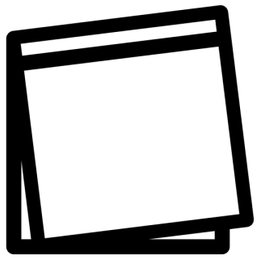


¿Qué es eXtreme Programming (XP)?

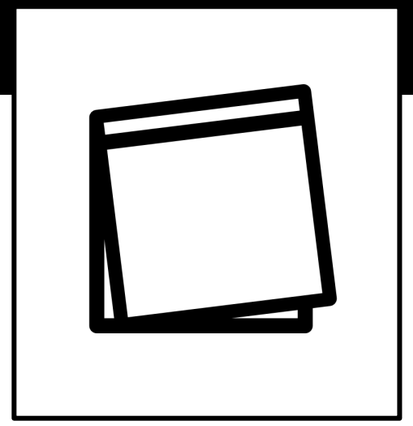
Es una **filosofía para el desarrollo de software**, basado en un conjunto de **prácticas útiles y técnicas probadas** para implementar código de forma que su diseño, arquitectura y codificación permitan incorporar modificaciones y añadir nuevas funcionalidades sin demasiado impacto en la calidad del mismo.

LAS PROMESAS DE XP

1. **Reducir el riesgo** del proyecto.
2. Mejorar la **respuesta ante cambios** en el negocio.
3. **Mejorar la productividad** a lo largo de toda la vida del software.
4. Los equipos XP producen software de calidad a un **ritmo sostenible**.

SE CARACTERIZA POR:

- Funciona con equipos de **cualquier tamaño**. Los valores y principios detrás de XP son aplicables a cualquier escala.
- Se adapta a **requisitos vagos o que cambian rápidamente**. Sin embargo, también aplica en proyectos en los que los requisitos no son tan volátiles.
- Sus **cortos ciclos de desarrollo**, que dan como resultado feedback tempranos, concretos y continuos.
- Su aproximación a la **planificación incremental**, que genera un plan general que se espera que evolucione a lo largo de la vida del proyecto.
- Su capacidad para **programar de manera flexible** la implementación de las funcionalidades, respondiendo a las necesidades cambiantes del negocio.
- Su **dependencia de pruebas automatizadas, desarrollada con TDD,** para monitorear el progreso del desarrollo y permitir que el sistema evolucione detectando los defectos lo más pronto posible.
- Su dependencia de la **comunicación oral**, **las pruebas y el código fuente** para comunicar la estructura y la intención del sistema. Para ello usa la figura del Sistema de Metáforas (System Metaphor) para describir el sistema de forma sencilla.
- Se basa en un proceso de **diseño evolutivo** que dura tanto como el sistema.
- Su dependencia de la **estrecha colaboración de personas** activamente comprometidas con talento y la excelencia.



VALORES

- Los **valores** son los criterios a gran escala que usamos para juzgar lo que vemos, pensamos y hacemos.
- Los **valores aportan un propósito a las prácticas** y deben hacerse explícitas para que las prácticas no se vuelvan rutinarias.

Ej. *Un desarrollador en una Retrospectiva da un feedback poco detallado y ambiguo. Ya ha dado un feedback pero este no es transparente y no revela lo que realmente opina por temor a ser juzgado y que sus palabras se vuelvan en su contra posteriormente. Realmente se está negando a dar un feedback más profundo de cómo ha visto el Sprint ya que no quiere tener responsabilidades por haber sido injustamente acusado en el pasado. En este caso el desarrollador valora la protección por encima de la comunicación.*

PRÁCTICAS

- Las **prácticas** son las actividades que se realizan dentro del equipo de forma continuada en el día a día y conforme a unas **reglas claras y objetivas**.
- Su ejecución da como resultado la adquisición de **nuevas habilidades y experiencia**.
- Las prácticas son la **evidencia de los valores** y pueden cambiar dependiendo de la situación.

