



eXtreme Programming



Hola!



Soy José Julián Ariza

Fundador de **Impacto Tecnológico** desde 2006.

Líder estratégico de Equipos de Desarrollo Ágil en modalidad Online y Presencial

Arquitecto de Software por naturaleza y DevOps por hobby

@JJArizaV – josejulian@impactotecnologico.net

@impactotecno



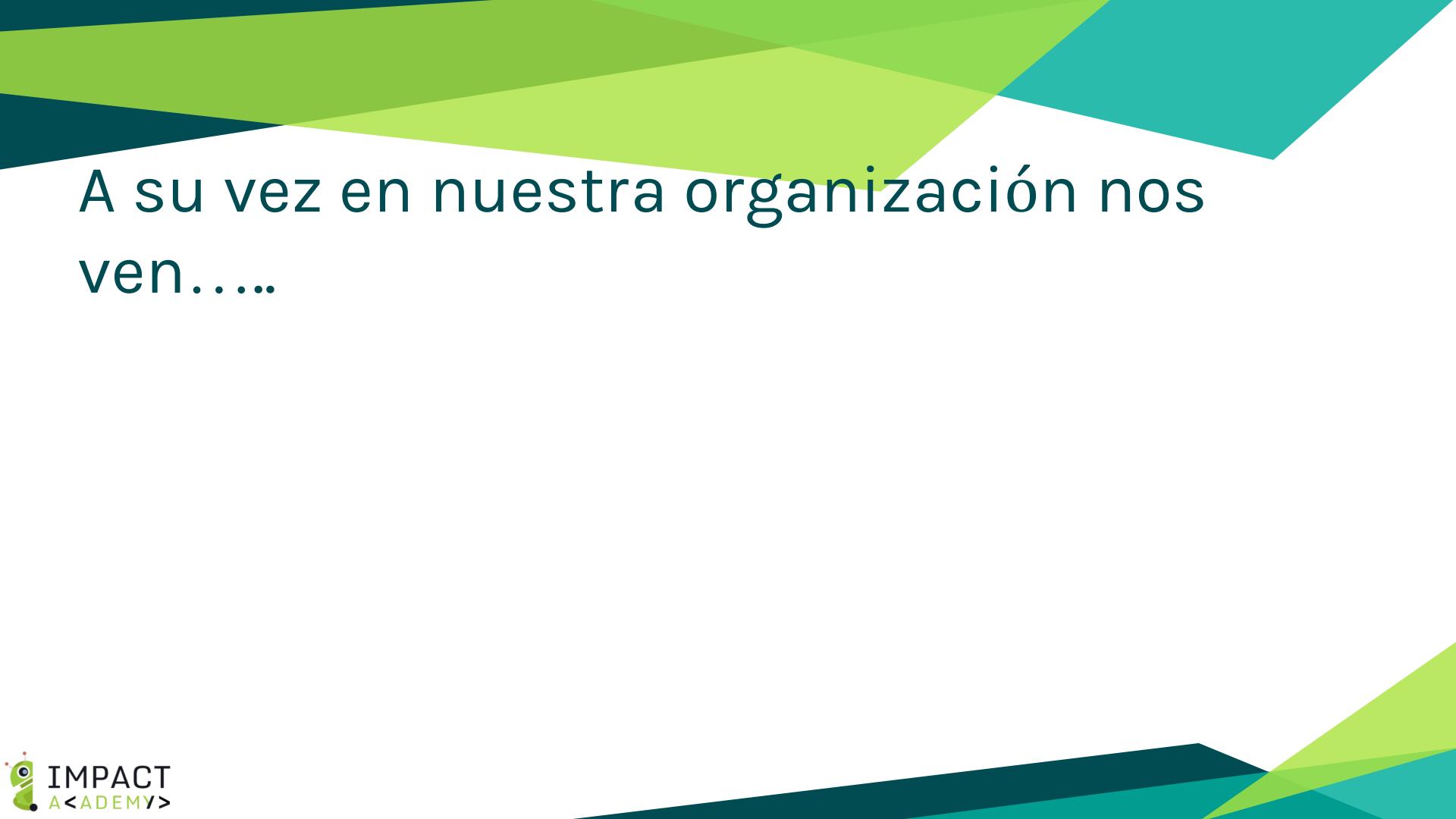
¿Qué necesitaremos?

