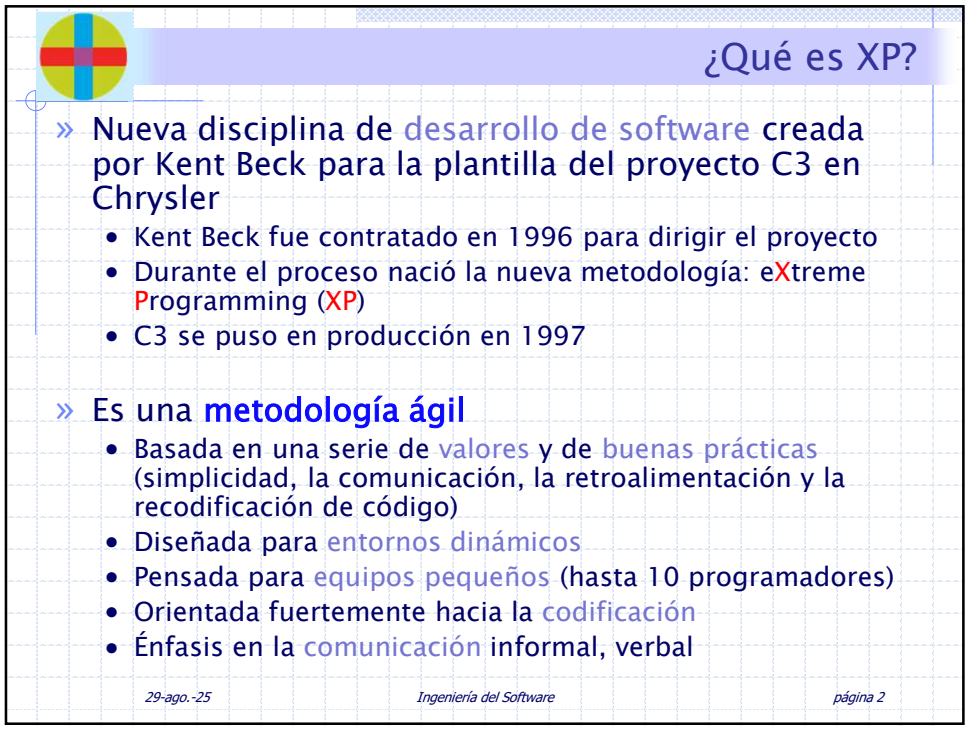
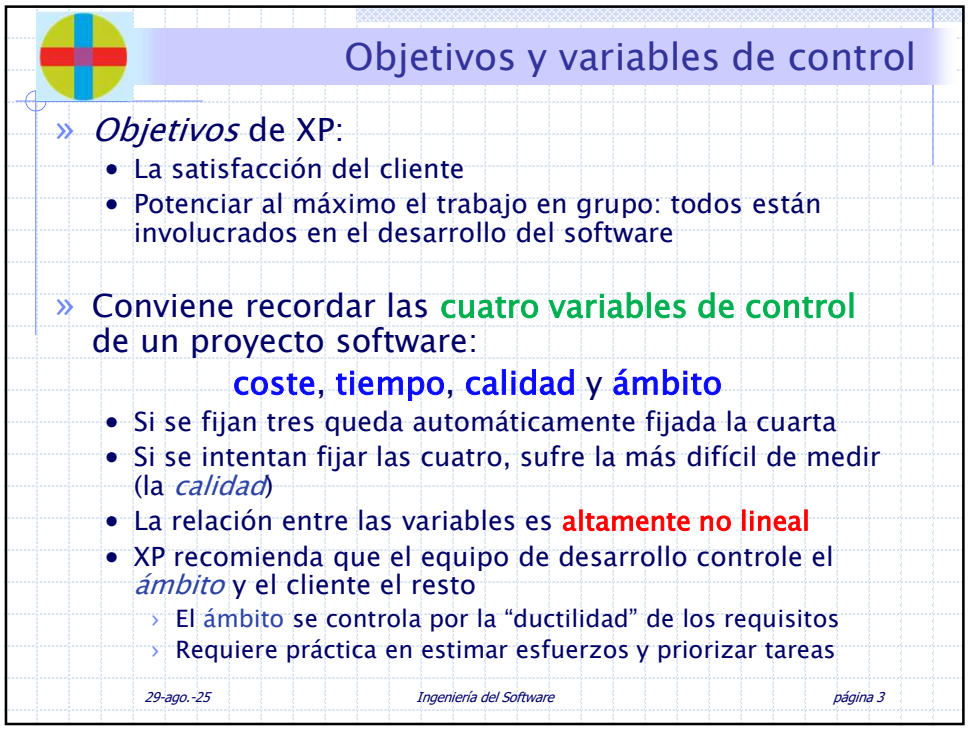

eXtreme Programming
Ingeniería del Software
Curso 2025/2026
Universidad San Pablo-CEU
Escuela Politécnica Superior
Campus de Montepríncipe
Extreme Programming

1


¿Qué es XP?
» Nueva disciplina de desarrollo de software creada por Kent Beck para la plantilla del proyecto C3 en Chrysler
• Kent Beck fue contratado en 1996 para dirigir el proyecto
• Durante el proceso nació la nueva metodología: eXtreme Programming (XP)
• C3 se puso en producción en 1997
» Es una metodología ágil
• Basada en una serie de valores y de buenas prácticas (simplicidad, la comunicación, la retroalimentación y la recodificación de código)
• Diseñada para entornos dinámicos
• Pensada para equipos pequeños (hasta 10 programadores)
• Orientada fuertemente hacia la codificación
• Énfasis en la comunicación informal, verbal
29-ago.-25 Ingeniería del Software página 2

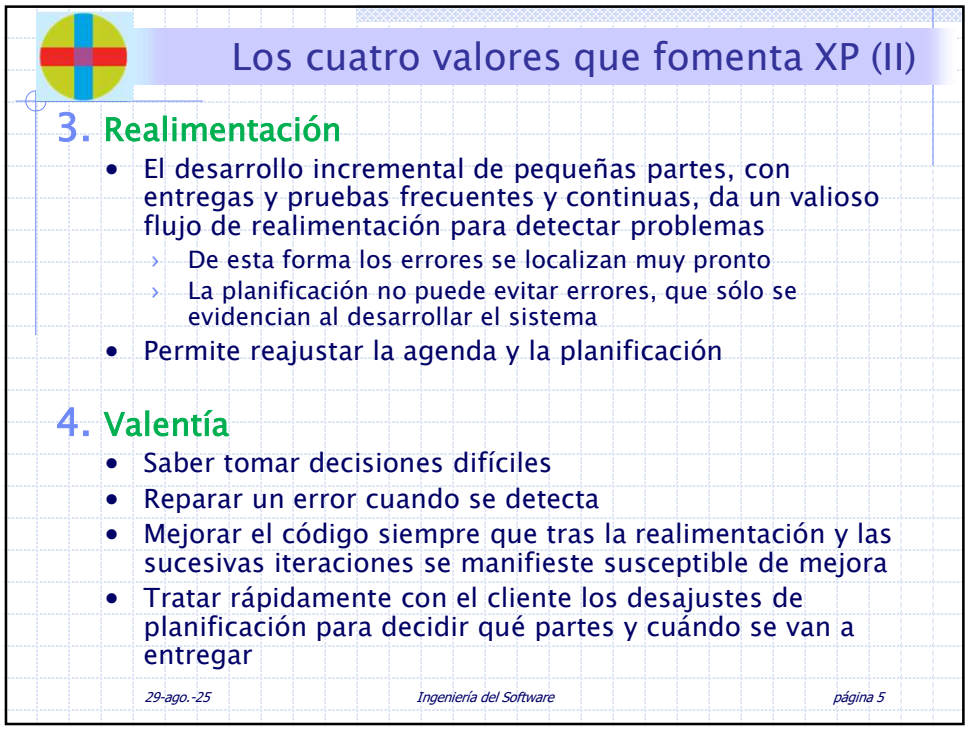
2


Objetivos y variables de control
» Objetivos de XP:
• La satisfacción del cliente
• Potenciar al máximo el trabajo en grupo: todos están involucrados en el desarrollo del software
» Conviene recordar las cuatro variables de control de un proyecto software:
coste, tiempo, calidad y ámbito
• Si se fijan tres queda automáticamente fijada la cuarta
• Si se intentan fijar las cuatro, sufre la más difícil de medir (la calidad)
• La relación entre las variables es altamente no lineal
• XP recomienda que el equipo de desarrollo controle el ámbito y el cliente el resto
› El ámbito se controla por la "ductilidad" de los requisitos
› Requiere práctica en estimar esfuerzos y priorizar tareas
29-ago.-25 Ingeniería del Software página 3

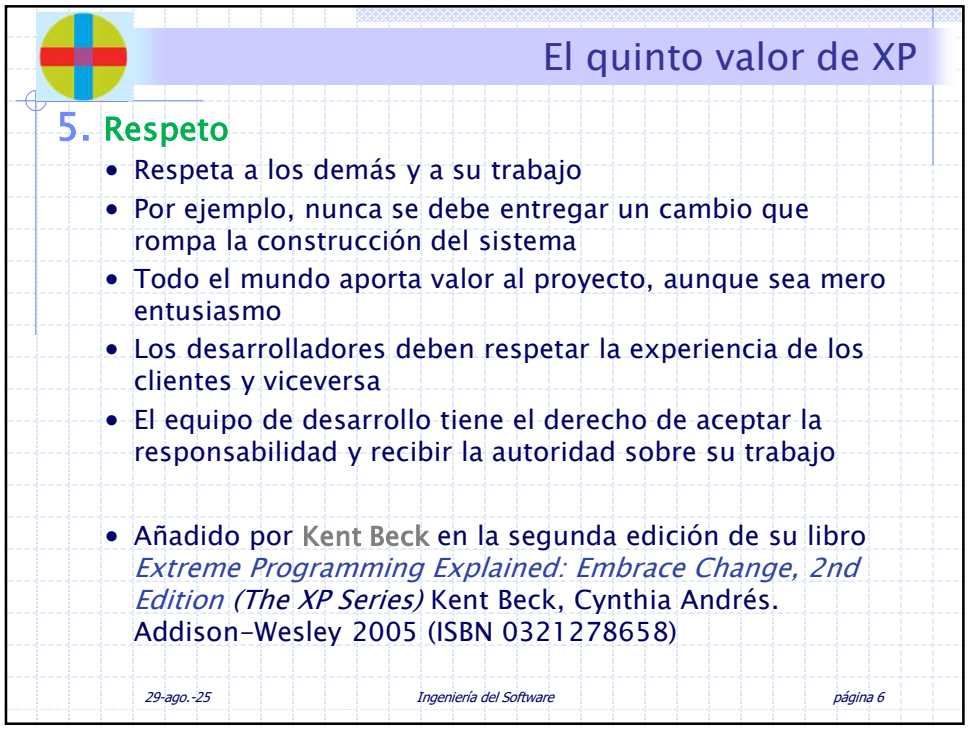
3


Los cuatro valores que fomenta XP (I)
1. Comunicación
• XP pone en comunicación directa y continua a clientes y desarrolladores
• El cliente se integra en el equipo para establecer prioridades y resolver dudas
› De esta forma ve el avance día a día, y es posible ajustar la agenda y las funcionalidades de forma consecuente
2. Sencillez
• Desarrollar sólo lo que realmente se necesita
• Implica resolver en cada momento sólo las necesidades actuales, reales
• Los costes y la complejidad de predecir el futuro son muy elevados, y la mejor forma de acertar es esperar al futuro, no tratar de predecirlo
29-ago.-25 Ingeniería del Software página 4

4


Los cuatro valores que fomenta XP (II)
3. Realimentación
• El desarrollo incremental de pequeñas partes, con entregas y pruebas frecuentes y continuas, da un valioso flujo de realimentación para detectar problemas
› De esta forma los errores se localizan muy pronto
› La planificación no puede evitar errores, que sólo se evidencian al desarrollar el sistema
• Permite reajustar la agenda y la planificación
4. Valentía
• Saber tomar decisiones difíciles
• Reparar un error cuando se detecta
• Mejorar el código siempre que tras la realimentación y las sucesivas iteraciones se manifieste susceptible de mejora
• Tratar rápidamente con el cliente los desajustes de planificación para decidir qué partes y cuándo se van a entregar
29-ago.-25 Ingeniería del Software página 5

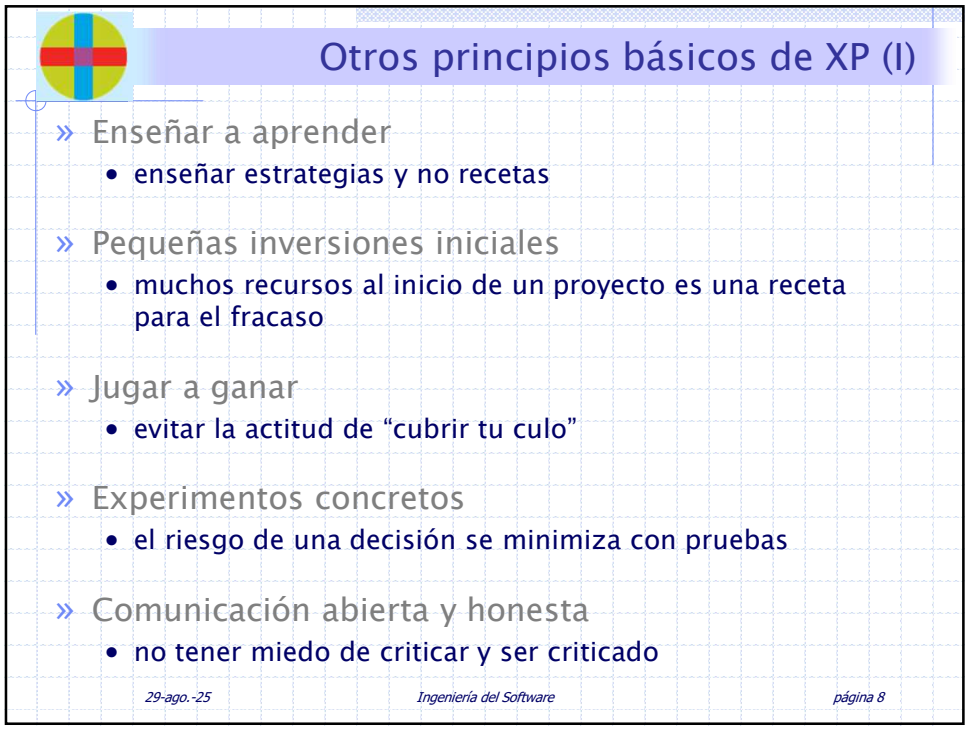
5


El quinto valor de XP
5. Respeto
• Respeta a los demás y a su trabajo
• Por ejemplo, nunca se debe entregar un cambio que rompa la construcción del sistema
• Todo el mundo aporta valor al proyecto, aunque sea mero entusiasmo
• Los desarrolladores deben respetar la experiencia de los clientes y viceversa
• El equipo de desarrollo tiene el derecho de aceptar la responsabilidad y recibir la autoridad sobre su trabajo
• Añadido por Kent Beck en la segunda edición de su libro Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series) Kent Beck, Cynthia Andrés. Addison-Wesley 2005 (ISBN 0321278658)
29-ago.-25 Ingeniería del Software página 6

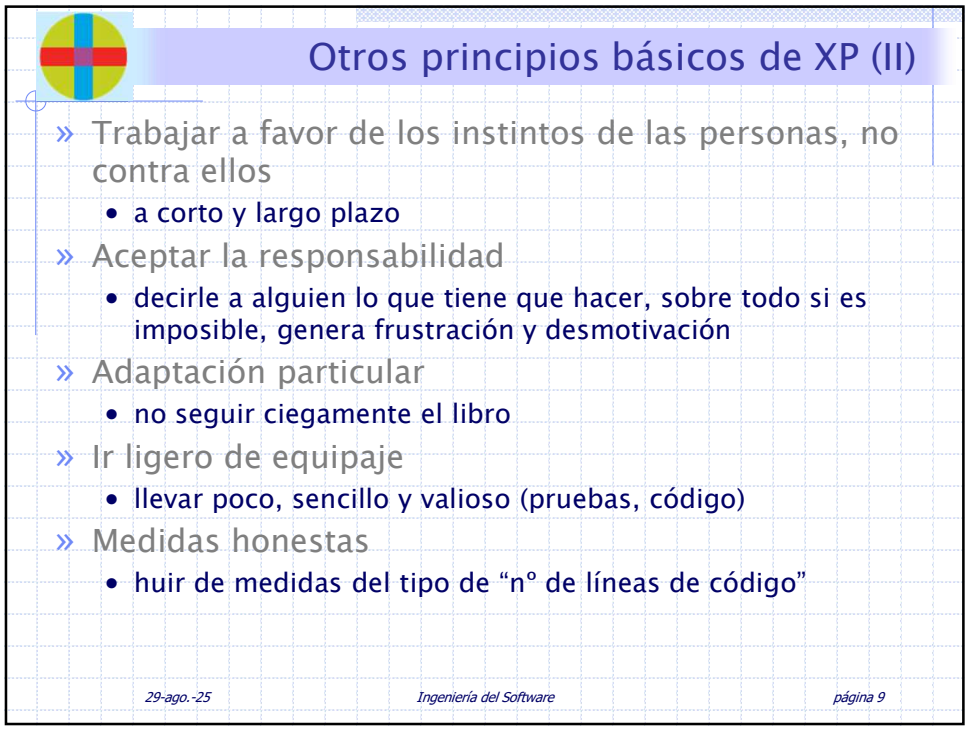
6


Los 5 principios fundamentales de XP
1. Realimentación rápida
• fundamental para la mejora del sistema
2. Asumir la sencillez
• el más complicado para un programador
3. Cambio incremental
• los grandes cambios de una vez no funcionan
4. Aceptar el cambio
• no luchar contra él
5. Trabajar con calidad
• a nadie le gusta trabajar de forma descuidada
29-ago.-25 Ingeniería del Software página 7