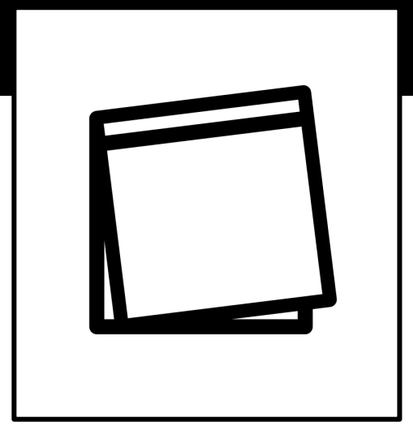
Ej. *Si valoras la comunicación, algunas de tus prácticas probablemente sean: asistir a la Stand Up Daily Meeting a diario, o bien te preocuparás cuando recibas información confusa y preferirás una reunión cara a cara, al envío de un email para aclarar las dudas. Por tanto, darás soporte a todo tipo de prácticas que mejoren y fomenten la comunicación.*

PRINCIPIOS

- Los **principios** tienden un puente entre los valores y las prácticas.
- Son un **sistema de normas o leyes** que deben ser cumplidas para que un sistema funcione adecuadamente.

Ej. *Un jardinero que pretenda repoblar una montaña sabrá que si planta pinos, la hierba no crecerá ya que al competir por agua las raíces tan extensas de los pinos no les permitirán la supervivencia. Por tanto, se basará en el Principio de compañía para identificar qué plantas y árboles pueden crecer fuertes y sanos juntas.*

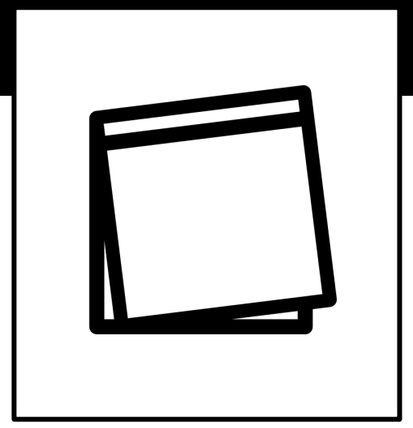
Se puede aplicar el mismo principio en la creación de un equipo de desarrollo a la hora de seleccionar los integrantes.



Valores XP

XP está basado en valores. No se trata de un conjunto de reglas, sino de una **forma de trabajar de acuerdo a tus valores personales y corporativos**. XP recomienda comenzar con los valores listados a continuación y posteriormente agregar aquellos que se alineen con el comportamiento del equipo.

COMUNICACIÓN	La comunicación **es el valor más importante** dentro de un equipo de desarrollo. Aunque algunos **problemas** son causados por la falta de conocimiento **la mayoría surgen por falta de comunicación**. Por tanto, cada uno es parte del equipo y **se comunican cara a cara** en la daily. **Se trabajará juntos en todo**, desde la definición de requisitos hasta la codificación y **se creará la mejor solución al problema entre todos.**
SIMPLICIDAD	La simplicidad es la esencia de XP, **es es el más intelectual de los valores de XP**. Por tanto, **nos centraremos en hacer lo que se necesita y se nos pide pero nada más**. Esto **maximizará el valor** creado para la inversión hecha hasta la fecha. Por esta razón, se tomarán pequeños pasos que nos hagan avanzar a nuestra meta y se mitigarán los fallos según se den. De modo que **crearemos algo de lo que estemos orgullosos** de mantener a largo plazo a un coste razonable.
FEEDBACK	El feedback es una **parte crítica de la comunicación**. Dado que **los cambios** son inevitables estos **generan la necesidad de obtener feedback**. Por consiguiente, usaremos este feedback **para acercarnos** más y más **a nuestros objetivos** y en consecuencia los equipos de XP se esforzarán por obtener la mayor cantidad de feedback que puedan manejar lo más rápido posible.
CORAJE	El coraje es la fuerza de **voluntad que puede desarrollar una persona**, sin miedo al fracaso, **para superar ciertos impedimentos** y que a veces se manifiesta como la tendencia a la acción. El coraje como valor sólo es peligroso pero junto con otros se hace poderoso. En XP se emplea para decir la verdad sobre el progreso y las estimaciones. No se teme a nada porque nunca se trabaja sólo.
RESPETO	Si los miembros de un equipo no se preocupan por el resto ni el trabajo que hacen, XP no funcionará. Por ello, **cada uno da y siente el respeto que ellos se merecen como un miembro valorado del equipo**. Cada uno aporta valor incluso si es simple entusiasmo. Los desarrolladores respetan la experiencia de los clientes y viceversa y la dirección respeta nuestro derecho a aceptar responsabilidad y recibe autoridad sobre nuestro propio trabajo.