- ◆ Cuenta en github.com
- ◆ Cuenta en codeanywhere.com
- ◆ Cuenta en codefactor.io asociada al usuario github
- ◆ JDK 1.7 o superior
- ◆ Eclipse Oxygen o superior
- ◆ Algún visualizador de MongoDB

- ◆ Cuando en nuestra organización queremos implantar metodologías ágiles solemos vernos como....







A su vez en nuestra organización nos
ven.....





Agilismo

Agile = Anarquía

No sirve para proyectos grandes ni para los muy pequeños

No hay documentación

No existe planificación

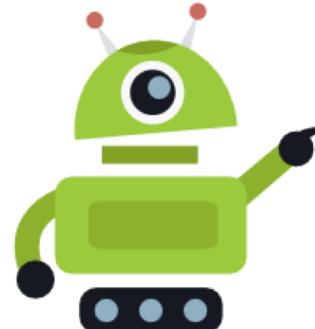
Solo sirve para equipos avanzados

Solo funciona si eres Google o Facebook

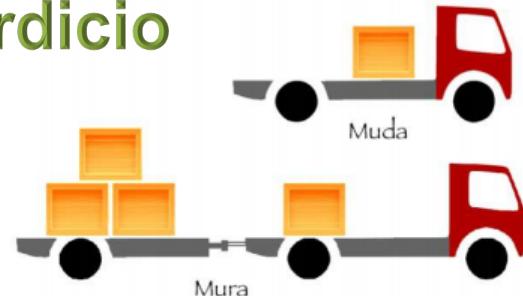
No es formal

Agilidad... orígenes

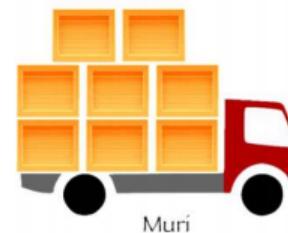
- ◆ Eliminar ineficiencias (Muda, Mura, Muri)
- ◆ Construir con calidad
- ◆ Crear conocimiento
- ◆ Entrega rápida
- ◆ Respetar a las personas
- ◆ Optimizar el conjunto



Desperdicio



Inconsistencia



Sobrecarga

Estadísticas sobre Agilismo en Empresas

- ◆ 20-50% de incremento en la productividad
- ◆ + del 50% de incremento en la calidad
- ◆ 30-75% más rapidez en lanzar productos al mercado