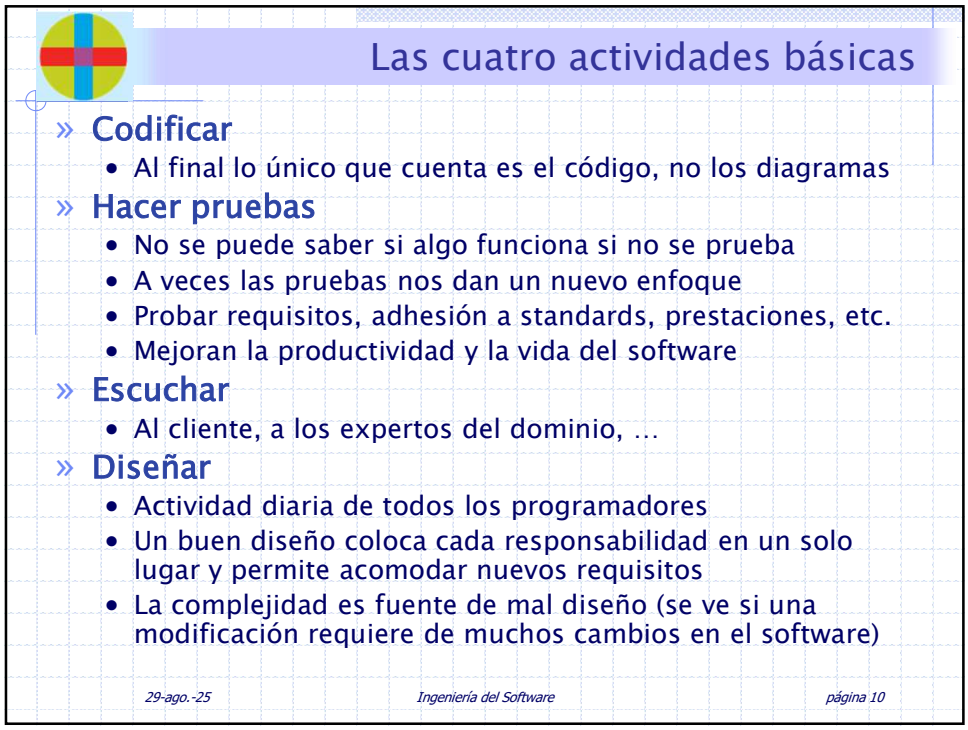
7


Otros principios básicos de XP (I)
» Enseñar a aprender
• enseñar estrategias y no recetas
» Pequeñas inversiones iniciales
• muchos recursos al inicio de un proyecto es una receta para el fracaso
» Jugar a ganar
• evitar la actitud de “cubrir tu culo”
» Experimentos concretos
• el riesgo de una decisión se minimiza con pruebas
» Comunicación abierta y honesta
• no tener miedo de criticar y ser criticado
29-ago.-25 Ingeniería del Software página 8

8


Otros principios básicos de XP (II)
» Trabajar a favor de los instintos de las personas, no contra ellos
• a corto y largo plazo
» Aceptar la responsabilidad
• decirle a alguien lo que tiene que hacer, sobre todo si es imposible, genera frustración y desmotivación
» Adaptación particular
• no seguir ciegamente el libro
» Ir ligero de equipaje
• llevar poco, sencillo y valioso (pruebas, código)
» Medidas honestas
• huir de medidas del tipo de "nº de líneas de código"
29-ago.-25 Ingeniería del Software página 9

9

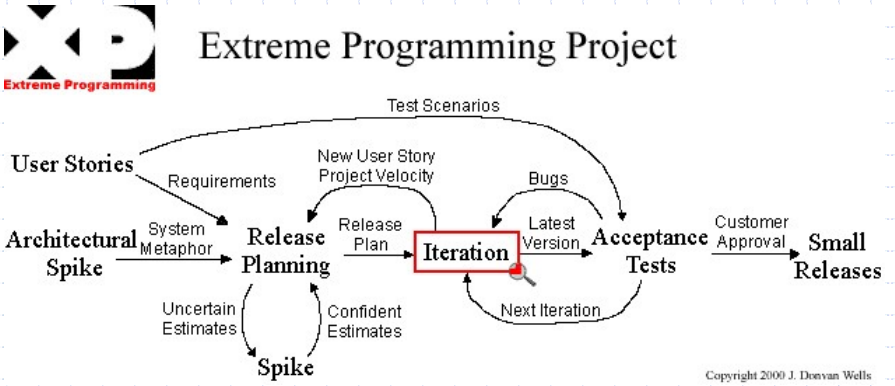

Las cuatro actividades básicas
» Codificar
• Al final lo único que cuenta es el código, no los diagramas
» Hacer pruebas
• No se puede saber si algo funciona si no se prueba
• A veces las pruebas nos dan un nuevo enfoque
• Probar requisitos, adhesión a standards, prestaciones, etc.
• Mejoran la productividad y la vida del software
» Escuchar
• Al cliente, a los expertos del dominio, …
» Diseñar
• Actividad diaria de todos los programadores
• Un buen diseño coloca cada responsabilidad en un solo lugar y permite acomodar nuevos requisitos
• La complejidad es fuente de mal diseño (se ve si una modificación requiere de muchos cambios en el software)
29-ago.-25 Ingeniería del Software página 10

10

## Las 4 actividades y el ciclo de vida XP

» "Listening, Testing, Coding, Designing. That's all there is to software. Anyone who tells you different is selling something" – Kent Beck

*"Escuchar, hacer pruebas, codificar, diseñar. En eso consiste el software. El que te diga otra cosa te está vendiendo una moto"*

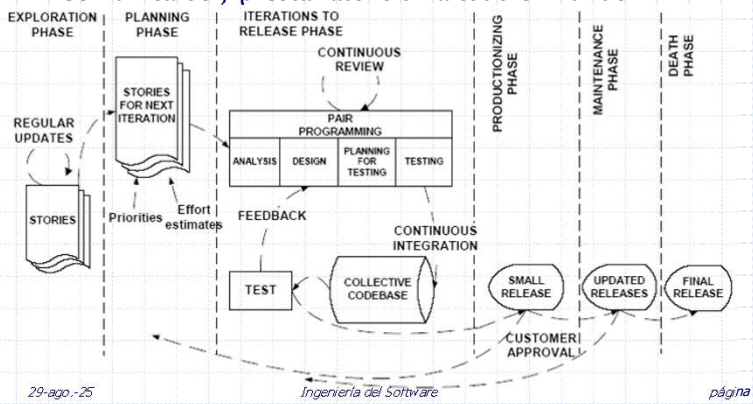





11

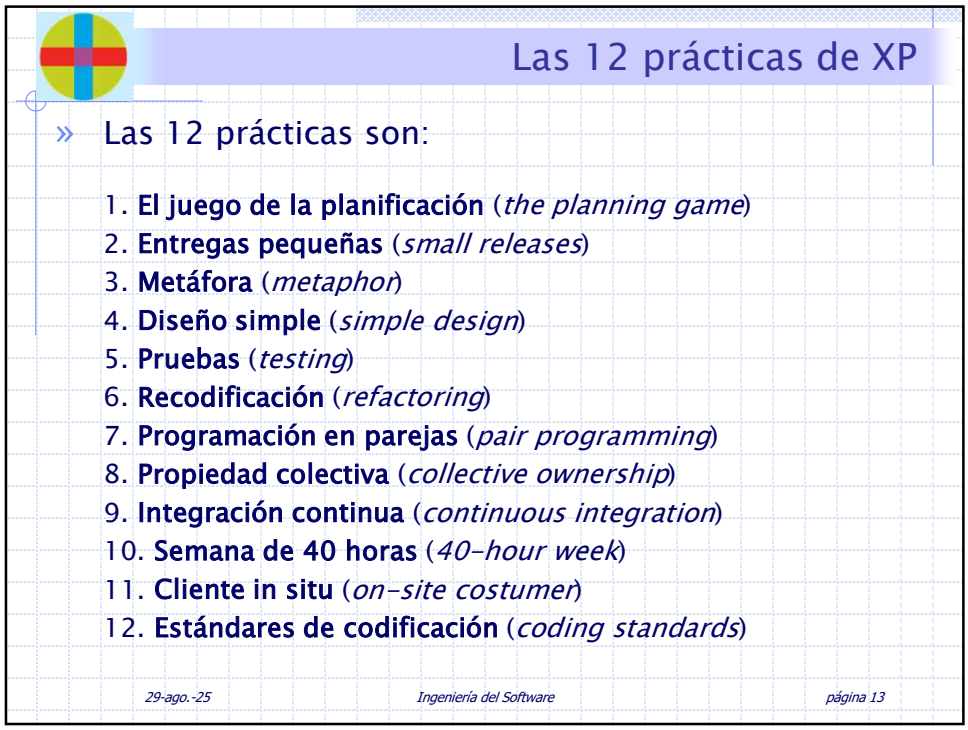
## Ciclo de vida del proceso XP

- Estrategia general a seguir:
  - "Analiza un poco, diseña un poco, programa un poco, prueba un poco"
  - Empezar por las características más importantes
  - Mantener la sencillez
  - Comunicarse y prestar atención a todo el mundo…

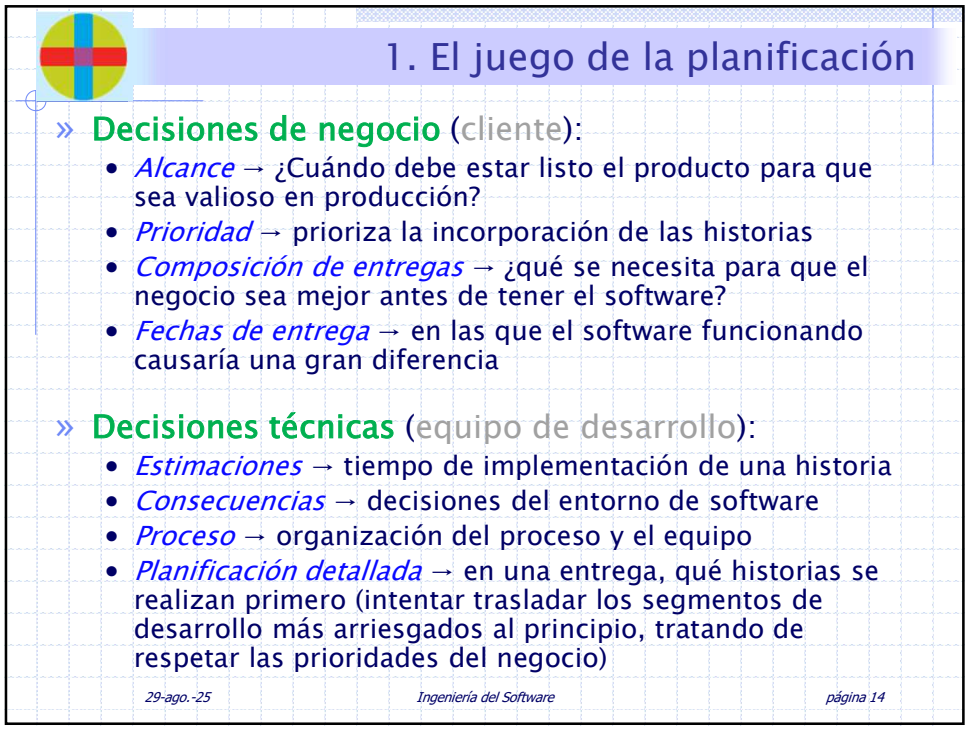





12


Las 12 prácticas de XP
Las 12 prácticas son:
1. El juego de la planificación (the planning game)
2. Entregas pequeñas (small releases)
3. Metáfora (metaphor)
4. Diseño simple (simple design)
5. Pruebas (testing)
6. Recodificación (refactoring)
7. Programación en parejas (pair programming)
8. Propiedad colectiva (collective ownership)
9. Integración continua (continuous integration)
10. Semana de 40 horas (40-hour week)
11. Cliente in situ (on-site costumer)
12. Estándares de codificación (coding standards)
29-ago.-25
Ingeniería del Software
página 13

13


1. El juego de la planificación
Decisiones de negocio (cliente):
Alcance → ¿Cuándo debe estar listo el producto para que sea valioso en producción?
Prioridad → prioriza la incorporación de las historias
Composición de entregas → ¿qué se necesita para que el negocio sea mejor antes de tener el software?
Fechas de entrega → en las que el software funcionando causaría una gran diferencia
Decisiones técnicas (equipo de desarrollo):
Estimaciones → tiempo de implementación de una historia
Consecuencias → decisiones del entorno de software
Proceso → organización del proceso y el equipo
Planificación detallada → en una entrega, qué historias se realizan primero (intentar trasladar los segmentos de desarrollo más arriesgados al principio, tratando de respetar las prioridades del negocio)
29-ago.-25
Ingeniería del Software
página 14

14

## 1. El juego de la planificación: reunión

- Reunión diaria XP ("*Stand-up Meeting*")
  - Todo el equipo
    - Problemas
    - Soluciones
  - De pie en un círculo
    - Evitar discusiones largas
    - Sin conversaciones separadas



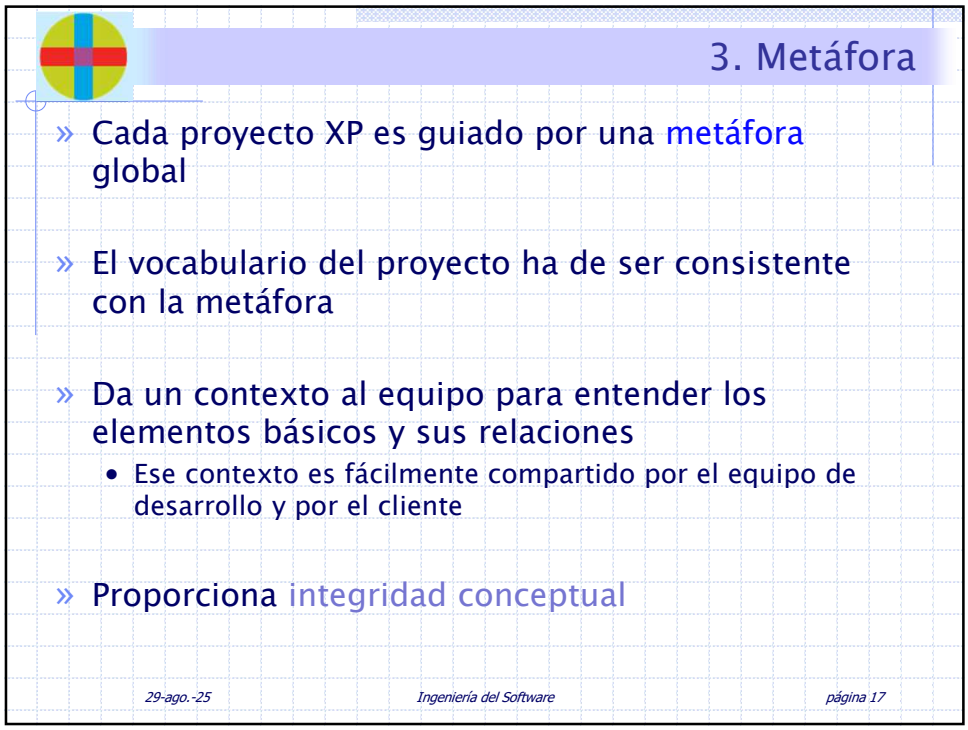
15

## 2. Entregas pequeñas

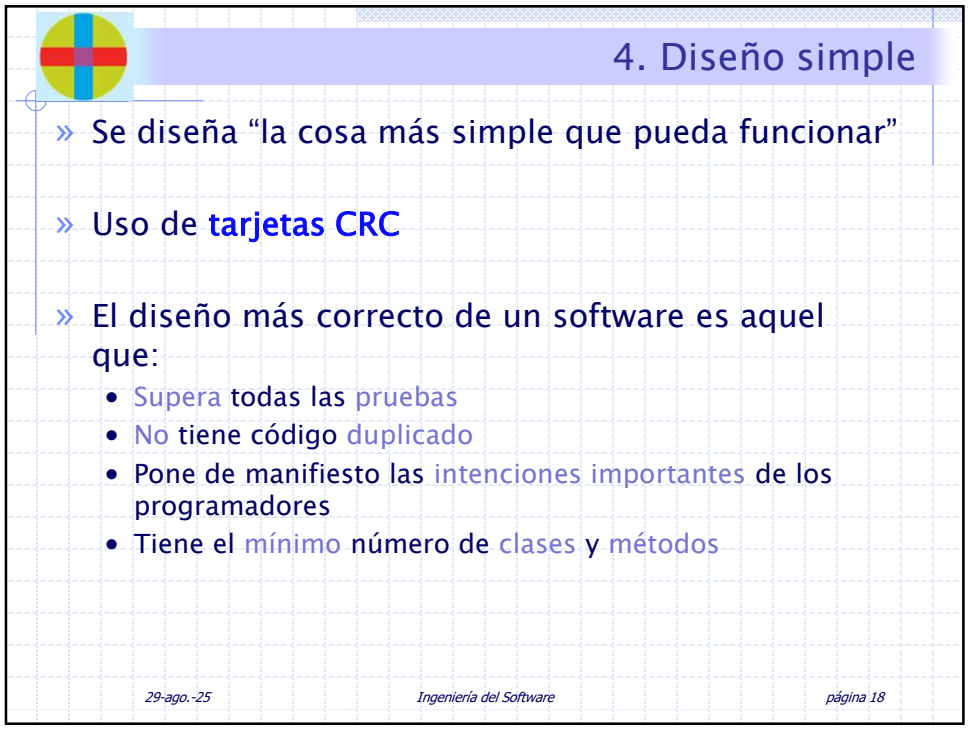
- La idea es poner rápidamente en producción un sistema muy sencillo, y después entregar nuevas versiones en ciclos muy cortos
- Cada entrega es lo más rápida y pequeña posible:
  - Contiene los requisitos más valiosos del sistema (básicos)
  - Reduce el riesgo → mayor realimentación desde el cliente, y más frecuente
- Minimizar el número de requisitos que componen una entrega → no realizar requisitos a medias



16


3. Metáfora
» Cada proyecto XP es guiado por una metáfora global
» El vocabulario del proyecto ha de ser consistente con la metáfora
» Da un contexto al equipo para entender los elementos básicos y sus relaciones
• Ese contexto es fácilmente compartido por el equipo de desarrollo y por el cliente
» Proporciona integridad conceptual
29-ago.-25 Ingeniería del Software página 17

17


4. Diseño simple
» Se diseña "la cosa más simple que pueda funcionar"
» Uso de tarjetas CRC
» El diseño más correcto de un software es aquel que:
• Supera todas las pruebas
• No tiene código duplicado
• Pone de manifiesto las intenciones importantes de los programadores
• Tiene el mínimo número de clases y métodos
29-ago.-25 Ingeniería del Software página 18

18

## 5. Pruebas

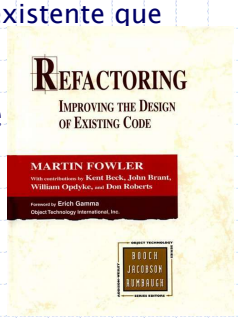
- Una característica sin una prueba automática asociada simplemente no existe
- Las pruebas unitarias se escriben ANTES que el código, y al principio deben fallar
  - Pruebas **automatizadas**
- Permiten el desarrollo de proyectos de forma rápida y segura
  - Equipo de desarrollo → **pruebas unitarias**
  - Cliente → **pruebas funcionales**
- Resultado → un programa cada vez más seguro



19

## 6. Recodificación

- **Recodificación** = mejora del código
- Antes de añadir un cambio, hay que preguntarse:
  - ¿Hay alguna forma de modificar el código existente que haga más fácil añadir el cambio?
- Después de añadir un cambio, hay que preguntarse:
  - ¿Se puede simplificar el programa?
    - eliminar complejidad innecesaria
    - eliminar duplicación de código
- Sólo se recodifica cuando el sistema lo requiere




www.refactoring.com



20

## 7. Programación en parejas

- Todo el código se escribe **en parejas**
  - Dos personas trabajando en la *misma máquina*
  - Roles en cada pareja
    - Uno piensa en la **táctica** (implementación)
    - Otro en la **estrategia**
      - ¿Es una buena aproximación?
      - ¿Va a fallar alguna prueba?
      - ¿Se puede hacer esto de forma más sencilla?