Principios XP

Los principios permiten cerrar la brecha entre valores y prácticas, proveyendo pautas específicas de dominio para encontrar prácticas en armonía con los valores. **Los principios enumerados aquí no son los únicos posibles para guiar el desarrollo de software, pero estos son los principios que guían a XP.**

14 PRINCIPIOS

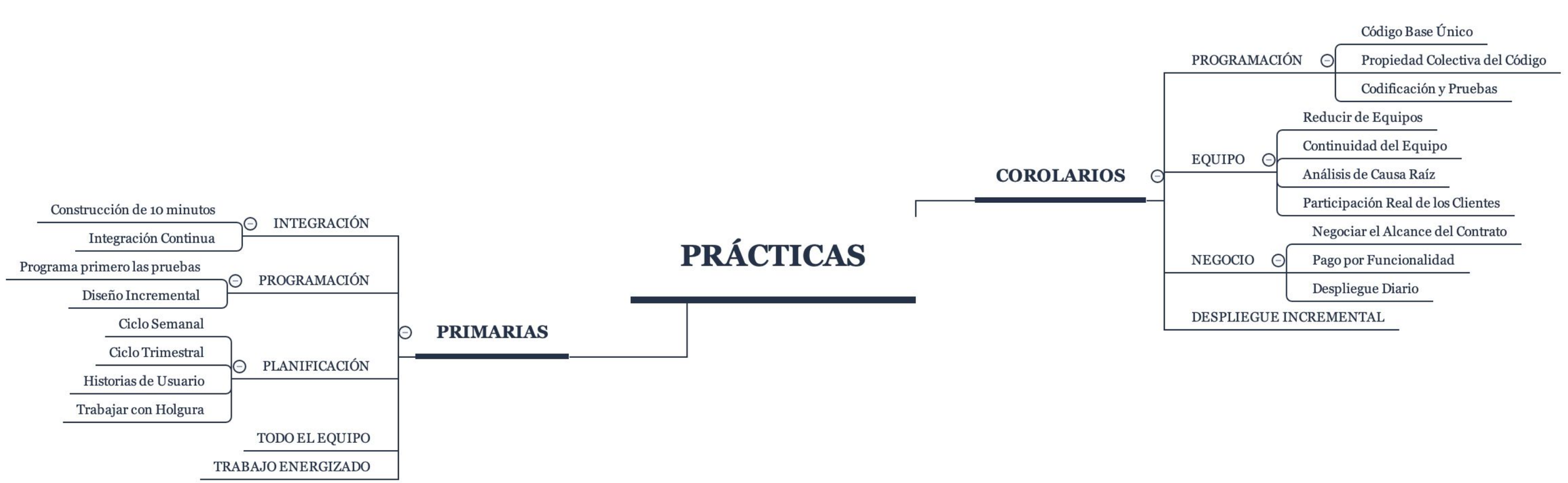
- **Humanidad**: El software lo desarrollan personas y es importante tener presente que los factores humanos son la clave para crear un software de calidad.
- **Economía**: El producto que se cree debe ser rentable tanto a corto como a largo plazo, debe producir beneficios.
- **Beneficio mutuo**: Uno de los principales principios de XP y de los más difíciles de poner en práctica. Propone pensar siempre en el beneficio de todas las partes implicadas en un desarrollo.
- **Auto-similitud**: Buscar soluciones similares en diferentes contextos. Los patrones de la forma de trabajo deben repetirse adaptándolos a la situación en la que estemos.
- **Mejora**: Excelencia en el desarrollo a través de la mejora. Se busca perfeccionar, no lo perfecto.
- **Diversidad**: Dos ideas sobre un diseño presentan una oportunidad, no un conflicto.
- **Reflexión**: Los buenos equipos piensan cómo y por qué hacen el trabajo, no esconden errores.
- **Flujo**: Entrega continua de valor.
- **Oportunidad**: Los problemas son oportunidades de cambio.
- **Redundancia**: Los problemas difíciles deben poder resolverse de diferentes maneras.
- **Fallo**: Un fallo no es un desperdicio si me sirve para aprender algo.
- **Calidad**: No vas a ir más rápido aceptando rebajar la calidad.
- **Pequeños pasos**: ¿Qué es lo menos que puedes hacer que sea reconocible y que vaya en la dirección correcta?
- **Responsabilidad aceptada**: El que acepta algo se hace responsable de ello.

