Roles de XP (2)



- Perseguidor (calidad)
  - Monitoriza el progreso de los programadores, toma acción si las cosas tienden a salirse de su senda.
  - Las acciones incluyen reuniones con el Cliente, solicitar ayuda al Tutor u otro Programador, ...
- Verificador
  - Implementa y corre las Pruebas Funcionales (¡no Pruebas Unitarias!)
  - Presenta gráficas de los resultados y se asegura de que la gente conoce cuándo los resultados empiezan a decaer.
- "Agorero"
  - Se asegura que todos conocen los riesgos que existen
  - Se asegura que las malas noticias no se ocultan, se disculpan o se reducen de proporción.

Roles de XP (3)

- Gestor
  - Planifica las reuniones (por ej., plan de iteraciones, plan de lanzamientos -releases-), se asegura que el proceso de las reuniones se sigue, anota los resultados de la reunión para futuros informes y los pasa al Perseguidor.
  - Posiblemente responsable ante el "Propietario de Oro"
  - Asiste a las reuniones, aporta información útil anterior.
- "Propietario de Oro"
  - La persona que paga el proyecto, que puede ser o no la misma que el Cliente

Un episodio sobre desarrollo ...

- *Miro a mi pila de tarjetas. La primera dice "Retención en exportaciones del trimestre hasta la fecha". Recuerdo que tu dijiste que habías terminado el cálculo del trimestre hasta la fecha. Te pregunto (mi colaborador potencial) si tienes tiempo para ayudarme con las exportaciones. "Por supuesto". La regla es: si alguien te pide ayuda, la respuesta debe ser "Si". Ya somos parjea de programación.*
- *Discutimos durante unos minutos el trabajo que hiciste ayer. Me preguntas: "¿cuáles son las pruebas para esta tarea?" Yo respondo: "cuando corramos esta prueba los valores de exportación deben ser los de esta casilla". Tu preguntas: "¿qué campos tienen que ser rellenados? "No se, preguntemos a Eddie".*
- *Interrumpimos a Eddie 30 seg. y nos explica cómo funciona.*
- *Implementamos la prueba basándose en otros casos existentes, hacemos refabricación, corremos la prueba ... todo funciona.*
- *Vemos que se puede mejorar y digo "vamos a ...". Me interrumpes: "Finalicemos lo que tenemos entre manos" ...*
- *Yo escribo el código (o tu, ... el que tenga la idea más clara). Tu ves que se puede hacer más simple, ... Intentas explicarme cómo ... Te paso el teclado para que tu sigas ...*

Kem Beck. eXtreme Programming eXplained. Embrace Change, Addison Wesley, 2000

Prácticas XP

- XP está basad en 12 prácticas claves:

Todas aplicadas *hasta el extremo*

- El "Juego de la Planificación"
  - Planificación de los Lanzamientos (Releases)
  - Planificación de las iteraciones
- Lanzamientos pequeños y frecuentes
- Metáfora del Sistema (System Metaphor)
- Desarrollo gobernado por las pruebas (Test Driven Development)
- Diseño Simple
- Refabricación (Refactoring)
- Programación en pareja (Pair Programming)
- Propiedad colectiva del código (Collective Code Ownership)
- Integración continua (Continuous Integration)
- Avance sostenido (Sustainable Pace)
- Cliente en el lugar de desarrollo (On-site Customer)
- Estándares de Codificación (Coding Standard)

¿Por qué extrema?

