Desde el Desarrollo Evolutivo a las Metodologías Ágiles

Desarrollo Evolutivo
Principios Ágiles
Metodologías Ágiles: XP, FDD y Crystal Clear

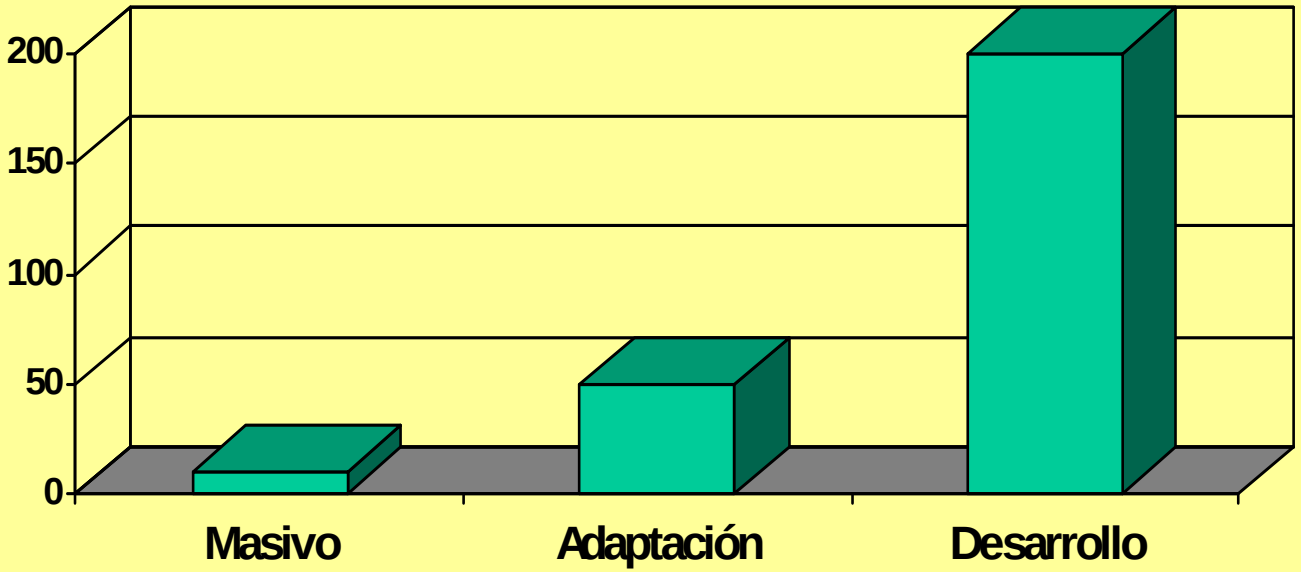
© 2004-2008 Pablo Straub

Desarrollo evolutivo

© 2004-2008 Pablo Straub

Desarrollo de Tecnología: ¿Un mal necesario?

- Si necesitas software, ¿qué conviene?
  - Comprar un software de mercado masivo
  - Adaptar una solución ya hecha
  - Desarrollar a partir de cero



© 2004-2008 Pablo Straub

Condiciones de aplicación de los métodos estándares

- Es posible determinar al comienzo las necesidades del futuro
- Mercados y necesidades relativamente estables
- Cliente ayuda poco al desarrollo (tiene poco tiempo)
- Equipo de desarrollo puede incluir algunos miembros con muy poca experiencia
  - El proceso definido y los controles de calidad son la red de seguridad
- No hay un gran beneficio de tener un sistema limitado funcionando desde antes del sistema completo (todo o nada)

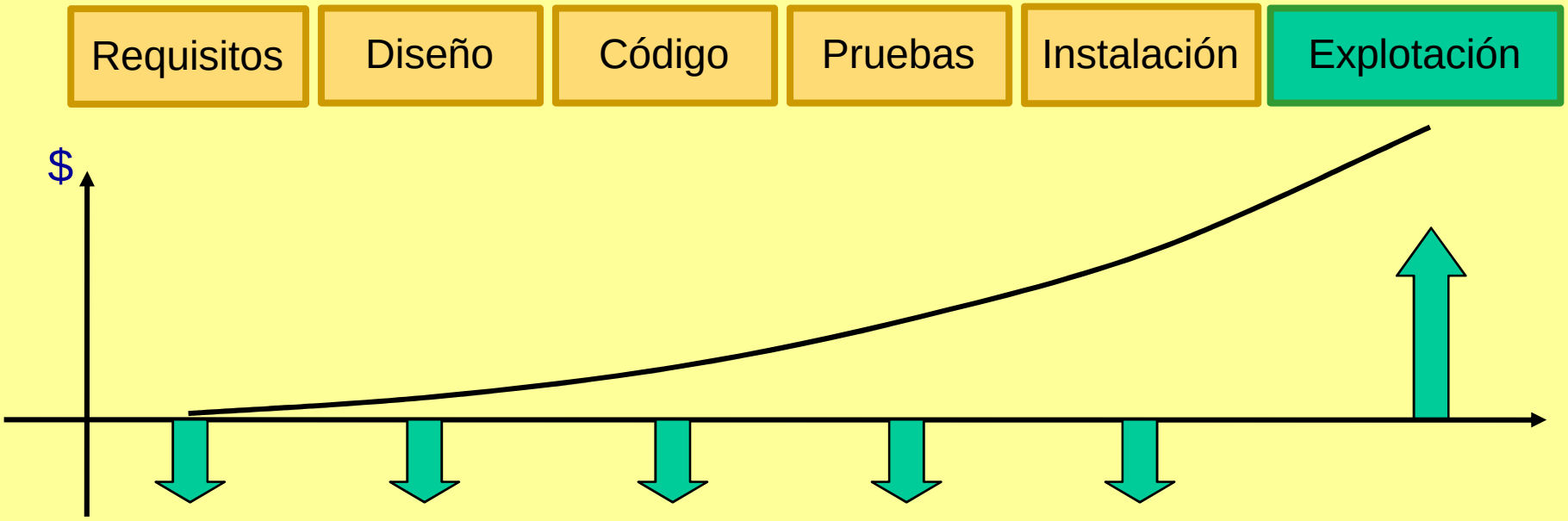
© 2004-2008 Pablo Straub

Un enfoque diferente para trabajar

- La inversión en tecnología de la información usualmente se hace con un estilo “big bang”
  - Adelanta los costos
  - Posterga beneficios
  - Aumenta el riesgo
  - Disminuye la flexibilidad
- Esto no es necesario: lo podemos hacer mucho mejor
  - Invertir en forma incremental
  - Obtener beneficios tempranamente
  - Disminuir riesgo tecnológico y de negocios
  - Postergar decisiones de gerencia
- Esto se llama desarrollo evolutivo (abreviado a evo) y fue introducido por Tom Gilb en 1989