Prácticas

Las prácticas son un elemento que influye en el comportamiento de las personas y las **hace ir en una dirección determinada desde donde están hasta donde pueden estar con XP**. Aunque aplicar una práctica es una elección, **las** que se indican son **prácticas** que **funcionan** y lo hacen aún **mejor cuando se aplican juntas**, ya que las interacciones entre las prácticas amplifican su efecto.

TIPOS DE PRÁCTICAS

Las prácticas se dividen en dos tipos: **13 prácticas primarias** y **11 corolarios.** Las **prácticas principales son útiles independientemente de lo que esté haciendo**. Cada una puede brindarle una mejora inmediata. Puedes comenzar con seguridad con cualquiera de ellas. Sin embargo, **los corolarios pueden resultar difíciles de aplicar sin dominar antes las prácticas primarias**. En las fichas posteriores se describe en qué consisten.



XP - Prácticas Primarias I

autentia

SENTARSE JUNTOS (SIT TOGETHER)

- Disponer de un **espacio abierto** en el que todo el equipo pueda trabajar.
- Una forma de satisfacer la necesidad de espacio privado es **limitando el número de horas de trabajo**.
- El espacio de trabajo debe promover que florezca la comunicación, **elimina los cubículos y las barreras que te impiden comunicarte** con las personas de tu alrededor.
- Cuantas **más reuniones cara-a-cara,** más humano y productivo se convierte el proyecto.

TODO EL EQUIPO (WHOLE TEAM)

- La gente necesita tener la **sensación de equipo**:
  - *Todos somos un equipo*
  - *Estamos en esto juntos*
  - *Apoyamos el trabajo, el crecimiento y el aprendizaje de todos*
- **Los equipos son dinámicos**. Si se requiere un conjunto de habilidades o actitudes, trae a una persona con esas habilidades al equipo. Si a alguien no se le requiere más, se libera y puede ir a otros lugares.

ESPACIO DE TRABAJO INFORMATIVO

- Hacer que el **espacio de trabajo se empape del proyecto** en el que estamos trabajando.
- Colocar las tarjetas con **las historias de usuario en las paredes**, agrupadas por estado.
  - *Proporcionan información rápidamente*
- Colocar también en un lugar visible distintas **gráficas de evolución del proyecto.**
  - *Información activa e importante*

PROGRAMA PRIMERO LAS PRUEBAS (TDD)