- Debe realizarse *dinámicamente* (cambio de parejas)
  - Se produce código de mayor calidad
  - Extiende el conocimiento
- "Se realiza el trabajo de una persona en casi la mitad del tiempo y mejor" (cuestionable)

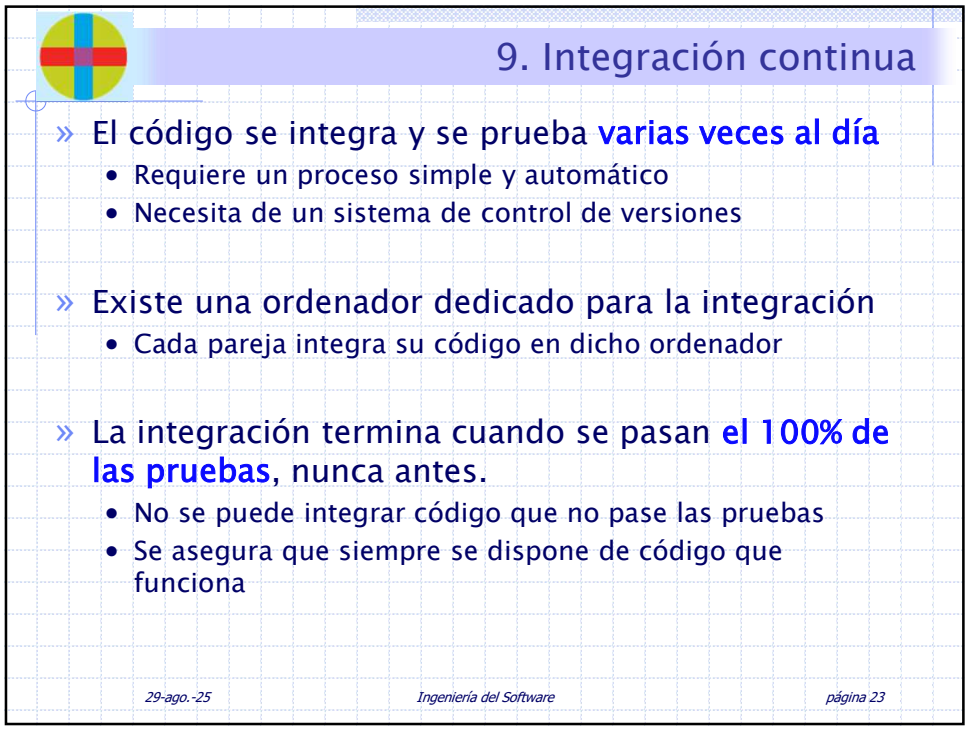
www.pairprogramming.com



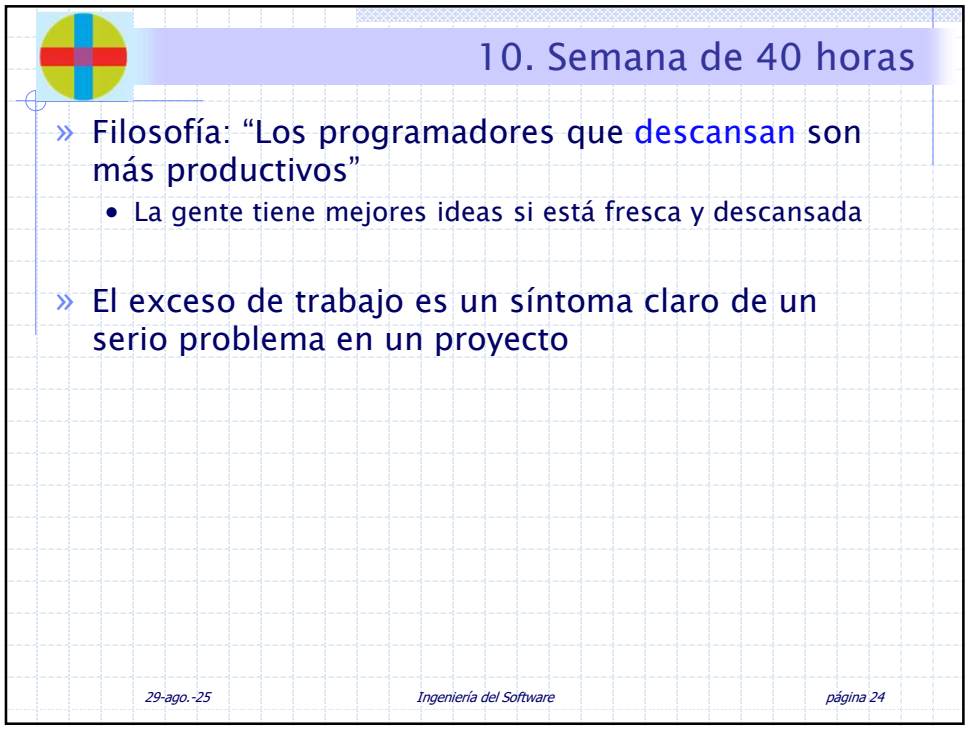


## 8. Propiedad colectiva

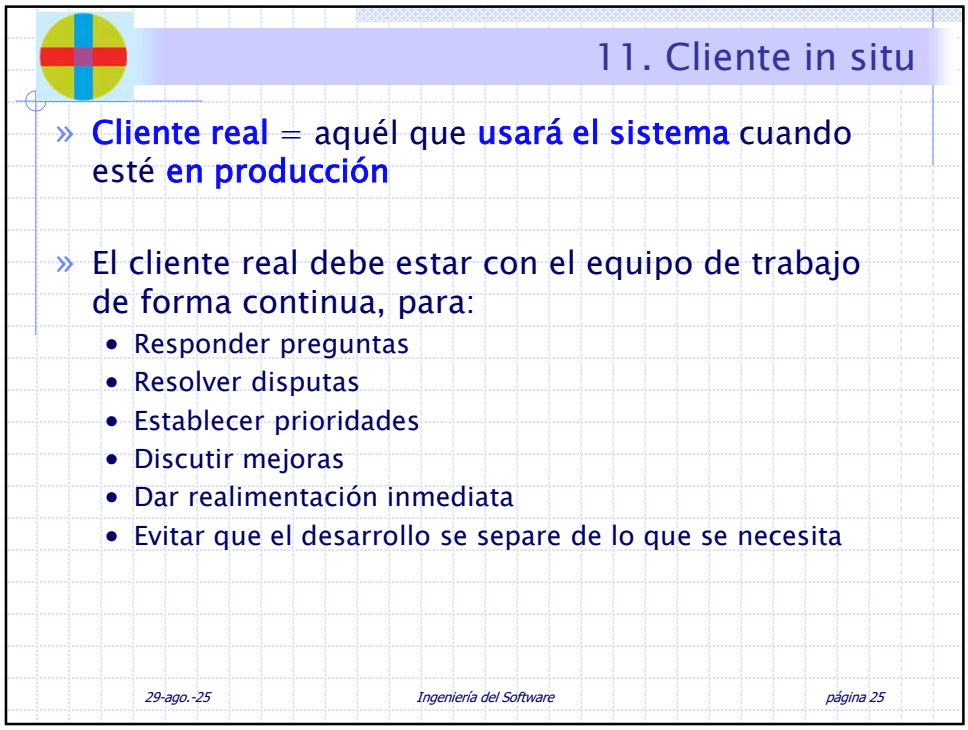
- Cualquiera puede modificar el código en cualquier momento
  - Se evitan cuellos de botella en la codificación
  - De hecho, si observa algún problema DEBE modificar el código para arreglarlo
- Todo el equipo de desarrollo
  - Asume las responsabilidades sobre el conjunto del sistema
  - Conoce algo sobre todas las partes y conoce muy bien aquéllas en las que trabaja
- El caos se evita mediante las pruebas automáticas


9. Integración continua
» El código se integra y se prueba varias veces al día
• Requiere un proceso simple y automático
• Necesita de un sistema de control de versiones
» Existe una ordenador dedicado para la integración
• Cada pareja integra su código en dicho ordenador
» La integración termina cuando se pasan el 100% de las pruebas, nunca antes.
• No se puede integrar código que no pase las pruebas
• Se asegura que siempre se dispone de código que funciona
29-ago.-25 Ingeniería del Software página 23

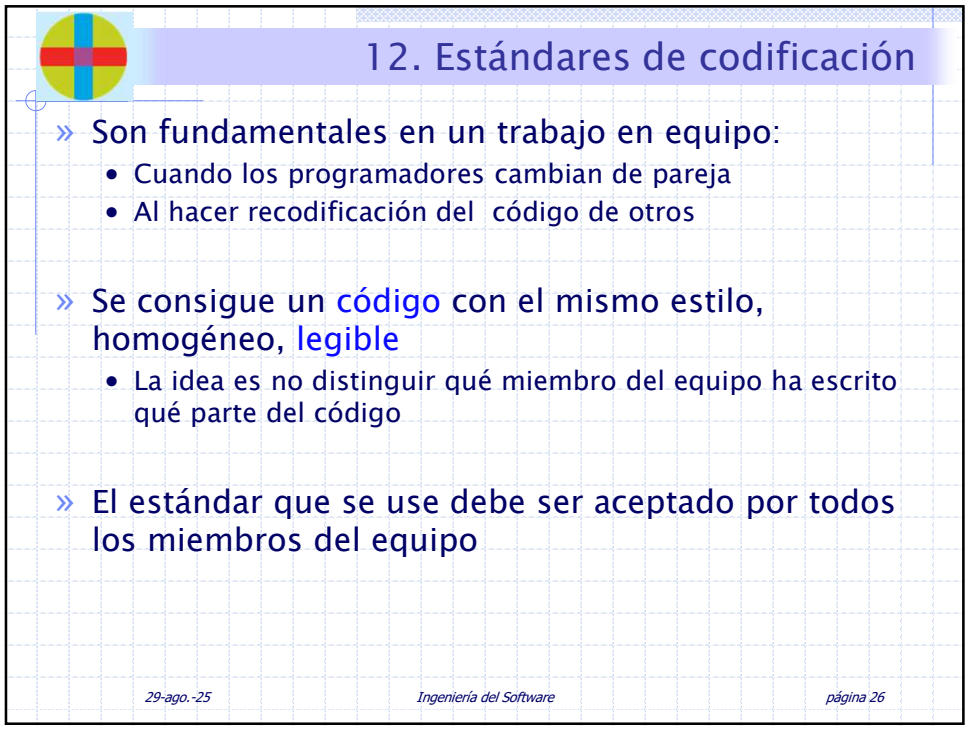
23


10. Semana de 40 horas
» Filosofía: “Los programadores que descansan son más productivos”
• La gente tiene mejores ideas si está fresca y descansada
» El exceso de trabajo es un síntoma claro de un serio problema en un proyecto
29-ago.-25 Ingeniería del Software página 24

24


11. Cliente in situ
» Cliente real = aquél que usará el sistema cuando esté en producción
» El cliente real debe estar con el equipo de trabajo de forma continua, para:
• Responder preguntas
• Resolver disputas
• Establecer prioridades
• Discutir mejoras
• Dar realimentación inmediata
• Evitar que el desarrollo se separe de lo que se necesita
29-ago.-25 Ingeniería del Software página 25

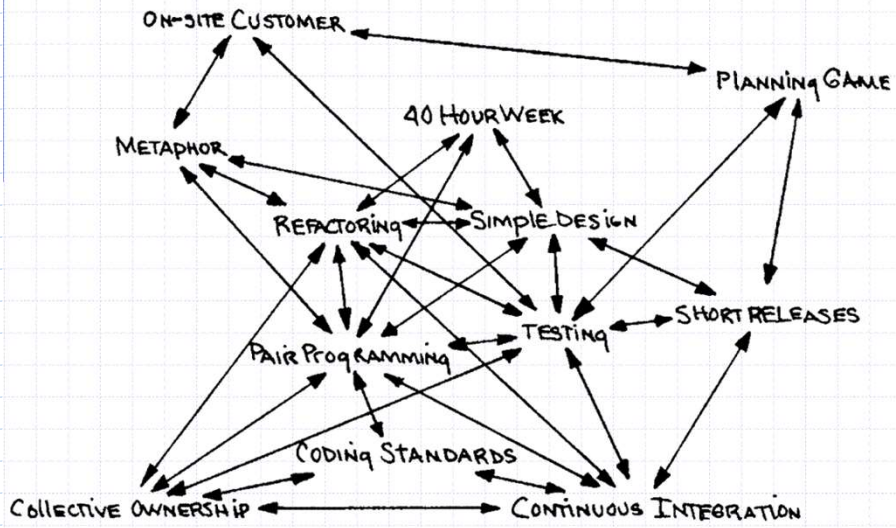
25


12. Estándares de codificación
» Son fundamentales en un trabajo en equipo:
• Cuando los programadores cambian de pareja
• Al hacer recodificación del código de otros
» Se consigue un código con el mismo estilo, homogéneo, legible
• La idea es no distinguir qué miembro del equipo ha escrito qué parte del código
» El estándar que se use debe ser aceptado por todos los miembros del equipo
29-ago.-25 Ingeniería del Software página 26

26

## Interacción entre las 12 prácticas

» Diagrama de interacción (Kent Beck):






27

## Segunda edición (2ª ed.)

» En la **segunda edición** (2ªed.) del libro "*Extreme Programming Explained*":

» Aparece un **quinto valor** (**Respeto**)

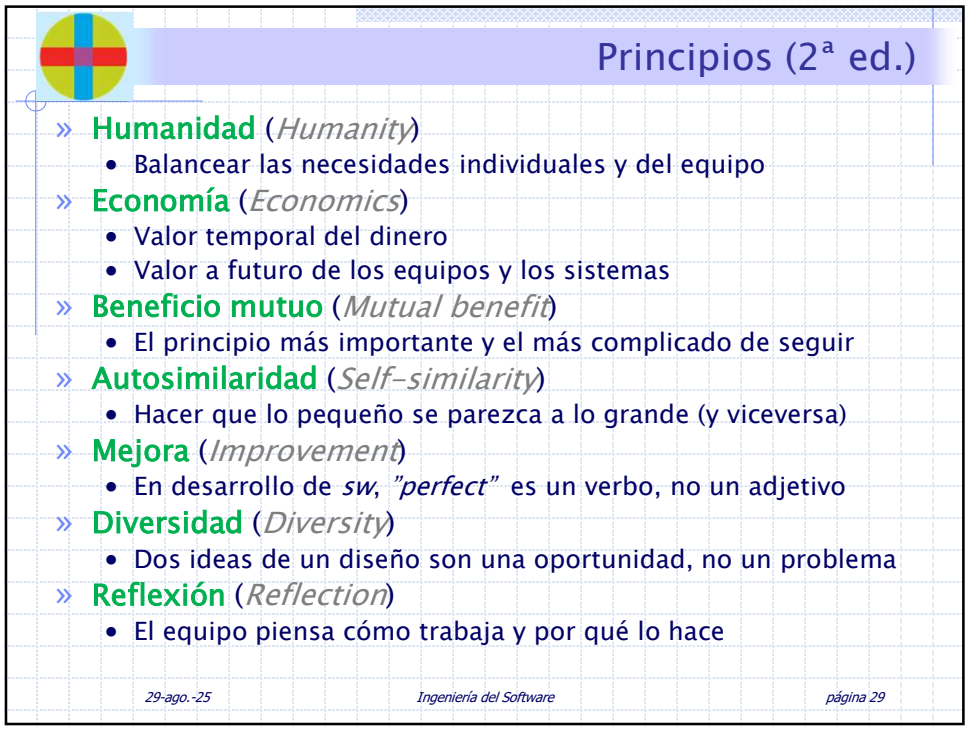
» Hay **más prácticas**, organizadas en dos grupos
- Prácticas **primarias** (*primary practices*)
- Prácticas **corolario** (*corollary practices*)

» Ambas trabajan en conjunto para que adoptar XP sea algo más incremental que el "hacer todas las 12 prácticas" de la primera edición

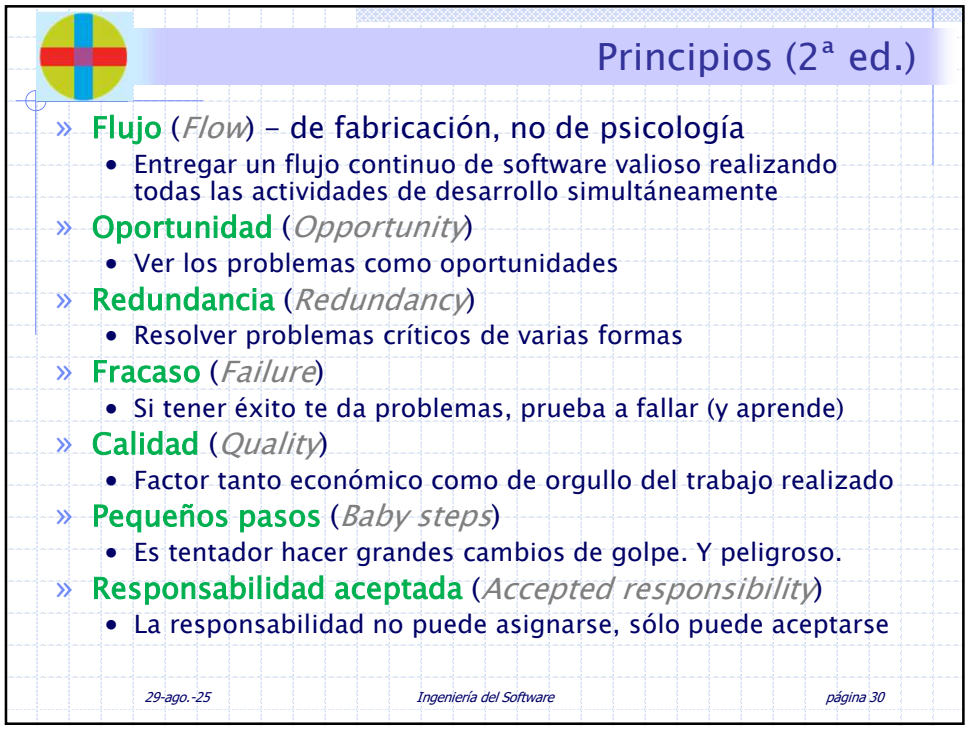
» La aproximación a la **planificación** es **más sencilla**