- Refleja la intensidad y falta de miedo de los atletas en deportes extremos.
- Las prácticas XP ponen el tono a un nivel más alto que los proyectos tradicionales

XP: Codificación (simplificada)

- El cliente está siempre disponible
- Código conforme a estándares
  - patrones de diseño
  - mejores prácticas (“best practices”)
- Codificar primero las pruebas unitarias
  - las pruebas unitarias sirven también como documentación
- Programación en pareja
  - contra-intuitiva, pero ¡funciona!
- Propiedad colectiva del código
- Lo último optimizar

El “Juego de la Planificación”

1

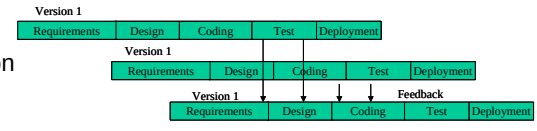
- Planificación XP:
  - Diálogo entre lo posible y lo deseable
  - Conjunto de reglas para gobernar la construcción de relaciones de respeto.
  - Metáfora: “Juego”
- El Objetivo:
  - Máximizar el valor del SW producido por el equipo.
- La Estrategia:
  - Poner la funcionalidad de mayor valor ...
  - en producción tan pronto como se pueda …
  - con mínima inversión inicial …
  - abordando primero la parte de mayor riesgo.
- Las piezas:
  - Tarjetas que contienen “historias de usuario”

Etapas de un proyecto XP (Lanzamientos pequeños y frecuentes)

2

- Inicio
  - Historias de Usuario
- Plan de lanzamientos (releases)
- Lanzamiento (cada lanzamiento lleva típicamente entre 1-6 meses)
  - Iteración 1 (típicamente 1- 3 semanas)
    - Desarrollo
    - Despliegue
    - Prueba de Aceptación
  - Iteración 2
    - Desarrollo
    - Despliegue
    - Prueba de Aceptación

      :
  - Iteración n



Captura de requisitos

- Responsabilidades
  - Punto clave: El Cliente es el responsable de los requisitos
  - Programadores ayudan en la captura y clarificación de los requisitos. Los clientes necesitan ayuda especial con los requisitos no funcionales y en la resolución de los detalles de las pruebas de aceptación.
- Documentación
  - Historias de Usuario
  - Casos de pruebas de aceptación

Historias de Usuario

- Una breve descripción del comportamiento del sistemas desde el punto de vista del usuario
- Emplea la terminología del Cliente sin jerga técnica
- Una por cada característica principal del sistema
- Debe estar escrita por los usuarios
- Se emplean para hacer estimaciones de tiempo para el plan de lanzamientos
- Reemplazan un gran Documento de Requisitos
- Gobiernan la creación de las Pruebas de Aceptación:
  - Debe haber una o más pruebas para verificar que una historia ha sido implementada de forma correcta
- Son una cosa distinta que los Requisitos:
  - Deben proporcionar sólo el detalle suficiente como para poder hacer una razonable estimación de cuánto tiempo requiere la implementación de la historia.

Historias de Usuario (2)

- Difiere de los Casos de Uso:
  - Escritos por el Cliente, no los Programadores, empleando terminología del Cliente.
  - Más “amigables” que los Casos de Uso formales.
- Las Historias de Uso tienen tres aspectos cruciales:
  - Tarjeta
    - Suficiente información para identificar la historia
  - Conversación
    - Cliente y Programadores discuten la historia para ampliar los detalles
    - Verbal cuando sea posible, pero documentada cuando se requiera
  - Confirmación
    - Pruebas de Aceptación para confirmar que la historia ha sido implementada correctamente

Historias de Usuario: Ejemplos

Un usuario quiere acceder al sistema, para ello debe buscar un administrador del sistema, quien introduce el Nombre del usuario, Apellidos, Dirección de e-mail, Nombre_Usuario (único) y Número de Teléfono.

Riesgo: Bajo **Coste**: 2 puntos

El usuario debe ser capaz de buscar un libro.

Riesgo: Alto **Coste**: (¡muy grande!)

Historias de Usuario: Ejemplos (2)

El usuario debe ser capaz de buscar un libro por Título, presentándole los resultados como una lista.

Riesgo: Med. **Coste**: 1 punto

El usuario debe ser capaz de buscar un libro por Autor, presentándole los resultados como una lista.

