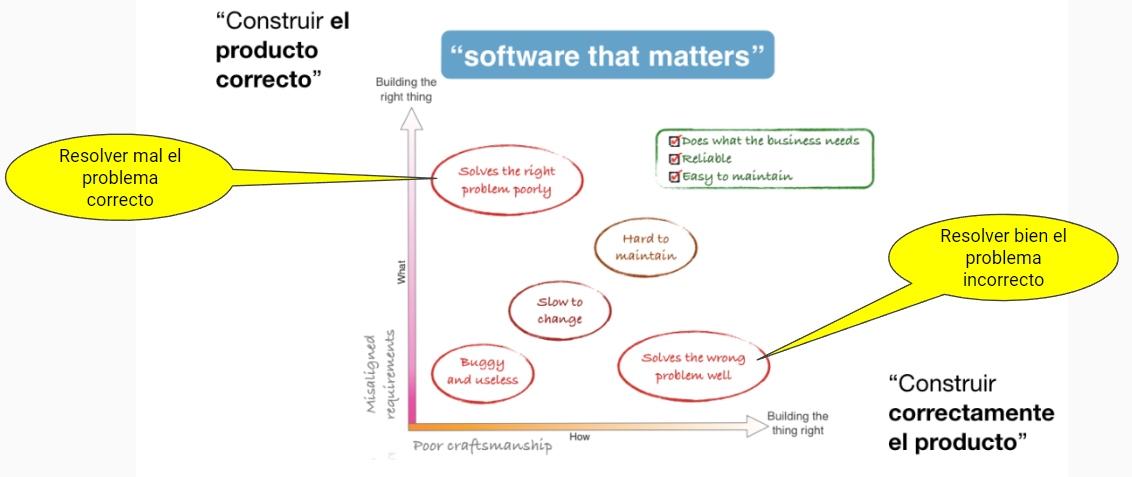
**Metodologías Agiles – Ciclo unificado – Fundamentos Agiles**

**El factor humano – Comunicación y colaboración:**

* En el desarrollo de software es fundamental el “**factor humano”.**
* La ingeniería es el arte de resolver problemas y mejorar la vida de las personas.
* Para construir un sistema es necesario entender como las personas van a utilizarlo.
* Una gran parte del trabajo de desarrollo de software es comunicación y colaboración.
* Gran parte de las herramientas de las met. agiles no son sino formas de gestionar la comunicación y la colaboración.

**Las dos caras del desarrollo de software:**

**Software como producto a distribuir y desplegar:** (Puesta en funcionamiento)

* Modelos de distribución y despliegue
* Aplicación web en internet
* App móvil
* Aplicación de escritorio
* En cualquier modelo es necesario que la empresa de desarrollo **mantenga todas las versiones del software entregadas** y usadas por los usuarios. Es **imprescindible un sistema de control de versiones.**

**Software como producto a usar:** (La importancia de lo funcional del software)

* El objetivo del software es ser usado y **proporcionar valor** en ese uso.
* La flexibilidad del software hace complicado diseñar el producto final.
* Muchas veces hay que **descubrir** el producto final.ee
* Ciclo de mejora continua: IDEA 🡪 PRODUCTO 🡪 EVALUACION

**Calidad en el software**

* Nuestra responsabilidad es desarrollar software de calidad, mejorando los procesos de desarrollo y el resultado final.
* El producto no funciona cuando:
* La aplicación solo cumple parcialmente lo que necesita el usuario.
* La aplicación no entiende el contexto del usuario.
* El software está mal desarrollado cuando:
* Es lento, poco eficiente, poco robusto.
* El código es poco modificable, poco entendible, con **deuda técnica.**

**Las 3 variables de un desarrollo software**

* El **alcance:** cuantos requisitos o tareas hay que realizar.
* El **tiempo:** cuánto durará el desarrollo.
* El **coste:** cuánto dinero se dedicará al proyecto.

Se denomina el **“triángulo de hierro”** y muestra la relación entre las 3 variables. No podemos mejorar una de ellas sin tocar alguna de las otras o afectar la calidad.

**Las 4 dimensiones del desarrollo software**

El resultado final depende de cuatro dimensiones principales:

* **Personas**
* **Proceso**
* **Producto**
* **Tecnología**

**Metodologías de desarrollo**

**¿Qué es una metodología de desarrollo?**

* Define pasos a seguir, modelo del ciclo de vida del software.
* Roles y responsabilidades de los distintos actores que intervienen en el proceso.