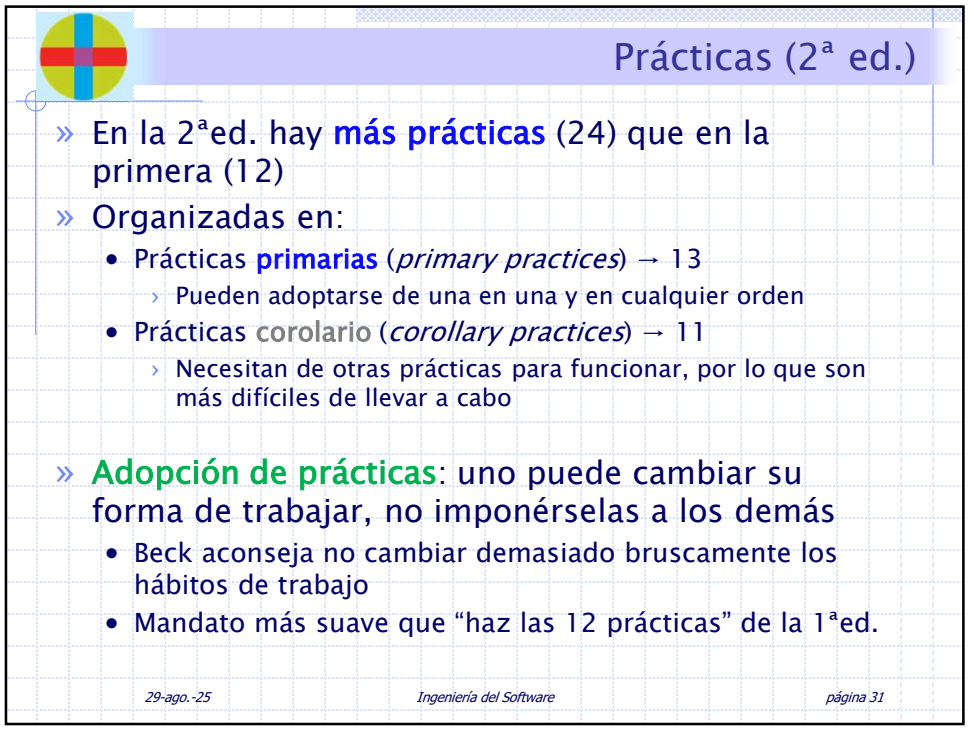
28


Principios (2ª ed.)
» Humanidad (Humanity)
• Balancear las necesidades individuales y del equipo
» Economía (Economics)
• Valor temporal del dinero
• Valor a futuro de los equipos y los sistemas
» Beneficio mutuo (Mutual benefit)
• El principio más importante y el más complicado de seguir
» Autosimilaridad (Self-similarity)
• Hacer que lo pequeño se parezca a lo grande (y viceversa)
» Mejora (Improvement)
• En desarrollo de sw, "perfect" es un verbo, no un adjetivo
» Diversidad (Diversity)
• Dos ideas de un diseño son una oportunidad, no un problema
» Reflexión (Reflection)
• El equipo piensa cómo trabaja y por qué lo hace
29-ago.-25 Ingeniería del Software página 29

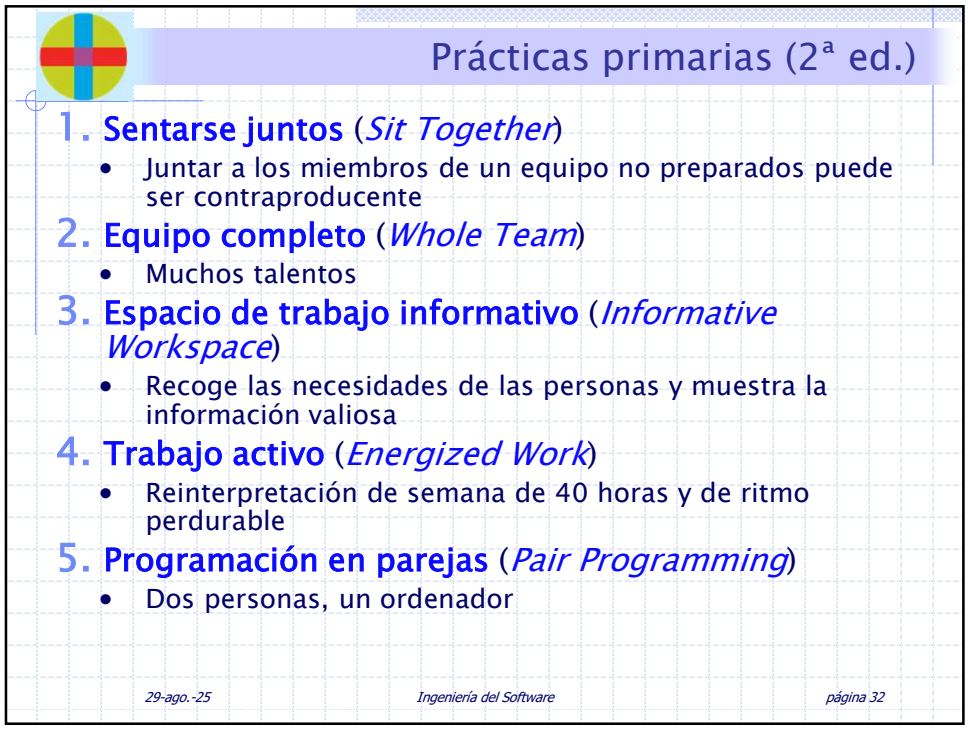
29


Principios (2ª ed.)
» Flujo (Flow) – de fabricación, no de psicología
• Entregar un flujo continuo de software valioso realizando todas las actividades de desarrollo simultáneamente
» Oportunidad (Opportunity)
• Ver los problemas como oportunidades
» Redundancia (Redundancy)
• Resolver problemas críticos de varias formas
» Fracaso (Failure)
• Si tener éxito te da problemas, prueba a fallar (y aprende)
» Calidad (Quality)
• Factor tanto económico como de orgullo del trabajo realizado
» Pequeños pasos (Baby steps)
• Es tentador hacer grandes cambios de golpe. Y peligroso.
» Responsabilidad aceptada (Accepted responsibility)
• La responsabilidad no puede asignarse, sólo puede aceptarse
29-ago.-25 Ingeniería del Software página 30

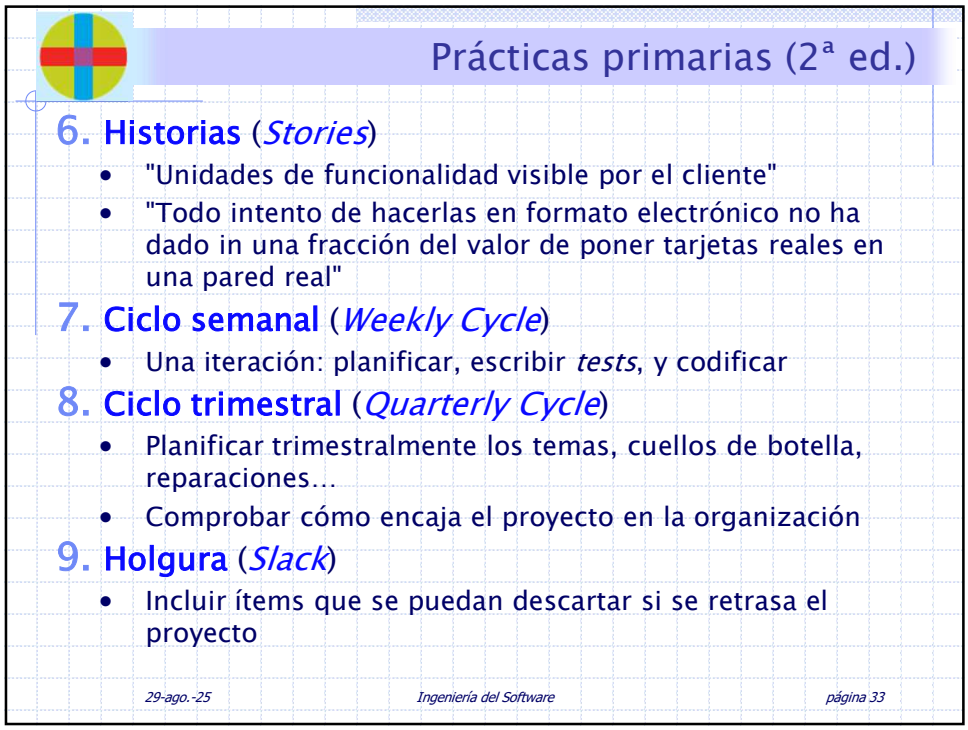
30


Prácticas (2ª ed.)
» En la 2ªed. hay más prácticas (24) que en la primera (12)
» Organizadas en:
• Prácticas primarias (primary practices) → 13
› Pueden adoptarse de una en una y en cualquier orden
• Prácticas corolario (corollary practices) → 11
› Necesitan de otras prácticas para funcionar, por lo que son más difíciles de llevar a cabo
» Adopción de prácticas: uno puede cambiar su forma de trabajar, no imponérselas a los demás
• Beck aconseja no cambiar demasiado bruscamente los hábitos de trabajo
• Mandato más suave que "haz las 12 prácticas" de la 1ªed.
29-ago.-25 Ingeniería del Software página 31

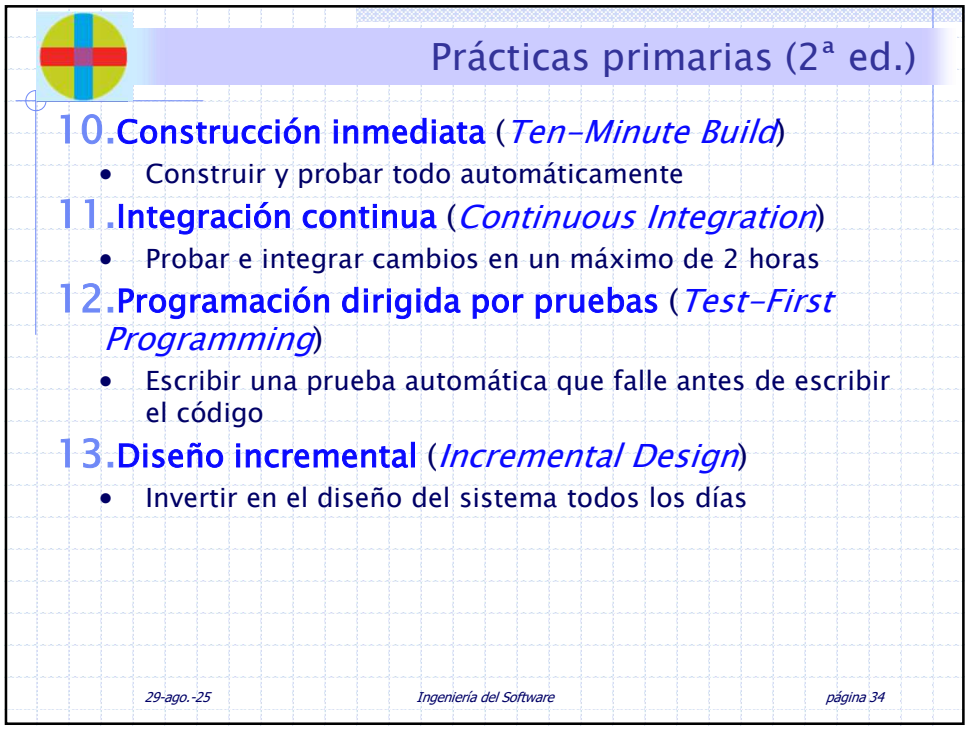
31


Prácticas primarias (2ª ed.)
1. Sentarse juntos (Sit Together)
• Juntar a los miembros de un equipo no preparados puede ser contraproducente
2. Equipo completo (Whole Team)
• Muchos talentos
3. Espacio de trabajo informativo (Informative Workspace)
• Recoge las necesidades de las personas y muestra la información valiosa
4. Trabajo activo (Energized Work)
• Reinterpretación de semana de 40 horas y de ritmo perdurable
5. Programación en parejas (Pair Programming)
• Dos personas, un ordenador
29-ago.-25 Ingeniería del Software página 32

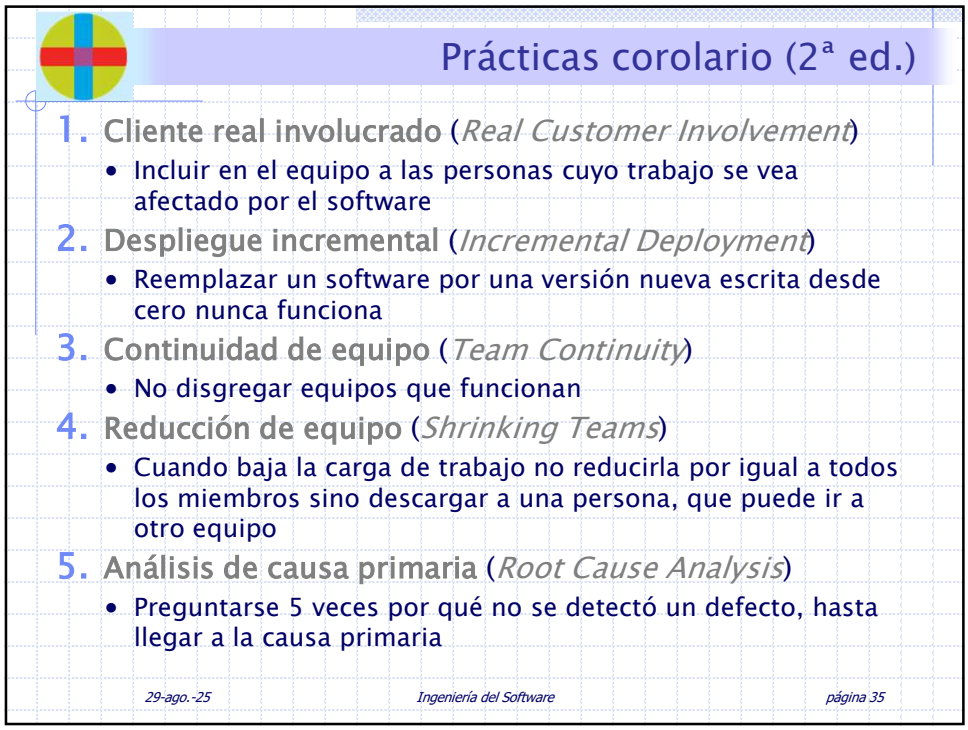
32


Prácticas primarias (2ª ed.)
6. Historias (Stories)
• "Unidades de funcionalidad visible por el cliente"
• "Todo intento de hacerlas en formato electrónico no ha dado in una fracción del valor de poner tarjetas reales en una pared real"
7. Ciclo semanal (Weekly Cycle)
• Una iteración: planificar, escribir tests, y codificar
8. Ciclo trimestral (Quarterly Cycle)
• Planificar trimestralmente los temas, cuellos de botella, reparaciones…
• Comprobar cómo encaja el proyecto en la organización
9. Holgura (Slack)
• Incluir ítems que se puedan descartar si se retrasa el proyecto
29-ago.-25 Ingeniería del Software página 33

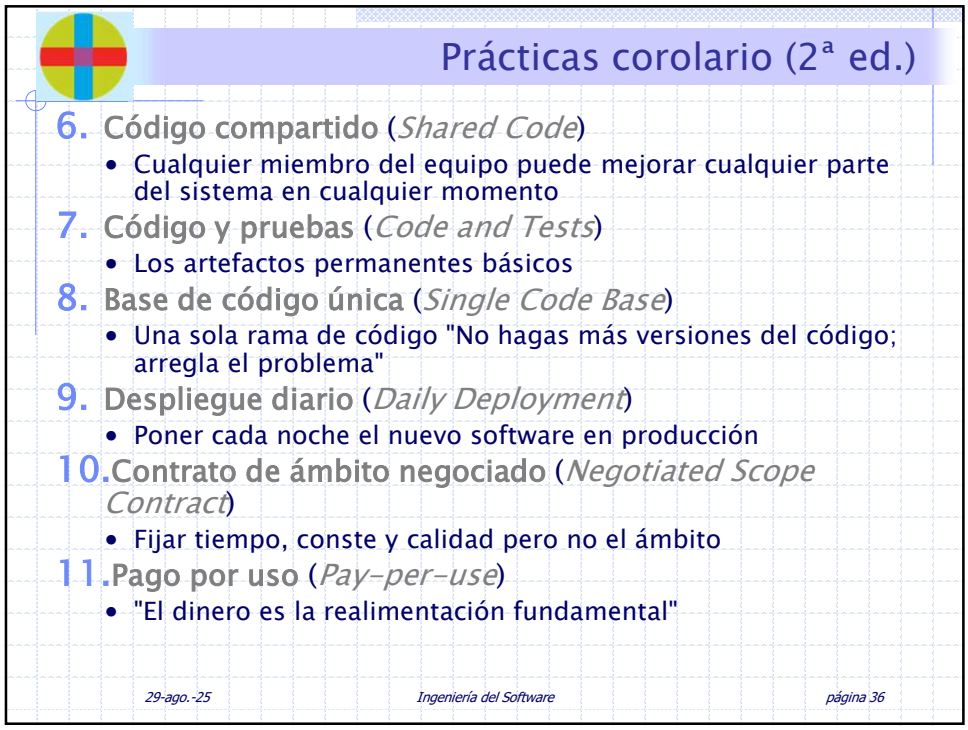
33


Prácticas primarias (2ª ed.)
10. Construcción inmediata (Ten-Minute Build)
• Construir y probar todo automáticamente
11. Integración continua (Continuous Integration)
• Probar e integrar cambios en un máximo de 2 horas
12. Programación dirigida por pruebas (Test-First Programming)
• Escribir una prueba automática que falle antes de escribir el código
13. Diseño incremental (Incremental Design)
• Invertir en el diseño del sistema todos los días
29-ago.-25 Ingeniería del Software página 34

34


Prácticas corolario (2ª ed.)
1. Cliente real involucrado (Real Customer Involvement)
• Incluir en el equipo a las personas cuyo trabajo se vea afectado por el software
2. Despliegue incremental (Incremental Deployment)
• Reemplazar un software por una versión nueva escrita desde cero nunca funciona
3. Continuidad de equipo (Team Continuity)
• No disgregar equipos que funcionan
4. Reducción de equipo (Shrinking Teams)
• Cuando baja la carga de trabajo no reducirla por igual a todos los miembros sino descargar a una persona, que puede ir a otro equipo
5. Análisis de causa primaria (Root Cause Analysis)
• Preguntarse 5 veces por qué no se detectó un defecto, hasta llegar a la causa primaria
29-ago.-25 Ingeniería del Software página 35

35


Prácticas corolario (2ª ed.)
6. Código compartido (Shared Code)
• Cualquier miembro del equipo puede mejorar cualquier parte del sistema en cualquier momento
7. Código y pruebas (Code and Tests)
• Los artefactos permanentes básicos
8. Base de código única (Single Code Base)
• Una sola rama de código "No hagas más versiones del código; arregla el problema"
9. Despliegue diario (Daily Deployment)
• Poner cada noche el nuevo software en producción
10. Contrato de ámbito negociado (Negotiated Scope Contract)
• Fijar tiempo, conste y calidad pero no el ámbito
11. Pago por uso (Pay-per-use)
• "El dinero es la realimentación fundamental"
29-ago.-25 Ingeniería del Software página 36