- Antes de escribir cualquier código escribir las pruebas o lo que es lo mismo **aplica un Desarrollo Orientado a Pruebas (Test Driven Development). Ventajas:**
  - ***Las pruebas sirven de especificación y aclaran el alcance*** *del código que tenemos que escribir.*
  - ***Cohesión y acoplamiento*** *- si es difícil escribir el test, es una señal de que tenemos un problema de diseño. El código débilmente acoplado y altamente cohesionado es más fácil de probar.*
  - ***Confianza*** *- escribiendo código limpio que funciona y demostrando tus intenciones con las pruebas construye una relación de confianza con tus compañeros.*
  - ***Ritmo*** *- es muy fácil perderse durante horas cuando se está programando. Con el enfoque de test-first, está claro lo que hay que hacer a continuación: o escribimos una prueba o hacemos que funcione una prueba rota (broken test). El ciclo que se genera se convierte pronto en algo natural y eficiente: test, code, refactor, test, code, refactor.*

TRABAJO ENERGIZADO (ENERGIZED WORK)

- **Trabajar** a pleno rendimiento, pero solo **tantas horas como se pueda ser productivo de forma sostenida.**
- Con suficiente **cafeína es muy fácil llegar al punto en el que el código que se escribe quita valor al proyecto** y (lo más peligroso) no darse cuenta de ello.
- Es posible aumentar la productividad **gestionando mejor el tiempo.**
  - *Recomendación de uso de la técnica Pomodoro*
  - ***Por ejemplo,*** *declarar una franja de dos horas como Tiempo de Programación, desconectar los e-mails y los móviles y dedicarse sólo a eso.*

XP - Prácticas Primarias II

autentia

PROGRAMACIÓN EN PAREJA (PAIR PROGRAMMING)

- Todo **el código** que va a producción **debe ser escrito por dos personas sentadas frente a la máquina.**
- Pair programming es un **diálogo entre dos personas** que están simultáneamente analizando, diseñando, probando e intentando programar mejor.
- **Ventajas**
  - *Se mantienen centrados mutuamente*
  - *Se clarifican ideas*
  - *Se vencen los bloqueos individuales*
  - *Se cumplen mejor los estándares del equipo*
- **Consejos**
  - *Rotar las parejas*
  - *No juntar dos programadores novatos*
  - *No invadir el espacio personal del otro (monitores grandes)*

HISTORIAS DE USUARIO (STORIES)

- Escribir las **historias de usuario en tarjetas pequeñas**, indicando un nombre, una descripción y una estimación del tiempo.
  - Recomendamos el uso de post it.
- **Es más adecuado hablar de "historias" que de "requisitos"** (palabra con connotaciones de "inmutabilidad" y "permanencia" que no son compatibles con "abrazar el cambio").
- **Al estimar cada historia es fácil** darse cuenta del coste de cada una de ellas y podemos combinarlas, **priorizarlas**.
- **Colocar las tarjetas** en la pared, **en un sitio visible**, no en un programa.
  - Recomendamos el uso de un tablero Kanban.

CICLO SEMANAL (WEEKLY CYCLE)

- **Reunión al comienzo de cada semana**.
  - *Revisión del progreso hasta la fecha, incluyendo si el progreso de la semana previa se corresponde con lo previsto*
  - *Hacer que el cliente escoja historias que sumen una semana de tamaño para la semana actual*
  - *Fraccionar las historias en tareas, los miembros del equipo se apuntan a las tareas y las estiman*
- **Comenzar la semana escribiendo las pruebas automáticas** que se realizarán cuando las historias estén completas y pasar el resto de la semana implementándolas e integrándolas en el proyecto.
- **Al final de la semana las nuevas historias deben estar disponibles para ser desplegadas.**
- **La semana es un ciclo corto para poder hacer experimentos** del estilo "vamos a usar la técnica del Pomodoro durante una semana" o "esta semana cambiamos de pareja cada hora".