Una metodología es más que un conjunto de reglas, también se compone por valores, principios, venas prácticas y herramientas.

**Modelo de desarrollo de cascada (Waterfall)**

* Fuerte énfasis en la planificación detallada y predecible
* El análisis y diseño se hace en gran parte antes de la implementación, prueba y puesta en producción.
* Cada fase debe producir un **artefacto** que es analizado, corregido y que se da como entrega a la próxima fase.
* Pruebas
* Se realizan al final del desarrollo.
* Cada fase tiene sus propias validaciones.
* Cada artefacto se valida antes de la siguiente fase.
* Las pruebas de código se plantean como una verificación.
* No se contempla el cambio, evolución ni automatización de pruebas.

**Brooks: El cuello de botella del desarrollo de software**

Hay cuatro propiedades de los elementos esenciales de un proyecto de software:

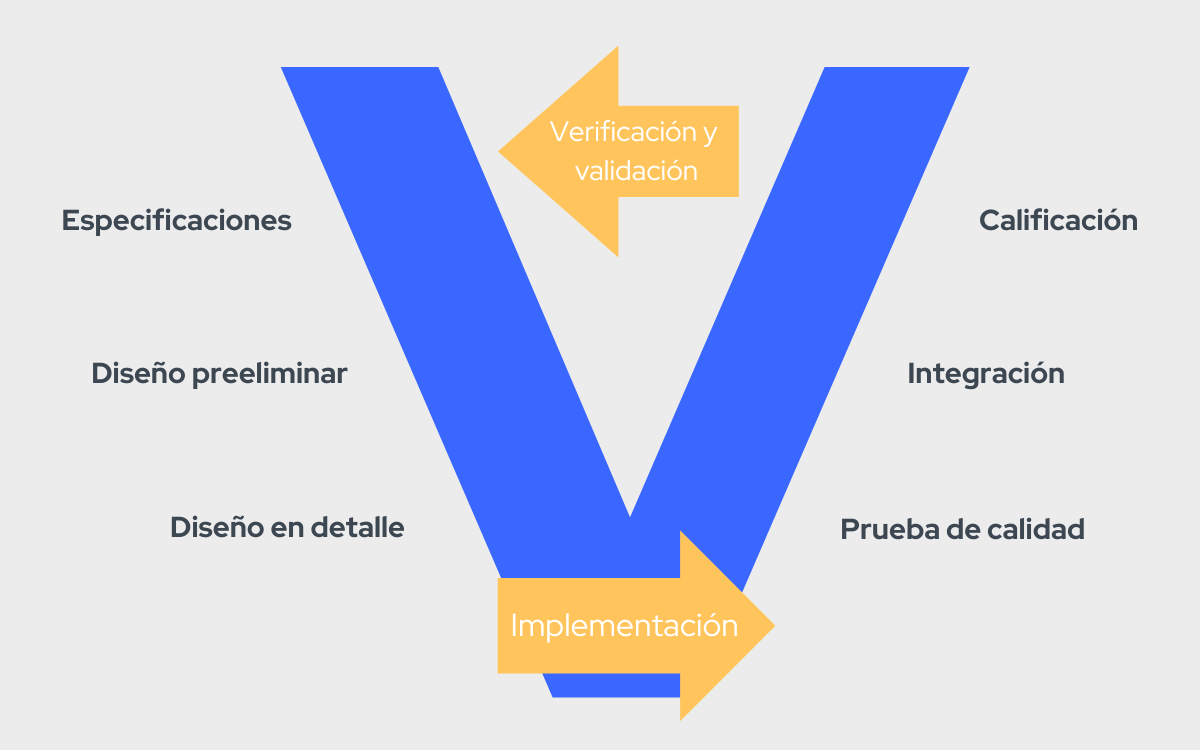
* Complejidad
* Conformidad
* Cambiabilidad
* Invisibilidad

Algunas ideas de cómo sería posible optimizar el desarrollo de los elementos esenciales del software:

* **Comparar frente a construir:** utilizar software ya existente.
* **Prototipado rápido y refinamiento de los requisitos.**
* **Desarrollo incremental – crecer no construir el software.**

**Modelo V**

* Cada fase del ciclo define un conjunto de pruebas que hay que realizar sobre el producto final.
* El eje horizontal representa el tiempo o el grado de terminación.
* El eje vertical representa el nivel de abstracción.
* Evolución a mejor del modelo cascada:
* Énfasis en pruebas.
* Distintos tipos de pruebas.
* Relación entre fases iniciales.