36

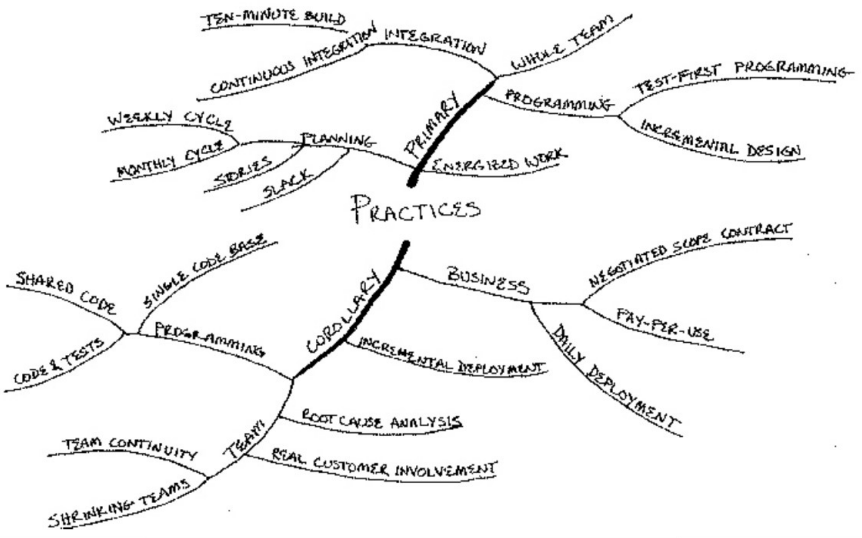
## Comparación con las 12 prácticas

1. El juego de la planificación: más sencilla en 2ªed.; ver Ciclo trimestral y Ciclo semanal
2. Entregas pequeñas: más explícitas en Despliegue incremental y Despliegue diario
3. Metáfora: siempre fue la peor comprendida; desaparece en 2ªed.
4. Diseño simple: Diseño incremental y, en menor medida, Base de código única
5. Pruebas: Programación dirigida por pruebas
6. Recodificación: no aparece explícitamente en 2ªed.; implícita en Diseño incremental
7. Programación en parejas: idéntica
8. Propiedad colectiva: ahora se llama Código compartido
9. Integración continua: idéntica
10. Semana de 40 horas: Trabajo activo y, en cierta medida, Holgura
11. Cliente in situ: cubierta por Sentarse juntos, Equipo completo y Cliente real involucrado
12. Estándares de codificación: no explícita; se deduce de Código compartido y Programación en parejas



37

## Interacción entre las prácticas (2ª ed.)






38

## Estrategias XP

- Vamos a revisar cinco estrategias de XP

1. Instalaciones físicas
2. Planificación
3. Desarrollo
4. Diseño
5. Pruebas

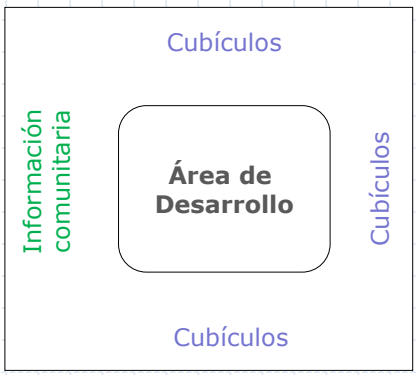


39

## 1. Instalaciones físicas

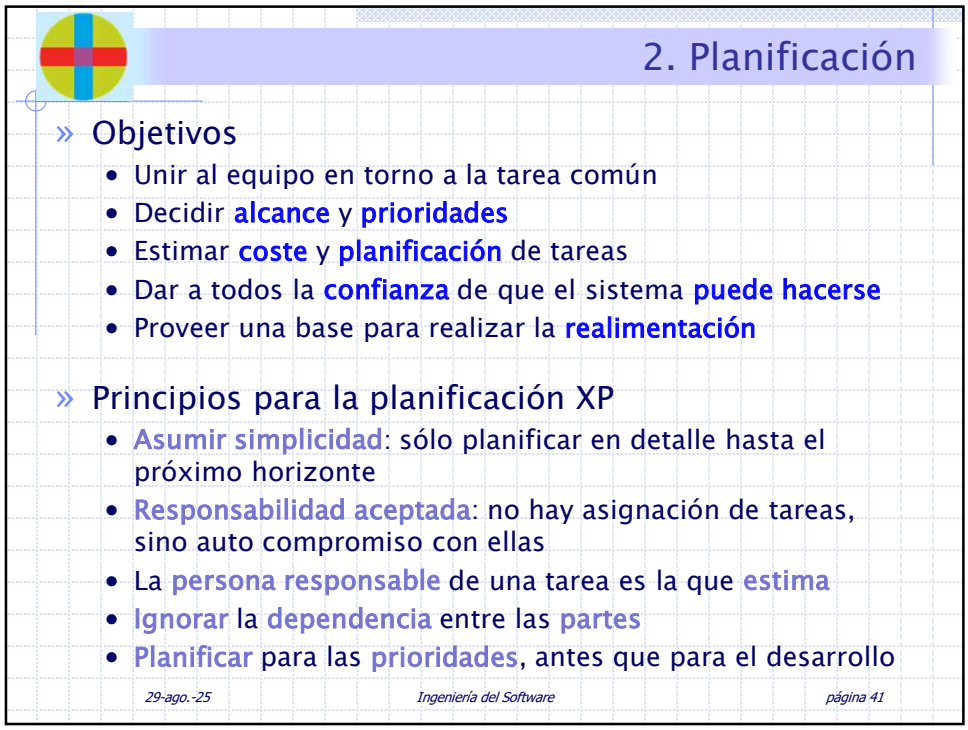
- **Planta abierta**, sin grandes muros
  - En la periferia: mini-cubículos para trabajo individual
    - pizarras
  - En el centro: PC's más poderosos (dos puestos por PC)
    - mesas

Espacio de trabajo del proyecto C3 de Chrysler

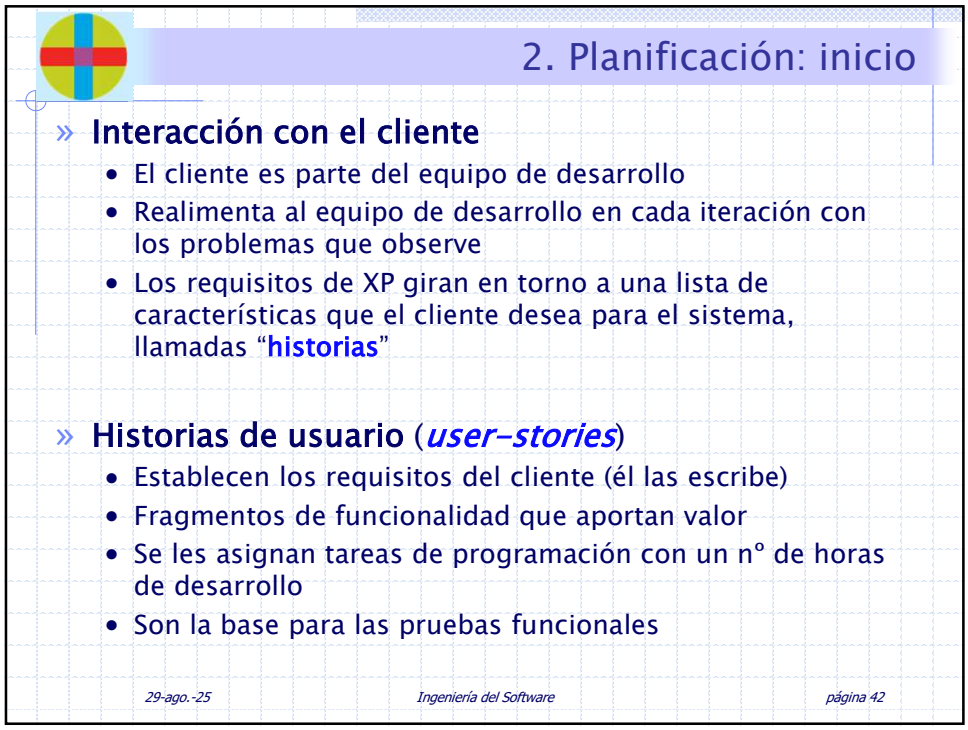





40

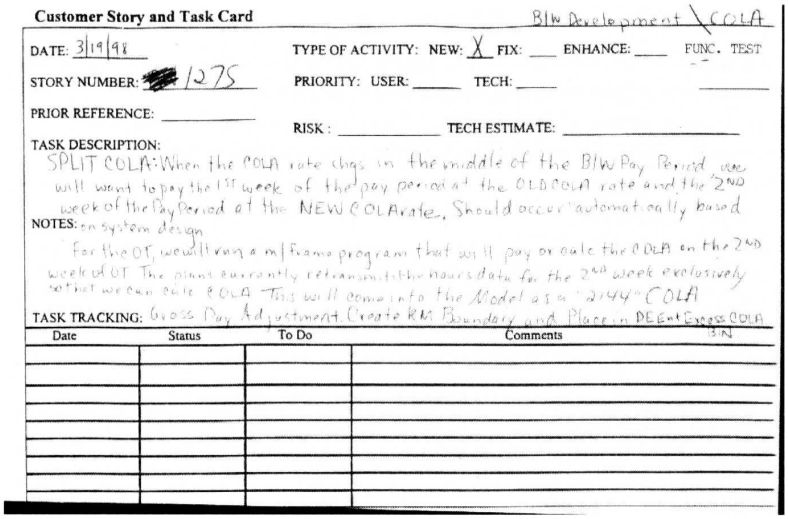

2. Planificación
» Objetivos
• Unir al equipo en torno a la tarea común
• Decidir alcance y prioridades
• Estimar coste y planificación de tareas
• Dar a todos la confianza de que el sistema puede hacerse
• Proveer una base para realizar la realimentación
» Principios para la planificación XP
• Asumir simplicidad: sólo planificar en detalle hasta el próximo horizonte
• Responsabilidad aceptada: no hay asignación de tareas, sino auto compromiso con ellas
• La persona responsable de una tarea es la que estima
• Ignorar la dependencia entre las partes
• Planificar para las prioridades, antes que para el desarrollo
29-ago.-25 Ingeniería del Software página 41

41


2. Planificación: inicio
» Interacción con el cliente
• El cliente es parte del equipo de desarrollo
• Realimenta al equipo de desarrollo en cada iteración con los problemas que observe
• Los requisitos de XP giran en torno a una lista de características que el cliente desea para el sistema, llamadas "historias"
» Historias de usuario (user-stories)
• Establecen los requisitos del cliente (él las escribe)
• Fragmentos de funcionalidad que aportan valor
• Se les asignan tareas de programación con un nº de horas de desarrollo
• Son la base para las pruebas funcionales
29-ago.-25 Ingeniería del Software página 42

42

## 2. Planificación: una ficha de historia



Customer Story and Task Card

B1W Development \ COLA

DATE: 3/19/98

TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST

STORY NUMBER: 1275

PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___

RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:

SPLIT COLA: When the COLA rate chgs in the middle of the B1W Pay Period we will want to pay the 1ST week of the pay period at the OLD COLA rate and the 2ND week of the Pay Period at the NEW COLA rate. Should occur automatically based on system design.

NOTES:

For the OT, we will run a mainframe program that will pay or calc the COLA on the 2ND week of OT. The plant currently retains in the hours data for the 2ND week explosively so that we can calc COLA. This will come into the Model as a "2ND" COLA

TASK TRACKING: Gross Pay Adjustment. Create RBT Boundary and Place in DEEmtExpressCOLA B1N

| Date | Status | To Do | Comments |
|---|---|---|---|
| | | | |

FIGURE 6. A story card





## 2. Planificación: entregas

» Planificación por **entregas** (*releases*)
- Se priorizan aquellas historias que el cliente selecciona porque son más importantes para el negocio
- De esta forma, tanto el equipo de gestión, el equipo de desarrollo y el cliente se sientan parte de la decisión

» **Entregas**:
- Muchas y frecuentes
- Lo más pequeñas posible
- Se dividen en iteraciones (iteración = 2 ó 3 semanas)
- Están compuestas por historias
- A los desarrolladores se les asignan las tareas de las historias que componen la entrega
  - › Esto no debe significar horas extras para los desarrolladores

## 2. El juego de la planificación

- Jugado por dos partes: **clientes** y **desarrolladores**
- Objetivo: **maximizar el valor** del software producido por el equipo
- Estrategia:
  - **invertir lo menos posible** para
  - poner la **funcionalidad más valiosa** en producción
  - lo **antes posible**
  - ... junto con las estrategias de **Desarrollo** y **Pruebas**





## 2. El juego de la planificación (II)

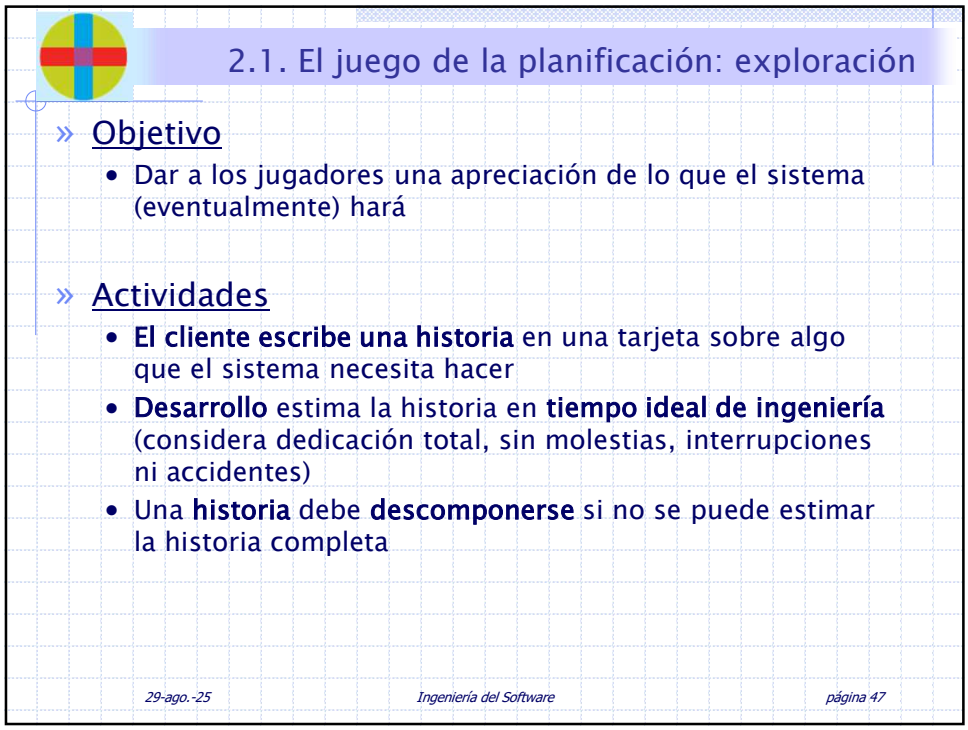
- Piezas
  - *Tarjetas de historias*
- Jugadores
  - **Clientes**
    - Usuarios reales del producto
    - Grupo de enfoque
    - Vendedores
  - **Desarrolladores**
- Actividades
  - **Exploración**: hallar las nuevas cosas que el sistema podría hacer
  - **Compromiso**: decidir qué subconjunto de requisitos se va a acometer
  - **Ajustar**: el desarrollo a medida que la realidad moldea el plan

| | | | |
|---|---|---|---|
| **Story Title:** | Generate a Unique Track Number | **Priority:** | H |
| **Creation Date:** | 10/25/00 | **Risk:** | M |
| **Card No.:** | 11.7 | **IEDs:** | 3 |

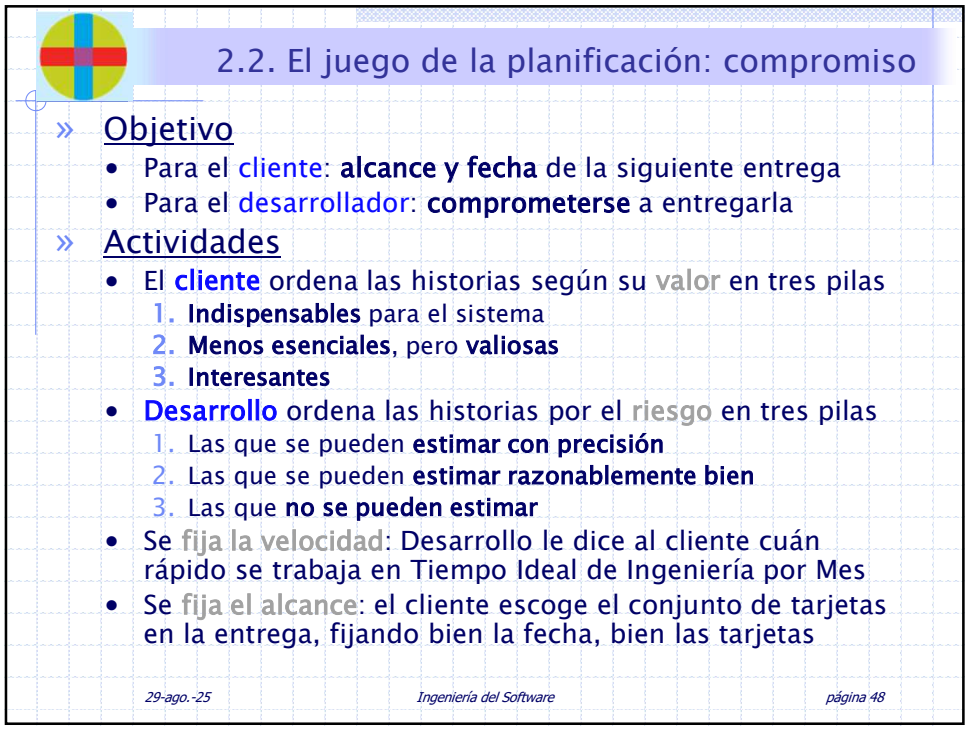
**Description:** There is a unique number assigned to each music track or media within a partner space so that music tracks or media are unique within Partners, and the Track Source ID is unique across all Partners.