© 2004-2008 Pablo Straub

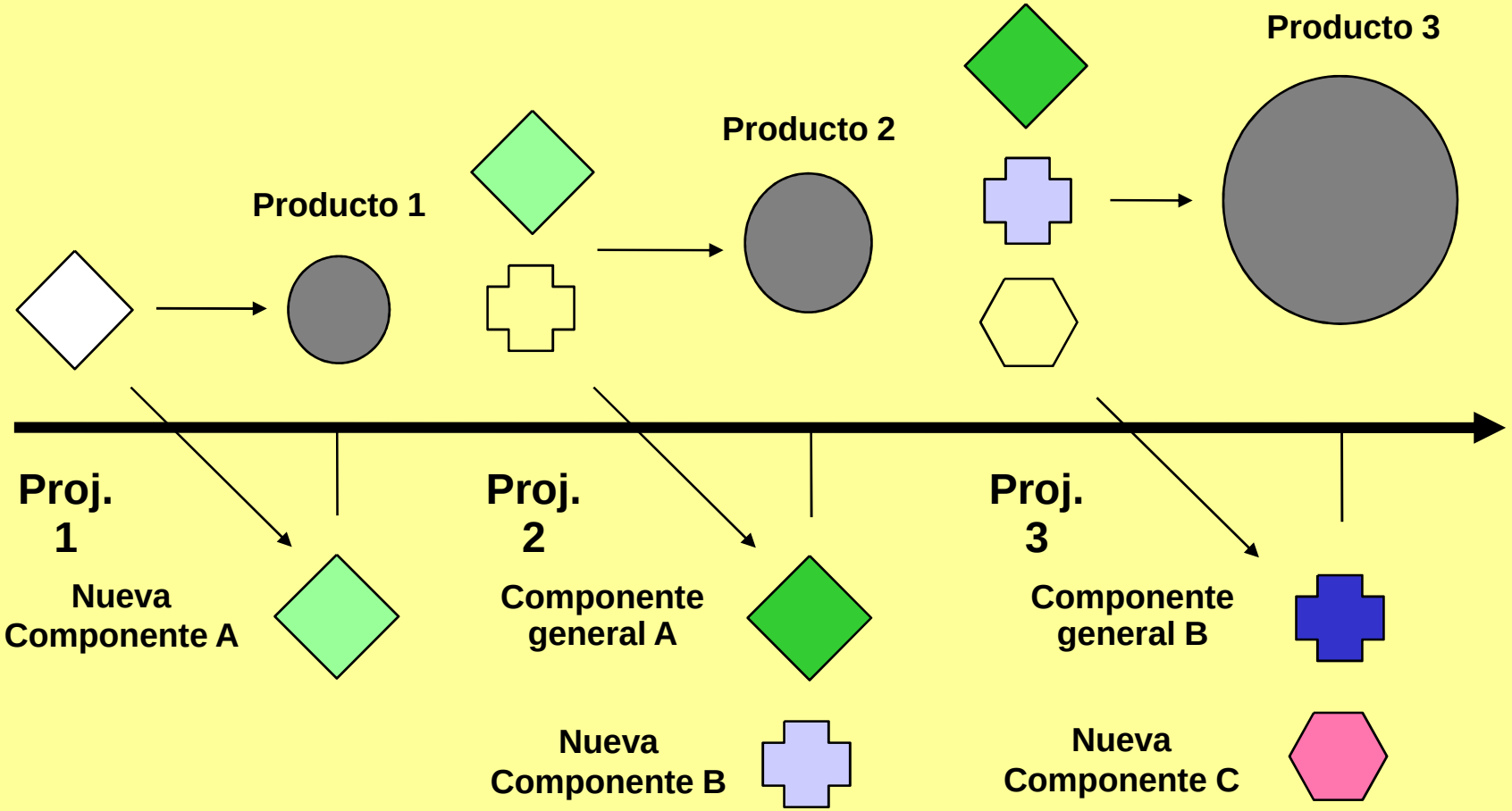
Costos y beneficios: El desarrollo en cascada



Adelanta los costos
Posterga beneficios
Los cambios son cada vez más costosos
Aumenta el riesgo
Disminuye la flexibilidad

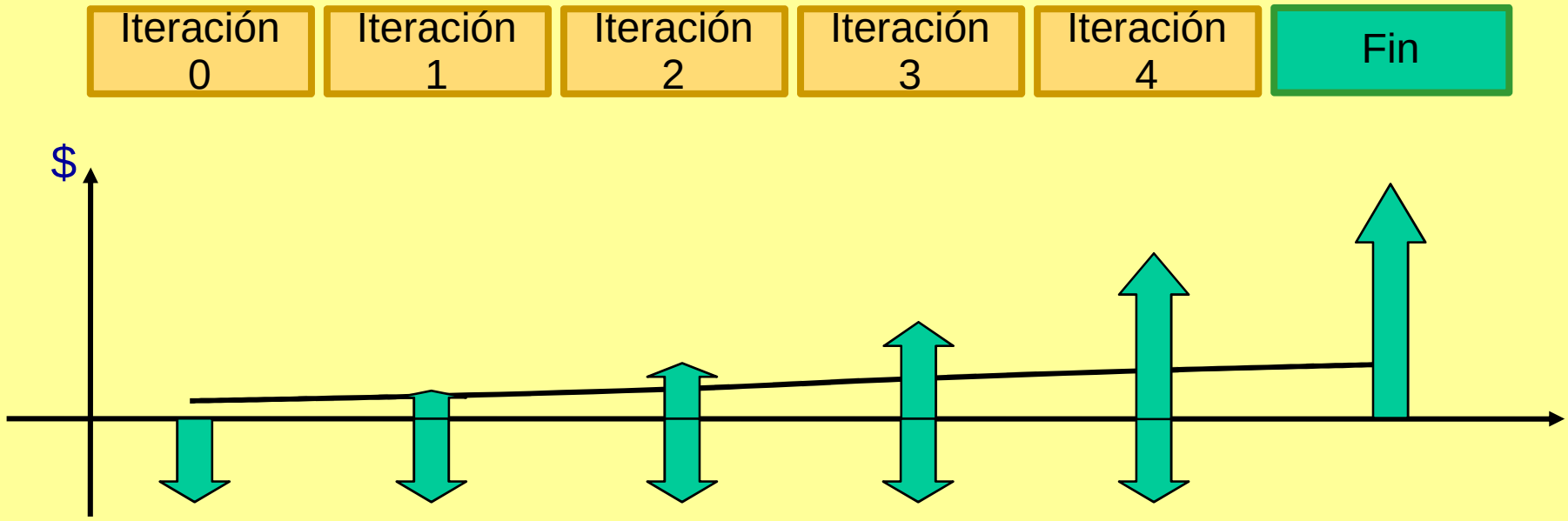
© 2004-2008 Pablo Straub

Desarrollo Evolutivo



© 2004-2008 Pablo Straub

Costos y Beneficios: El desarrollo evolutivo



Adelanta los beneficios
Los cambios son poco costosos
disminuye el riesgo por realimentación de usuarios
Aumenta la flexibilidad por posibilidad de cancelar un proyecto sin perder lo invertido