Riesgo: Med. **Coste**: 1 punto

Historias de Usuario: Ejemplos (3)

El usuario debe ser capaz de buscar un libro por su número ISBN, presentándole los resultados como una lista.

Riesgo: Med. **Coste**: 1 punto

El usuario debe ser capaz de buscar un libro por Categoría, presentándole los resultados como una lista.

Riesgo: Med. **Coste**: 2 puntos

Pruebas de Aceptación

- Pruebas formales para determinar si un sistema satisface sus criterios de aceptación, es decir, las Historias de Usuario.
- Debe ser automatizado, pero puede ser una simple serie de pasos repetidos.
- Al menos debe haber una Prueba de Aceptación por cada Historia.

Plan de Lanzamiento (Release)

- El Cliente define el valor para el negocio de las características deseadas (Historias de Usuario)
- Los Programadores proporcionan estimaciones de 1, 2 o 3 “puntos”.
- Historias mayores de 3 puntos deben dividirse en otras más pequeñas.
- El Cliente decide qué historias deben ser incluidas en un lanzamiento.
- Poner el foco en completar primero las Historias con mayor riesgo y mayor valor para el negocio.

Plan de Lanzamiento (Release) - 2

- Las Historias para un Lanzamiento se organización en iteraciones de 1-3 semanas.
- Las Historias de mayor riesgo y mayor prioridad se incluyen en las primeras iteraciones.
- Para un nuevo sistema, la iteración “0” define el esqueleto básico de la aplicación y la infraestructura requerida.
- El Lanzamiento y las Iteraciones tienen fechas fijas para su consecución - las fechas son fijas, el alcance es variable.
- Este es el Plan de Lanzamiento

Plan de Iteración

- Las Historias para una Iteración son divididas en Tareas por los Programadores
- Las Tareas son estimadas por todos los Programadores como grupo.
- Los Programadores aceptan Tareas y hacen una estimación del tiempo requerido para completarlas.
- Pueden aceptar como mucho tantos puntos como completaron en la última Iteración
- Una vez que comienza el desarrollo, la Velocidad del Proyecto mide el progreso.

Metáfora del Sistema

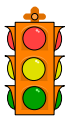
3

- Buscar una metáfora que valga para todo el sistema y apoye la búsqueda de un diseño simple.
  - Metáfora = Marco de referencia que indique cómo pensar en relación con el sistema, proporcionando una forma sencilla de pensar en el comportamiento del mismo.
  - Ejemplo: “el sistema es como una pizarra, en la que varias partes del sistema pueden escribir o borrar en ella. Al concluir, el usuario puede guardar el contenido o borrarlo.”

Diseño gobernado por las Pruebas del Programador

4

- Pruebas automatizadas escritas para probar el comportamiento de las clases individuales
- Fundamental para la XP y para mantener una curva de coste plana
- XP emplea una mentalidad de “Primero Probar”; escribir la prueba para a continuación escribir el código que pase dicha prueba.
  - “Nunca escriba una línea de código sin tener una prueba que haya fallado” (Kent Beck).
- Ningún código pasa a producción salvo que tenga las pruebas asociadas.



Pruebas del Programador (2)

- Las Pruebas se escriben primero
  - Empezar por una sola prueba para un sólo método de una sola clase.
  - Escribir una prueba para un solo escenario del método y a continuación escribir el código que pase dicha prueba.
  - Escribir otra prueba para otro escenario, observar que falla la prueba y escribir a continuación el código que permita superarla.
  - Utilizar esta aproximación para construir un juego de pruebas unitarias que puedan ser disparadas automáticamente.
- Experiencia práctica:
  - Relación código Prueba/Producción: 1/2
  - La escritura de pruebas fuerza a escribir métodos más pequeños.
  - Caso de prueba = documentación
  - Tener muchos casos de prueba facilita el atreverse a cambiar el código de otras personas.
  - Reutilización sistemática de las pruebas preparadas por otros.