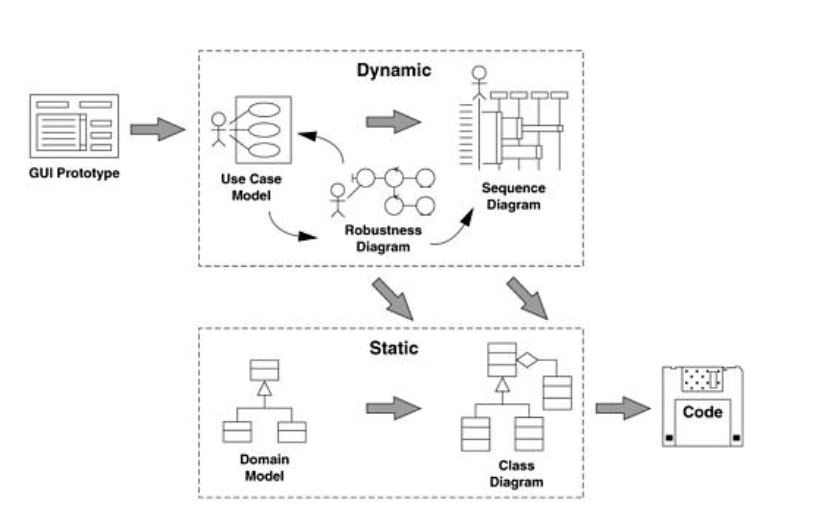


**Definición del proceso ICONIX**

* Metodología de desarrollo de software.
* Solución de compromiso entre RUP y XP.
* Basado en UML.
* Guiado por casos de uso.
* Usa sólo 4 diagramas UML.

**Etapas**

* **Análisis de requerimientos.**
* Identificar Casos de Uso.
* Producir un Modelo de Dominio.
* Hacer prototipos UI.
* **Análisis y Diseño preliminar.**
* Describir textualmente los Casos de Uso.
* Efectuar un Análisis de Robustez.
* **Diseño detallado**
* Construir un Diagrama de Clases a partir del MD.
* Construir Diagramas de Secuencia a partir de los CU.
* **Implementación**
* Escribir Casos de Prueba.
* Codificación a partir de los DCD y DSD.



**ICONIX logra:**

* Uso racionalizado de UML.
* Alto grado de trazabilidad.
* Iterativo e incremental.

**El Proceso Unificado**

**¿Por qué implementarlo?**

* Inefectividad, ineficacia y/o ineficiencia.
* Porque tiene malas variables en su economía.
* Porque tiene mala calidad de software.
* Porque tiene mala mantenibilidad.
* Porque tiene complejidad arbitraria.

**¿Qué es?**

* Proceso de desarrollo de software basado en componentes, construido por componentes interconectados y bien definidos.
* Dirigido por Casos de Uso.
* Centrado en la Arquitectura.
* Iterativo e incremental.

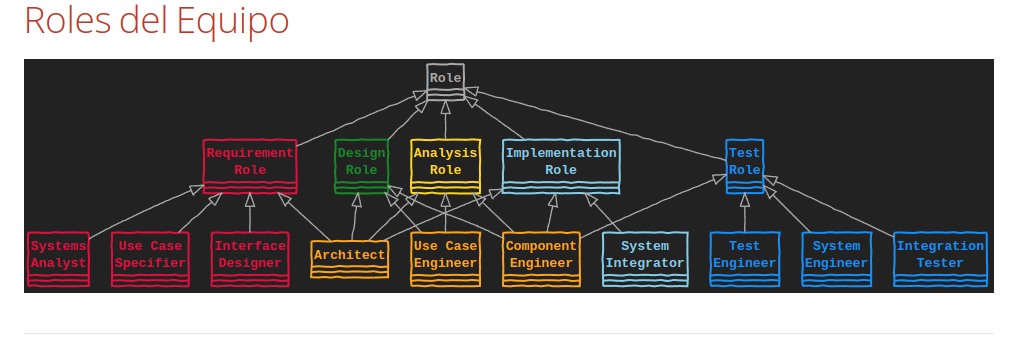
**¿Para qué implementarlo?**

* Efectividad, eficacia y eficiencia.