© 2004-2008 Pablo Straub

Contratos, aprobación de proyectos, y los impedimentos para el desarrollo evolutivo

Hipótesis

- La forma en que las organizaciones aprueban proyectos dificulta el desarrollo evolutivo, causando costos mayores, flexibilidad menor, y calidad menor

Reflexión

- ¿Qué podemos hacer al respecto?
- ¿Podemos aprovechar algunas de las ventajas del desarrollo evolutivo aún cuando el mercado nos empuja a la cascada?

© 2004-2008 Pablo Straub

Métodos Ágiles

© 2004-2008 Pablo Straub

Programación ágil según Dilbert

© Scott Adams, Inc./Dist. by UFS, Inc.

© 2004-2008 Pablo Straub

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over *processes and tools*
Working software over *comprehensive documentation*
Customer collaboration over *contract negotiation*
Responding to change over *following a plan*

That is, while there is value in the items on
the right, we value the items on the left more.

http://agilemanifesto.com

Manifiesto por un Desarrollo de Software Ágil

Estamos descubriendo formas mejores de desarrollar software, desarrollando y ayudando a otros a desarrollar. A través de este trabajo hemos empezado a valorar:

Individuos e interacciones por sobre *procesos y herramientas*
Software funcionando por sobre *documentación completa*
Cooperación con el cliente por sobre *negociación de contratos*
Reacción frente al cambio por sobre el *seguimiento de un plan*

Es decir, si bien hay valor en los elementos de la derecha, valoramos más los elementos de la izquierda.

http://agilemanifesto.com

Principios tras el Manifiesto Ágil

- Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software valioso
- Dele la bienvenida a los cambios de requisitos, incluso al final del desarrollo
  - Los procesos ágiles usan el cambio para darle una ventaja competitiva al cliente
- Entregue software funcionando frecuentemente, desde cada dos semanas a cada dos meses, con preferencia al ritmo más rápido

© 2004-2008 Pablo Straub

Principios tras el Manifiesto Ágil

- Los responsables del negocio y los desarrolladores deben trabajar juntos durante todo el proyecto
- Construya proyectos con personas motivadas
  - Deles el ambiente y el apoyo que necesitan, y confíe en que harán su trabajo
- La forma más eficiente y efectiva de traspasar información hacia y desde un equipo de desarrollo es la conversación cara-a-cara

© 2004-2008 Pablo Straub

Principios tras el Manifiesto Ágil

- El software que funciona es la principal medida de avance
- Los procesos ágiles promueven desarrollo sustentable
  - Los auspiciadores, desarrolladores y usuarios deben ser capaces de mantener un paso constante indefinidamente
- La atención constante por la excelencia técnica y el buen diseño aumenta la agilidad

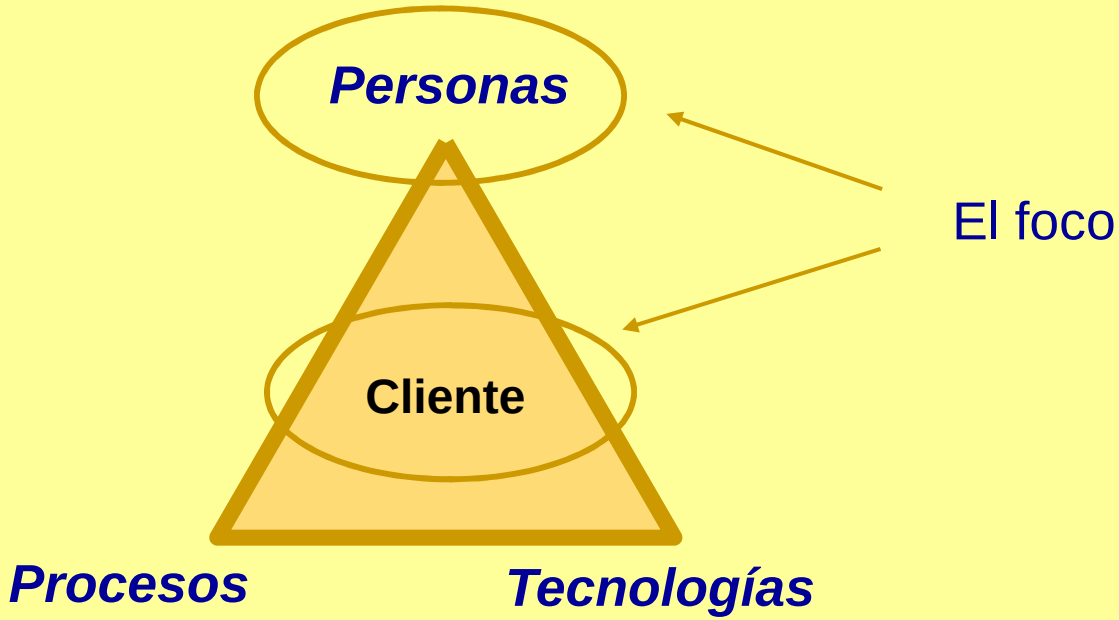
© 2004-2008 Pablo Straub

Principios tras el Manifiesto Ágil

- La simplicidad –el arte de maximizar la cantidad de trabajo no hecho– es esencial
- Las mejores arquitecturas, requisitos y diseños emergen de los equipos auto-organizados
- A intervalos regulares, el equipo reflexiona sobre cómo hacerse más efectivo, luego afina y ajusta su comportamiento según eso

© 2004-2008 Pablo Straub

De qué depende la calidad



- Conviene enfocarse prioritariamente en las personas
  - Los procesos y las tecnologías ayudan a las personas
  - Mejoras en equipos de personas sí son apropiables

© 2004-2008 Pablo Straub

Principios del Desarrollo Liviano

1. Iniciar pronto
2. Aprender constantemente
3. Postergar las decisiones
4. Entregar rápidamente
5. Eliminar pérdidas
6. Empoderar al equipo
7. Diseñar con integridad
8. Evitar suboptimizaciones

Fuente: Mary Poppendieck. Lean Development & the Predictability Paradox © 2003 Poppendieck LCC

Principios de los Métodos *Crystal*

1. Cara a cara
2. El peso es costoso
3. Metodologías más pesadas para equipos más grandes
4. Más ceremonia para sistemas más críticos
5. Más realimentación y comunicaciones, menos entregables intermedios
6. Disciplina, habilidades, comprensión del proceso, formalidad, documentación
7. La eficiencia no es importante en actividades no críticas

Fuente: Alistair Cockburn. What is *Crystal?* © 1999 Humans and Technology, Inc.

