XP Programming

ADS 2025

XP (eXtreme Programming)

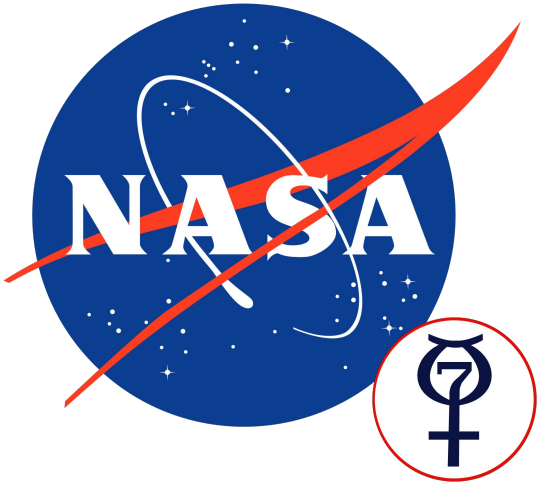
Es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999).

La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. (Metodología ágil).

Historia y origen

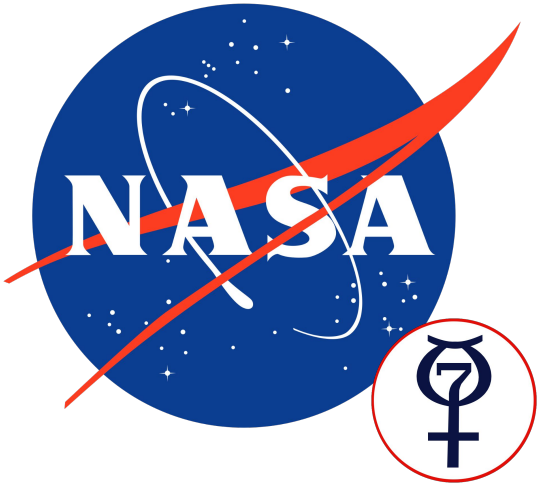
Kent Beck desarrolló una “programación extrema” durante su trabajo en el proyecto de nóminas para el Sistema de Compensación Integral de Chrysler. A medida que su trabajo iba avanzando, comenzó a refinar la metodología de desarrollo utilizada en el proyecto y escribió un libro sobre la metodología (1999).

Muchas de las prácticas de la programación extrema existen desde hace algún tiempo; la metodología lleva las "mejores prácticas" a niveles “extremos”. Por ejemplo, la "práctica del desarrollo de la prueba primero, la planificación y la escritura de pruebas antes de cada microincremento" se utilizó ya en el Proyecto Mercury de la NASA (primer programa espacial de EEUU), a principios de la década de 1960...



Historia y origen

... Para acortar el tiempo total de desarrollo, se desarrollaban algunos documentos de prueba formales (como para las pruebas de aceptación) en paralelo con (o poco antes) que el software estuviera listo para la prueba. Un grupo de prueba independiente de la NASA podía escribir los procedimientos de prueba, basados en requisitos formales y límites lógicos, antes de que los programadores escriban el software y lo integren con el hardware. XP lleva este concepto al nivel extremo, escribiendo pruebas automatizadas (a veces dentro de módulos de software) que validan el funcionamiento de incluso pequeñas secciones de codificación de software, en lugar de solo probar las funciones más grandes.



Influencias

Dos influencias importantes dieron forma al desarrollo de software en la década de 1990:

Internamente, la programación orientada a objetos reemplazó a la programación procedimental (secuencial) como el paradigma de programación preferido por algunos desarrolladores.

Externamente, el auge de Internet y el auge de las puntocom enfatizaron la velocidad de comercialización y el crecimiento de la empresa como factores comerciales competitivos.

Los requisitos rápidamente cambiantes exigían ciclos de vida más cortos del producto y, a menudo, chocaban con los métodos tradicionales de desarrollo de software.