Pruebas del Programador (3)

- Las Pruebas determinan qué código debe escribirse.
- Las Pruebas del Programador deben correrse al 100% antes de que el código pase a integración.
- Como mucho una prueba fallida en cualquier momento.
- Pruebas de caja-gris
- Apoyar con “Refabricación” (promover el ánimo)
- Existen entornos de Prueba para muchos lenguajes:
  - JUnit para Java
  - CPPUnit para C++
  - NUnit para todos los lenguajes .Net

Muestra Prueba JUnit

- Primero, escribir la Prueba:

```
import junit.framework.TestCase;

public class TestMovieList extends TestCase {
    public void testEmptyList() {

    MovieList emptyList = new MovieList();

    assertEquals( "Empty list should have size of 0",
                0, emptyList.size() );
    }
}
```


Muestra Prueba Junit (2)

Ahora escribir el código para que pase la Prueba.

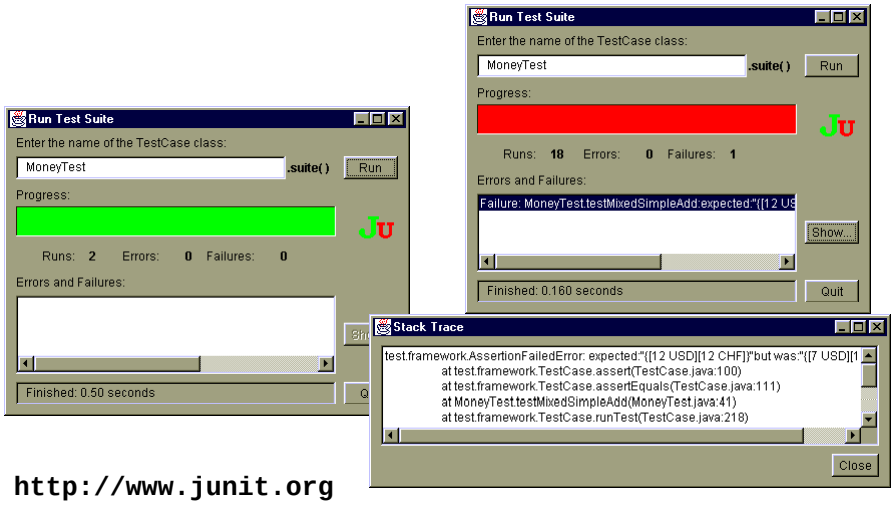
- Primero, no hay clase MovieList:

```
public class MovieList {
}
```

- Ahora necesitamos un método ‘size’ que devuelva un ‘int’:

```
public class MovieList {
    public int size() {
        return 0;
    }
}
```

JUnit



http://www.junit.org

Diseño Simple

5

- XP: Se busca el diseño más simple que permita correr el actual juego de pruebas:
  - Código + pruebas comunican todo lo que se quiere comunicar
  - El sistema no contiene código duplicado
  - El sistema tiene el menor número posible de clases
  - El sistema tiene el menor número posible de métodos
- *Se consigue mediante un proceso continuo de Refabricación*

Refabricación (Refactoring)

6

- Refabricación = un cambio realizado ...
  - en la estructura interna del SW …
  - para que sea más fácil de entender …
  - y más barato de modificar …
  - sin que cambie su comportamiento observable.

Introducir *patrones de diseño*

- Refabricar:
  - aplicar series de refabricaciones

Refabricación (2)

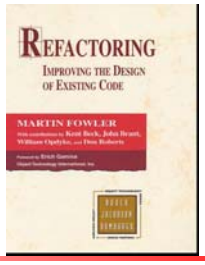
- "Refabricación es el proceso de cambiar el sistema SW de tal forma que no altere el comportamiento externo del código aunque si mejore su estructura interna." (Martin Fowler)
- Mejora la calidad y mantenibilidad del código.
  - Ayuda a localizar errores y programar más rápido.
- Soportado por las Pruebas del Programador - Los Programadores pueden verificar rápidamente que la refabricación no ha creado errores de regresión.
- Debe eliminar malas prácticas tales como:
  - Código duplicado
  - Métodos largos
  - Clases grandes
  - Listas largas de parámetros
  - Falta de comentarios
  - …..