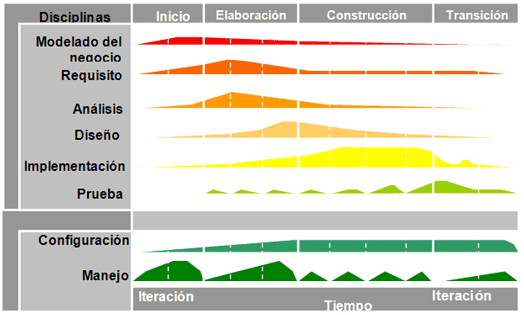
**Beneficios**

* Construir el sistema en el tiempo de forma incremental.
* Manejar mejor los requisitos cambiantes.
* Disponer de una estructura para guiar el desarrollo.
* Gestión temprana del riesgo.

**Roles del equipo:**



**Fases**



**Fundamentos Ágiles**

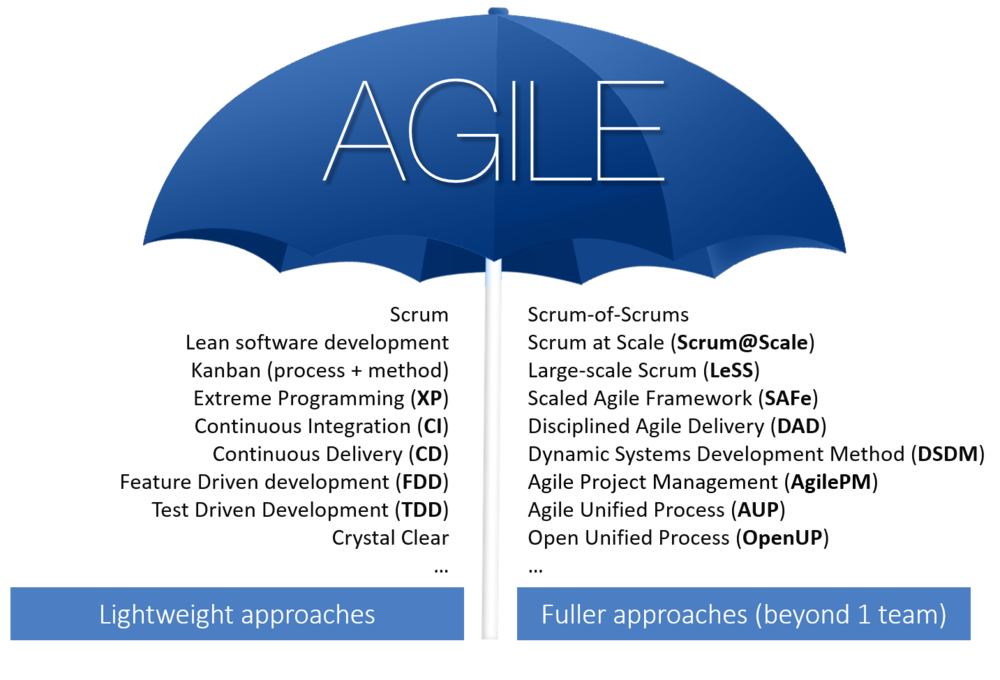
**Modelo espiral**

* Se aprovecha de que el software es fácilmente modificable.
* Sistema mínimo funcionando desde el principio.
* Cada iteración añade un conjunto de funcionalidades y si es necesario refina lo existente.
* Al final de cada iteración se obtiene un prototipo.

**La propuesta ágil: ciclos cortos**

* Cada ciclo corto incorpora todas las fases del ciclo de vida.
* Retroalimentación temprana para obtener las nuevas características para el siguiente ciclo.
* Cada iteración tiene un nuevo producto al que se le han añadido características nuevas 🡪 Prototipo.
* El producto evoluciona y adopta los cambios sugeridos por el cliente hasta que se complete.