Resumen de Principios Ágiles

- Agregación de valor explícita
  - Sólo se implementan *prestaciones* o *features* con valor conocido para los clientes
  - Esto impacta mucho al diseño de arquitectura
- Aprendamos junto al cliente qué agrega valor
  - Ciclos cortos de desarrollo evolutivo
  - Sistema potencialmente entregable (*potentially shippable*)
  - Énfasis en la excelencia, la simplicidad y el aprendizaje
- Gestión ágil
  - Énfasis en comunicación intergrupal e intragrupal
  - Ingenieros y equipos con poder de toma de decisiones
  - Gerente como facilitador

© 2004-2008 Pablo Straub

Técnicas que permiten los métodos ágiles

- **Refactorización** (*refactoring*) – Cambios en el diseño sobre el sistema implementado
- **Pruebas automáticas** – Pruebas exhaustivas del sistema cuyos resultados se comparan con resultados esperados
- **Integración continua** – Automatización de la integración de sistemas de modo de permitir que las pruebas automáticas se ejecuten muy frecuentemente

- Estas técnicas y otras sirven de malla de seguridad para permitir los métodos ágiles

© 2004-2008 Pablo Straub

Cambios en la Arquitectura de Software

- El enfoque tradicional de arquitectura de software ***no funciona*** con Desarrollo Evolutivo ni mucho menos con Métodos Ágiles
- Se necesitan ***arquitecturas flexibles*** o mejor aún la posibilidad de ***cambiar la arquitectura*** de sistemas en producción

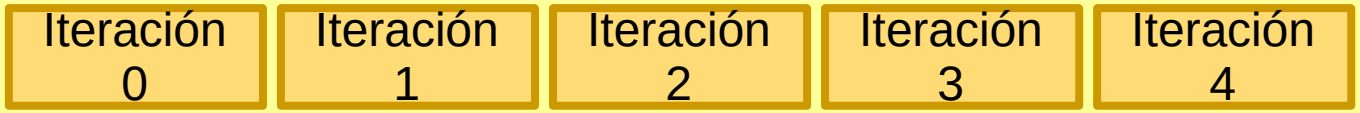
© 2004-2008 Pablo Straub

Arquitectura de software: Desarrollo en cascada

- Dado que están todos los requisitos se diseña una arquitectura apropiada para todos los requisitos
- Respuesta a cambios futuros
  - Cambios consistentes con la arquitectura serán simples
  - Cambios que contradicen la arquitectura serán difíciles y caros
  - El cliente no entenderá por qué algunos cambios “simples” son tan complejos para los informáticos cuando hay otros cambios mucho más “grandes” que los completaron rápido
  - Esto genera desconfianza

© 2004-2008 Pablo Straub

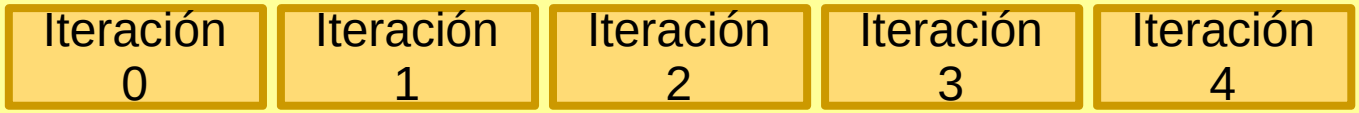
Arquitectura de software: Desarrollo evolutivo



- Aquí reconocemos que habrán muchos cambios
- La solución planteada por evo son las arquitecturas flexibles
  - Sistemas basados en mensajes
  - Desacoplar interfaces de núcleo funcional
  - Desacoplar servidores de los servicios que implementan
  - Muchas otras técnicas
- Esto es mejor que el método de cascada, pero tiene problemas:
  - La arquitectura suele ser más compleja de lo necesario
  - Es difícil predecir los tipos de cambios que se necesitarán
  - Algunos cambios no se adaptan a la arquitectura

© 2004-2008 Pablo Straub

Arquitectura de software: Desarrollo ágil



- Se parte con una arquitectura muy simple orientada a los requisitos implementados en la iteración
- Se hace evolucionar la arquitectura, cambiando la arquitectura sobre un sistema ya implantado
  - La arquitectura nunca es más compleja de lo estrictamente necesario
  - Si un cambio no se adapta a la arquitectura, la arquitectura se cambia

Advertencia

Esto lo hacen profesionales. No intente hacer esto en su casa.

© 2004-2008 Pablo Straub

Riesgo en los métodos ágiles

- Cambios de arquitectura significativos de sistemas en operación son como ejercicios de trapecio elevado
- ¿Queremos hacer esto sin malla de seguridad?

Advertencia

Esto lo hacen profesionales. No intente hacerlo en su casa.