- When a track is added its number is validated for uniqueness and to fit within 28 bits.
- The TrackSourceID is now:
  - 2 bits for Version
  - 12 bits for Partner Number
  - 28 bits for Track Number
- The Activity log parsing is revised for this new scheme

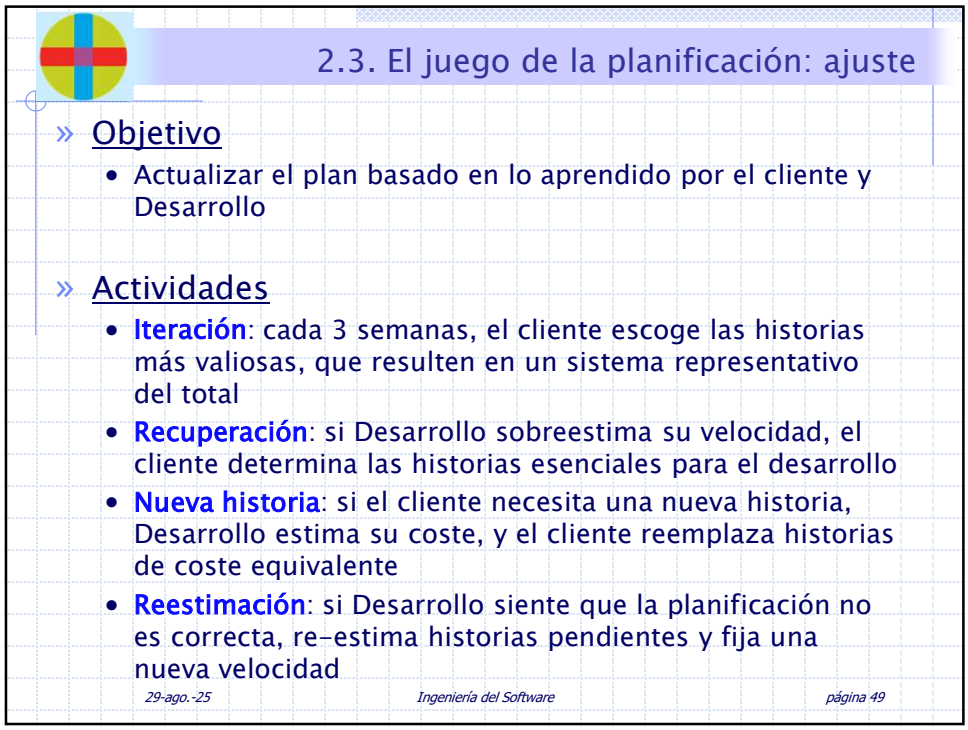
**Completion Criteria:** Verify that a unique number is generated when a new track is added, and it is 28 bits long. The TrackSourceID is unique and complies with the definition, and is properly parsed and written to the Activity Log.


2.1. El juego de la planificación: exploración
» Objetivo
• Dar a los jugadores una apreciación de lo que el sistema (eventualmente) hará
» Actividades
• El cliente escribe una historia en una tarjeta sobre algo que el sistema necesita hacer
• Desarrollo estima la historia en tiempo ideal de ingeniería (considera dedicación total, sin molestias, interrupciones ni accidentes)
• Una historia debe descomponerse si no se puede estimar la historia completa
29-ago.-25 Ingeniería del Software página 47

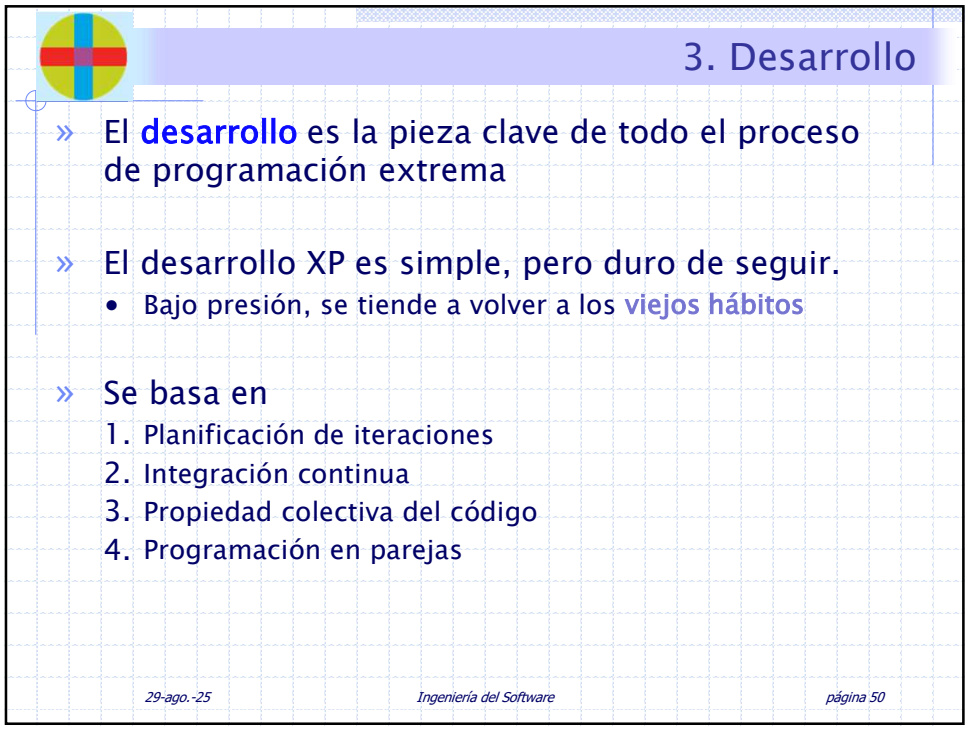
47


2.2. El juego de la planificación: compromiso
» Objetivo
• Para el cliente: alcance y fecha de la siguiente entrega
• Para el desarrollador: comprometerse a entregarla
» Actividades
• El cliente ordena las historias según su valor en tres pilas
1. Indispensables para el sistema
2. Menos esenciales, pero valiosas
3. Interesantes
• Desarrollo ordena las historias por el riesgo en tres pilas
1. Las que se pueden estimar con precisión
2. Las que se pueden estimar razonablemente bien
3. Las que no se pueden estimar
• Se fija la velocidad: Desarrollo le dice al cliente cuán rápido se trabaja en Tiempo Ideal de Ingeniería por Mes
• Se fija el alcance: el cliente escoge el conjunto de tarjetas en la entrega, fijando bien la fecha, bien las tarjetas
29-ago.-25 Ingeniería del Software página 48

48


2.3. El juego de la planificación: ajuste
Objetivo
Actualizar el plan basado en lo aprendido por el cliente y Desarrollo
Actividades
Iteración: cada 3 semanas, el cliente escoge las historias más valiosas, que resulten en un sistema representativo del total
Recuperación: si Desarrollo sobreestima su velocidad, el cliente determina las historias esenciales para el desarrollo
Nueva historia: si el cliente necesita una nueva historia, Desarrollo estima su coste, y el cliente reemplaza historias de coste equivalente
Reestimación: si Desarrollo siente que la planificación no es correcta, re-estima historias pendientes y fija una nueva velocidad
29-ago.-25 Ingeniería del Software página 49

49


3. Desarrollo
El desarrollo es la pieza clave de todo el proceso de programación extrema
El desarrollo XP es simple, pero duro de seguir.
Bajo presión, se tiende a volver a los viejos hábitos
Se basa en
1. Planificación de iteraciones
2. Integración continua
3. Propiedad colectiva del código
4. Programación en parejas
29-ago.-25 Ingeniería del Software página 50

50

## 3.1 Planificación de iteraciones

- Piezas
  - Se utilizan **tarjetas de tareas** (*task cards*) en vez de tarjetas de historias
- Jugadores
  - Cada uno de los programadores
- Las fases son similares al juego de la planificación

| Task Title: | Servlet for Track Numbers | End Date: | 11/3/00 |
|---|---|---|---|
| Card No.: | 11.7.1 | IED Plan: | 3 |
| Start Date: | 11/1/00 | IED Actual: | 3.5 |

**Description:** Create new servlet for assigning track numbers and modify Add/UpdateTrack to include track number and partner track name.

| To do: | Done: |
|---|---|
| Up-date Newdb.cmd to create track id | Y |
| Create new servle to get unique track id | Y |
| Up-date DTD for AddTrack | Y |
| Up-date AddTrack and ChangeTrack related classes | Y |
| Run all tests | Y |
| Up-date weblogic & deploy properties | Y |
| " API documentation & Sourcesafe | Y |
| Release code, send completion email, enter metrics | Y |



51

## 3.1 Planificación de iteraciones: ficha de tareas

Engineering Task Card

DATE: 3/17/98 BIW Based on Conversation w/REBECCA Smalltalk/Future NEW

STORY NUMBER: X923 SOFTWARE ENGINEER: _____ TASK ESTIMATE: _____

TASK DESCRIPTION:
Composite Bin - Regular Base Needs to Be Displayed on GUI. We have the hidden bin for Regular Base (Lost Time) to display NOT the auto gen bin but the BIN that composites the Auto Pay & the Lost Time. There is a separate composite bin started that needs to be completed??

SOFTWARE ENGINEER'S NOTES:

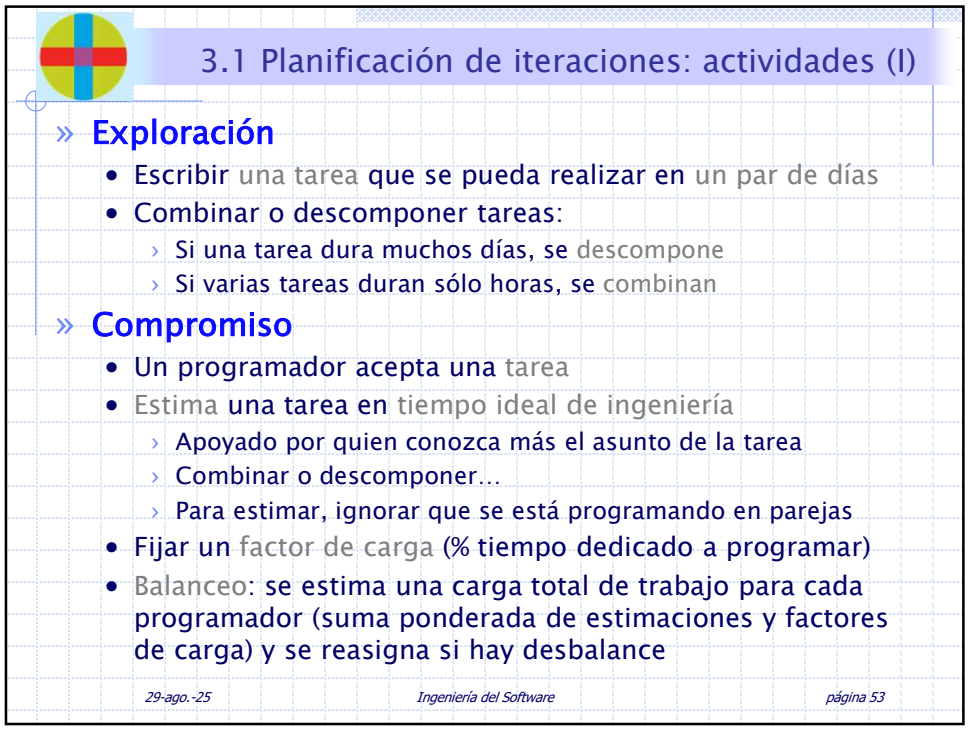
TASK TRACKING:

| Date | Done | To Do | Comments |
|---|---|---|---|
| | | | |

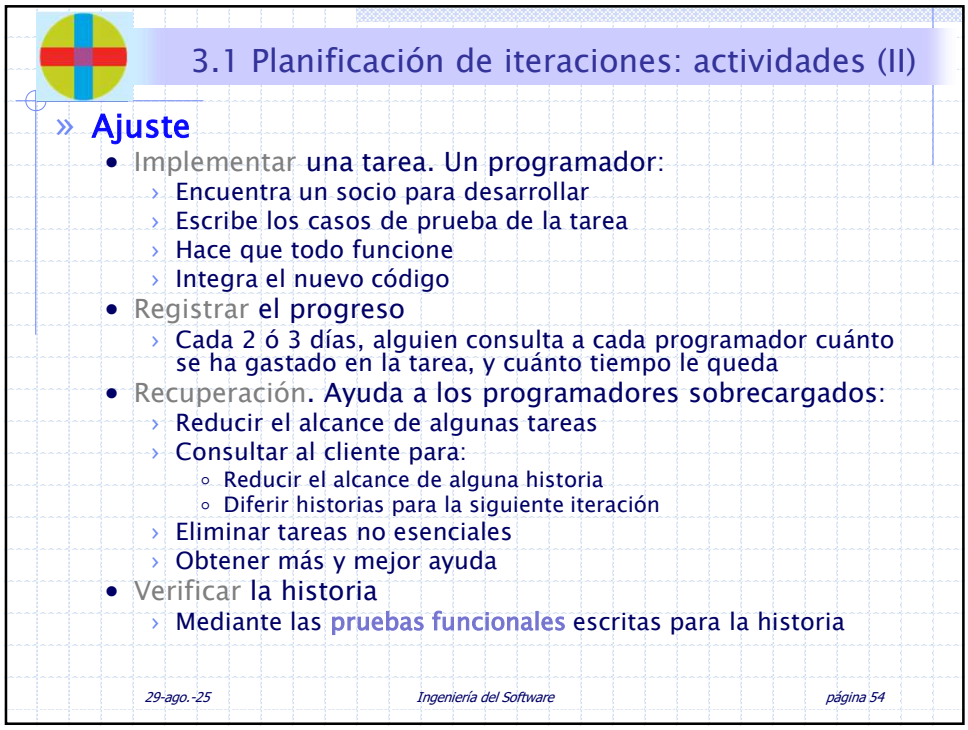
FIGURE 7. A task card



52


3.1 Planificación de iteraciones: actividades (I)
» Exploración
• Escribir una tarea que se pueda realizar en un par de días
• Combinar o descomponer tareas:
› Si una tarea dura muchos días, se descompone
› Si varias tareas duran sólo horas, se combinan
» Compromiso
• Un programador acepta una tarea
• Estima una tarea en tiempo ideal de ingeniería
› Apoyado por quien conozca más el asunto de la tarea
› Combinar o descomponer…
› Para estimar, ignorar que se está programando en parejas
• Fijar un factor de carga (% tiempo dedicado a programar)
• Balanceo: se estima una carga total de trabajo para cada programador (suma ponderada de estimaciones y factores de carga) y se reasigna si hay desbalance
29-ago.-25 Ingeniería del Software página 53

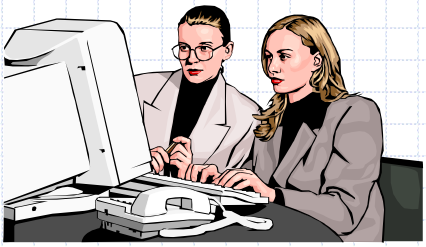
53

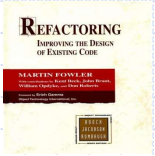

3.1 Planificación de iteraciones: actividades (II)
» Ajuste
• Implementar una tarea. Un programador:
› Encuentra un socio para desarrollar
› Escribe los casos de prueba de la tarea
› Hace que todo funcione
› Integra el nuevo código
• Registrar el progreso
› Cada 2 ó 3 días, alguien consulta a cada programador cuánto se ha gastado en la tarea, y cuánto tiempo le queda
• Recuperación. Ayuda a los programadores sobrecargados:
› Reducir el alcance de algunas tareas
› Consultar al cliente para:
◦ Reducir el alcance de alguna historia
◦ Diferir historias para la siguiente iteración
› Eliminar tareas no esenciales
› Obtener más y mejor ayuda
• Verificar la historia
› Mediante las pruebas funcionales escritas para la historia
29-ago.-25 Ingeniería del Software página 54

54

## 3.4 Desarrollo: programación en parejas

- La programación de tareas se realiza **por parejas**
- La pareja diseña, prueba, implementa e integra el código de la tarea
- Código dirigido por las pruebas
- Código modular, intentando recodificar siempre que sea posible



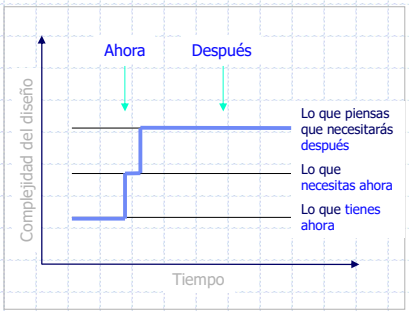




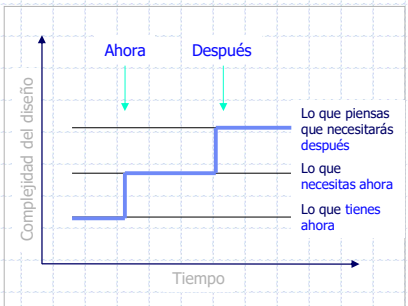

55

## 4. Diseño XP

- Lo más simple que corra la suite de pruebas actual
  - Es decir, **lo más simple que pueda funcionar**
- El diseño se debe revisar y mejorar continuamente
  - La comunicación es la clave (metáfora)
  - Uso de metodologías como las *tarjetas CRC*