**¿Qué es ágil?**

****

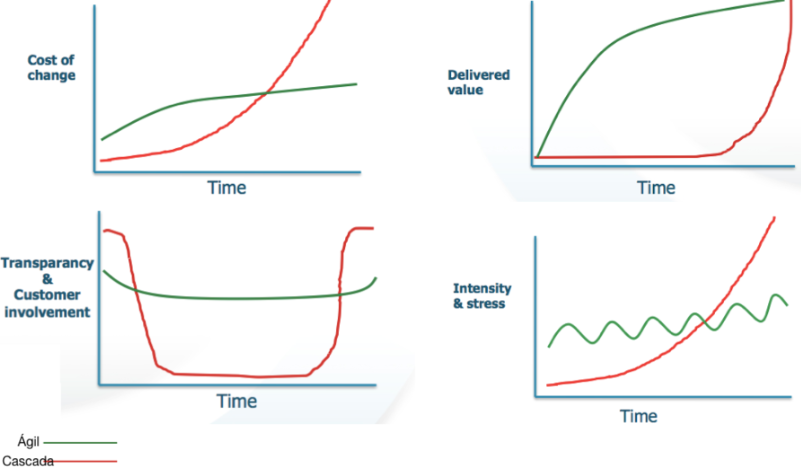
**Desarrollo incremental**

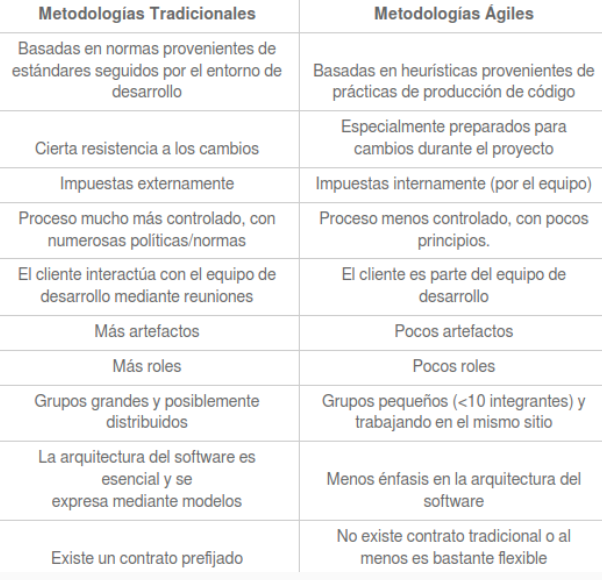
* En todas se adopta un desarrollo de iteraciones cortas.
* Al final de cada una debe obtenerse un incremento.
* Debe permitir una integración rápida de las nuevas características.

**Producto mínimo viable (MVP)**

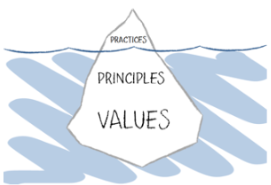
* Idea para construir startups aplicable al desarrollo de software.
* Hacer primero lo que más incertidumbre genera.

**Agile vs Cascada (Tradicional):**

****



**Valores, principios y practicas**

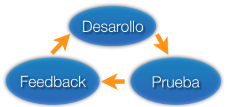
* Las practicas tiene sentido en el marco de un conjunto de valores y principios.
* Las metodologías agiles no son dogmáticas, responden a principios y valores.
* Si se intenta copiar las practicas sin aplicar los principios subyacentes se caerá en la repetición vacía.
* Para aplicar las prácticas los valores y principios deben ser compartidos por toda la organización y no solo por el equipo de desarrollo.
* Los valores y principios son generales y se pueden aplicar a múltiples situaciones. Mientras que las prácticas son concretas para determinado contexto de aplicación.

**Los 4 valores del manifiesto ágil:**

1. **Individuos e interacciones** sobre proceso y herramientas.
2. **Software que funciona** sobre documentación exhaustiva.
3. **Colaboración con el cliente** frente a negociación exhaustiva.
4. **Responder al cambio** frente a seguir un plan.

**Los 12 principios del manifiesto ágil:**

1. Satisfacer al cliente mediante entrega temprana y continua de software con valor, esa es la mayor prioridad. Uso del método Kaizen, mejora continua:



1. Aceptar que los requisitos cambien, incluso en etapas tardías.
2. Entregar software funcional frecuentemente. Entre dos semanas y dos meses, con preferencia al periodo más corto posible. Las iteraciones cortas entregan valor al cliente y permiten obtener la retroalimentación (feedback) para el equipo.
3. Los responsables de negocio y desarrolladores trabajan en conjunto.
4. Los proyectos se desarrollan en torno a individuos motivados.
5. El método más efectivo de comunicación es el cara a cara.
6. El software funcionando es la medida principal de progreso.
7. Los procesos ágiles promueven el desarrollo sostenible.
8. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
9. La simplicidad en el código a desarrollar es esencial.
10. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
11. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo.

