

qué es XP?

“un estilo de desarrollo de software centrado en la aplicación excelente de técnicas de programación, comunicación clara y trabajo en equipo que permite conseguir objetivos antes impensables”

- basada en valores (comunicación, realimentación, simplicidad, valentía, respeto)
- con un cuerpo de prácticas útiles
- con un conjunto de principios complementarios
- con una comunidad que comparte todo lo anterior

Fuente: [Extreme Programming Explained: Embrace Change](#), Kent Beck

¿qué lo distingue de otras metodologías?

- ciclos de desarrollo cortos
- enfoque de planificación incremental
- habilidad para responder al cambio con flexibilidad
- confianza en **tests** automatizados escritos por desarrolladores, clientes y testers

¿qué lo distingue de otras metodologías?

- confianza en comunicación oral, tests y código para **comunicar intención y estructura** de los sistemas
- confianza en diseño evolutivo
- confianza en colaboración estrecha de individuos con **talento ordinario**
- confianza en prácticas en sintonía con los instintos a corto plazo de los miembros del equipo y los intereses a largo plazo del proyecto

¿cómo gestiona los riesgos?

riesgo

desviaciones en la
planificación

cancelación de
proyecto

el sistema sale mal

XP

ciclos cortos y funcionalidad más
importante primero

maximizar valor de las entregas

los tests evitan la acumulación de
problemas

¿cómo gestiona los riesgos?

riesgo

XP

tasa de defectos

testeo de programadores y clientes

mala comprensión del negocio

implicación directa del cliente

cambios en requisitos

reducción de los ciclos de entrega

¿cómo gestiona los riesgos?

riesgo	XP
características no utilizadas	XP se centra en las tareas prioritarias que aportan mayor valor
rotación del personal	los programadores asumen responsabilidad y se reduce su frustración por "pedirles lo imposible"

valores

- dan propósito a las prácticas
- son universales
- idealmente, son los mismos en el trabajo y en la vida

comunicación

- importante para crear sentido de equipo y cooperación efectiva
- ¿qué comunicación necesitamos para enfocar el problema?
- ¿qué comunicación necesitamos para no volver a tener este problema?

simplicidad

qué es lo más simple que podría funcionar?

eliminar complejidad sobrante

YAGNI: "no lo vamos a necesitar"

se realimenta con la comunicación

el código simple:

pasa los tests

expresa toda idea que necesita expresar

lo dice todo una sola vez

no tiene partes superfluas

By [Ward Cunningham](#)

realimentación

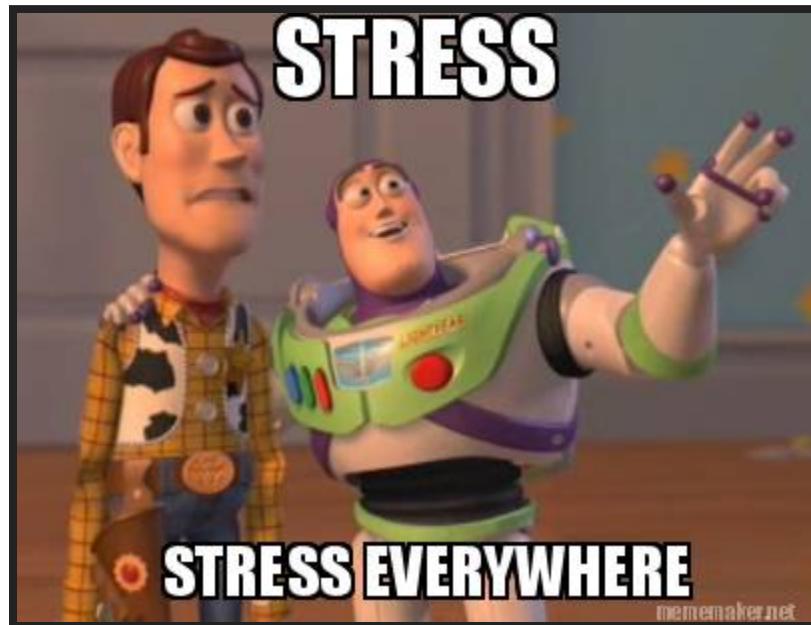
- los cambios son inevitables, pero necesitan de realimentación
- la realimentación nos acerca a nuestros objetivos
- cuanta más mejor: cuanto antes sepamos, antes podemos adaptarnos

formas de realimentación

- opiniones del equipo sobre una idea
- "pinta" del código una vez implementada la idea
- si los tests se han podido escribir fácilmente
- si los tests se ejecutan
- si la idea funciona una vez puesta en marcha

¡ojo!

demasiada realimentación...