Concepción de los métodos

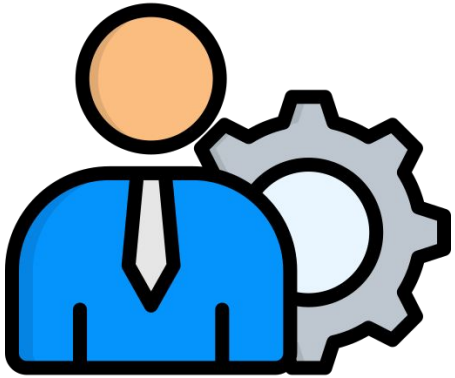
Chrysler contrata a Kent Beck para su Sistema de Compensación Integral. Dentro del proceso de desarrollo, Beck nota varios problemas, y aprovechó esta oportunidad para proponer e implementar algunos cambios en las prácticas de desarrollo. Beck describe la concepción temprana de los métodos así:

“La primera vez que me pidieron que liderara un equipo, les pedí que hicieran algunas de las cosas que pensaba que eran sensatas, como pruebas y revisiones. La segunda vez había mucho más en juego. Pensé: "Al diablo con los torpedos, al menos esto será un buen artículo", y le pedí al equipo que subiera todas las perillas a 10 en las cosas que pensaba que eran esenciales y omitiera todo lo demás.”

Valores, Características, y roles.

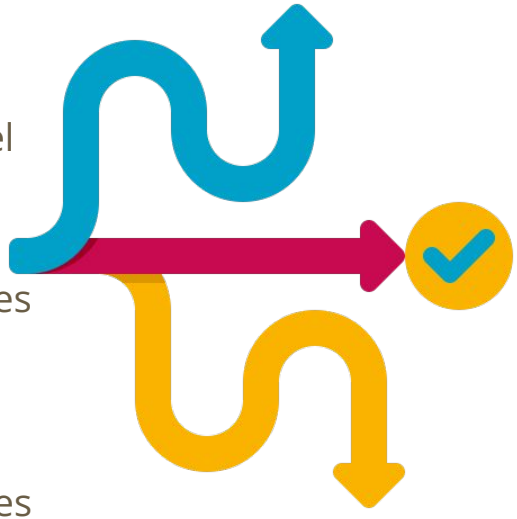
Al crear la serie de libros sobre XP, Beck remarca tres elementos, con subelementos por cada uno, que se van a ir interrelacionando para crear el mejor desarrollo XP posible, y alcanzar los objetivos. Estos son:

- **Valores**:
  - Simplicidad
  - Comunicación
  - Retroalimentación (feedback)
  - Coraje o valentía
  - Respeto
- Características:
  - ... ya las vamos a ir viendo.
- Roles:
  - ... también los vamos a ver ahora.



Valores: Simplicidad

XP propone el principio de hacer la cosa más simple que pueda funcionar, en relación al proceso y la codificación. Es mejor hacer hoy algo simple, que hacerlo complicado y probablemente nunca usarlo mañana.



La simplicidad es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores hacen que la complejidad aumente exponencialmente.

Para mantener la simplicidad es necesaria la refactorización del código, esta es la manera de mantener el código simple a medida que crece.

También se aplica la simplicidad en la documentación, de esta manera el código debe comentarse en su justa medida, intentando eso sí que el código esté autodocumentado. Para ello se deben elegir adecuadamente los nombres de las variables, métodos y clases. Los nombres largos no decrementan la eficiencia del código ni el tiempo de desarrollo gracias a las herramientas de autocompletado y refactorización que existen actualmente.

Aplicando la simplicidad junto con la autoría colectiva del código y la programación por parejas se asegura que cuanto más grande se haga el proyecto, todo el equipo conocerá más y mejor el sistema completo.

Valores: Comunicación

Algunos problemas en los proyectos tienen origen en que alguien no dijo algo importante en algún momento. XP hace casi imposible la falta de comunicación.

La comunicación se realiza de diferentes formas. Para los programadores el código comunica mejor cuanto más simple sea. Si el código es complejo hay que esforzarse para hacerlo legible. El código autodocumentado es más fiable que los comentarios ya que estos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse solo aquello que no va a variar, por ejemplo el objetivo de una clase o la funcionalidad de un método.

Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de como utilizar su funcionalidad. Los programadores se comunican constantemente gracias a la programación por parejas. La comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo. El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

Valores: Retroalimentación o feedback

Retroalimentación concreta y frecuente del cliente, del equipo y de los usuarios finales da una mayor oportunidad de dirigir el esfuerzo eficientemente.

Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real.

Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante.

Considérense los problemas que derivan de tener ciclos muy largos. Meses de trabajo pueden tirarse por la borda debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo. El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el estado de salud del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

Valores: Coraje o valentía

El coraje (valor) existe en el contexto de los otros 3 valores. (si funciona...mejoralo).

Muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana. Esto es un esfuerzo para evitar empantanarse en el diseño y requerir demasiado tiempo y trabajo para implementar el resto del proyecto. La valentía le permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario. Esto significa revisar el sistema existente y modificarlo si con ello los cambios futuros se implementarán más fácilmente. Otro ejemplo de valentía es saber cuándo desechar un código: valentía para quitar código fuente obsoleto, sin importar cuanto esfuerzo y tiempo se invirtió en crear ese código. Además, valentía significa persistencia: un programador puede permanecer sin avanzar en un problema complejo por un día entero, y luego lo resolverá rápidamente al día siguiente, solo si es persistente.

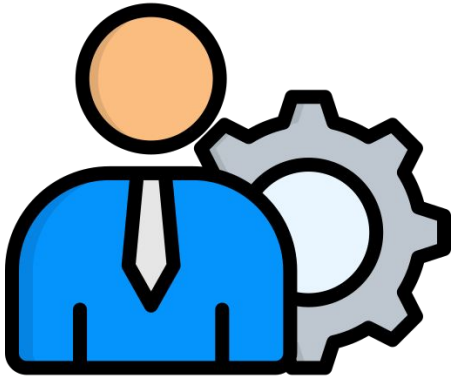
Valores: Respeto

El respeto se manifiesta de varias formas. Los miembros del equipo se respetan los unos a los otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros. Los miembros respetan su trabajo porque siempre están luchando por la alta calidad en el producto y buscando el diseño óptimo o más eficiente para la solución a través de la refactorización del código. Los miembros del equipo respetan el trabajo del resto no haciendo menos a otros, una mejor autoestima en el equipo eleva su ritmo de producción.

Características

Al crear la serie de libros sobre XP, Beck remarca tres elementos, con subelementos por cada uno, que se van a ir interrelacionando para crear el mejor desarrollo XP posible, y alcanzar los objetivos. Estos son:

- Valores:
  - ... ya los vimos 👍
- **Características**:
  - Equipo completo
  - Pruebas unitarias continuas
  - Programación en parejas
  - Refactorización del código
  - Propiedad del código compartida
  - Versiones pequeñas
  - Integración continua
  - Metáforas
  - Ritmo sostenible
  - Simplicidad en el código (como la del valor)
- Roles:
  - ... ya los vamos a ver ahora.



Características: Equipo completo

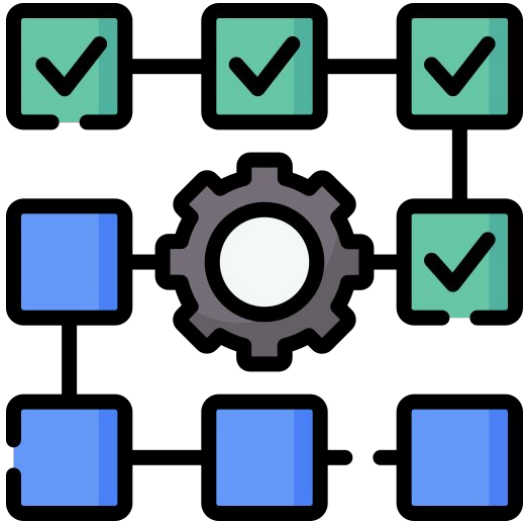
Forman parte del equipo todas las personas que tienen algo que ver con el proyecto, incluido el cliente y el responsable del proyecto.

Todos los miembros del equipo de desarrollo tienen habilidades y conocimientos suficientes para realizar cualquier tarea necesaria para el proyecto. Esto significa que los desarrolladores no están limitados a roles específicos y están dispuestos a asumir diferentes responsabilidades según sea necesario.

El concepto de equipo completo se basa en la premisa de que la colaboración y la comunicación efectiva son fundamentales para el éxito del proyecto. Al tener a todos los miembros del equipo capacitados y comprometidos para asumir diferentes tareas, se fomenta un ambiente de trabajo colaborativo y se promueve la flexibilidad en la asignación de trabajo.

