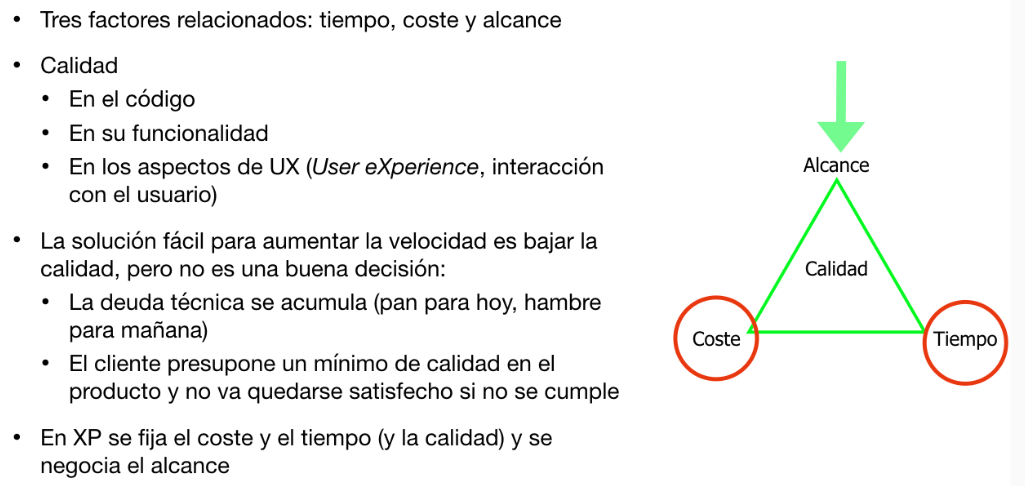
**Extreming Programming (XP)**

**Definiciones:**

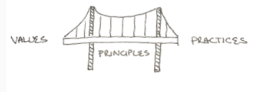
* *“Working Smart is more important than working hard”*
* Promesas:
  + Reducir el riesgo del proyecto.
  + Mejorar la respuesta antes cambios en el negocio.
  + Mejorar la productividad a lo largo de toda la vida del software.
  + Hacer más divertido el trabajo del equipo construyendo el software.
* Triangulo de hierro:





* La enseñanza mas importante de XP es el énfasis en las pruebas y en concreto en la práctica de **TDD**.

**Los 5 valores de XP:**

* Comunicación.
* Simplicidad.
* Retroalimentación.
* Coraje/Valentía.
* Respeto.

**Los 14 principios de XP:**

1. **Humanidad**: el software lo desarrollan personas.
2. **Economía**: Asegura el valor económico de lo que haces.
3. **Beneficio mutuo**: Buscamos prácticas que beneficien a todos ahora.
4. **Auto-similitud**: aplica patrones conocidos a múltiples problemas.
5. **Mejora**: excelencia en el desarrollo a través de la mejora. Buscamos perfeccionar, no lo perfecto.
6. **Diversidad**: dos ideas sobre un diseño presentan una oportunidad, no un conflicto.
7. **Reflexión**: los buenos equipos piensan cómo y por qué hacen el trabajo, no esconden errores.
8. **Flujo**: entrega continua de valor.
9. **Oportunidad**: los problemas son oportunidades de cambio.
10. **Redundancia**: los elementos complejos deberían estar duplicados, las partes más complicadas se deben mirar dos veces.
11. **Fallo**: un fallo no es un desperdicio si me sirve para aprender algo.
12. **Calidad**: no vas a ir más rápido aceptando rebajar la calidad.
13. **Pequeños** **pasos**: ¿Qué es lo menos que puedes hacer que sea reconocible y que vaya en la dirección correcta?
14. **Responsabilidad aceptada**: el que acepta algo se hace responsable de ello.

**Las 13 prácticas de XP:**

1. **Sit Together**: Trabajar en un espacio abierto para todo el equipo, promoviendo la comunicación y aumentando la productividad mediante reuniones cara a cara.
2. **Whole Team:** Fomentar una sensación de equipo donde todos apoyan el trabajo, el crecimiento y el aprendizaje de cada miembro, manteniendo equipos dinámicos y multifuncionales que se adaptan a las necesidades del proyecto.
3. **Informative Workspace:** Crear un espacio de trabajo inmerso en el proyecto, utilizando tarjetas en la pared para historias de usuario y gráficos visibles para mostrar la evolución del proyecto.
4. **Energized Work**: Trabajar a pleno rendimiento de manera sostenible, gestionando el tiempo eficazmente y dedicando periodos específicos exclusivamente a la programación para aumentar la productividad.
5. **Pair Programming:** Todo el código debe ser escrito por dos personas trabajando juntas en una misma estación, analizando y diseñando en equipo para mejorar la calidad y mantener los estándares del equipo.
6. **Stories:** Escribir historias de usuario en tarjetas pequeñas con nombre, descripción y estimación de tiempo, priorizándolas y colocándolas en un lugar visible para facilitar la gestión y adaptación al cambio.
7. **Weekly Cycle**: Reunión al comienzo de la semana para revisar el progreso, seleccionar historias para la semana, fraccionarlas en tareas, escribir pruebas automáticas aplicando TDD para luego de lo que queda de la semana hacer la implementación asegurando que al final de la semana estén listas para ser desplegadas.
8. **Quarterly Cycle**: Reuniones trimestrales para revisar la evolución del proyecto, identificar cuellos de botella, planificar temas y seleccionar historias para el trimestre, enfocándose en la visión general del proyecto y su valor añadido.
9. **Slack:** Trabajar con un margen de tiempo para suavizar la tensión de cumplir compromisos, con una atmósfera de confianza y comunicación honesta para eliminar errores, tensiones y mejorar la credibilidad. Evitar trabajar con plazos no realistas.
10. **Ten-Minute Build:** Construir y probar automáticamente todo el sistema en 10 minutos, asegurando que todo el proceso de construcción y pruebas esté automatizado, incluyendo pruebas de partes nuevas y existentes.
11. **Continuous Integration:** Integrar cambios de código antes de que pase dos horas, asegurando que el sistema sea construido y probado continuamente para detectar errores rápidamente y mantener el software listo para su despliegue.
12. **Test-First Programming**: Escribir pruebas antes de escribir el código, lo que sirve de especificación y guía para el desarrollo, mejorando la cohesión, el acoplamiento y la confianza en el código producido.
13. **Incremental Design:** Diseño gradual con pequeños y seguros pasos, mejorando continuamente el sistema en su conjunto y eliminando duplicación, para facilitar cambios y mejorar el diseño basado en la experiencia adquirida.