valentía

acción efectiva frente al miedo

debe ser usada conjuntamente con el resto de valores

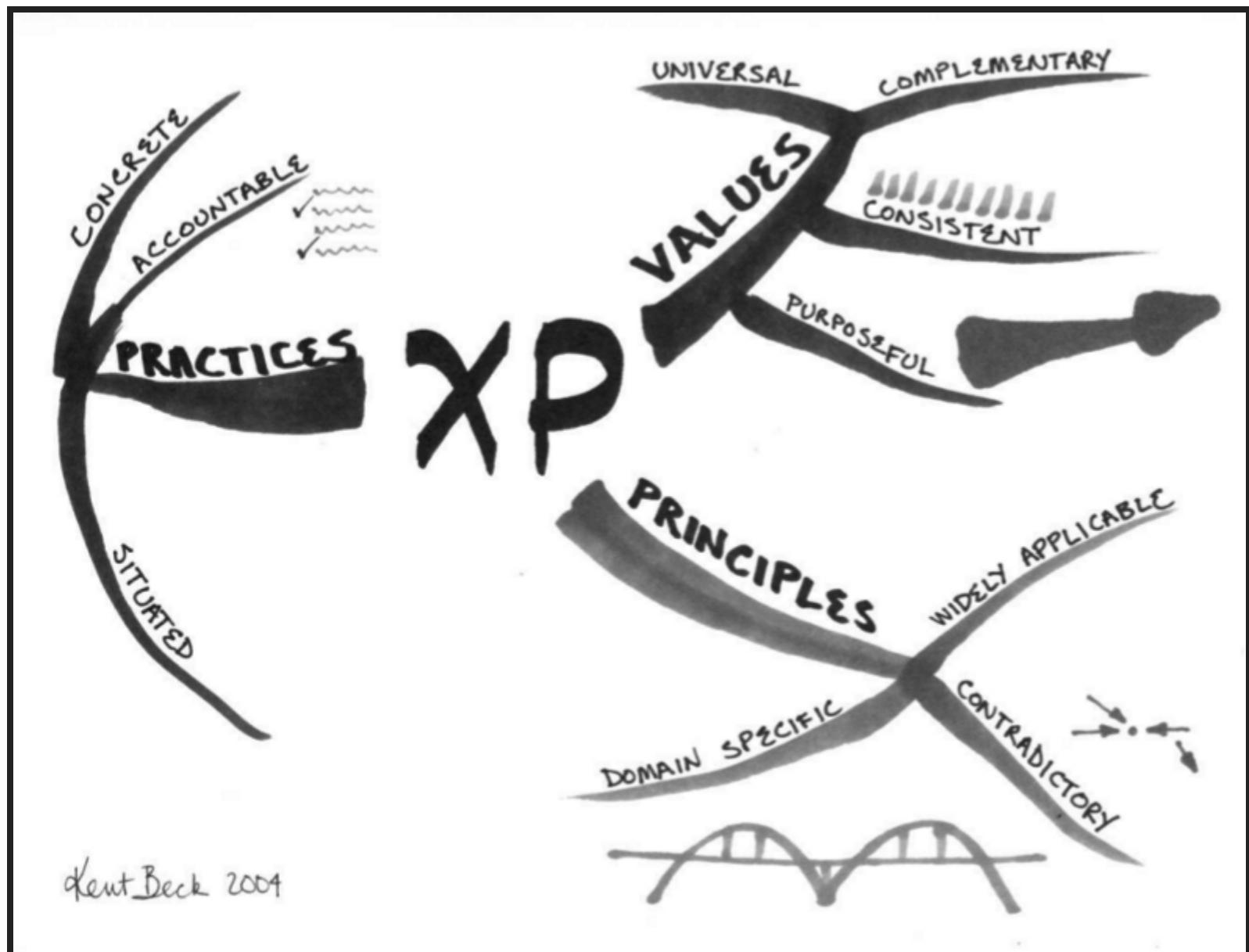
a veces significa actuar, a veces ser paciente

respeto

“si los miembros del equipo no se preocupan del resto y de lo que están haciendo, XP no funcionará”

yo soy importante, y tú también

¿otros?



principios

- llenan el vacío entre valores y prácticas
- los principios son directrices específicas para la vida
 - no son tan abstractos como los valores

humanidad

¿qué necesitamos para ser buenos programadores?

- seguridad básica
- realización
- sentimiento de pertenencia
- crecimiento
- intimidad

economía

un euro hoy vale más que un euro mañana: el desarrollo de software aporta más valor cuanto antes produzca beneficio

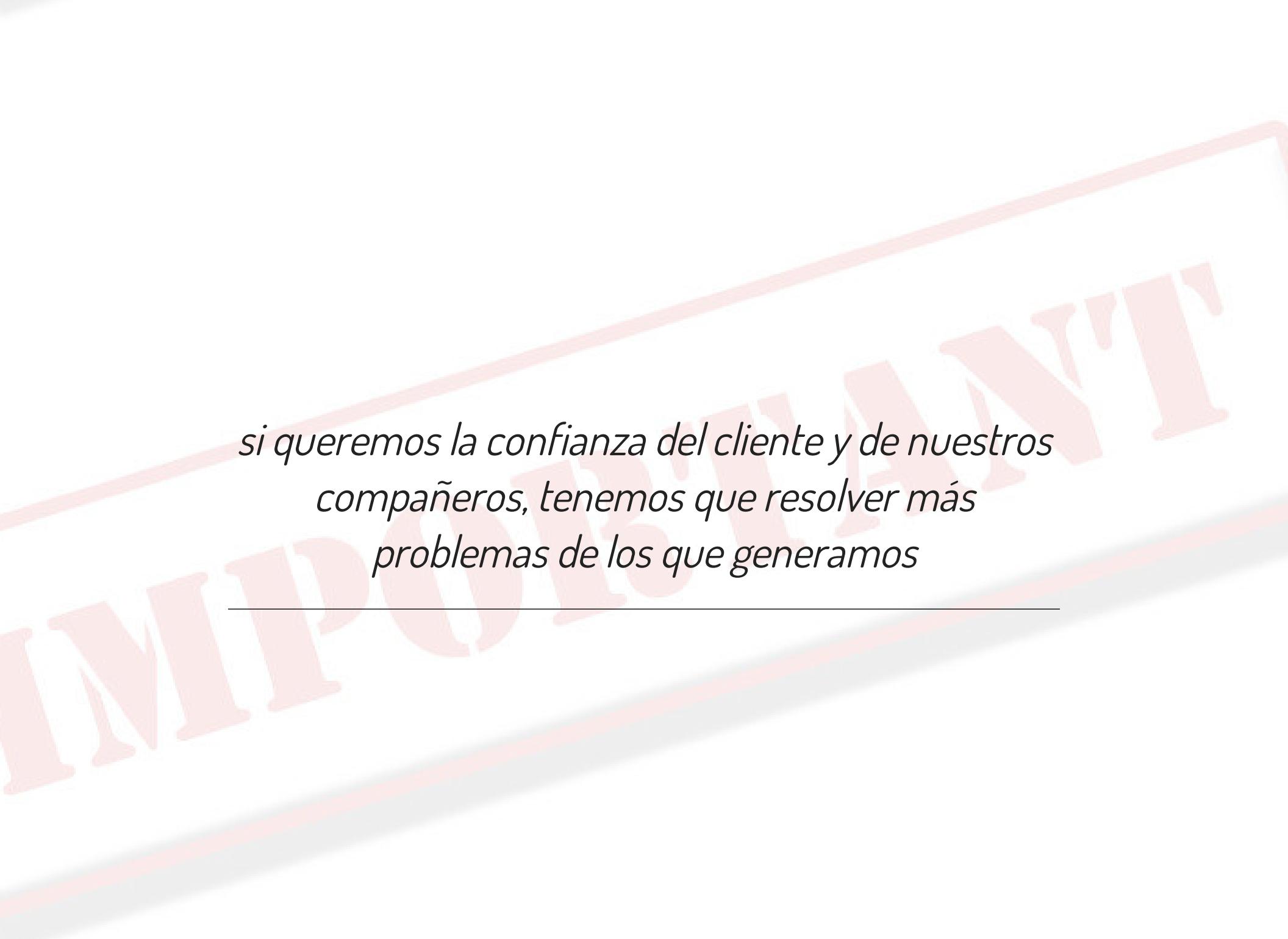
valor de las opciones: mi software vale más si puedo usarlo para otros fines distintos de su propósito (!)