Principales Métodos Ágiles

Programación extrema o XP

Feature Driven Development o FDD

Crystal Clear

© 2004-2008 Pablo Straub

eXtreme Programming - XP
Programación eXtrema

- Iniciado en 1996, primer libro publicado en 1999
- Se llama extremo, porque elimina técnicas/procesos de poco valor y sobre-enfatiza procesos de mucho valor
  - “Poner todas las perillas en 10”
- Iteraciones de 1 a 4 semanas
- Historias de usuarios
- Cliente en las instalaciones
- Mucho énfasis en pruebas automáticas
- Hágalo lo más simple posible
  - No lo vas a necesitar (NLVAN)
  - *You aren't gonna need it (YAGNI)*

© 2004-2008 Pablo Straub

XP - Programación eXtrema

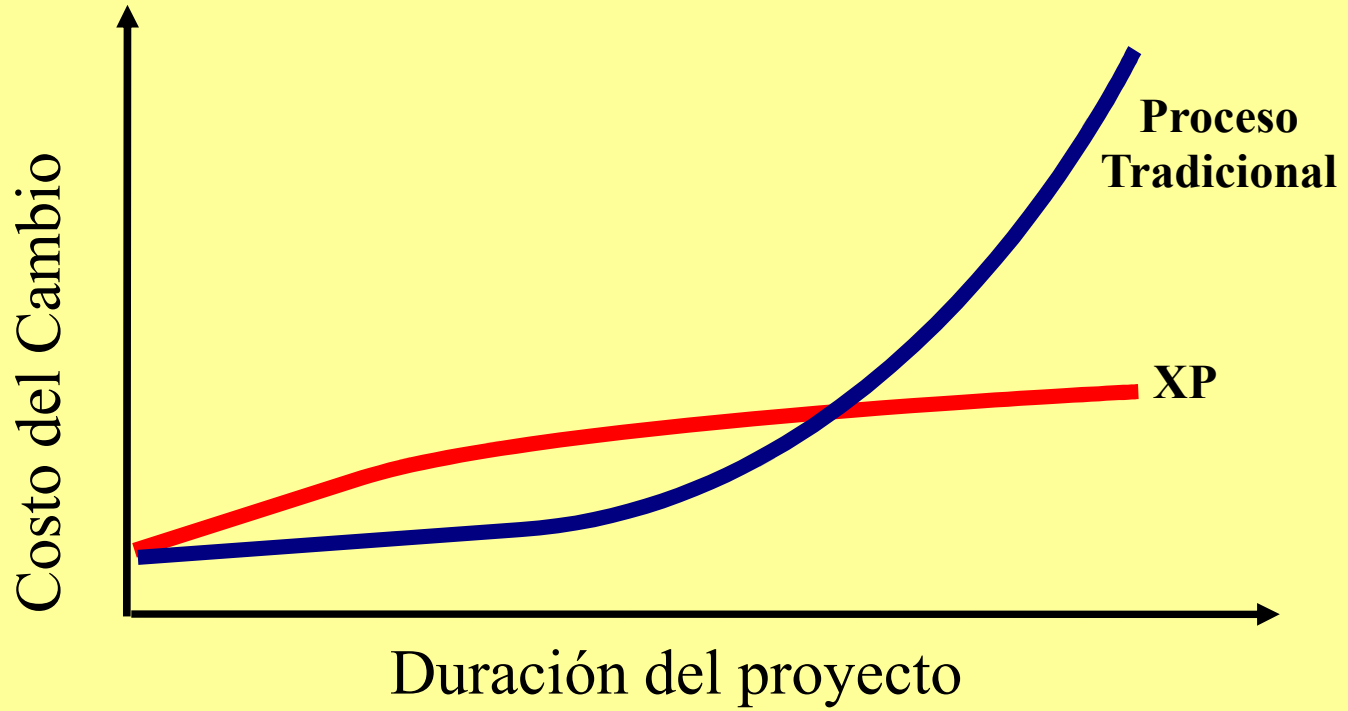
- Se basa en implementar cuentos del usuario (user stories) en forma iterativa
  - Planificación: elegir cuáles cuentos implementar y aclararlos
  - Codificación: implementar el cuento
  - Pruebas automáticas: a lo menos una por cada clase/método
  - Prueba de aceptación: si pasa, la funcionalidad nueva se acepta; si no, el cuento correspondiente se rehace en la próxima iteración
- Los requisitos son poco detallados
  - El cliente debe estar entre los desarrolladores (customer on-site)
  - Si hay problemas se corrige a muy bajo costo
  - Una user story es una oración escrita en una tarjeta

© 2004-2008 Pablo Straub

El costo de un cambio

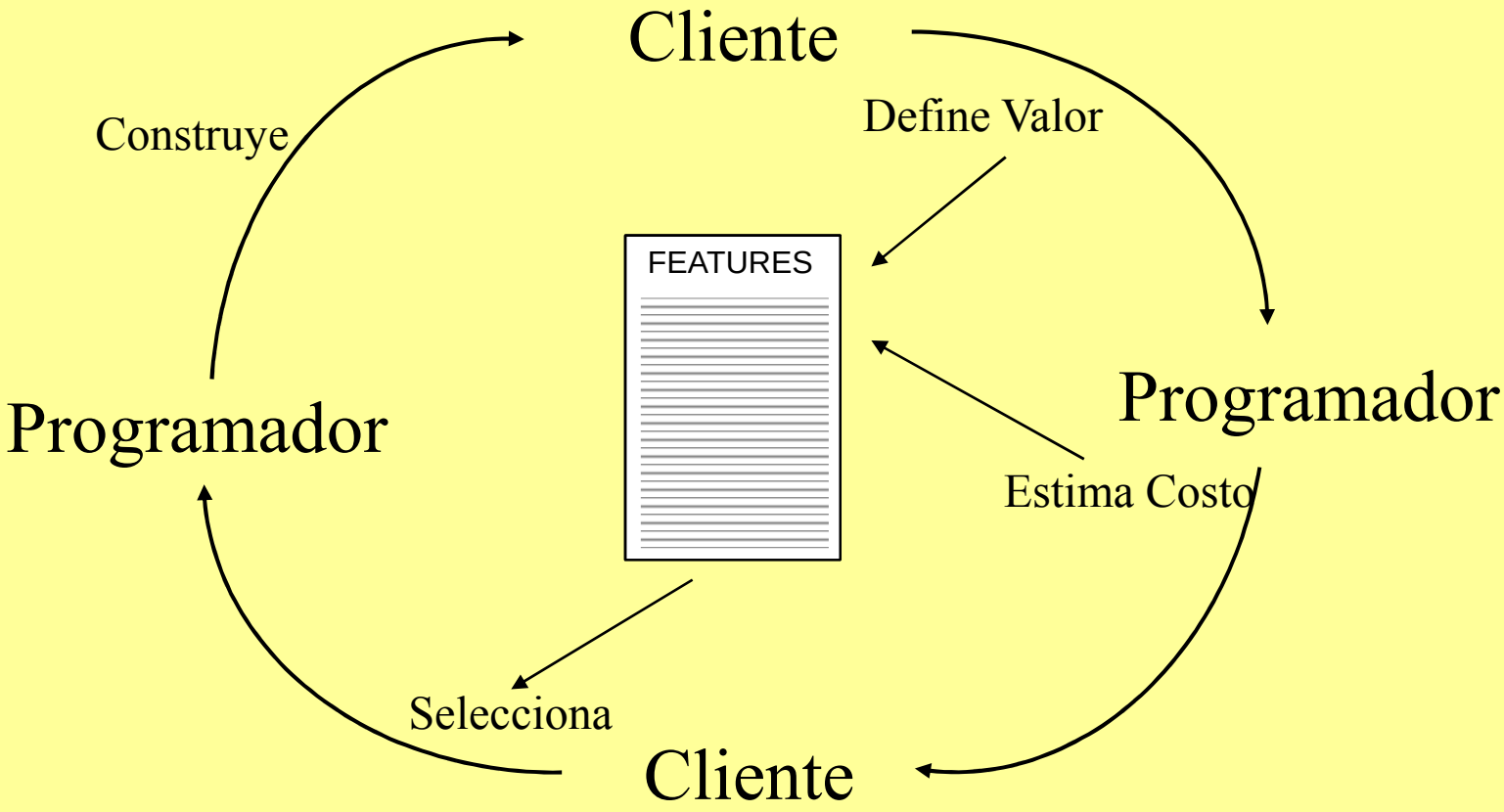
- "El error es usualmente 100 veces más caro de corregir en la fase de mantenimiento que en la fase de requisitos."