Características: Pruebas unitarias continuas

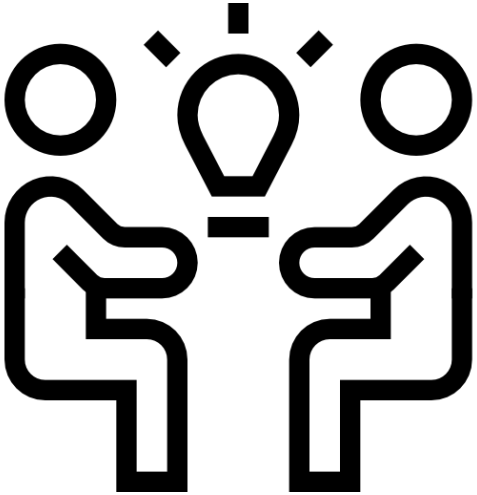
Una característica primordial es la de las pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación. Véase, por ejemplo, las herramientas de prueba JUnit orientada a Java, DUnit orientada a Delphi, NUnit para la plataforma.NET o PHPUnit para PHP. Estas tres últimas inspiradas en JUnit, la cual, a su vez, se inspiró en SUnit, el primer framework orientado a realizar tests, realizado para el lenguaje de programación Smalltalk.



Características: Programación en parejas

Se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. La mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.

A su vez, en la programación en parejas, uno de los programadores actúa como el "conductor" mientras que el otro es el "navegante". El conductor es responsable de escribir el código, mientras que el navegante revisa y ofrece sugerencias. Estos roles se pueden intercambiar periódicamente para mantener una colaboración equilibrada y asegurarse de que ambos programadores estén involucrados activamente en el proceso.



Características: Refactorización del código

Reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.

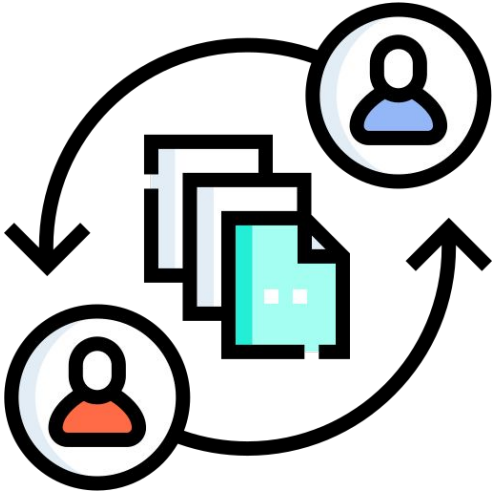
La refactorización se enfoca en abordar los llamados "**malos olores**" en el código, que son señales de diseño o implementación deficiente. Estos pueden incluir duplicación de código, código largo y complicado, métodos o clases demasiado acoplados, entre otros. Identificar y eliminar estos malos olores mejora la mantenibilidad, la extensibilidad y la comprensión del código.

La refactorización puede implicar una variedad de técnicas, como la extracción de métodos o clases para eliminar duplicación de código, la simplificación de estructuras de control complejas, la mejora de nombres de variables y métodos para una mayor legibilidad, y la reorganización de la estructura del código para una mejor modularidad. XP enfatiza la importancia de utilizar técnicas de refactorización seguras y respaldadas por pruebas para garantizar que los cambios no introduzcan errores en el código.

Características: Propiedad del código compartida

Cualquiera puede y debe tocar y conocer cualquier parte del código.

En vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.

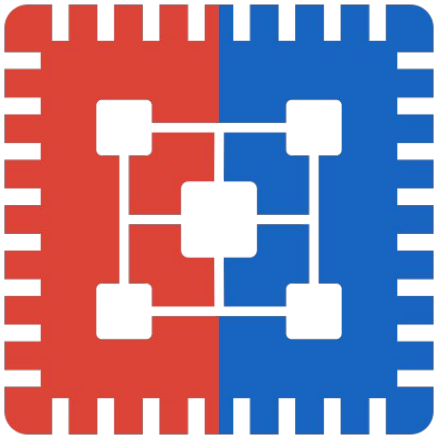


A su vez, la propiedad del código compartida fomenta una colaboración activa entre los miembros del equipo. Se alienta a todos a revisar, analizar y brindar retroalimentación sobre el código desarrollado por otros. Esto promueve una mayor comunicación y comprensión del código en todo el equipo, lo que a su vez mejora la calidad general del software.

Al compartir la propiedad del código, se fomenta la transferencia de conocimiento entre los miembros del equipo. Cada miembro tiene la oportunidad de comprender diferentes partes del sistema y aprender de las habilidades y experiencias de otros. Esto no solo mejora la capacidad del equipo para mantener y mejorar el código, sino que también crea un ambiente de aprendizaje continuo.

Características: Versiones pequeñas o microincrementos

Las mini-versiones deben ser lo suficientemente pequeñas como para poder hacer una cada pocas semanas. Deben ser versiones que ofrezcan algo útil al usuario final y no trozos de código que no pueda ver funcionando.