CICLO TRIMESTRAL (QUARTERLY CYCLE)

- Es conveniente **hacer reuniones con un ciclo superior al semanal para ver la evolución del proyecto** en conjunto.
- Cada 3 meses:
  - *Identificar los cuellos de botella, especialmente los que no están controlados por el equipo*
  - *Planificar el tema o los temas para el cuatrimestre*
  - *Escoger historias que sumen un trimestre para cumplir con los temas escogidos*
  - *Centrarse en una visión general de cómo el proyecto encaja en la organización y del valor que añade*
- Los trimestres también **son un buen ciclo para reflexionar sobre el equipo, proponer y evaluar experimentos** que duren más de una semana.

XP - Prácticas Primarias III

autentia

TRABAJAR CON HOLGURA (SLACK)

- Trabajar con un cierto colchón que **suavice la tensión de cumplir compromisos imposibles.**
- Es importante **establecer una atmósfera de confianza** en la que los que piden y los que desarrollan el producto se comunican de forma clara y honesta.
- **Trabajar con plazos no realistas introduce errores inmanejables**, mina la moral y construye relaciones antagonistas.
- **Cumplir los compromisos**, incluso modestos, **elimina desperdicios** (waste), suaviza las tensiones, mejora la credibilidad y construye unas relaciones basadas en la comunicación honesta.

CONSTRUCCIÓN DE 10 MINUTOS (10' BUILD)

- **10 minutos para construir automáticamente todo el sistema y ejecutar todos los tests.**
- **Todo el proceso** de build y de paso de las pruebas **debería estar automatizado.**
- No **comprobar** sólo **las pruebas** de la nueva parte añadida al sistema, sino **todas** las anteriores.
- En **continuous delivery** se va un paso más allá y se automatiza también el despliegue y el lanzamiento de las nuevas funcionalidades en producción.

INTEGRACIÓN CONTINUA (CONTINUOUS INTEGRATION)

- **No dejar pasar más de dos horas sin integrar los cambios** que hemos programado.
- La programación en equipo es un problema de **"divide, vencerás e integrarás".**
- La **integración es un paso no predecible** que puede costar más que el propio desarrollo.
- **Integración síncrona:** cada pareja después de un par de horas sube sus cambios y espera a que se complete el build y se hayan pasado todas las pruebas sin ningún problema de regresión.
- **Integración asíncrona:** cada noche se hace un build diario en el que se construye la nueva versión del sistema. Si se producen errores se notifica con alertas de emails.
- **El sistema resultante debe ser un sistema listo para lanzarse sin demasiados problemas.** Si el objetivo es desplegar una web, hay que desplegar la web (en el entorno de staging).

DISEÑO INCREMENTAL (INCREMENTAL DESIGN)

- **Diseño gradual,** con pasos pequeños y seguros.
- **El diseño no debe hacerse sólo al principio**, sino conforme vamos adquiriendo experiencia.
- **Invertir cada día en el diseño del sistema,** luchando por ver cómo la parte que estamos construyendo puede mejorarlo en su conjunto.
- **Pensar todo el tiempo en el problema en conjunto**, buscando formas de diseñar mejor todo el sistema. Cuando tengamos claro un diseño mejor, trabajar gradual pero continuamente para alinear el sistema real con este diseño.
- **Los equipos XP trabajan** duro **para** crear las condiciones bajo las **que el coste de cambiar el software sea bajo.**
- Una regla sencilla para mejorar el diseño: **eliminar la duplicación**.

XP - Prácticas Corolarios I

autentia

PARTICIPACIÓN REAL DE LOS CLIENTES

- **Haz que las personas** cuyas vidas y negocios se vean afectados por tu sistema **formen parte del equipo.**
  - Tendrás resultados diferentes con clientes reales.
- En general, **cuanto más se acerquen las necesidades del cliente** a las funcionalidades del desarrollo, **más valioso se volverá el desarrollo.**
- La inclusión de clientes en el proceso de desarrollo **fomenta la confianza y la mejora continua.**

DESPLIEGUE INCREMENTAL

- Es una práctica corolario que **puede aplicarse tanto al reemplazamiento de sistemas Legacy como migraciones masivas de datos.**
- Consiste en **modificar primero algunas de sus funcionalidades** del producto ya existentes sobre el que se está trabajando **y poco a poco** ir **modificando el resto** de forma incremental.