¿Cuándo Refabricar?

- Al tercer golpe … se refabrica
  - Primera vez: simplemente se hace
  - Segunda vez con algo similar: duplicar
  - Tercera vez con algo similar: refabricar
- Cada cambio incluye refabricación
  - Al añadir una carácterística.
  - Al arreglar un fallo
  - Al revisar código
- Cada refabricación mejora/repara el diseño
- Grandes refabricaciones se realizan por necesidad
- Cada refabricación = mini fase de diseño
  - => Muerte del gran diseño (fase)
  - => No hay separación programador/arquitecto

Proceso de Refabricación

- Modificar el código implica:
  - Prueba (añadir si es necesario)
  - Refabricar para aumentar la comprensión del código
  - Prueba
  - Añadir pruebas para los cambios que se hagan, modificar hasta que pase
  - Prueba
  - Refabricar para llevar el código a su estado más simple
  - Prueba



Muestra de Refabricación

- Extraer Método
  - Se tiene un código fragmentado que puede ser agrupado. Convertir el fragmento en un método cuyo nombre explique el objeto del método

```
void printOwing() {
    printBanner();
    // Print Details
    System.out.println( “Name " + _name);
                System.out.println( “Amount " +
  getOutstanding());
}
```


Muestra de Refabricación (2)

- El resultado:

```
void printOwing() {
    printBanner();
    printDetails( getOutstanding() );
}

void printDetails (double outstanding) {
    System.out.println( “Name: " + _name);
    System.out.println( “Amount: " + outstanding);
}
```

(Fuente: “Refactoring”, by Martin Fowler)

Programación en pareja

7

- Todo el código de producción se escribe con dos personas mirando a una máquina, con un solo teclado y un solo ratón.
- Aunque la revisión del código sea buena, se sigue haciendo.
- Experiencia práctica
  - Resistencia (inicial)
  - Cambio de parejas
  - Discusión del diseño, refabricación, pruebas
  - Forma de comunicar el conocimiento sobre el sistema
  - Mejora mucho la calidad
  - La codificación es más divertida, más tranquila y menos solitaria
  - Más productiva (>2)

Propiedad colectiva del código

8

- Antes:
  - Desarrolladores tenían la propiedad (asumían la responsabilidad) de trozos del código de un sistema …
  - a fin de asegurar la coherencia del diseño.
  - Si se iba el desarrollador, el código quedaba sin dueño ...
  - o si estaba fuera, nadie podía evolucionar/reparar dicho código
- XP:
  - Todo el código es propiedad de todos
  - Cualquier pareja puede cambiar cualquier código.
  - Es más, debido a la práctica de refabricación, se supone que las parejas modificarán código que no escribieron.
  - La coherencia se mantiene por realizarse un desarrollo gobernado por las pruebas (un juego robusto de pruebas asegura que los cambios no introducen efectos laterales no previstos).

Integración continua

9

- Tener un sistema corriendo desde el primer día.
- Integrar las historias una a una (hacer al menos una integración diaria) - Integración casi continua
  - Desarrollador completa un cambio …
  - se verifica el cambio en el repositorio del código fuente …
  - se detecta el cambio y se inicia una construcción (integración) …
  - se corre automáticamente un juego de pruebas …
  - Si falla alguna prueba se informa al desarrollador …
  - Los problemas de integración se arreglan uno a uno, en pequeños paquetes tan pronto como se detectan.