beneficio mutuo

toda actividad debería beneficiar a todos los implicados

comunicación con el futuro:

- pruebas automatizadas que mejoran diseño e implementación hoy
- refactorización para eliminar complejidad adicional
- elección coherente de nomenclatura



*si queremos la confianza del cliente y de nuestros
compañeros, tenemos que resolver más
problemas de los que generamos*

auto-semejanza

intentemos copiar la estructura de una solución en un contexto nuevo

(!)

mejora

no existe el proceso perfecto

pero sí podemos perfeccionar los procesos

hoy: una solución justa

mañana: una solución mejor

mejorar el diseño y eliminar lo sobrante iterativamente

pero nunca una solución perfecta

good enough is fine



a algunos les gusta buscar soluciones complejas para los problemas...

una idea mejor: maximizar eficiencia y minimizar esfuerzo

solución judo, mejor que:

- no hacer nada mientras se espera a la solución perfecta
- desperdiciar recursos

y algo lo suficientemente bueno se puede convertir en mejor después

Fuente: [Rework](#): *good enough is fine*, David Heinemeier Hanson, Jason Fried

diversidad

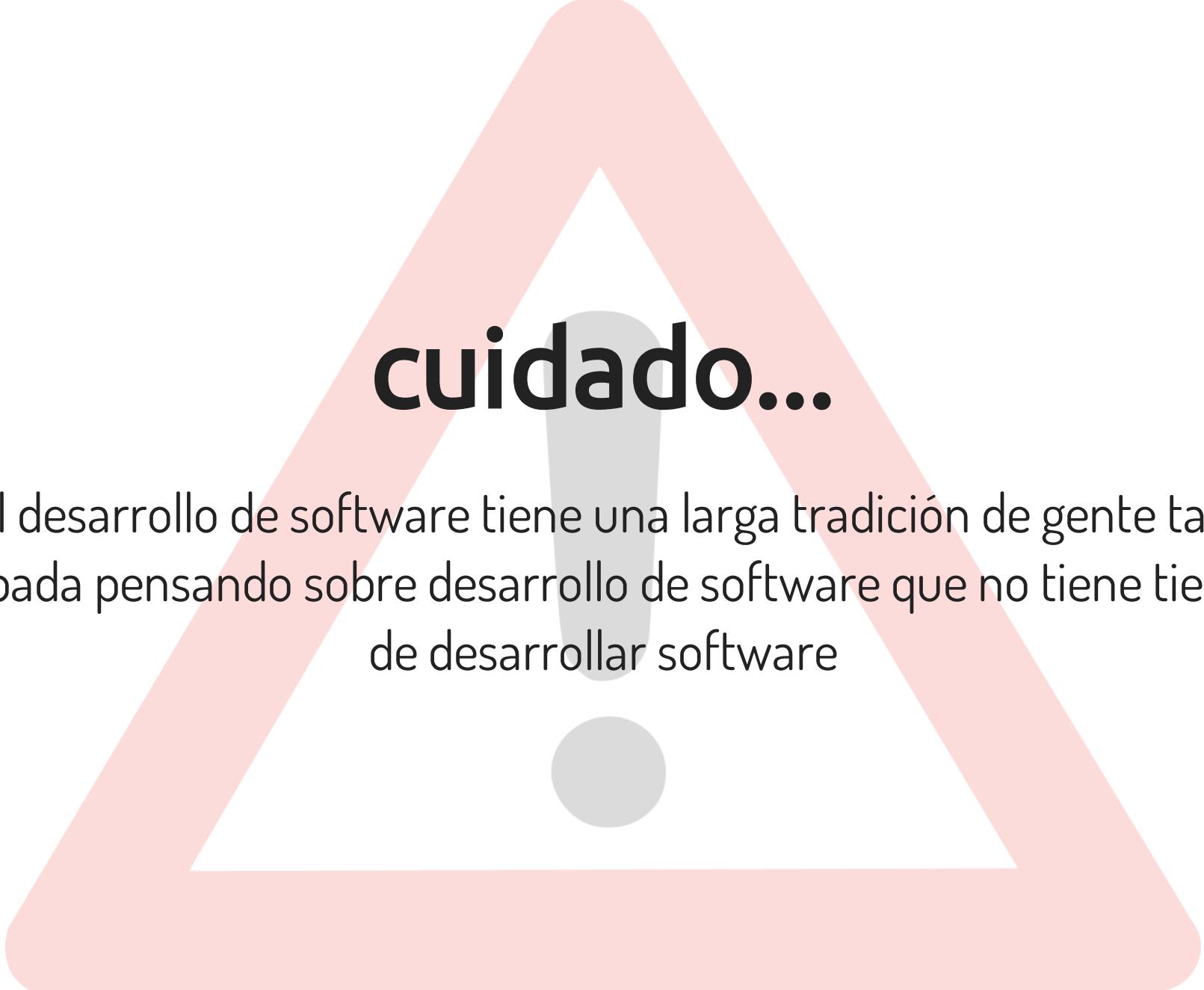
los equipos necesitan variedad de perfiles, habilidades y actitudes...