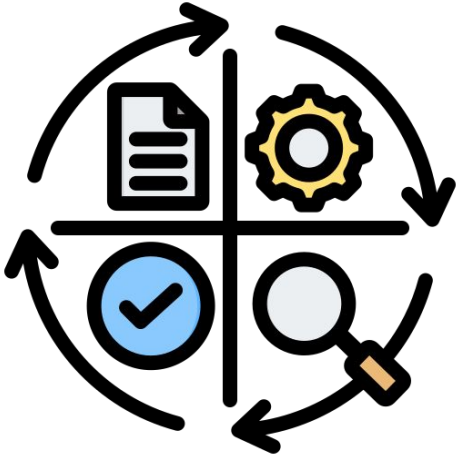
Para lograr las versiones pequeñas, los equipos de XP trabajan en ciclos de desarrollo cortos llamados iteraciones o sprints. Estas iteraciones típicamente duran de una a tres semanas y se enfocan en entregar un conjunto de características o mejoras definidas y probadas.

La práctica de versiones pequeñas prioriza la entrega de funcionalidad clave y de alto valor para el cliente. Se busca identificar las características más importantes y desarrollarlas en incrementos pequeños y manejables. Esto permite una mayor adaptabilidad a medida que los requisitos pueden cambiar o surgir nuevas prioridades.

Al entregar versiones pequeñas, el equipo de desarrollo puede obtener retroalimentación temprana del cliente o usuario final. Esto facilita la identificación de posibles problemas, errores o cambios necesarios. El equipo puede ajustar rápidamente su enfoque y hacer mejoras en función de la retroalimentación recibida, lo que resulta en un producto final más alineado con las necesidades y expectativas del cliente.

Características: Integración continua

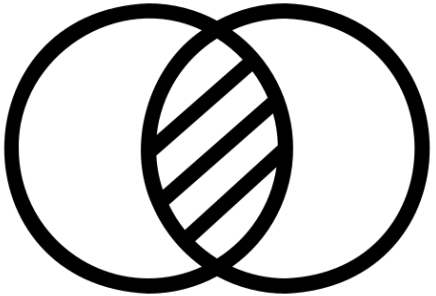
Debe tenerse siempre un ejecutable del proyecto que funcione y en cuanto se tenga una nueva pequeña funcionalidad, debe recompilarse y probarse. Es un error mantener una versión congelada dos meses mientras se hacen mejoras y luego integrarlas todas de golpe. Cuando falle algo, no se sabe qué es lo que falla de todo lo que hemos metido.



Características: Metáforas

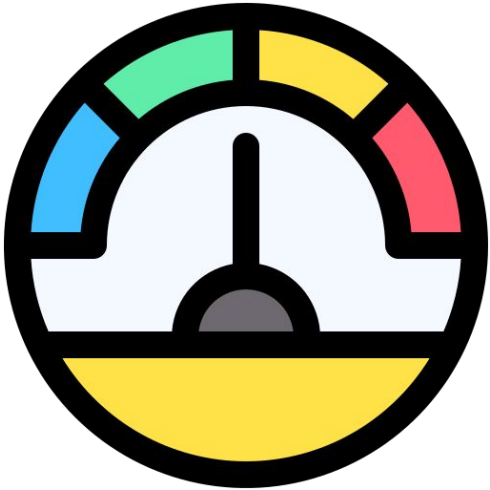
Hay que buscar unas frases o nombres que definan cómo funcionan las distintas partes del programa, de forma que sólo con los nombres se pueda uno hacer una idea de qué es lo que hace cada parte del programa. Un ejemplo claro es el "recolector de basura" de java. Ayuda a que todos los programadores (y el cliente) sepan de qué estamos hablando y que no haya malentendidos.

La metáfora puede ayudar a simplificar el diseño y la arquitectura del sistema. Al relacionar el sistema con algo familiar y comprensible, se pueden tomar decisiones de diseño más intuitivas y coherentes. Además, la metáfora puede proporcionar una estructura conceptual que oriente el desarrollo y facilite la comprensión del sistema en su conjunto.