eXtreme Programming

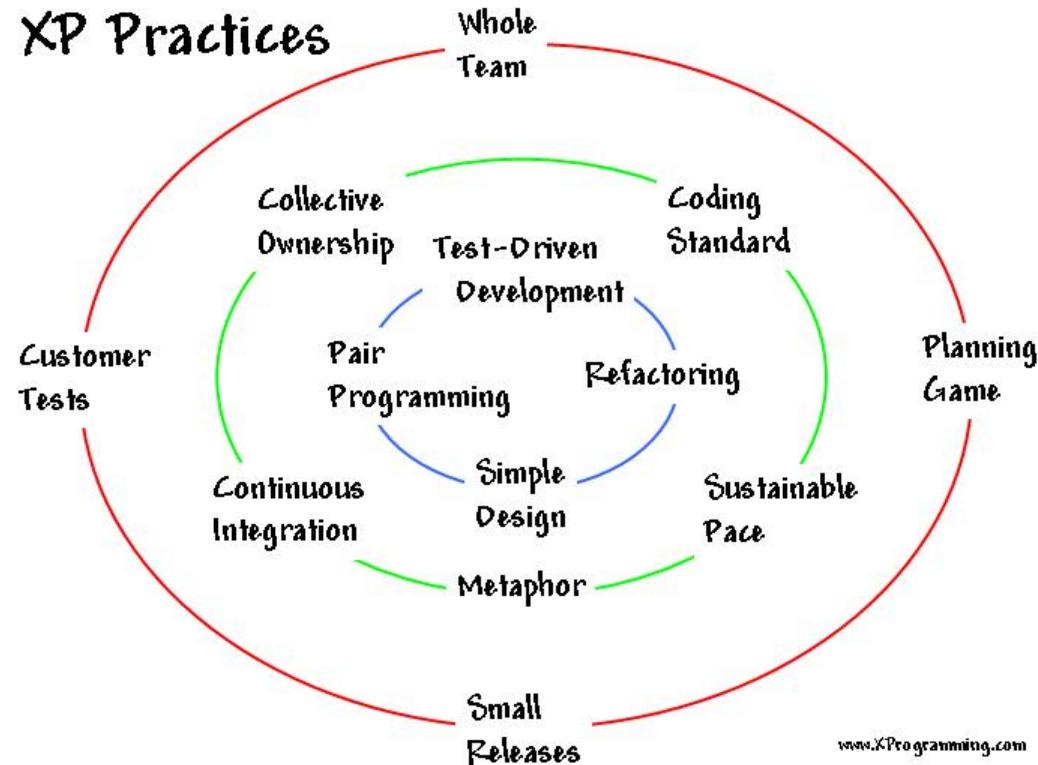




*Es una metodología de desarrollo ágil
que tiene como principal objetivo
aumentar la productividad a la hora de
desarrollar un proyecto software.*

PROGRAMACIÓN EXTREMA (XP)

- ◆ Está basada en un conjunto de reglas y buenas prácticas para el desarrollo de software en ambientes muy cambiantes con requisitos imprecisos, por ende está enfocada en la retroalimentación continua entre el equipo de desarrollo y el cliente.



Características

SIMPLICIDAD

Retroalimentación

Respeto

Coraje

Comunicación



Principio de pruebas:

- ◆ Lo primero que se debe hacer es establecer un período de pruebas de aceptación del programa, en el cual se definirán las entradas y salidas del sistema.
- ◆ Básicamente se define lo que debe hacer el software desarrollado.



Planificación

- ◆ Junto con el cliente se establecen sus necesidades para definir las actividades que el sistema debe realizar.
- ◆ Debe generarse un documento que contendrá **historias de usuario** que forman el **release plan**, el cual define los tiempos de entrega de la aplicación para poder recibir feedback por parte del cliente.