...lo que conlleva **conflicto** (varias formas de resolver un problema)

un conflicto debe enfocarse como una oportunidad y no como un problema

reflexión

los buenos equipos no sólo hacen su trabajo;
piensan **cómo** lo están haciendo
piensan **por qué** lo están haciendo
no esconden sus errores, sino que los exponen y aprenden de ellos



cuidado...

el desarrollo de software tiene una larga tradición de gente tan
ocupada pensando sobre desarrollo de software que no tiene tiempo
de desarrollar software

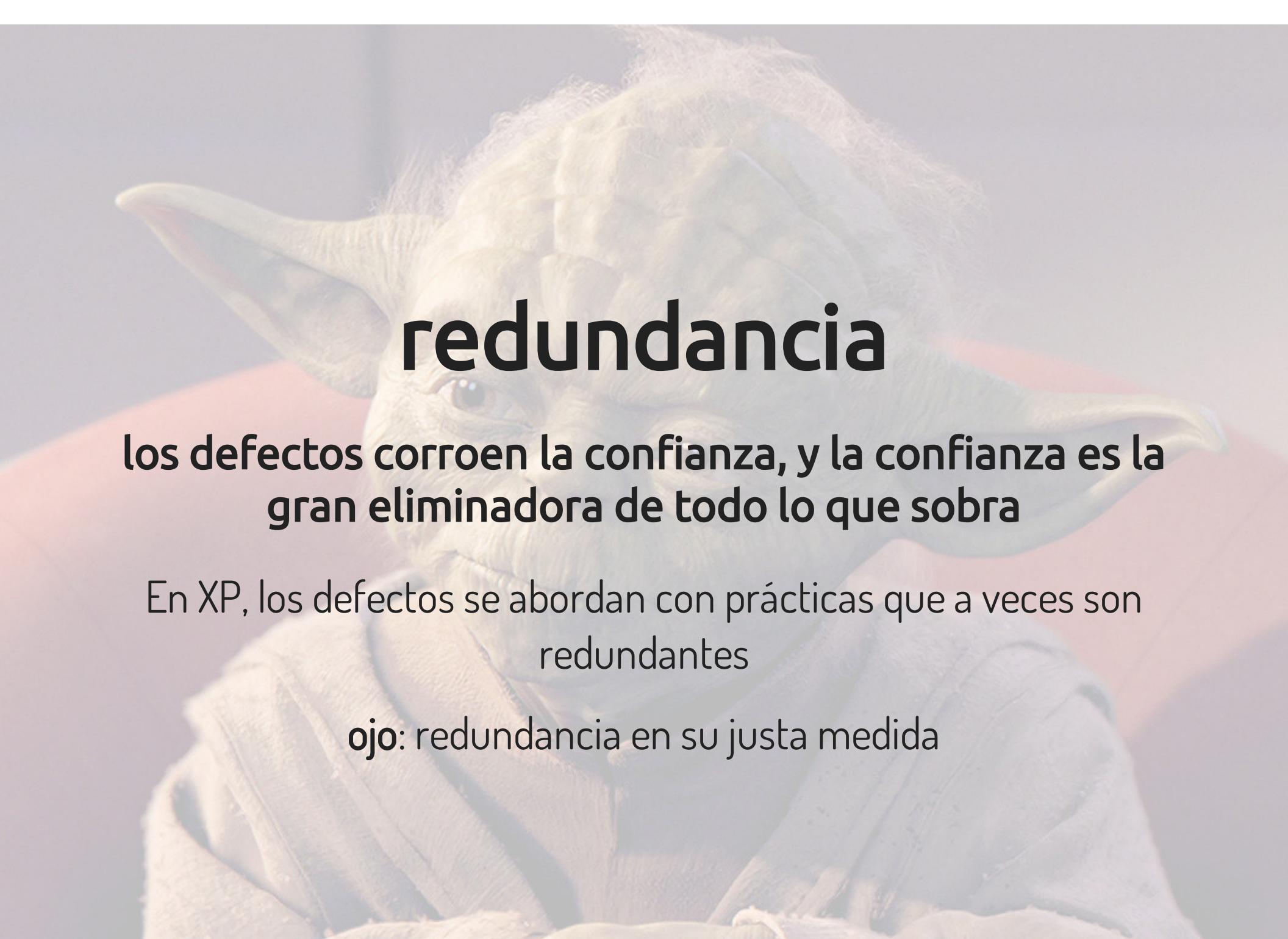
flujo

“entregar un flujo continuo de software valioso mediante la participación en todas las actividades de desarrollo de forma simultánea”

- evitar el "Big Bang"
- integrar y entregar con frecuencia en pequeñas cantidades

oportunidad

para alcanzar la excelencia, los problemas han de transformarse en oportunidades para el aprendizaje y mejora, no sólo en supervivencia



redundancia

los defectos corroen la confianza, y la confianza es la gran eliminadora de todo lo que sobra