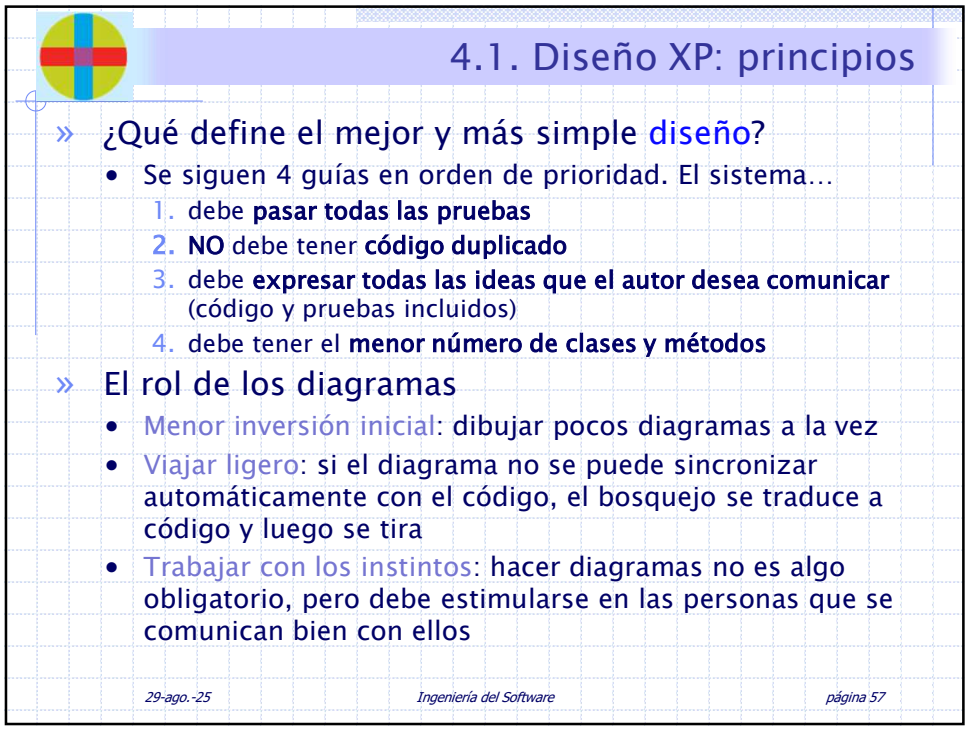

En un proyecto común



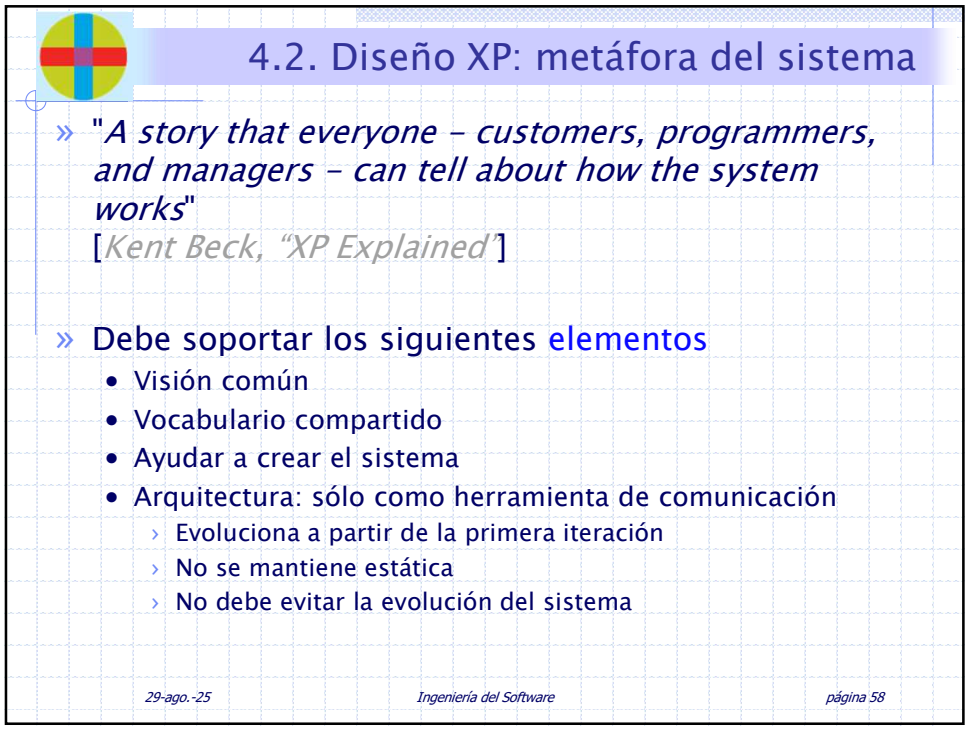

En XP



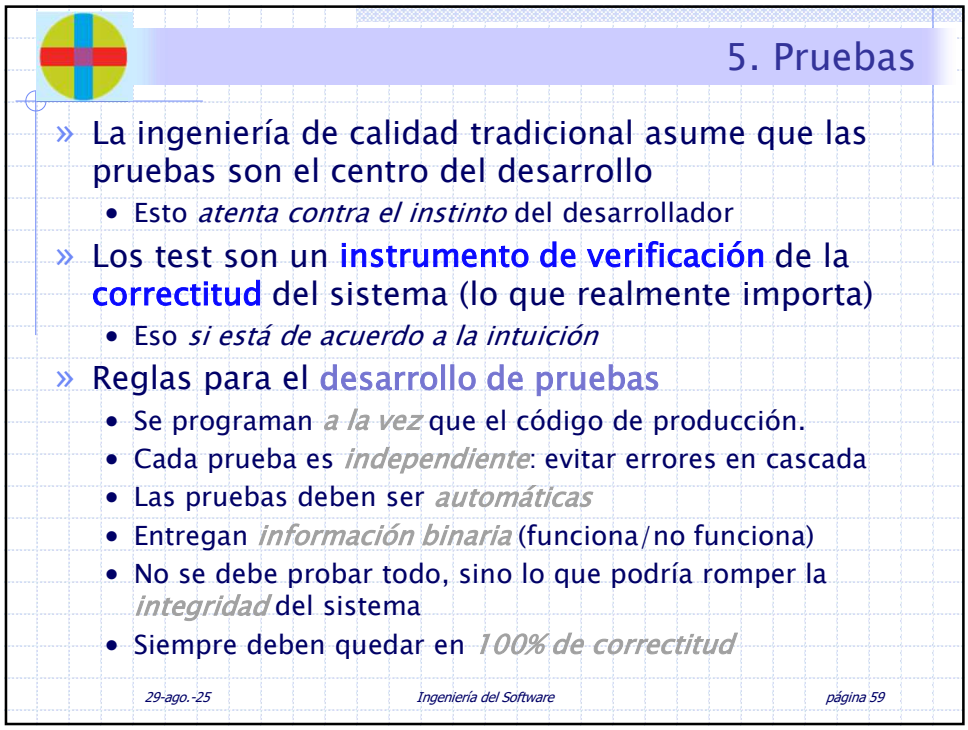
56


4.1. Diseño XP: principios
» ¿Qué define el mejor y más simple diseño?
• Se siguen 4 guías en orden de prioridad. El sistema…
1. debe pasar todas las pruebas
2. NO debe tener código duplicado
3. debe expresar todas las ideas que el autor desea comunicar (código y pruebas incluidos)
4. debe tener el menor número de clases y métodos
» El rol de los diagramas
• Menor inversión inicial: dibujar pocos diagramas a la vez
• Viajar ligero: si el diagrama no se puede sincronizar automáticamente con el código, el bosquejo se traduce a código y luego se tira
• Trabajar con los instintos: hacer diagramas no es algo obligatorio, pero debe estimularse en las personas que se comunican bien con ellos
29-ago.-25 Ingeniería del Software página 57

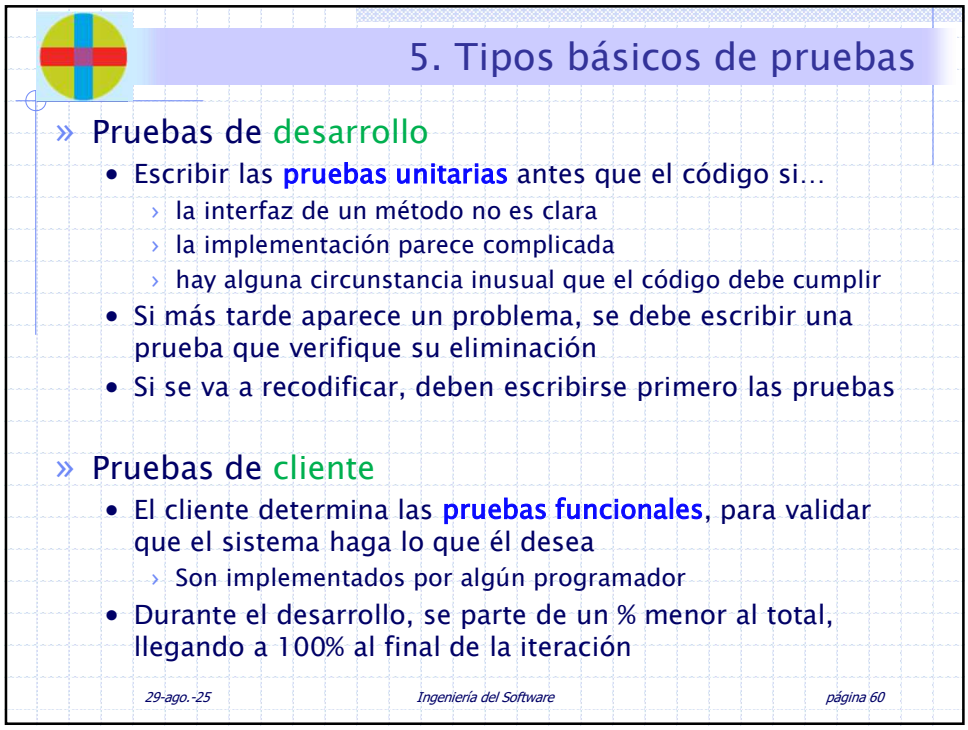
57


4.2. Diseño XP: metáfora del sistema
» "A story that everyone – customers, programmers, and managers – can tell about how the system works"
[Kent Beck, "XP Explained"]
» Debe soportar los siguientes elementos
• Visión común
• Vocabulario compartido
• Ayudar a crear el sistema
• Arquitectura: sólo como herramienta de comunicación
› Evoluciona a partir de la primera iteración
› No se mantiene estática
› No debe evitar la evolución del sistema
29-ago.-25 Ingeniería del Software página 58

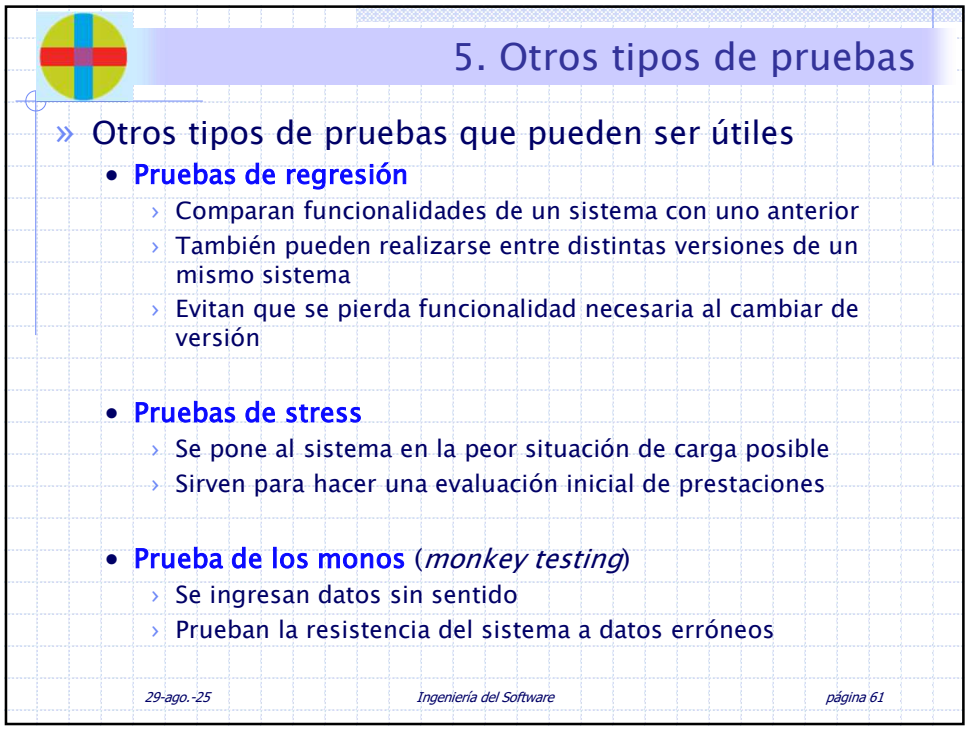
58


5. Pruebas
» La ingeniería de calidad tradicional asume que las pruebas son el centro del desarrollo
• Esto *atenta contra el instinto* del desarrollador
» Los test son un **instrumento de verificación** de la **correctitud** del sistema (lo que realmente importa)
• Eso *si está de acuerdo a la intuición*
» Reglas para el desarrollo de pruebas
• Se programan *a la vez* que el código de producción.
• Cada prueba es *independiente*: evitar errores en cascada
• Las pruebas deben ser *automáticas*
• Entregan *información binaria* (funciona/no funciona)
• No se debe probar todo, sino lo que podría romper la *integridad* del sistema
• Siempre deben quedar en *100% de correctitud*
29-ago.-25 Ingeniería del Software página 59

59


5. Tipos básicos de pruebas
» Pruebas de desarrollo
• Escribir las **pruebas unitarias** antes que el código si…
› la interfaz de un método no es clara
› la implementación parece complicada
› hay alguna circunstancia inusual que el código debe cumplir
• Si más tarde aparece un problema, se debe escribir una prueba que verifique su eliminación
• Si se va a recodificar, deben escribirse primero las pruebas
» Pruebas de cliente
• El cliente determina las **pruebas funcionales**, para validar que el sistema haga lo que él desea
› Son implementados por algún programador
• Durante el desarrollo, se parte de un % menor al total, llegando a 100% al final de la iteración
29-ago.-25 Ingeniería del Software página 60

60


5. Otros tipos de pruebas
Otros tipos de pruebas que pueden ser útiles
Pruebas de regresión
Comparan funcionalidades de un sistema con uno anterior
También pueden realizarse entre distintas versiones de un mismo sistema
Evitan que se pierda funcionalidad necesaria al cambiar de versión
Pruebas de stress
Se pone al sistema en la peor situación de carga posible
Sirven para hacer una evaluación inicial de prestaciones
Prueba de los monos (monkey testing)
Se ingresan datos sin sentido
Prueban la resistencia del sistema a datos erróneos
29-ago.-25
Ingeniería del Software
página 61

61


Roles XP
Roles que aparecen en XP
Jefe de Proyecto (Manager)
Cliente (Customer)
Programador (Programmer)
Encargado de Pruebas (Tester)
Encargado de Seguimiento (Tracker)
Entrenador (Coach)
Consultor (Consultant)
29-ago.-25
Ingeniería del Software
página 62

62

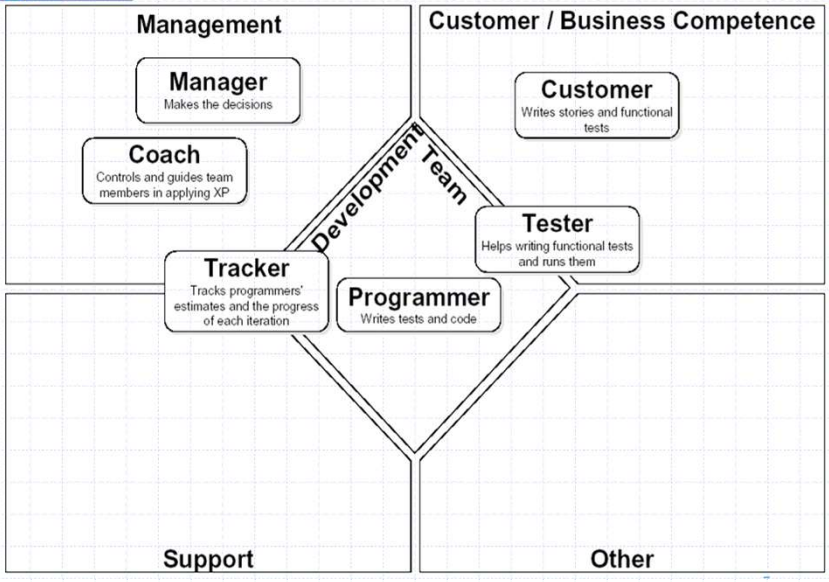

Roles XP (I)
» Programador (*Programmer*)
• Responsable de decisiones técnicas
• Responsable de construir el sistema
• Sin distinción entre analistas, diseñadores o codificadores
• En XP, los programadores diseñan, programan y realizan las pruebas
» Jefe de Proyecto (*Manager*)
• Organiza y guía las reuniones
• Asegura condiciones adecuadas para el proyecto
» Cliente (*Customer*)
• Es parte del equipo
• Determina qué construir y cuándo
• Establece las pruebas funcionales
http://wiki.c2.com/?ExtremeRoles
29-ago.-25 Ingeniería del Software página 63

63


Roles XP (II)
» Encargado de Pruebas (*Tester*)
• Ayuda al cliente con las pruebas funcionales
• Se asegura de que las pruebas funcionales se superan
» Rastreador (*Tracker*)
• *Metric Man*
• Observa sin molestar
• Conserva datos históricos
» Entrenador (*Coach*)
• Responsable del proceso
• Tiende a estar en un segundo plano a medida que el equipo madura
» Consultor (*Consultant*)
• Realiza labores de apoyo y consultoría en temas específicos
http://wiki.c2.com/?ExtremeRoles
29-ago.-25 Ingeniería del Software página 64