Avance sostenido

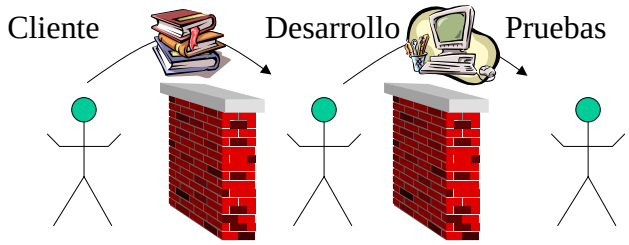
10

- Los equipos XP deben trabajar con paso regular. Así se conseguirá avanzar más que si se hace con una velocidad que no se podrá mantener a lo largo del tiempo.
- Esto no implica 40 horas exactamente (cada equipo determinará cuál es su paso adecuado, el cual dependerá de cada miembro del equipo).
- La Programación en pareja y el desarrollo gobernado por las pruebas son efectivos porque focalizan la mente de la pareja de forma intensa en el código que están creando y esta atención no se puede mantener mucho tiempo.
- Típicamente un equipo dedica unas 6 hr diarias a realizar trabajo en pareja y el resto a otras actividades.
- El Tutor debe evitar que los equipos “se quemen” y reconducir tal situación cuando la detecte.

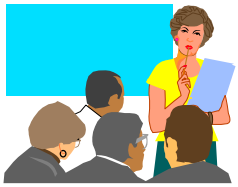
Cliente en el lugar de desarrollo

11

- Antes:



- Ahora (XP): Cliente en el lugar de desarrollo
  - Escribe las historias de uso y las pruebas de aceptación
  - Contesta las dudas que surjan en el día a día.
  - Si no está presente → retrasos



Estándares de Codificación

12

- Como el código es colectivo, es fundamental seguir unos estándares de codificación.
- Estándares: reglas y convenciones que deben seguir los desarrolladores al escribir el código:
  - cómo se deben nombrar las variables y métodos
  - cómo deben ser formateadas las líneas
  - ….
- Formato:
  - Informal, si el equipo es pequeño.
  - Escrito, a poco que crezca el tamaño del equipo (aunque breve y centrándose en lo fundamental).

Claves para implantar bien XP

- Tener un “propietario de oro” (soporte ejecutivo)
- Asegurarse de que hay un cliente o grupo que hable con una sóla voz
- Focalizarse en:
  - Reducción del riesgo mediante realimentación (involucración del cliente, desarrollo iterativo)
  - Reducción del riesgo entregando primero aquellos requisitos de mayor valor para el negocio.
  - Reducción del riesgo por ciclos de entregas cortos, adaptándose a los cambios requeridos por el negocio.
  - Reducción del riesgo por la visibilidad del proceso -todos saben lo que pasa y, por tanto, los problemas no quedan ocultos durante meses-
  - Reducción del coste de desarrollo al poner el foco en las pruebas -más rápida entrega, producto de mayor calidad, esfuerzo reducido en tareas de aseguramiento de la calidad-

Estrategias

- Tamaño del equipo:
  - Conforme crece el tamaño del grupo se requiere más comunicación
  - Funciona mejor con interacción cara a cara (difícil con equipos grandes)
  - Óptimo: 4 desarrolladores/4 meses/US$ 500K (Standish Group)
  - Mantener espacios abiertos que favorezcan la comunicación
- Empezar por un proyecto piloto pequeño. Aplicar luego a proyectos mayores o abordar los grandes pequeños.
- Analizar lecciones aprendidas tras cada iteración y entrega → Qué mejorar/cambiar
- Mantenimiento incorporado al desarrollo cuanto antes (mismo equipo).
- Pruebas rigurosas es la red de protección: pruebas unitarias automáticas, pruebas de aceptación automáticas (de ser posible).