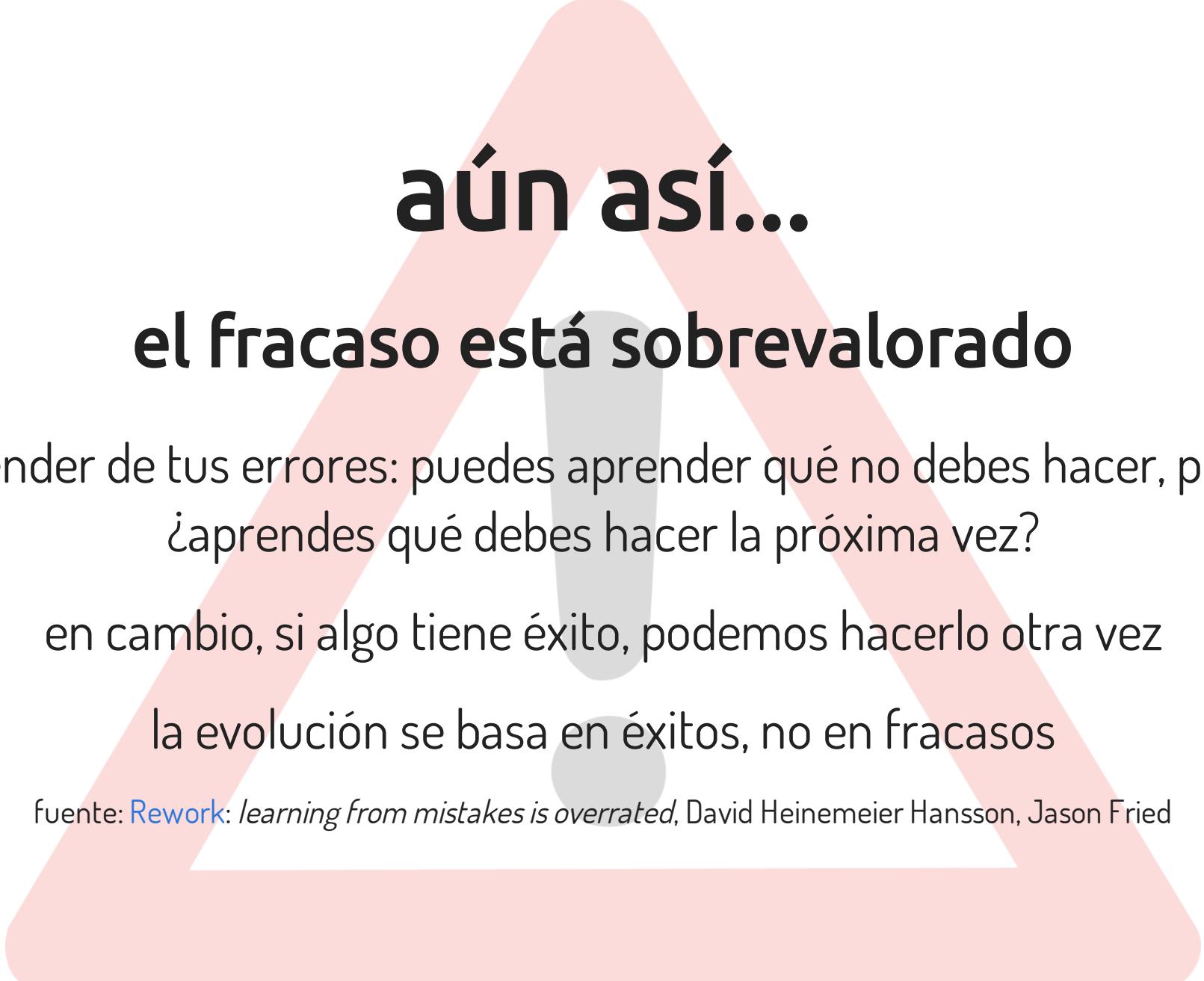
En XP, los defectos se abordan con prácticas que a veces son redundantes

ojo: redundancia en su justa medida

fracaso

probemos, probemos, probemos

el fracaso no es una pérdida de tiempo si produce conocimiento
si no se sabe qué hacer, arriesgarse al fracaso puede ser el camino más
corto para el éxito



aún así...

el fracaso está sobrevalorado

aprender de tus errores: puedes aprender qué no debes hacer, pero...

¿aprendes qué debes hacer la próxima vez?

en cambio, si algo tiene éxito, podemos hacerlo otra vez

la evolución se basa en éxitos, no en fracasos

Fuente: [Rework](#): *learning from mistakes is overrated*, David Heinemeier Hansson, Jason Fried

calidad

los proyectos no se terminan antes a costa de sacrificar la calidad, más bien al contrario.

en cambio:

- entregas más rápidas
- mayor productividad
- mayor efectividad

además de tratarse de un factor económico, las personas necesitan poder enorgullecerse de algo

Y ENTONCES...

**COMO CONTROLO EL
PROYECTO?**

memegenerator.net

tiempo y coste suelen estar fijos...

XP elige el **alcance** como medio principal de planificación, seguimiento
y gobernanza

como el alcance nunca se conoce con total precisión, parece una
buena elección

cuidado...

la preocupación por la calidad no es una excusa para la inacción

si se desconoce una forma limpia de llevar a cabo un trabajo,
hagámóslo lo mejor que podamos

si la mejor forma de llevarlo a cabo lleva mucho tiempo, hagámoslo en
el tiempo disponible de la mejor forma posible y luego evolucionémoslo

**la calidad es gratis
(Crosby)**

pequeños pasos

el *overhead* introducido desarrollando en pasos pequeños es menor que el derivado de recuperarse de una gran integración fallida

las personas digieren mejor pequeños cambios

ejemplo: ¡implantar esta metodología!