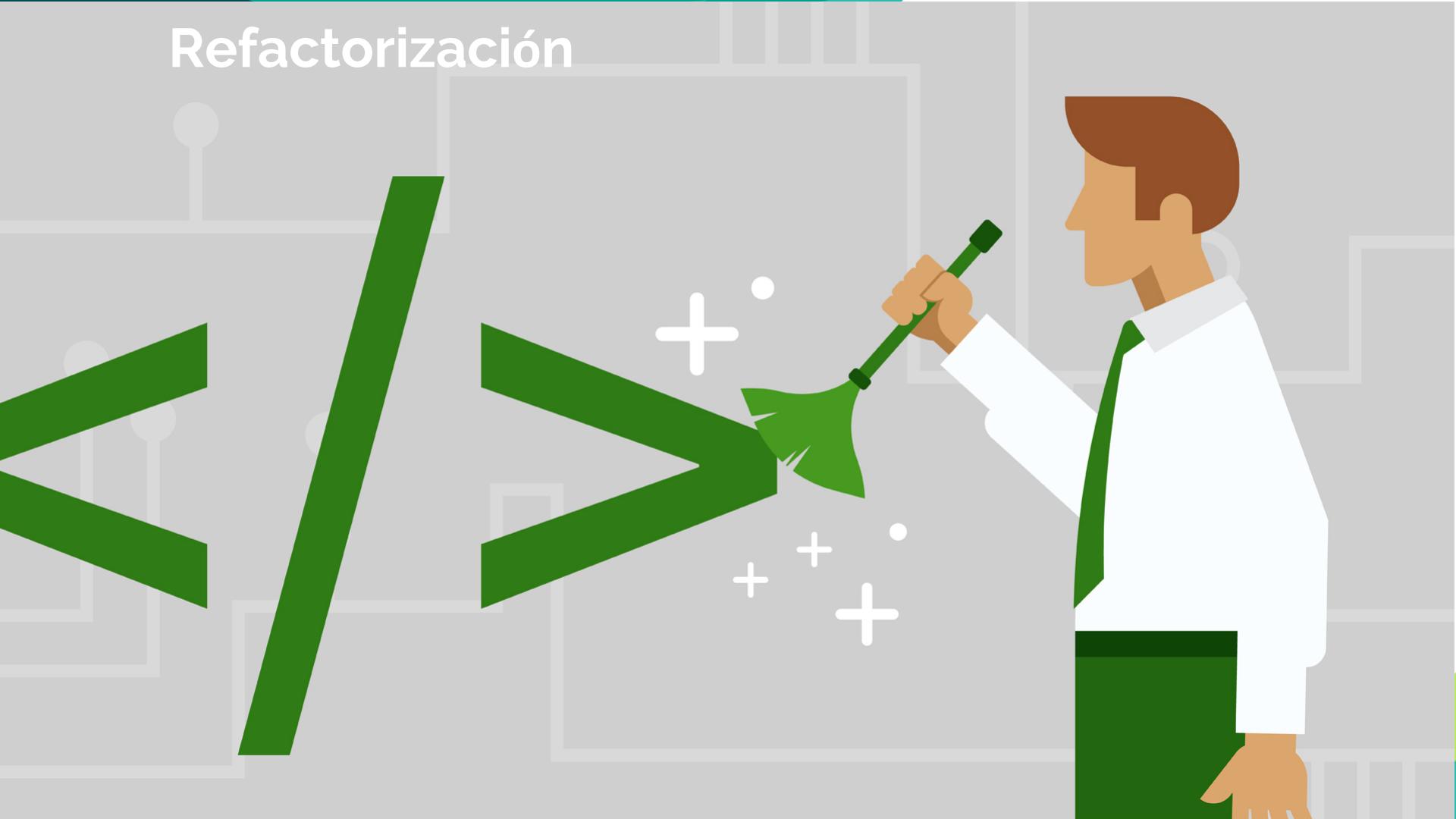


Pair-programming

- ◆ Consiste en **escribir código en parejas compartiendo una sola máquina.**
- ◆ Según los experimentos ya realizados sobre este método, se producen mejores y más consistentes aplicaciones a igual o menor coste