64

## XP Roles & Responsibilities



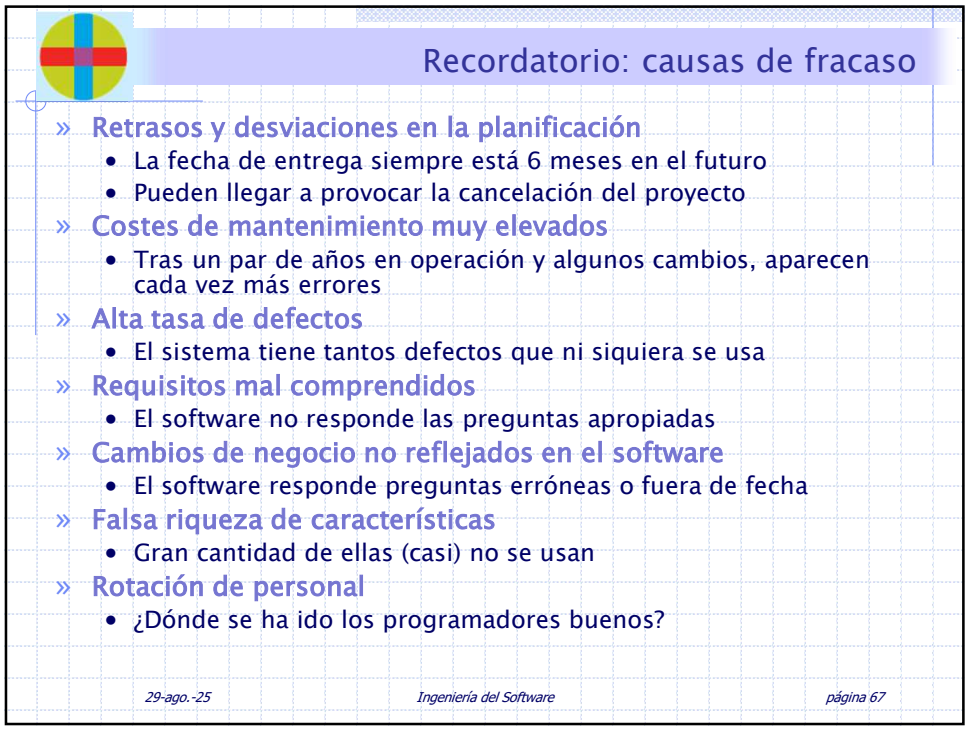



65

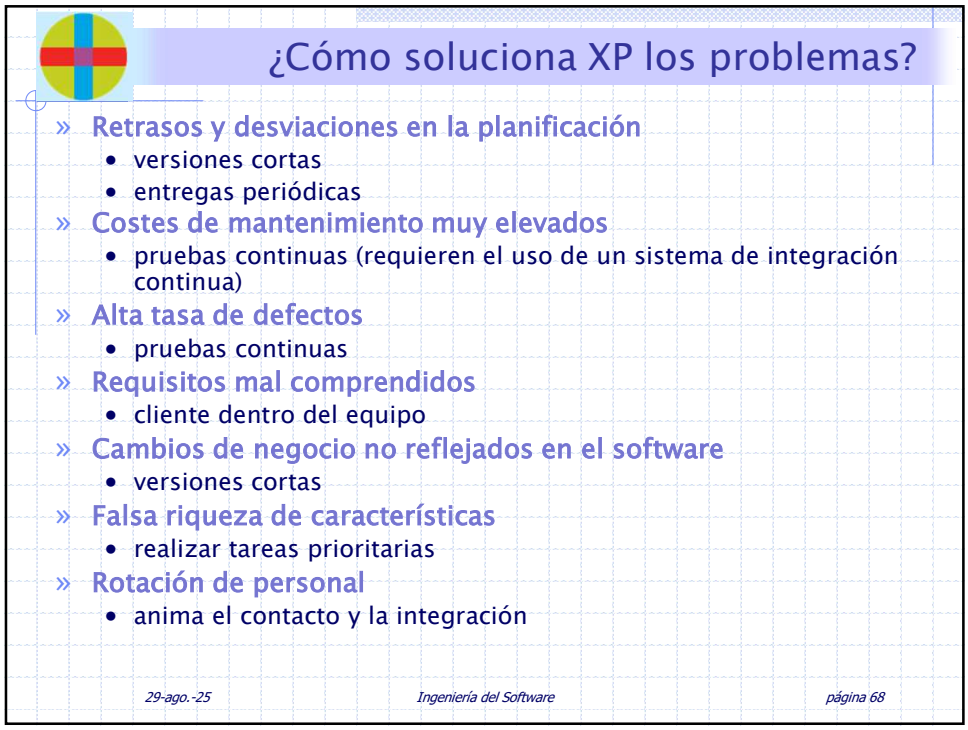
## Un día de trabajo en XP



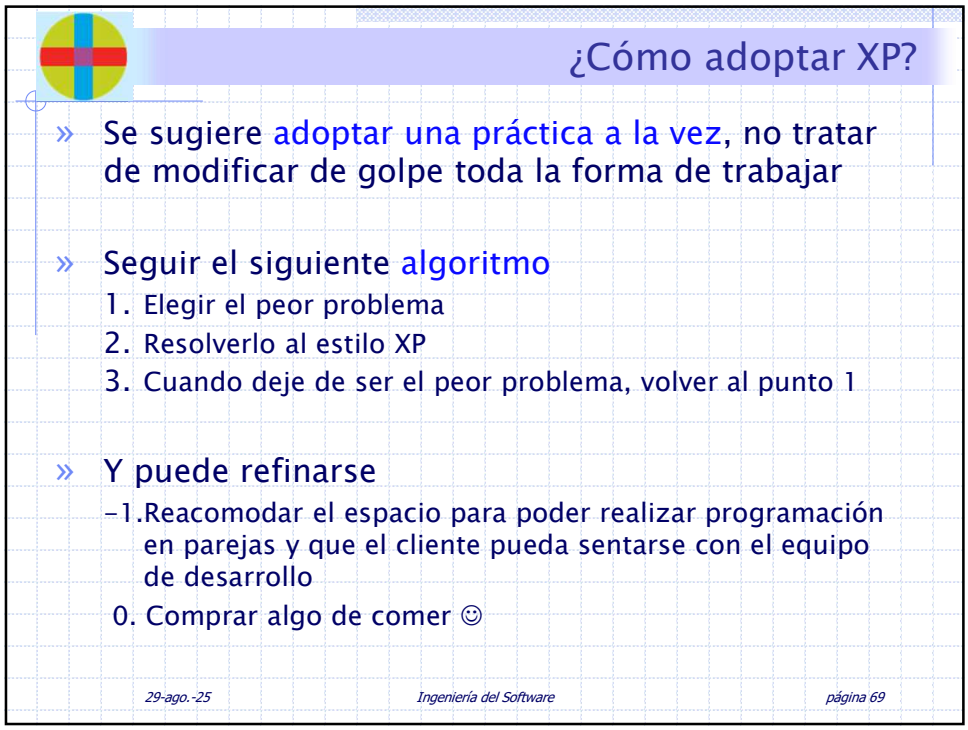
66


Recordatorio: causas de fracaso
» Retrasos y desviaciones en la planificación
• La fecha de entrega siempre está 6 meses en el futuro
• Pueden llegar a provocar la cancelación del proyecto
» Costes de mantenimiento muy elevados
• Tras un par de años en operación y algunos cambios, aparecen cada vez más errores
» Alta tasa de defectos
• El sistema tiene tantos defectos que ni siquiera se usa
» Requisitos mal comprendidos
• El software no responde las preguntas apropiadas
» Cambios de negocio no reflejados en el software
• El software responde preguntas erróneas o fuera de fecha
» Falsa riqueza de características
• Gran cantidad de ellas (casi) no se usan
» Rotación de personal
• ¿Dónde se ha ido los programadores buenos?
29-ago.-25 Ingeniería del Software página 67

67


¿Cómo soluciona XP los problemas?
» Retrasos y desviaciones en la planificación
• versiones cortas
• entregas periódicas
» Costes de mantenimiento muy elevados
• pruebas continuas (requieren el uso de un sistema de integración continua)
» Alta tasa de defectos
• pruebas continuas
» Requisitos mal comprendidos
• cliente dentro del equipo
» Cambios de negocio no reflejados en el software
• versiones cortas
» Falsa riqueza de características
• realizar tareas prioritarias
» Rotación de personal
• anima el contacto y la integración
29-ago.-25 Ingeniería del Software página 68

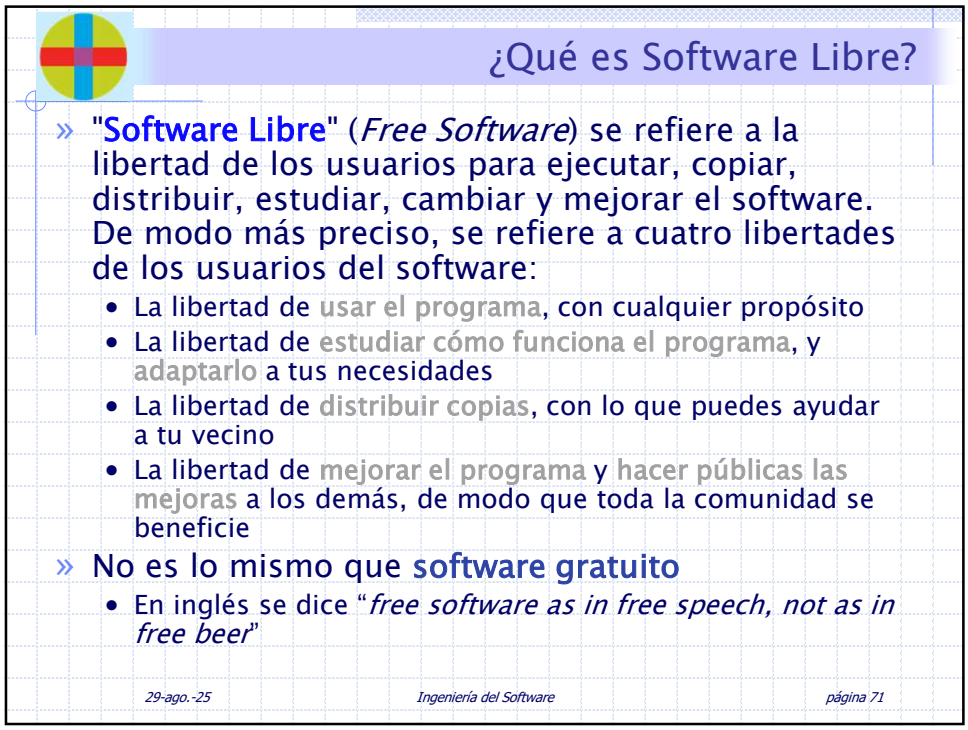
68


¿Cómo adoptar XP?
» Se sugiere adoptar una práctica a la vez, no tratar de modificar de golpe toda la forma de trabajar
» Seguir el siguiente algoritmo
1. Elegir el peor problema
2. Resolverlo al estilo XP
3. Cuando deje de ser el peor problema, volver al punto 1
» Y puede refinarse
-1. Reacomodar el espacio para poder realizar programación en parejas y que el cliente pueda sentarse con el equipo de desarrollo
0. Comprar algo de comer ☺
29-ago.-25 Ingeniería del Software página 69

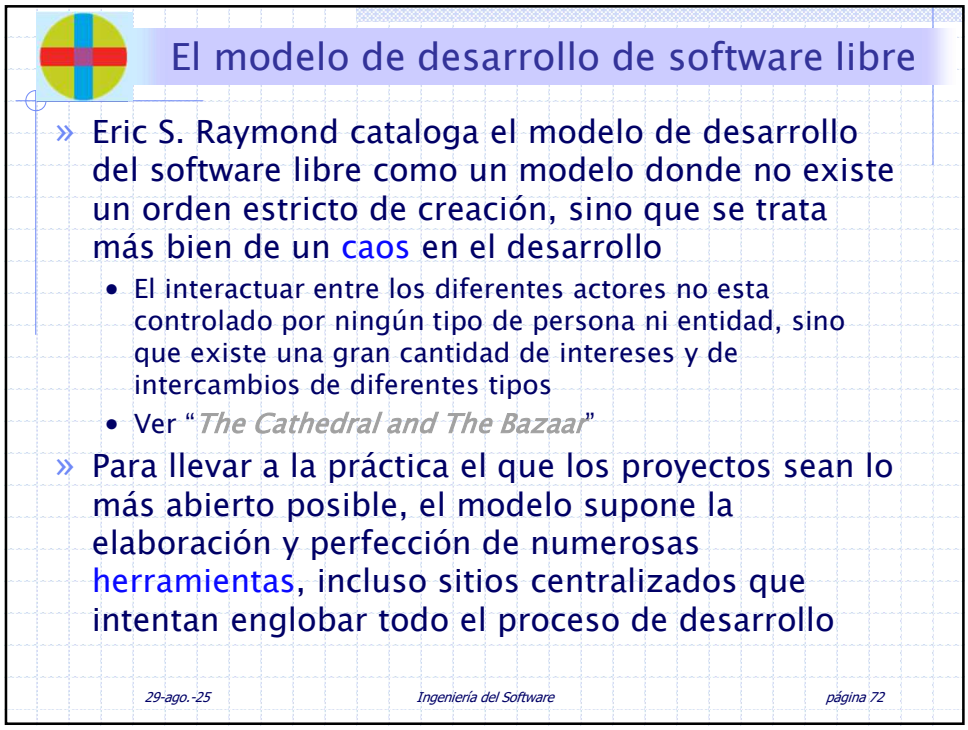
69


Programación eXtrema
y Software Libre

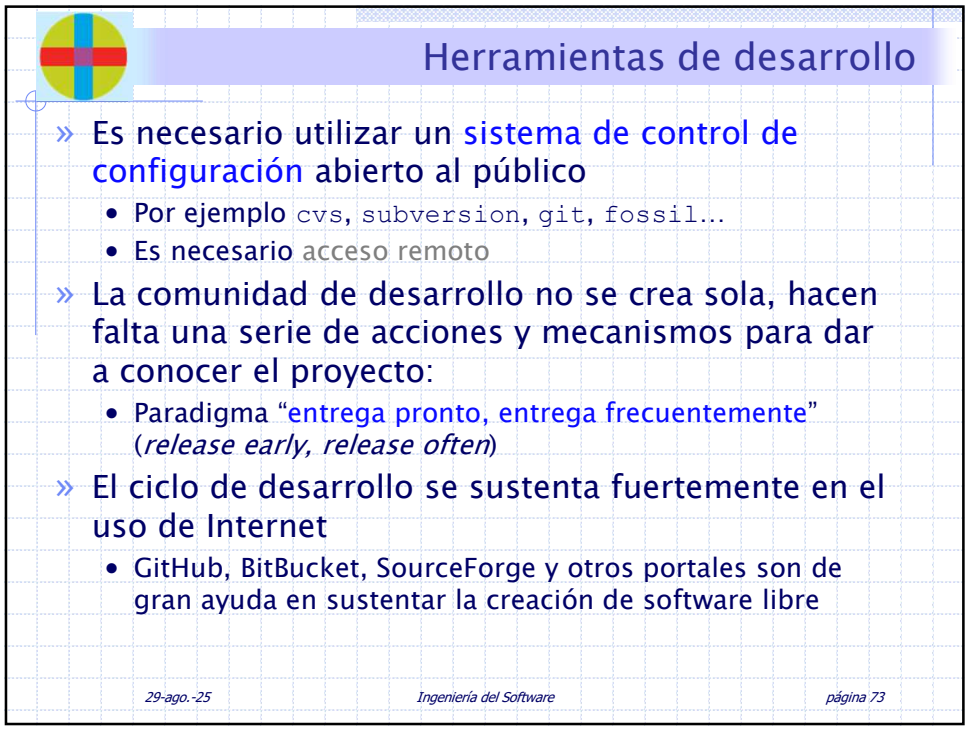
70


¿Qué es Software Libre?
» "Software Libre" (*Free Software*) se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. De modo más preciso, se refiere a cuatro libertades de los usuarios del software:
• La libertad de usar el programa, con cualquier propósito
• La libertad de estudiar cómo funciona el programa, y adaptarlo a tus necesidades
• La libertad de distribuir copias, con lo que puedes ayudar a tu vecino
• La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie
» No es lo mismo que software gratuito
• En inglés se dice "*free software as in free speech, not as in free beer*"
29-ago.-25 Ingeniería del Software página 71

71


El modelo de desarrollo de software libre
» Eric S. Raymond cataloga el modelo de desarrollo del software libre como un modelo donde no existe un orden estricto de creación, sino que se trata más bien de un caos en el desarrollo
• El interactuar entre los diferentes actores no esta controlado por ningún tipo de persona ni entidad, sino que existe una gran cantidad de intereses y de intercambios de diferentes tipos
• Ver "*The Cathedral and The Bazaar*"
» Para llevar a la práctica el que los proyectos sean lo más abierto posible, el modelo supone la elaboración y perfección de numerosas herramientas, incluso sitios centralizados que intentan englobar todo el proceso de desarrollo
29-ago.-25 Ingeniería del Software página 72

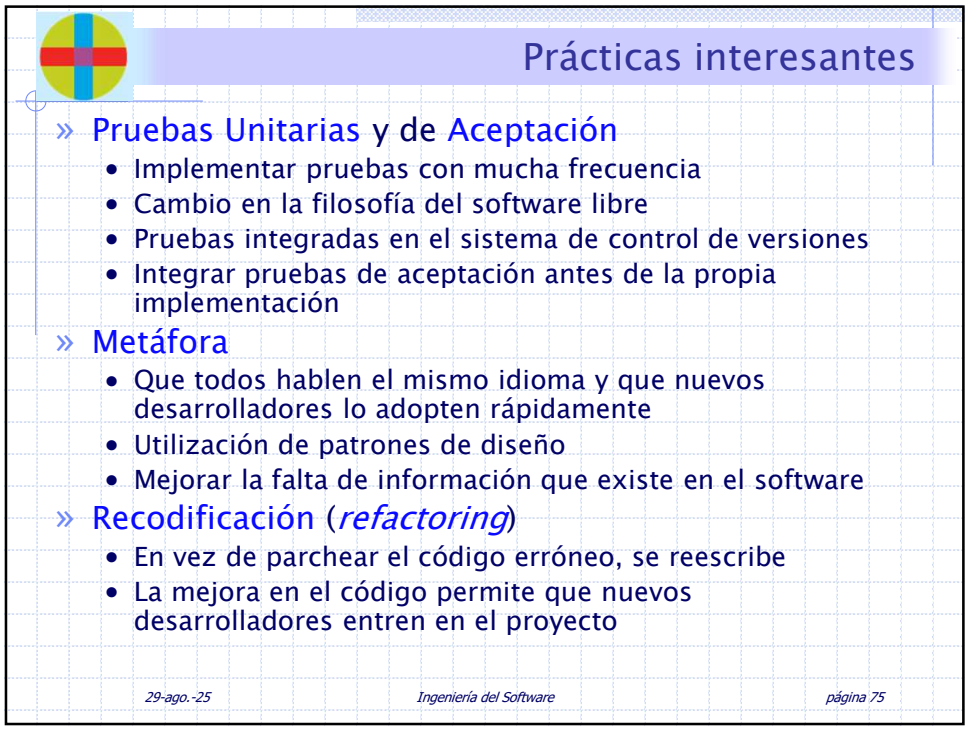
72


Herramientas de desarrollo
» Es necesario utilizar un sistema de control de configuración abierto al público
• Por ejemplo cvs, subversion, git, fossil...
• Es necesario acceso remoto
» La comunidad de desarrollo no se crea sola, hacen falta una serie de acciones y mecanismos para dar a conocer el proyecto:
• Paradigma "entrega pronto, entrega frecuentemente" (*release early, release often*)
» El ciclo de desarrollo se sustenta fuertemente en el uso de Internet
• GitHub, BitBucket, SourceForge y otros portales son de gran ayuda en sustentar la creación de software libre
29-ago.-25 Ingeniería del Software página 73

73


Software libre y Programación Extrema
» Características intrínsecas de XP en software libre:
• Propiedad colectiva del código (característica esencial del software libre)
• El paradigma "*release early, release often*" del software libre encaja con la idea de entregas frecuentes de XP
» Prácticas de difícil adaptación
• Carga de trabajo de 40 horas semanales
• Cliente en casa
• El juego de planificación
› No hay cliente dificulta la planificación
› Los proyectos se crean para satisfacer necesidades personales (no hay historias)
• Programación por parejas
29-ago.-25 Ingeniería del Software página 74

74


Prácticas interesantes
» Pruebas Unitarias y de Aceptación
• Implementar pruebas con mucha frecuencia
• Cambio en la filosofía del software libre
• Pruebas integradas en el sistema de control de versiones
• Integrar pruebas de aceptación antes de la propia implementación
» Metáfora
• Que todos hablen el mismo idioma y que nuevos desarrolladores lo adopten rápidamente
• Utilización de patrones de diseño
• Mejorar la falta de información que existe en el software
» Recodificación (refactoring)
• En vez de parchear el código erróneo, se reescribe
• La mejora en el código permite que nuevos desarrolladores entren en el proyecto
29-ago.-25 Ingeniería del Software página 75

75


Interrogantes y retos
» Compatibilidad hacia atrás y dependencias
» Interrogantes económicos y psicológicos
• dificultad de estimar cuánto va a costar un proyecto
• sería interesante tener alguna forma de hacerlo de manera más o menos exacta para el desarrollo de software libre
» Efectos de la recodificación
» Los desarrolladores de software libre suelen adaptarse rápidamente a nuevas ideas
» El éxito en la adopción de XP depende en buena medida de la creación de nuevas herramientas que la soporten o que las existentes la integren
29-ago.-25 Ingeniería del Software página 76

76

## Resumen

- Se ha visto cómo el método de desarrollo seguido por en el software libre y XP tienen muchas características comunes
- Existe una serie de prácticas (pruebas, metáfora y recodificación) que es muy interesante adoptar en el software libre
- La programación extrema ofrece métodos formales para plasmar información, métodos que no existen en tal forma en el software libre
- En definitiva, se ha podido ver cómo la programación extrema puede aportar nuevas formas en busca de optimizar el modelo de desarrollo de software libre



77

## ¿Preguntas?



78