Barry Boehm, *Software Engineering Economics*, 1981, p. 40.



© 2004-2008 Pablo Straub

XP - Ciclo de creación de valor



© 2004-2008 Pablo Straub

eXtreme Programming - Las prácticas

1. El juego del plan
2. Pequeñas entregas (*releases*)
3. Metáfora
4. Diseño Simple
5. Probar antes de codificar (*Test then code*)
6. Refactorizar (*Refactoring*)
7. Programación en pares
8. Responsabilidad compartida del código
9. Integración continua
10. Ritmo sostenible (40 horas a la semana)
11. Cliente disponible *on-site*
12. Estándares de codificación

© 2004-2008 Pablo Straub

Elija XP si...

- ... tiene requisitos vagamente definidos o volátiles
- ... tiene o puede desarrollar habilidades y prácticas de ingeniería potentes
- ... los clientes se pueden involucrar todos los días (horas)

Fuente: Michael Cohn. Selecting an Agile Process, *21 Septiembre 2004*.

© 2004-2008 Pablo Straub

Feature Driven Development FDD
Desarrollo dirigido por prestaciones

- Iniciado a mediados de los 90
- *Java Modeling in Color with UML*, Coad, Lefebvre y DeLuca, 1999
- *A practical Guide to Feature-Driven Development*, Palmer y Felsing, 2002

© 2004-2008 Pablo Straub

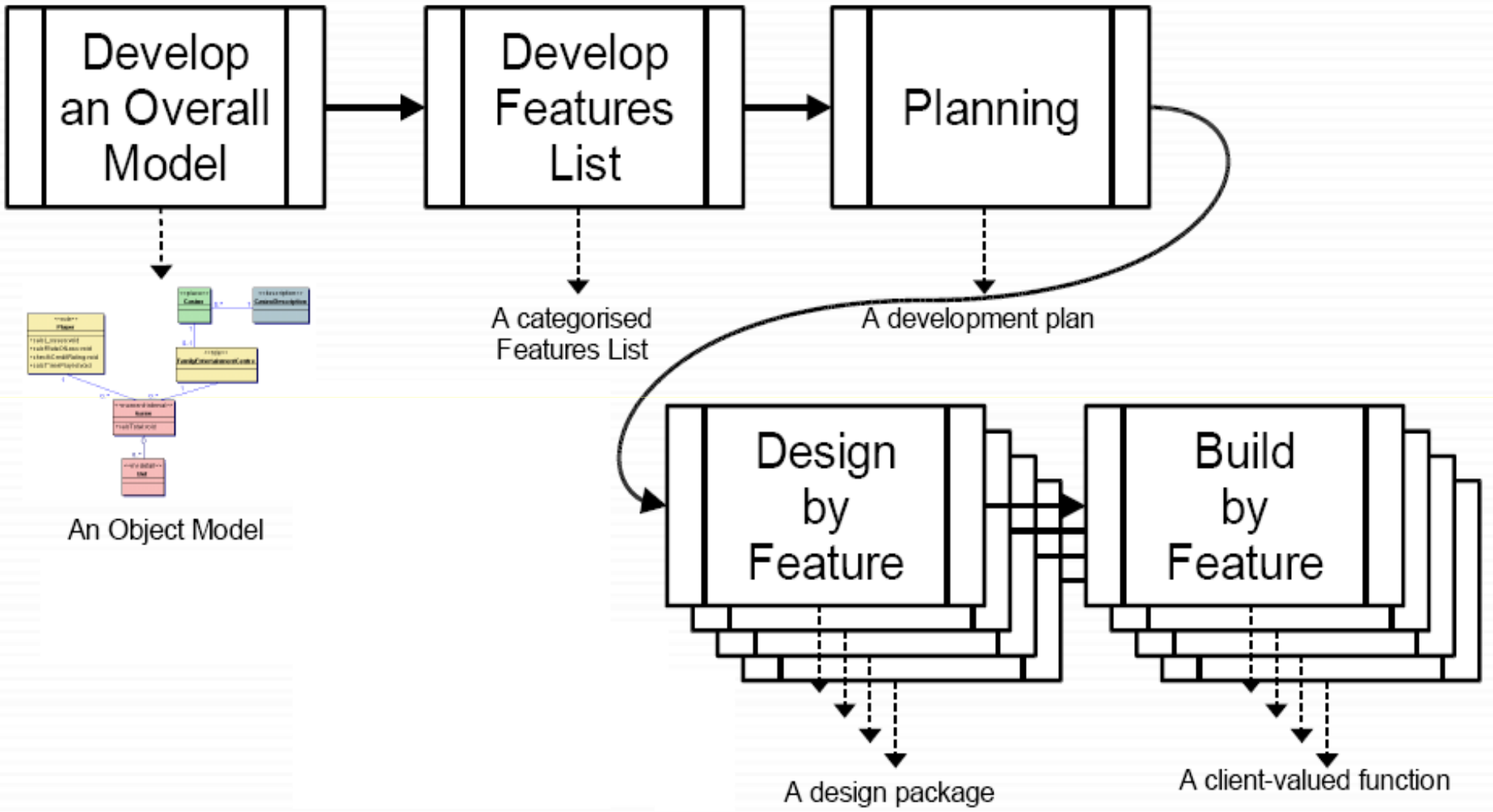
Feature Driven Development FDD

Desarrollo dirigido por prestaciones

- Se basa en implementar sistemas en forma iterativa
  - Descubrimiento de la lista de prestaciones (features)
    - Desarrollo de un modelo global
    - Lista priorizada de prestaciones
    - Plan de desarrollo
  - Implementación de las prestaciones
    - Diseño y revisión del diseño
    - Construcción y revisión de código
    - Reunión de liberación
- Los requisitos tienden a ser buenos porque incluyen el modelo que fue desarrollado en conjunto con los clientes
- Incluye un método de seguimiento cuantitativo del plan