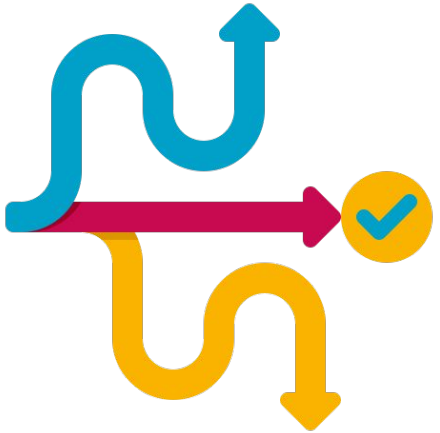
Características: Ritmo sostenible

Se debe trabajar a un ritmo que se pueda mantener indefinidamente. Esto quiere decir que no debe haber días muertos en que no se sabe qué hacer y que no se deben hacer un exceso de horas otros días. Al tener claro semana a semana lo que debe hacerse, hay que trabajar duro en ello para conseguir el objetivo cercano de terminar una historia de usuario o mini-versiones.



Características: Simplicidad en el código

Es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.



La simplicidad en el código promueve la legibilidad y la comprensión. Se busca escribir un código claro y conciso que pueda ser fácilmente entendido por otros miembros del equipo. Se prefieren nombres de variables y métodos significativos y se evita la complejidad innecesaria en la estructura y el flujo del código. Esto facilita la colaboración y el mantenimiento a largo plazo.

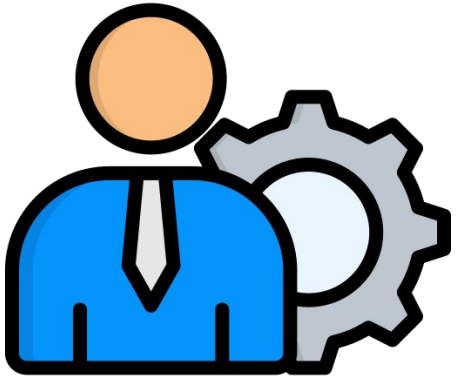
Se evita la sobrecarga y la complejidad innecesaria. Se enfoca en encontrar la solución más simple y directa para un problema en lugar de implementaciones complicadas o excesivamente abstractas. Esto facilita la comprensión, el razonamiento y la corrección de errores en el código.

(y todo lo de la diapositiva 8)

Roles

Al crear la serie de libros sobre XP, Beck remarca tres elementos, con subelementos por cada uno, que se van a ir interrelacionando para crear el mejor desarrollo XP posible, y alcanzar los objetivos. Estos son:

- Valores:
  - ... ya los vimos 👍
- Características:
  - ...tambien las vimos 👍
- **Roles (pag 105)**:
  - Programador
  - Test developer
  - Cliente
  - Tester
  - Tracker
  - Entrenador (coach)
  - Consultor
  - Gestor (Big Boss)



Roles

Programador: Produce el código del sistema. Es la esencia del equipo.

Test developer: Produce el código de los test unitarios del sistema. Es uno de los roles más importantes.

Cliente: Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar el mayor valor de negocio.

Tester: Interpreta el pedido del cliente y ayuda al equipo de desarrollo a escribir las pruebas funcionales. Ejecuta pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

Roles

Tracker: Es el encargado de seguimiento. Proporciona retroalimentación al equipo. Debe verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones.

Entrenador (coach): Responsable del proceso global. Guía a los miembros del equipo para seguir el proceso correctamente.

Consultor: Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Ayuda al equipo a resolver un problema específico. Además este tiene que investigar según los requerimientos.

Gestor (Big Boss): Es el dueño de la tienda y el vínculo entre clientes y programadores. Su labor esencial es la coordinación.

Características

En conclusión, Xp es una metodología ágil centrada en p**otenciar las relaciones interpersonales** como clave para el éxito en desarrollo de software, promoviendo **el trabajo en equipo**, preocupándose por el **aprendizaje de los desarrolladores**, y propiciando un **buen clima de trabajo**. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, **comunicación fluida** entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Ventajas y desventajas

Ventajas:

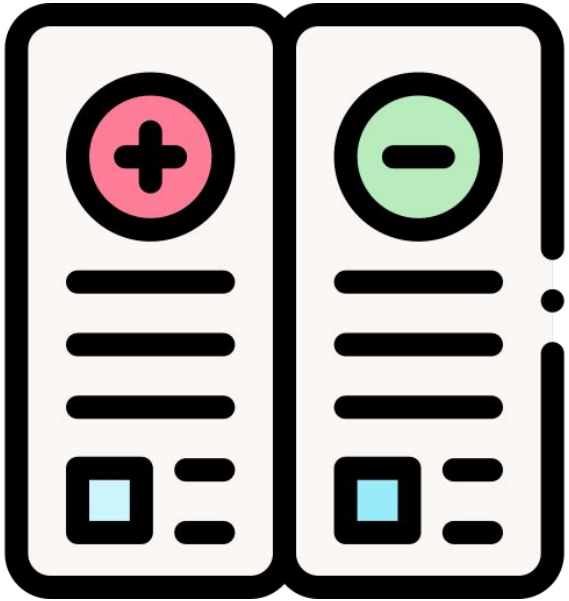
- Programación organizada.
- Menor tasa de errores.
- Satisfacción del programador.