XP y CMMi

Prácticas individuales			*Prácticas de Grupo*		
Actividad XP	**KPA* de CMM**	**Nivel CMM**	**Actividad XP**	**KPA* de CMM**	**Nivel CMM**
Prueba Unitaria	**Aseguramiento de la calidad**	**2**	**Integración continua**	**Aseguramiento de la calidad**	**2**
Diseño simple	**Gestión de Requisitos**	**2**	**Presencia permanente del cliente**	**Gestión de Requisitos**	**2**
Programación en pareja	**Revisión por compañeros**	**2**	**Metáfora de Sistema**	**Gestión de Requisitos**	**2**
Semanas laborables de 40 horas	**Seguimiento y vigilancia del proyecto**	**2**	**Propiedad colectiva**	**Gestión de Requisitos**	**2**
Estándares de codificación	**Aseguramiento de la calidad**	**2**	**El Juego de la planificación**	**Planificación del proyecto**	**2**

- Son las herramientas las que separan los procesos no las actividades.
  - Las actividades son “sentido común”
- Las únicas actividades especiales XP son:
  - Programación en pareja
  - Pruebas unitarias

KPA: Key Process Area

Los principios de XP

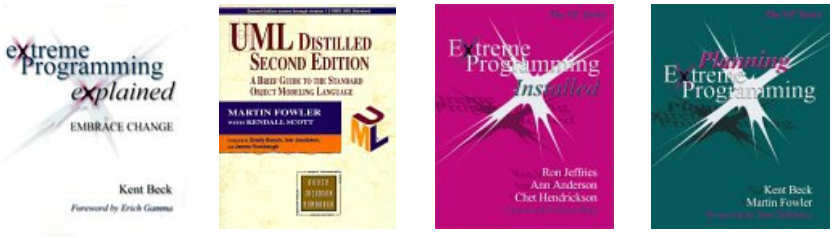
- Además de por sus valores y prácticas, la XP se caracteriza por sus 5 principios básicos:
  - Realimentacion rápida: a sus clientes y aprende de ella.
  - Asumir la simplicidad: intentar siempre una solución simple antes de abordar una compleja.
  - Cambio incremental: mejorar el SW a través de pequeños cambios.
  - Aceptar el cambio: ser expertos en acomodarse y adaptarse
  - Hacer trabajo de calidad: el SW debe mostrar de forma consistente los más altos niveles de destreza en calidad.
- Estos son la clave para hacer XP una reaidad, aunque no se siga alguna de las 12 prácticas.

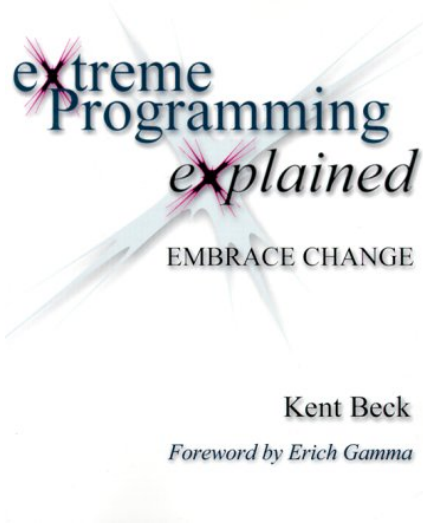
Herramientas Modernas

- Herramientas para gestión de proyectos
  - Diagramas PERT (Program Evaluation Review Technique), también denominados PERT / CRM (Critical Path Management)
  - Diagramas Gantt
  - software para gestión de proyectos
- Herramientas de desarrollo/CASE
  - Primero, y más importante, el entorno de pruebas xUnit
  - Entornos de desarrollo integrados (IDE*) que soporten refabricación del SW ("refactoring") (Visual Studio, JBuilder, Eclipse, etc.)
  - Herramientas para integración continua como CruiseControl.
- Herramientas de proyecto
  - Gestión de cambios y configuración
- Lenguajes de alto nivel

Bibliografía

- K. Beck -- eXtreme programming eXplained. Addison Wesley (2000).
- M. Fowler -- UML Distilled (2nd Ed.)
- www.c2.com/wiki - Extreme Programming Wiki
- www.xprogramming.com - El sitio de Ron Jeffries para todos los temas XP




extreme Programming explained
EMBRACE CHANGE
Kent Beck
Foreword by Erich Gamma