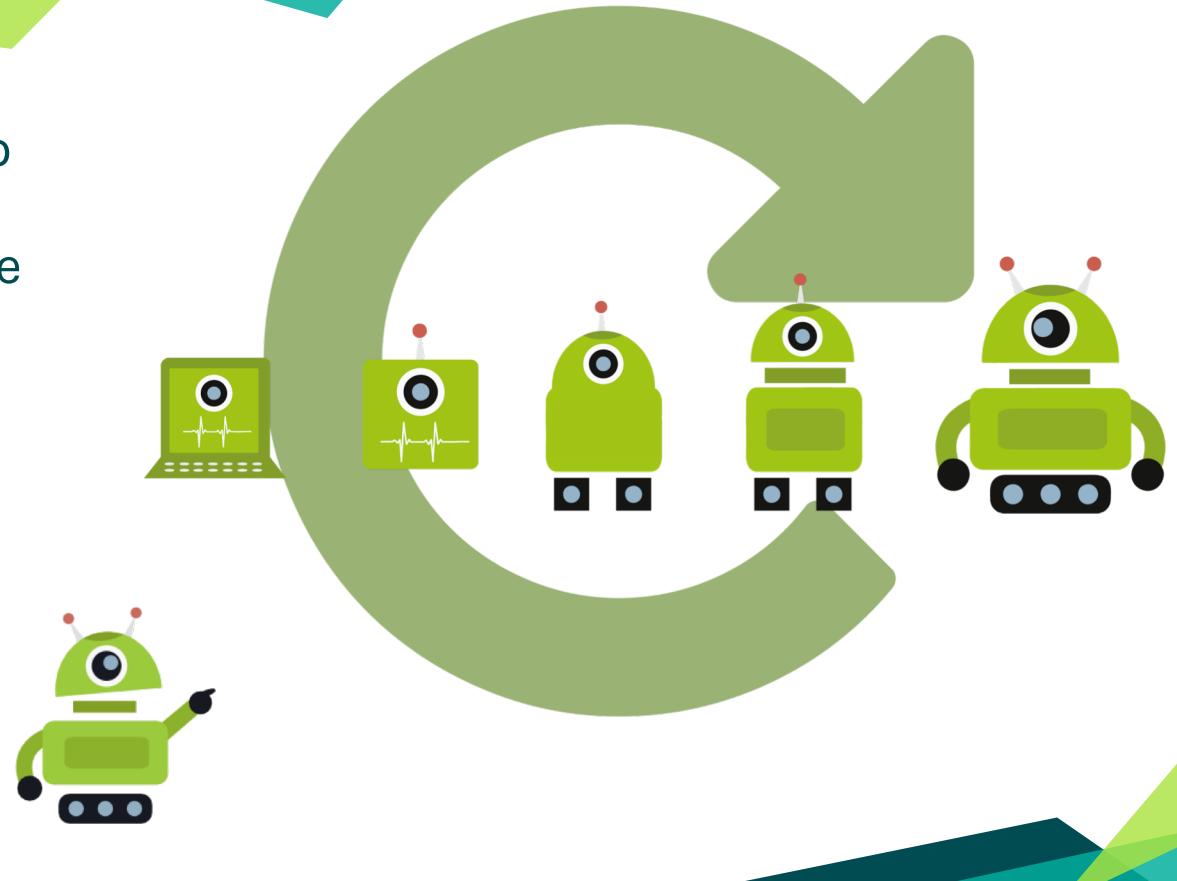


Refactorización



Entregas pequeñas

- ◆ El producto es evaluado en un ambiente real mediante la colocación de un sistema sencillo en producción el cual se actualizará rápidamente, es decir, cada 2 semanas (3 como máximo) el software será puesto en producción

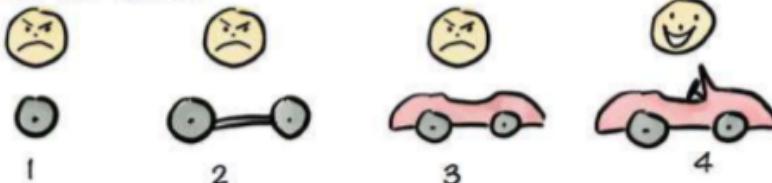


¿Qué es y qué no es Incremental?

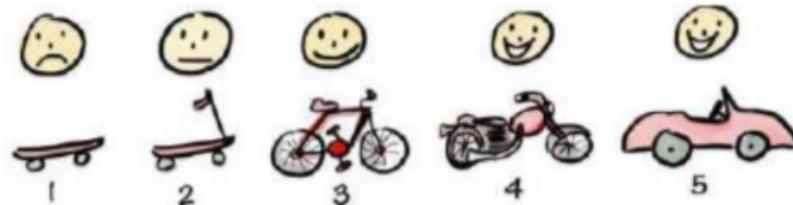
... no es fácil de conseguir



Not like this....



Like this!



Diseño simple

- ◆ El mejor programa será aquel que cumpla con los requisitos y sea más simple. Es importante proporcionar un software que cubra las necesidades de un cliente. Ni más ni menos.



**Clear,
feasible,
testable**



Propiedad colectiva del código

- ◆ El código tiene propiedad compartida. Nadie es propietario de nada, ni siquiera de lo que ha desarrollado. Todos los programadores son "dueños" de todo el código.
- ◆ Según esta metodología, cuantos más programadores haya trabajando en una parte de código, menos errores tendrá.



Estándar de programación



- ◆ Deben existir reglas para escribir y documentar código, además de cómo se comunican las diferentes piezas de código desarrolladas por diferentes equipos. El objetivo de esto es que parezca que el código ha sido escrito por una única persona.



Poniendo a punto nuestro entorno: Eclipse

- ◆ Descargar formatter Google Style:

<https://github.com/impactotecnologico/eclipse-formatter-googlestyle-java>

- ◆ Importar formatter en eclipse

- ◆ Configurar acciones básicas: Save Actions

- ◆ Instalando Checkstyle

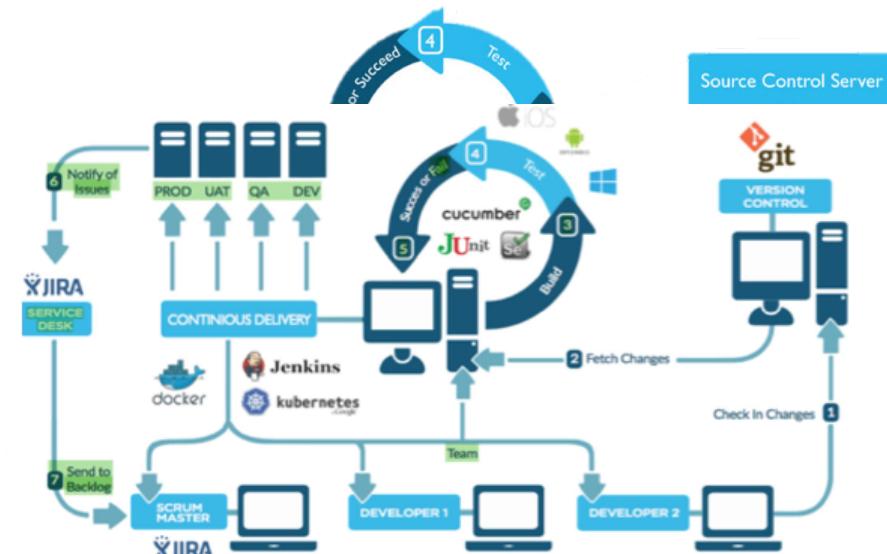
- ◆ Copiando la guía de estilo

- ◆ Configuración básica

- ◆ Proyecto ejemplo: <https://github.com/impactotecnologico/wso2-rest-base>

Integración continua

◆ Consiste en implementar progresivamente las nuevas características del software. En lugar de crear versiones estables en función de una planificación previamente realizada, los programadores reúnen su código y reconstruyen el proyecto varias veces al día si hace falta.



Ventajas del Pair Programming

- ◆ Muchos errores se detectan en el momento en que se escriben, en lugar de en las pruebas de control de calidad o en el campo.
- ◆ La cantidad de defecto final es estadísticamente menor.
- ◆ Los diseños son mejores y el código es más corto.
- ◆ El equipo resuelve los problemas más rápido.
- ◆ La gente aprende mucho más sobre el sistema y sobre el desarrollo de software.
- ◆ El proyecto termina con varias personas entendiendo cada parte del sistema.
- ◆ Las personas aprenden a trabajar juntas y a hablar más a menudo juntas, ofreciendo un mejor flujo de información y una mejor dinámica de equipo.
- ◆ La gente disfruta más de su trabajo.



- ◆ La Universidad de Utah realizó experimentos sobre pair programming. Los resultados revelaron:
- ◆ Los pares pasaron 15% más de tiempo programando que de manera individual
- ◆ El código escrito por los pares pasó consistentemente más casos de prueba que el código escrito desarrolladores individuales.
- ◆ Los pares implementaron consistentemente la misma funcionalidad producida por desarrolladores individuales en menos líneas de código.
- ◆ Aprender a programar en un entorno donde hay resultados tangibles rápidamente es divertido y permite aprender más rápido.



¿Por dónde empezamos?



¿Por dónde empezar?

- ◆ Williams and Robert R. Kessler, en su libro, ‘**All I Really Need to Know about Pair Programming I Learned in Kindergarten**’, explica bien cómo nutrir las habilidades que todos hemos aprendido en Kindergarten para establecer la cohesión del equipo, en general, y el pair programming:
 - ◆ Share everything
 - ◆ Play fair
 - ◆ Do not hit people
 - ◆ Put things back where you found them
 - ◆ Clean up your own mess
 - ◆ Say you are sorry when you hurt somebody