Desventajas:

- Es recomendable emplearlo solo en proyectos a corto plazo.

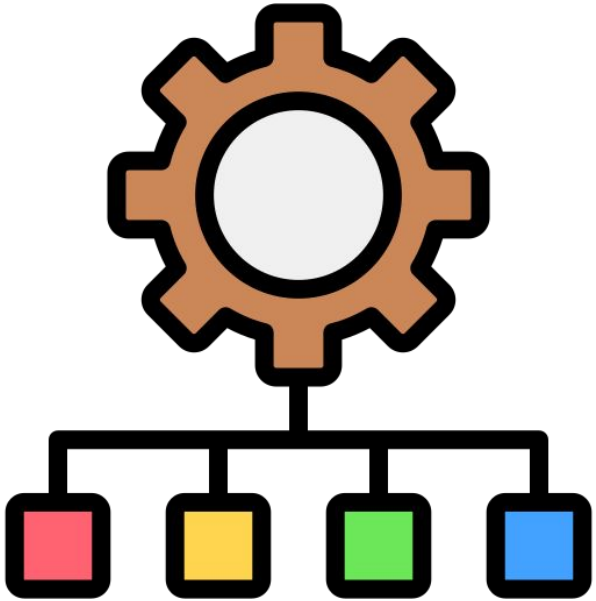
Adecuado para proyectos:

- De corto plazo.



Programación organizada

La programación en parejas, escritura de pruebas antes de implementar código, iteraciones cortas y retroalimentación continua permiten a los equipos abordar rápidamente cualquier problema o error que surja durante el desarrollo. A su vez, darle prioridad a la calidad y la corrección del código, hacen que XP ayude a minimizar los errores y aumentar la estabilidad y confiabilidad del producto final.



Satisfacción del programador

XP promueve un ambiente de trabajo colaborativo y de confianza, donde los programadores tienen una mayor participación en el proceso de toma de decisiones y en la planificación de las tareas. Esto ayuda a fomentar un sentido de propiedad y empoderamiento en el equipo de desarrollo, lo que a su vez aumenta la satisfacción laboral. Además, la programación en parejas y la revisión continua del código brindan oportunidades de aprendizaje y crecimiento profesional, ya que los programadores pueden compartir conocimientos y habilidades entre ellos. La colaboración estrecha también puede reducir el estrés y la presión individual, ya que el equipo se apoya mutuamente en el desarrollo del software. En general, XP busca crear un entorno de trabajo satisfactorio y enriquecedor para los programadores, lo que puede conducir a una mayor motivación, productividad y retención del talento.

Solamente adecuado para proyectos de corto plazo

Escalabilidad limitada: XP se centra en la colaboración y la comunicación constante entre los miembros del equipo. Si el equipo de desarrollo es grande, puede ser difícil mantener este nivel de interacción y coordinación eficientemente. A medida que el número de personas involucradas aumenta, la complejidad de la comunicación y la coordinación también crece, lo que dificulta la implementación efectiva de las prácticas de XP. Por lo tanto, a largo plazo o en proyectos de mayor envergadura, puede resultar más complicado aplicar las técnicas y principios de XP de manera efectiva.

Solamente adecuado para proyectos de corto plazo

Necesidad de compromiso y experiencia: XP requiere un alto nivel de compromiso por parte de todos los miembros del equipo, así como una sólida experiencia en desarrollo ágil y programación en parejas. No todos los equipos o desarrolladores están preparados para adoptar XP y aplicar sus prácticas de manera efectiva. Requiere un cambio de mentalidad y una comprensión profunda de los principios y técnicas de XP. Si un equipo no está adecuadamente capacitado o no tiene la experiencia necesaria en XP, puede resultar difícil implementar correctamente esta metodología y obtener los beneficios esperados.