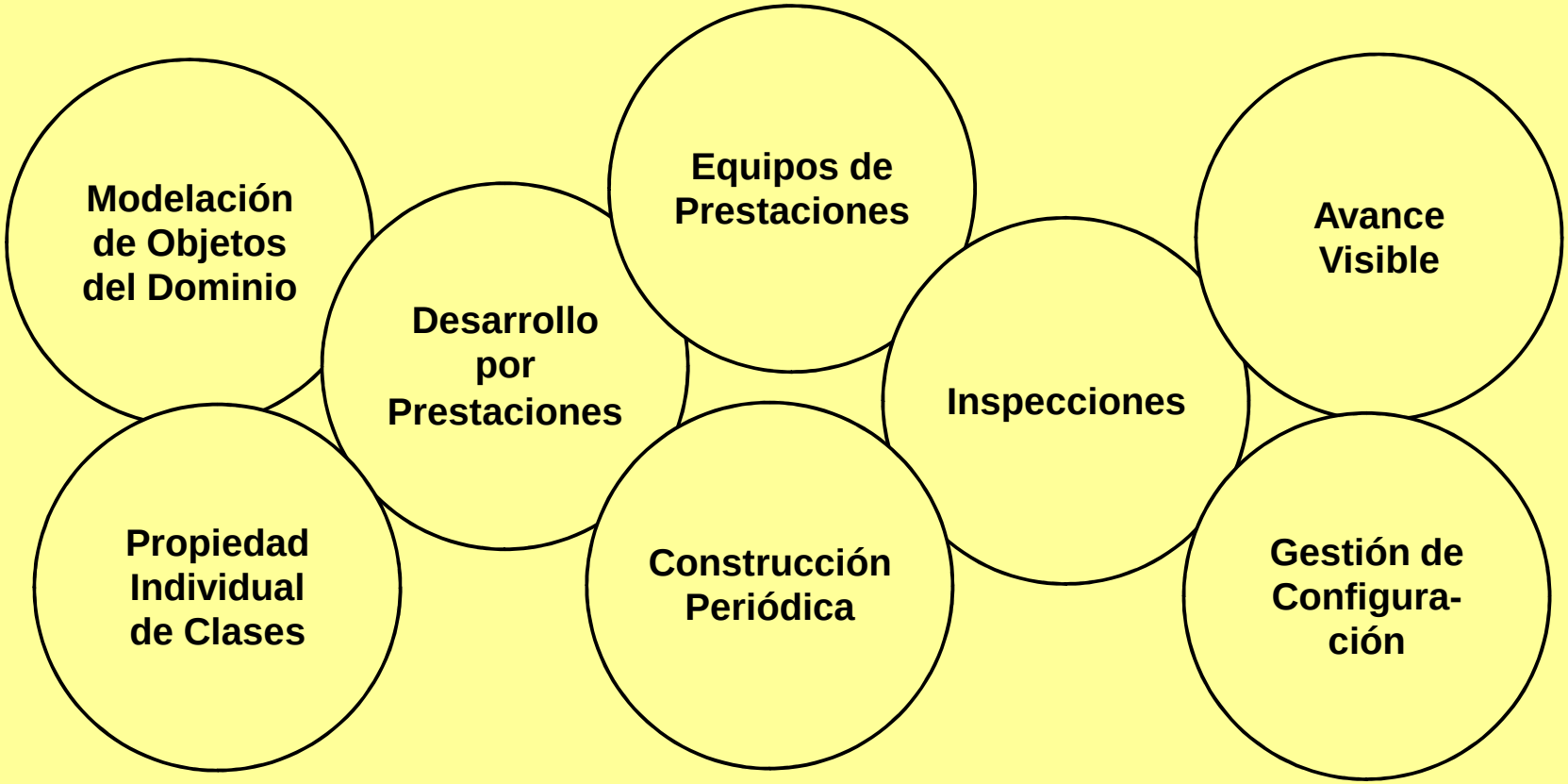
© 2004-2008 Pablo Straub

Proceso de FDD



© 2004-2008 Pablo Straub

Las ocho “mejores prácticas”



© 2004-2008 Pablo Straub

Elija FDD si...

- ... está dispuesto a entregar cierta agilidad a cambio de un proceso claramente escalable
- ... su organización tiene sólidas habilidades en UML
- ... la mayoría de los requisitos son conocidos desde el comienzo o son más o menos estables
- ... usted no considera que los equipos auto-organizados son un factor crítico de éxito

Fuente: Michael Cohn. Selecting an Agile Process, *21 Septiembre 2004*.