**Prácticas Ágiles:**

* **Prácticas del Gerente de Proyecto:**

1. **Tableros de tareas (Task boards):** Herramientas visuales utilizadas para rastrear el progreso de tareas y proyectos. A menudo usan columnas como "Por hacer", "En progreso" y "Hecho" para visualizar el estado de las tareas.
2. **Puntos de historia (Story points):** Una unidad de medida para estimar el esfuerzo requerido para implementar una historia de usuario, relativa a otras historias.
3. **Gráficos de quemado (Burn down charts):** Representaciones gráficas que muestran la cantidad de trabajo restante en un proyecto en comparación con el tiempo. Ayuda a los equipos a seguir su progreso y predecir si pueden completar su trabajo a tiempo.
4. **Velocidad del proyecto (Project velocity):** La cantidad de trabajo que un equipo puede completar en una iteración determinada, medida en puntos de historia. Ayuda a predecir el rendimiento futuro.
5. **Temas (Themes):** Objetivos amplios que guían la creación y priorización de historias de usuario. Los temas ayudan a asegurar que el trabajo esté alineado con los objetivos comerciales.
6. **Priorización (Prioritization):** El proceso de ordenar tareas o historias de usuario por importancia o valor para el proyecto o cliente.
7. **Estimación (Estimation):** La práctica de predecir la cantidad de tiempo o esfuerzo necesario para completar una tarea o proyecto.

* **Prácticas del Líder del Equipo:**

1. **Liderazgo de servicio (Servant leadership):** Una filosofía de liderazgo donde el objetivo principal del líder es servir al equipo. Implica priorizar las necesidades de los miembros del equipo y ayudarles a crecer y rendir de la manera más efectiva posible.
2. **Radiador de información (Information radiator):** Una gran pantalla, altamente visible, utilizada por los equipos para compartir información importante sobre el estado del proyecto con todas las partes interesadas.
3. **Trabajar juntos (Sit together):** La co-localización de los miembros del equipo para mejorar la comunicación y la colaboración.
4. **Comunicación osmótica (Osmotic Communication**): El flujo de información dentro de un equipo que ocurre de forma natural cuando los miembros del equipo trabajan en proximidad.
5. **Planning Poker (Planning Poker):** Una técnica de estimación basada en el consenso utilizada para estimar el esfuerzo o tamaño relativo de las historias de usuario en el desarrollo de software.

* **Prácticas del Product Owner:**

1. **Elemento del backlog (Backlog item):** Una tarea, requisito o característica específica que necesita completarse y que se agrega al backlog del producto.
2. **Historias de usuario (User Stories):** Descripciones breves y simples de una característica desde la perspectiva del usuario final, explicando quién, qué y por qué.
3. **Iteración (Iteration):** Un período de tiempo limitado durante el cual se completa un trabajo específico y se deja listo para revisión. Generalmente dura de una a cuatro semanas.
4. **Ranking relativo (Relative ranking):** Priorización de elementos del backlog según su importancia o valor relativo.
5. **Backlog del producto (Product backlog):** Una lista priorizada de trabajo para el equipo de desarrollo, derivada del roadmap y sus requisitos.

* **Prácticas del Desarrollador:**

1. **Refactorización (Refactoring):** El proceso de reestructurar el código existente sin cambiar su comportamiento externo para mejorar la legibilidad, reducir la complejidad o mejorar el mantenimiento.
2. **Programación en pareja (Pair programming):** Una técnica de desarrollo de software ágil en la que dos programadores trabajan juntos en una misma estación de trabajo, con uno escribiendo código y el otro revisándolo.
3. **Sistema de control de versiones (Version control system):** Un sistema que registra cambios en archivos a lo largo del tiempo para que versiones específicas puedan ser recuperadas más tarde. Ejemplos incluyen Git, Subversion y Mercurial.
4. **Integración continua (Continuous integration):** Una práctica en la que los desarrolladores fusionan frecuentemente sus cambios de código en un repositorio central, seguido de construcciones y pruebas automatizadas para detectar problemas temprano.
5. **Desarrollo basado en pruebas (Test-driven development):** Un proceso de desarrollo de software donde se escriben pruebas antes del código en sí. El desarrollo procede escribiendo código para pasar las pruebas, seguido de la refactorización.