responsabilidad aceptada

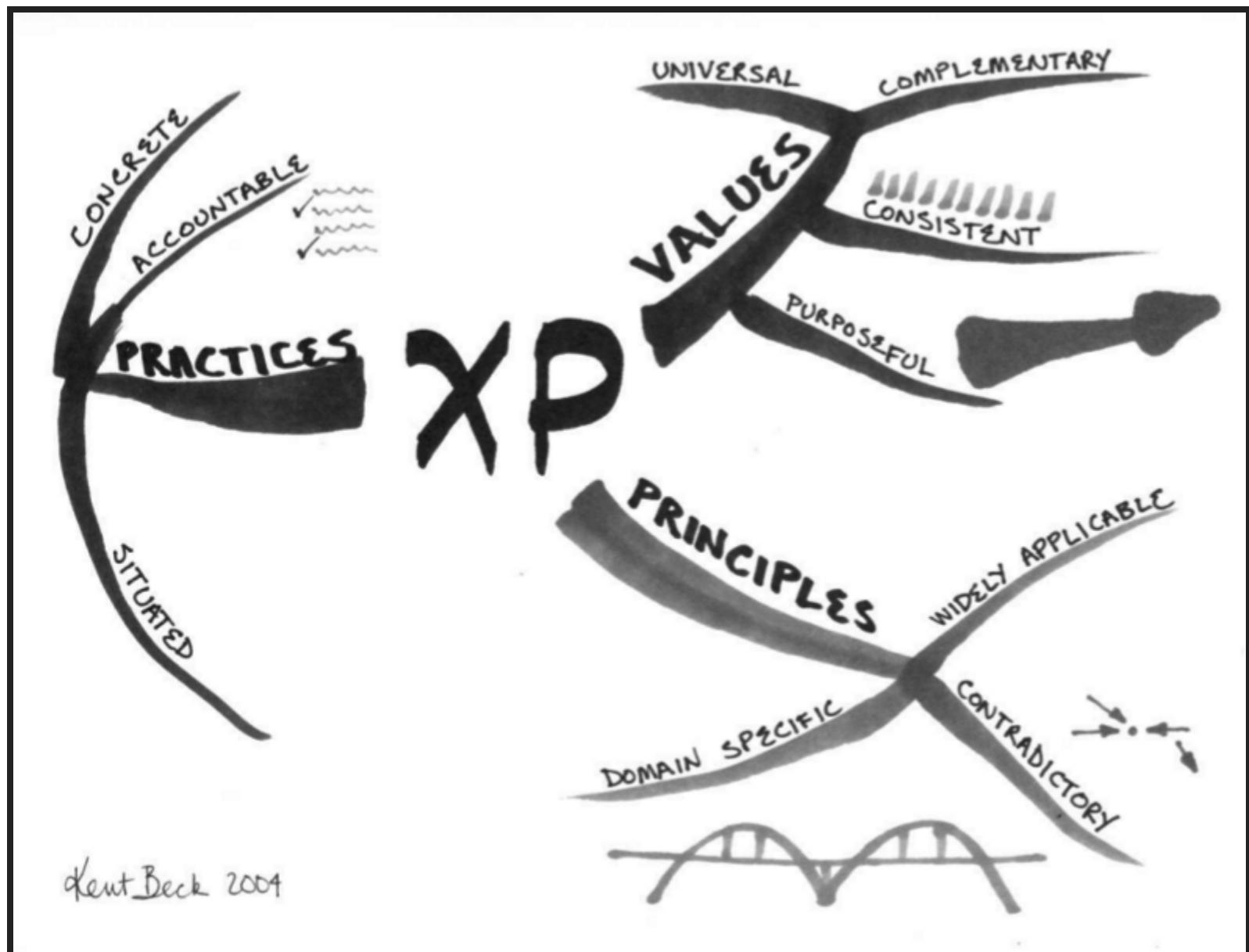
la responsabilidad no se asigna; se acepta

llevado a la práctica:

- quien hace el trabajo también lo estima
- quien implementa, también diseña y prueba

y con la responsabilidad viene la autoridad

si un experto en el proceso es capaz de decirme cómo he de trabajar pero no de participar en dicho trabajo o sus consecuencias, autoridad y responsabilidad dejan de estar alineadas.



prácticas

- la evidencia de los valores
- son claras
- son concretas

sentarse juntos

desarrollar en un espacio lo suficientemente amplio para todo el equipo

mantener espacios para conservar privacidad

si el sentimiento de seguridad de un equipo va atado a sus pequeños espacios propios, quitárselo antes de reemplazarlo por un sentimiento de logro compartido producirá resistencia al cambio

equipo completo

incluir en el equipo personas con todas las habilidades y perspectivas necesarias para que el proyecto tenga éxito:

equipos multifuncionales

¿y de qué tamaño?

Dos límites:

- 12: interacción cómoda en el día a día
- 150: con más personas, no se reconocen las caras

Fuente: The Tipping Point, Malcolm Gladwell

¿y si asigno a una misma persona a varios grupos?

Un alto porcentaje del tiempo disponible se pierde en **cambio de contexto**

Se pierde la sensacion de pertenencia

espacio de trabajo informativo

conseguir que un observador con interés pueda hacerse una idea del estado de los proyectos en 15 segundos:

pizarras, gráficas, indicadores, paredes... todo en un golpe de vista

trabajo con energía

quemarse no ayuda al equipo

cansados es difícil reconocer si se está restando valor al producto

¡¡Vete a dormir!!

obstinación, falta de creatividad, disminución de moral, irritabilidad...