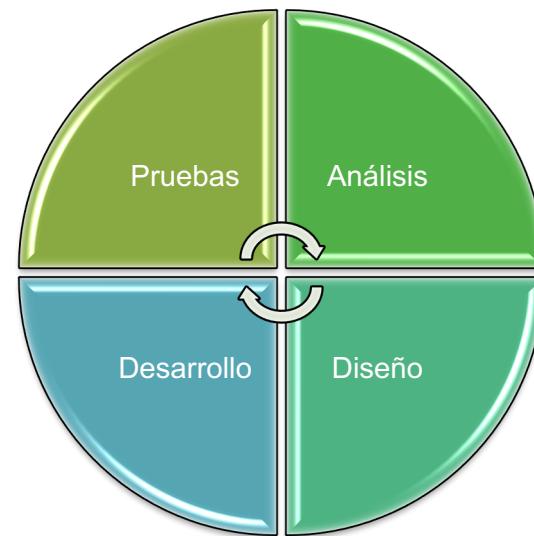


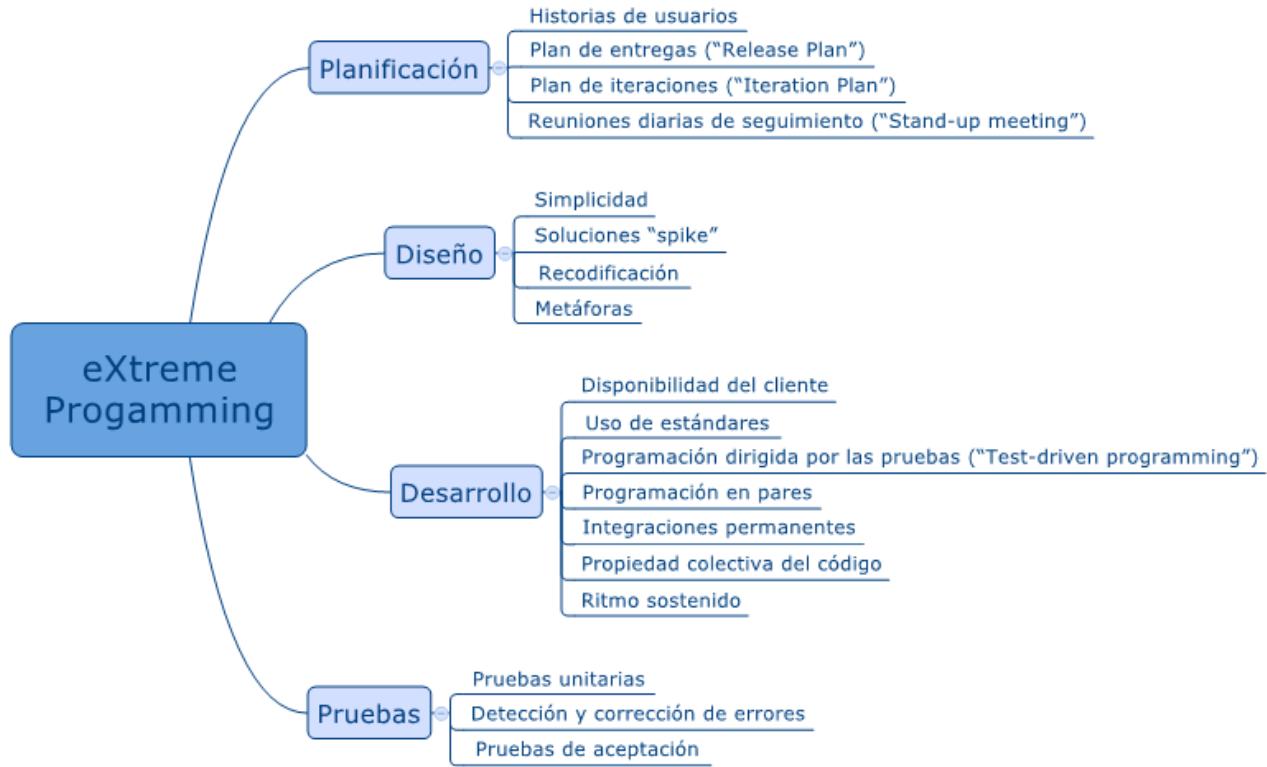
Ciclo de vida XP



Ciclo de vida XP

- Entender lo que el cliente necesita > Fase de Exploración
- Estimar el esfuerzo > Fase de Planificación
- Crear la solución > Fase de Iteraciones
- Entregar el producto final al cliente > Fase de puesta en producción





Roles en XP

Developer

- Estimar historias
- Definir tareas de historias
- Estimar tareas
- Escribir pruebas unitarias
- Escribir código para pasar las pruebas unitarias
- Realizar pruebas unitarias
- Refactor
- Integrar continuamente

Cliente

- Escribir historias de usuario
- Escribir pruebas funcionales
- Establecer prioridades en las historias
- Explicar historias
- Decidir sobre las historias

Roles en XP

Manager

- Definir las reglas de planificación.
- Familiariza al equipo y al cliente con las reglas
- Controla la planificación, corrige cualquier desviación, modifica las reglas
- Programa y realiza reuniones
- Participa en la estimación
- Asegura que el equipo esté trabajando correctamente
- Rastrea defectos de prueba funcionales.

Coach

- Comprende en profundidad la aplicación de Extreme Programming al proyecto.
- Identifica las prácticas de programación extrema que ayudan en caso de cualquier problema.
- Mantiene la calma incluso cuando todos los demás están en pánico.
- Observa al equipo en silencio e interviene solo cuando prevea un problema importante y hace que el equipo también vea el problema.
- Asegura que el equipo sea autosuficiente.
- Está listo para ayudar.