© 2004-2008 Pablo Straub

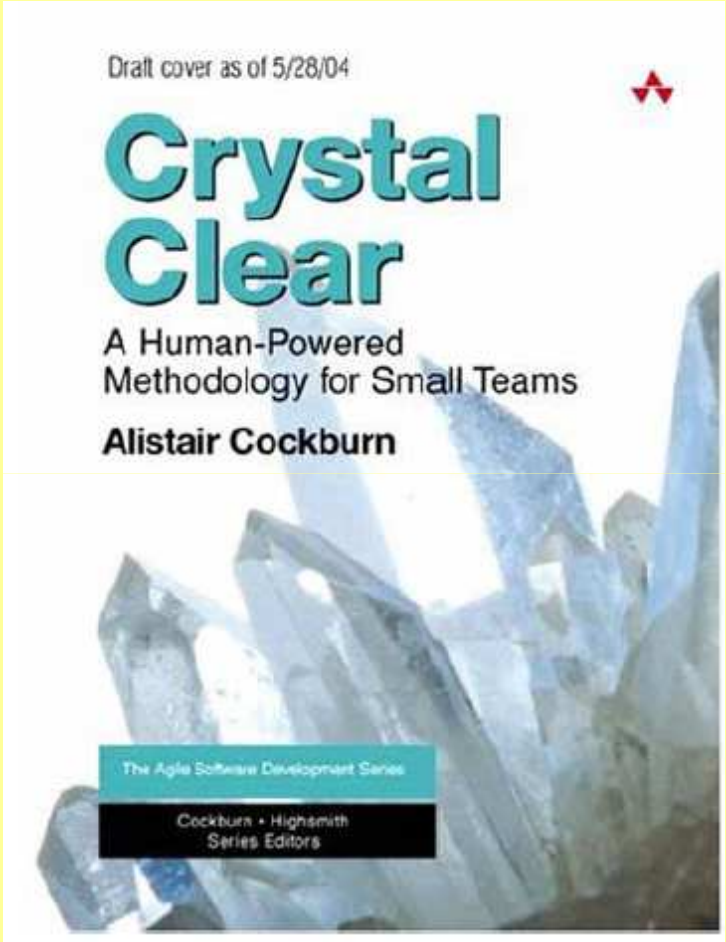
Comparación de XP y FDD

- Ambos métodos están basados en desarrollo evolutivo
- Los dos métodos son “livianos”, pero XP es más: sin documentación del código y con revisiones informales
- XP es mejor si hay requisitos cambiantes o mal definidos
- FDD tiene mejor escalabilidad a proyectos grandes
- FDD incluye seguimiento y reporte del estado del proyecto: mantiene calmados a los gerentes
- FDD permite variabilidad en la experiencia de los miembros, pero puede dejar a los menos expertos esperando al programador líder
- Ambas requieren mucha disciplina y un buen gerente

© 2004-2008 Pablo Straub

Crystal Clear

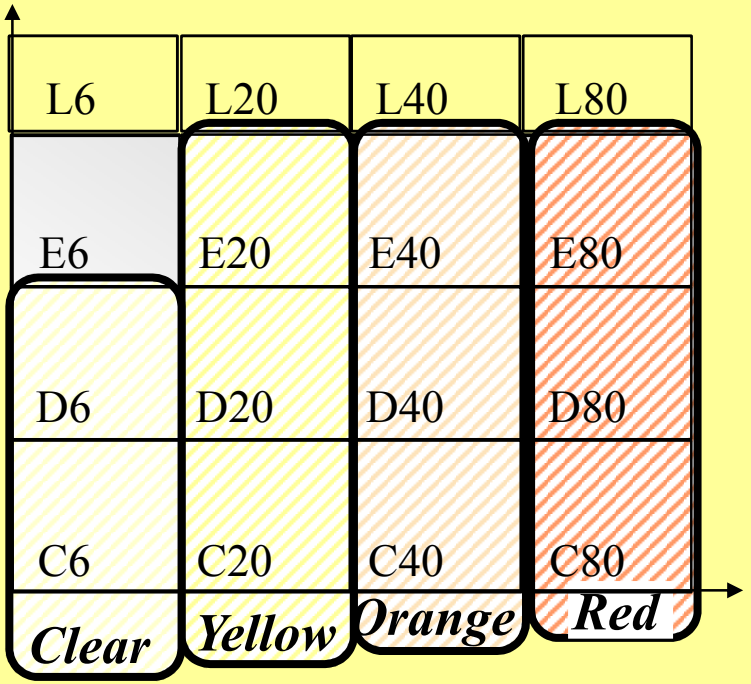
- Filosofía Crystal
  - Fuerte en comunicaciones
  - Liviano en entregables
  - Vías de comunicación cortas, ricas e informales
  - Entregar frecuentemente
  - Reducir el esfuerzo indirecto (overhead)
  - Usar las habilidades naturales de la gente (hablar, buscar), pero cuidado con las debilidades (descuido, indisciplina)
  - Aprender individualmente y como equipo



© 2004-2008 Pablo Straub

Crystal Clear: Uno de una familia de métodos

- Riesgo caracterizado por
  - C comodidad
  - D pérdida de dinero disponible
  - E pérdida de dinero esencial
  - L pérdida de vida
- Según el perfil de riesgo se elije el Crystal que corresponda
- Para Forum, Crystal Clear extendido a E8 debiera ser suficiente



Fuente: Alistair Cockburn. What is *Crystal?*
© 1999 Humans and Technology, Inc.

© 2004-2008 Pablo Straub

Crystal Clear: Alcance, roles y equipos

- Alcance
  - Para proyectos D6 (6 personas sin pérdida de dinero esencial)
  - Puede extenderse a E8 (8 personas sin riesgo vital)
  - No es para proyectos grandes (poca coordinación grupal)
  - No es para proyectos con riesgo vital (poca verificación)
- Roles
  - Requeridos: auspiciador, diseñador/programador, diseñador senior, usuario (dedicación parcial)
  - Roles combinados: coordinador, experto en el negocio, levantador de requisitos
- Equipo
  - Un único equipo de diseñadores/programadores
  - Todos sentados en la misma oficina o en oficinas adyacentes

© 2004-2008 Pablo Straub

Crystal Clear: Productos e hitos

- Productos
  - Todos se publican y se revisan
  - Mucha tolerancia en los productos de trabajo
    - Secuencia de entregas, calendario de vistas del usuario, entregas
    - Lista de actores y objetivos; casos de uso
    - Bosquejos de diseños y notas, borradores de las pantallas
    - Modelo común de objetos
    - Código ejecutable, código de migración, casos de uso
    - Manual de usuario
- Declaraciones
  - Requisitos lo bastante estables para diseñarlos
  - Interfaz con el usuario lo bastante estable para documentarla
  - Aplicación lo bastante correcta para entregarla

© 2004-2008 Pablo Straub

Crystal Clear: Estándares

- Política
  - Entregas al usuario cada 2 ± 1 meses
  - Seguimiento por hitos, no por productos de trabajo
  - Requisitos como casos de uso con anotaciones
  - Pruebas de regresión obligatorias de la funcionalidad
  - Revisión por pares del código
  - Modelo de propietarios de los productos de trabajo
  - Políticas son obligatorias pero se pueden sustituir (ej., Scrum)
- Estándares locales (de cada equipo)
  - Estilo de programación
  - Estilo de interfaz con el usuario
  - Mecanismo de pruebas de regresión

© 2004-2008 Pablo Straub

Elija Crystal Clear si...

- ... quiere hacer un montón de definición de proceso usted mismo
- ... tiene un equipo pequeño todos en la misma oficina
- ... los requisitos son principalmente conocidos o “conocibles” de antemano
- ... el proyecto arriesga pérdidas de dinero no esencial y nunca pérdida de vida humana

Fuente: Michael Cohn. Selecting an Agile Process, *21 Septiembre 2004*.

© 2004-2008 Pablo Straub