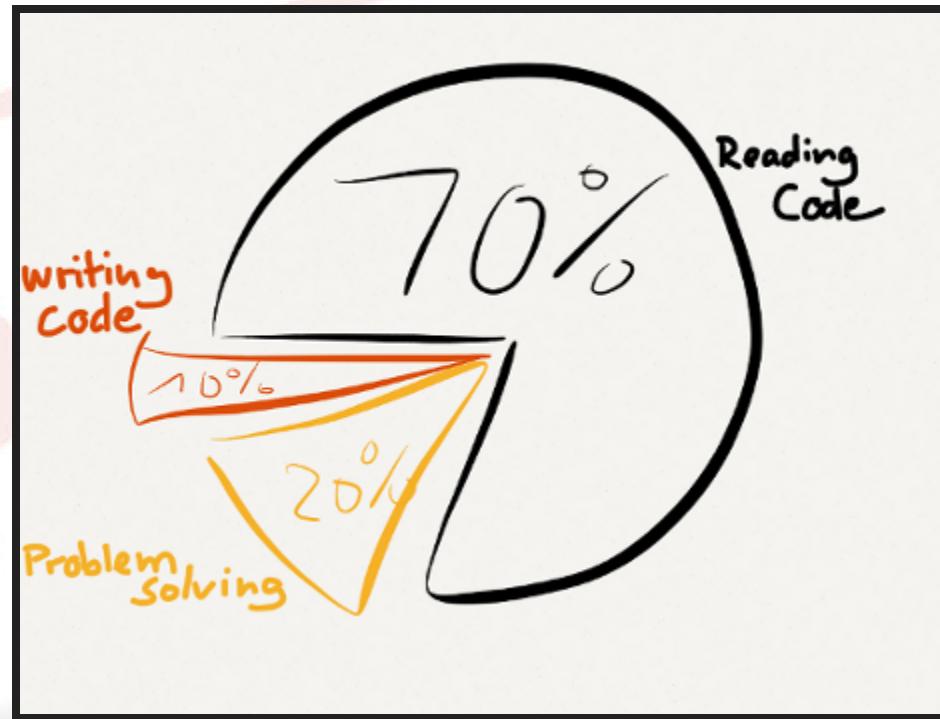
Fuente: [Rework](#): *go to sleep*, David Heinemeier Hanson, Jason Fried

programación en parejas

- realimentación en la tarea
- tormentas y clarificación de ideas
- iniciativa cuando el compañero está atascado
- responsabilidad compartida

¡no significa que no podamos pensar solos!

economía del pair-programming



fuente: Olaf Lewitz, Agile coach

herramientas

- teamviewer
- tmux/wemux

ejemplo

historias

planificar usando unidades de funcionalidad visible para el cliente
"requisitos": connotación absolutista y permanente
y luego, *estimar* (lo que le permite al equipo dividir, combinar o ampliar alcance basándose en el valor conocido de las características)

una historia es una promesa de tener una conversación

estructura

ID	COSTE
COMO <ROL>	
QUIERO <CARACTERÍSTICAS>	
PARA <VALOR>	

interpretación

no es un requisito funcional

- tiene menos detalle
- su objetivo es lograr interacción entre el equipo y el cliente

no es un caso de uso

- no proporciona contexto para el diseño
- no proporciona sentido de completitud
- no proporciona anticipación

¡¡podemos tener miles!!

alternativa

el caso de uso

- resuelve los problemas descritos
- no debe ocupar más de 1-2 páginas

se puede mapear con historias: un caso de uso puede contener 10 o más historias

características

- independiente: que pueda implementarse en cualquier orden
- negociable: sus detalles se acordarán en la conversación
- valiosa
- estimable con la precisión suficiente para ayudar al cliente a priorizar
- pequeña: días/persona
- comprobable: si el equipo no puede probar una historia nunca sabrá si la ha terminado

asignación de valor

MoSCoW

- M: must (debe tenerla)
- S: should (debería tenerla, aunque la solución no fallará si no existe)
- C: could (sería conveniente que la tuviera)
- W: won't (no está en los planes tenerla)

ciclos semanales

- revisión del progreso hasta la fecha (semana anterior)
- selección con el cliente de historias a implementar esta semana
- división de las historias en tareas y **estimación** (mal necesario)

todo el mundo se centra en el Viernes

¿y la asignación de tareas?

el desarrollador se convierte en propietario de la tarea

- el propio equipo reparte responsabilidades
- los desarrolladores se auto-asignan la siguiente tarea de la pila

¿pros y contras?

ciclos trimestrales

- identificar cuellos de botella (especialmente fuera del equipo)
- planificar el trimestre siguiente con temas
- centrarse en el panorama general (*the big picture*)
- propuestas de experimentos a largo plazo

la parte floja: *slack*

“comprometerse más de lo posible genera defectos, falta de confianza, baja moral y relaciones antagónicas”

incluir tareas menores que puedan dejarse atrás y fomenten la creatividad

Ej: Google Mail, AdSense..

construir en diez minutos

construir automáticamente el sistema **completo** y ejecutar **todos** sus tests en 10 minutos

integración continua

divide, integra y vencerás

- asíncrona: se pierde tiempo de reflexión
- síncrona (no más de 2 horas): se evitan interrupciones

la integración continua ha de ser lo suficientemente completa como para que el primer despliegue del sistema en producción no sea un problema

herramientas

- jenkins
- travis ci

los tests primero

- evita incluir código innecesario
- si es difícil escribir un test, hay un problema de diseño
- los tests que funcionan dan confianza
- se coge ritmo de trabajo: **test -> code -> refactor**

contra: sólo se trabaja con una parte muy pequeña del sistema

pro: se pueden escribir miles de ellos y construir en diez minutos

diseño incremental

tradicionalmente: diseño antes de implementación

justificación: Bohem (coste de corregir un defecto se incrementa exponencialmente con el tiempo)

pero, ¿y los cambios?

los equipos XP confían en su habilidad para adaptar el diseño a requisitos futuros: mantener las tareas de diseño en proporción a las necesidades del sistema

- ¿cómo? en pequeños pasos
- ¿cuándo? lo más tarde posible

otras prácticas

análisis causa-efecto: 5 whys

equipos más pequeños: liberar personal para formar otros equipos

único flujo de código: **evitar demasiadas ramas**

despliegues diarios: evitar el riesgo derivado de la diferencia de versiones entre el código en producción y el desarrollado por el programador

código y pruebas: artefactos permanentes que permiten **generar otros documentos**

involucrar realmente al cliente

hacer que las personas cuyas vidas o trabajo estén afectadas por nuestros sistemas formen parte del equipo

conviene que ciertos clientes "visionarios" participen en reuniones
trabajar sin clientes o con "proxies" de clientes llevará al desperdicio:

- funcionalidad que no se utiliza
- tests que no reflejan los criterios de aceptación reales
- pérdida de la oportunidad de construir relaciones reales

¿confían los clientes en nosotros?

despliegues incrementales

los grandes despliegues tienen altos riesgos y altos costes de recursos humanos y económicos

¿la alternativa?

dividir el sistema en pequeñas piezas de funcionalidad manejables y encontrar una forma de correrlas en paralelo con el sistema antiguo

contratos de alcance negociado

fijar coste, tiempo y calidad pero dejar un margen para el alcance
reducir riesgos sacando pequeños contratos en vez de uno grande

*un mecanismo que proporciona a todos los
participantes la valentía necesaria para hacer lo
que parece apropiado hoy, no algo ineficiente tan
sólo porque está en el contrato*

¡ojo con el fraccionamiento!

¿cómo escala XP?

en número

- transformar el problema en problemas más pequeños
 - aplicar soluciones sencillas
 - aplicar soluciones complejas si aún persiste algún problema
- partir en muchos problemas para muchos equipos puede traer problemas de integración...

solución: integrar frecuentemente

¿cómo escala XP?

en una organización grande

mantener la comunicación con el resto de la organización usando los canales habituales

presentar la información en una forma que la organización pueda absorber

¿cómo escala XP?

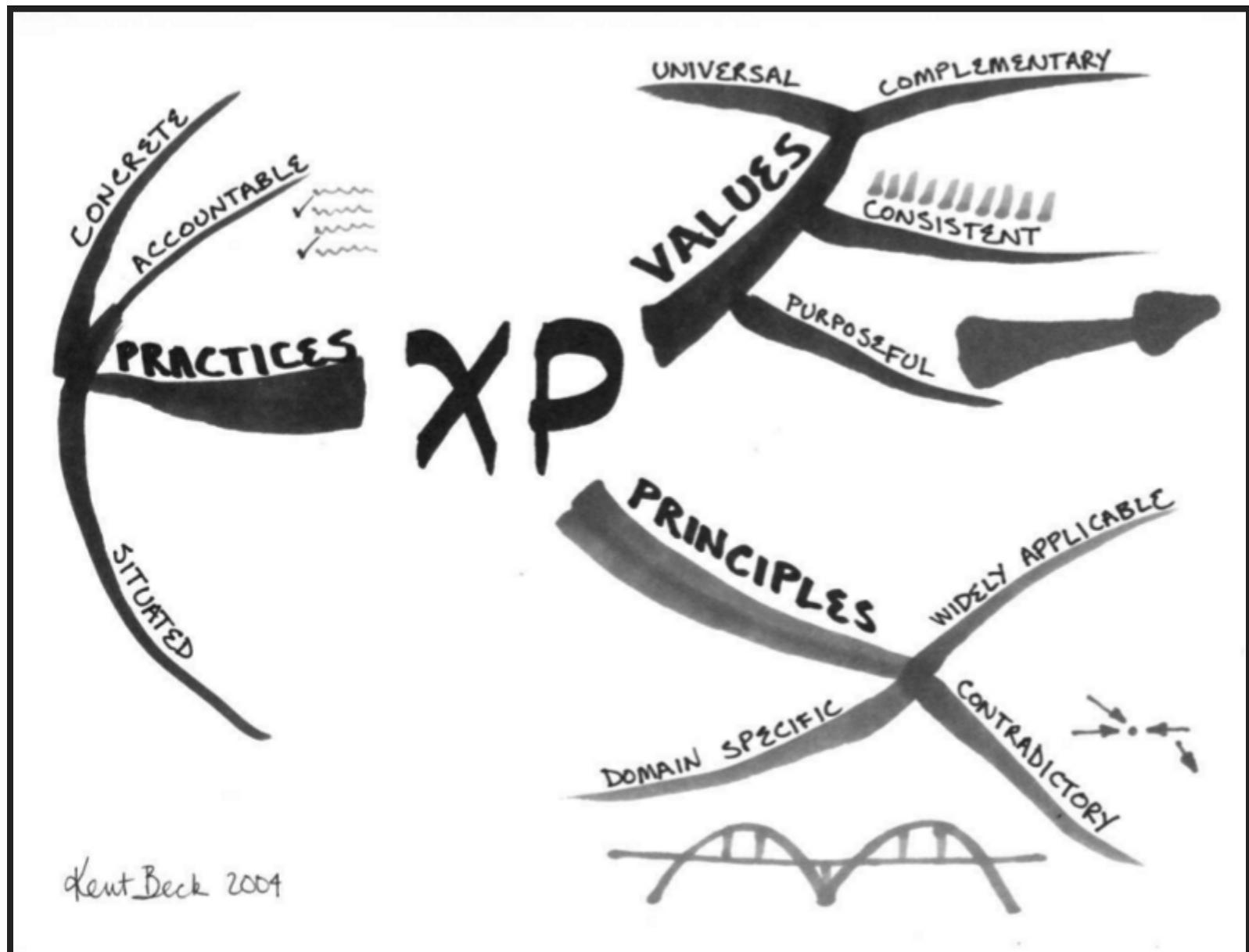
en el tiempo

el caso más simple: mantener la continuidad del equipo en todo el ciclo de vida del producto

los proyectos largos funcionan bien con XP porque los tests previenen muchos de los errores habituales de mantenimiento

un mensaje

*si hay un mensaje que me gustaría comunicar,
cualquiera que sea tu puesto de trabajo y aunque
el desarrollo de software sólo te afecte
tangencialmente, es el siguiente: el desarrollo de
software es capaz de mucho, mucho más de lo
que está entregando actualmente -Kent Beck*





[Acceso al repositorio del curso](#)

[Miguel Expósito Martín](#)