Actividades en XP

Coding

- En el pair programming, la codificación se considera el corazón del desarrollo.
- Si no codificamos, al final del día no tendremos hecho nada.

Testing

- En el pair programming, es necesario realizar pruebas para garantizar que la codificación esté completa.
- Si no se prueba, no sabe cuándo se ha terminado de codificar.

Listening

- En el pair programming, escuchas para saber qué codificar o qué probar.
- Si no escuchas, no sabes qué codificar o qué probar.

Designing

- En el pair programming diseñas para que puedas seguir codificando, probando y escuchando indefinidamente
- Un buen diseño permite la extensión del sistema, con cambios en un solo lugar.

Scaling Extreme Programming

- ◆ Inicialmente, se consideró que la Programación Extrema era efectiva en equipos pequeños, con un tamaño de equipo de hasta 12-16 desarrolladores.
- ◆ Más tarde, se observó que es posible escalar la programación extrema hasta equipos de 40-50 desarrolladores. Sin embargo, se recomienda hacer la escala mediante la construcción de equipos recursivos.
- ◆ Primero se construye un equipo pequeño, se hace crecer gradualmente, y se usa el primer equipo como base para crear nuevos equipos.

Documentación en XP

- ◆ eXtreme Programming no está en contra de la documentación.
- ◆ Se recomienda documentar lo que se requiere, cuando se requiere y solo para el detalle requerido y suficiente.
- ◆ Esto puede diferir de un proyecto a otro y corresponde al equipo decidir sobre el alcance de la documentación.
- ◆ Sin embargo, omitir la documentación no está permitido por las prácticas de Extreme Programming.



Conceptos Erróneos sobre XP

Conceptos erróneos	Aclaratorias
No se escribe documentación	La documentación mínima y suficiente debe estar allí. Sin embargo, no existen estándares formales para la cantidad o el tipo de documentos que se necesitan.
No se diseña	El diseño mínimo explícito y directo debe estar presente y evolucionar a medida que avanza el desarrollo.
Extreme Programming es solo pair programming y es fácil	Pair Programming es solo una de las prácticas de Programación Extrema. Requiere una gran disciplina y consistencia que se logra en conjunto con las otras prácticas de Programación Extrema.
Extreme Programming es la única y verdadera forma de construir software	Extreme Programming es solo efectiva en algunos proyectos, no en todos.

4 Manos a la Obra!!



¿Qué necesitamos?

- ◆ **Cuentas**
 - ◆ Cuenta propia en github
 - ◆ Cuenta propia en codeanywhere
- ◆ **Software**
 - ◆ Google Chrome
 - ◆ Algún visualizador de Mongo o consola bash
- ◆ **Pasos**
 - ◆ Compartir correo de github y de codenanywhere
 - ◆ Revisar repo: <https://github.com/impactotecnologico/spring-rest-base-1>

¿Qué tenemos?

- ◆ Accedemos a contenedor compartido vía URL
 - ◆ Proyecto Base Spring Boot
 - ◆ **mvn spring-boot:run**
- ◆ Accedemos a la BD remota vía consola bash o visualizador:
 - ◆ host: ds227481.mlab.com
 - ◆ DB: meetup-impacto-tecnológico
 - ◆ user: impacto
 - ◆ password: impacto.1
 - ◆ url:
`mongodb://impacto:impacto.1@ds227481.mlab.com:27481/meetup-impacto-tecnologico`

Requerimientos

- ◆ Crear un CRUD simple para las entidades
 - ◆ Producto
 - ◆ Usuario: puede ser de tipo cliente o admin
 - ◆ Pedido: tiene productos y usuarios cliente
- ◆ Empaqueado:
 - ◆ Debe empaquetarse en un war usando Maven
- ◆ Realizamos codereview
- ◆ Clonamos en local la versión final y aplicamos formateo y checkstyle.

Organizando el trabajo

- ◆ Organización por grupos
 - ◆ Asignación de developer inicial y observador
 - ◆ Asignación de code reviewer
- ◆ Segmentación de tareas por entidad
 - ◆ Uso de pair programming para el desarrollo
- ◆ Haremos code review para detectar puntos a mejorar



Gracias!

Alguna pregunta?

Pueden contactarnos mediante
info@impactotecnologico.net y apuntarse a
algunos de nuestros cursos