CONTINUIDAD DEL EQUIPO

- **Un equipo de desarrollo debe mantenerse aunque se cambie de proyecto**. De esta forma se mantiene la efectividad del equipo.
- **Las relaciones y los logros del equipo aportan valor al software.** Ignorar el valor de las relaciones y la confianza para simplificar los problemas de agenda es una forma errónea de economizar.
- **Que los equipos se mantengan unidos no implica que sean completamente estáticos.** En cualquier momento nuevos miembros se pueden incorporar fomentando una rotación razonable dentro del equipo proporcionando a la compañía los beneficios de los equipos estables así como la propagación del conocimiento y la experiencia en ella.

REDUCIR LOS EQUIPOS

- Con el tiempo un equipo será cada vez más productivo. Manteniendo la carga de trabajo, se podrá enviar a alguno de sus miembros a formar nuevos equipos.
- **Evita crear equipos cada vez más grandes** ya que el rendimiento del mismo será tan pobre que valdrá la pena considerar otras alternativas.

XP - Prácticas Corolarios II

autentia

ANÁLISIS DE CAUSA RAÍZ

- Cada vez que se encuentre un defecto se debe **eliminar el defecto y su causa. El objetivo** no es sólo eliminar ese defecto sino **que el equipo no cometa el mismo tipo de error de nuevo.**
- Un ejercicio simple que se puede aplicar para ello es usar la **técnica de los 5 Porqués** de Taiichi Ohno.

NEGOCIAR EL ALCANCE DEL CONTRATO

- **Escriba contratos que fijen el tiempo, los costos y la calidad, pero no el alcance preciso del sistema.** El alcance debe negociarse de forma continua mediante contratos cortos en lugar de uno largo.
- **Los contratos grandes y largos se pueden dividir en la mitad o en tercios.** Y si la solicitud de cambio en un contrato es muy costosa, se puede redactar con menos alcance fijo por adelantado y menores costos para los cambios.
- En contratos para el desarrollo de software resulta más útil y menos arriesgado **negociar varios contratos de duración más corta** y fijar de esta forma el alcance para cada uno de ellos.
- **Negociar el alcance del contrato es un mecanismo para alinear los intereses de proveedores y clientes para alentar la comunicación y la retroalimentación**, y para tener la oportunidad de mostrar el coraje de hacer lo que parece correcto hoy y no hacer algo ineficaz sólo porque está en el contrato.

PROPIEDAD COLECTIVA DEL CÓDIGO

- **Todos los miembros del equipo sienten el código como suyo y se responsabilizan del mismo por igual**. Cualquier miembro del equipo debe ser capaz de modificar cualquier parte del código.

CODIFICACIÓN Y PRUEBAS

- Los elementos clave del proyecto que **no se pueden dejar de mantener y mejorar nunca.** Este mantenimiento es la **Refactorización, en todo momento, en todo lugar.**

CÓDIGO BASE ÚNICO

- Si por algún motivo se necesita trabajar en paralelo al desarrollo principal, esto no debería prolongarse más de varias horas.

DESPLIEGUE DIARIO

- **Cada día debe ponerse el código desarrollado en producción** para evitar un desfase entre la última versión de software y la de cada máquina local. Para poder desarrollar esta práctica Corolario es imprescindible el dominio de las primarias así como el Corolario Despliegue Incremental.

PAGAR POR FUNCIONALIDAD

- Si se paga por funcionalidad, **se sabrá con precisión hacia dónde dirigir el desarrollo.**