Mujeres Digitales 2025

Metodologías Ágiles



AHK Deutsch-Kolumbianische Industrie- und Handelskammer
Cámara de Industria y Comercio Colombo-Alemana

Fomentado por:
Ministerio Federal de Economía y Energía
en virtud de una decisión del Bundestag alemán

ORGANIZACIÓN KEY



Metodologías Ágiles

1. Introducción a las metodologías ágiles pág 4
   1.1. ¿Qué son las metodologías ágiles y por qué importan? pág 5
   1.2. ¿Cómo se diferencian de los métodos tradicionales? pág 6
   1.3. Ventajas y desafíos de ser ágil en el desarrollo web pág 8
   1.4. Ágil más allá del software: aplicar agilidad en lo académico y personal
2. El Manifiesto Ágil pág 9
   2.1. Historia y contexto del manifiesto pág 10
   2.2. Los 4 valores ágiles
   2.3. Los 12 principios ágiles pág 12
   2.4. Cómo se traduce el manifiesto ágil en proyectos de desarrollo web pág 15
   2.5. ¿Por qué hay tantas metodologías ágiles diferentes? pág 16
3. Kanban para visualizar y organizar procesos pág 17
   3.1. ¿Qué es Kanban y cómo se diferencia de Scrum?
   3.2. Creando un tablero Kanban para un proyecto de desarrollo web pág 18
   3.3. Kanban personal: organiza tus tareas académicas o tu emprendimiento pág 20
   3.4. Ejemplos de Herramientas colaborativas pág 21
4. Marco de trabajo Scrum
   4.1. Estructura de Scrum y sus roles clave pág 22
      4.1.1. Scrum Master, Product Owner y equipo de desarrollo pág 23
      4.1.2. ¿Cómo se adapta Scrum a un equipo de desarrollo web?
      4.1.3. Casos reales: rol de un desarrollador web dentro de un equipo ágil



4.1.4. El rol del desarrollador en proyectos académicos y personales pág 24

4.2. Sprints: ¿Qué son y cómo organizar un sprint?

4.3. Backlog de producto y backlog del sprint aplicado a desarrollo web pág 25

4.4. Ceremonias Scrum - Eventos clave pág 26

4.4.1. Sprint Planning

4.4.2. Daily Scrum pág 27

4.4.3. Sprint Review

4.4.4. Sprint Retrospective pág 28

4.5. Herramientas digitales para implementar Scrum

5. Planificación ágil en desarrollo web pág 29

5.1. ¿Cómo estimar tareas y esfuerzo cuando aún estás aprendiendo? pág 30

5.2. ¿Qué es una historia de usuario y cómo escribirla en desarrollo web?

5.3. División de tareas técnicas pág 31

5.4. Velocidad del equipo y gestión de avances pág 38

6. Casos de uso reales: desarrollo web con metodologías ágiles

6.1. Proyecto académico ágil: desde el planteamiento hasta el modelo predictivo pág 39

6.2. Proyecto freelance o emprendimiento de desarrollo pág 46

6.3. ¿Cómo usar ágil si trabajas como desarrollador en una empresa? pág 48

6.4. Adaptar lo aprendido a proyectos de vida personal pág 49

7. Herramientas digitales para la gestión ágil

7.1. Trello: estructura básica y tableros ejemplo para proyectos de desarrollo pág 51

7.2. Notion: gestión personal y de equipo pág 53

7.3. Miro y FigJam para colaborar y mapear ideas pág 55

7.4. Plantillas útiles para proyectos de desarrollo web pág 56

8. Buenas prácticas y errores comunes al aplicar agilidad

8.1. Evitar la sobreplanificación y aprender a iterar pág 58

8.2. Perfeccionismo vs. entregas incrementales pág 59



8.3. Cuando el equipo no colabora:
¿Cómo adaptarse y seguir avanzando? pág 60
8.4. Ágil no significa “sin estructura” pág 61
9. Otras metodologías ágiles que debes conocer
9.1. XP (Extreme Programming):
calidad técnica y colaboración pág 63
9.2. Lean: elimina el desperdicio,
enfócate en lo esencial pág 64
9.3. Crystal: agilidad adaptativa según tu equipo pág 65
9.4. Combinación de metodologías pág 67
10. ¿Cómo integrar la agilidad a tu formación y carrera
como desarrollador web?
10.1. Aprender con agilidad: autogestión del conocimiento pág 68
10.2. Crear tu portafolio de desarrollo de forma ágil pág 69
10.3. Iterar tus proyectos para mejorar
con retroalimentación pág 70
10.4. Ágil como cultura de vida: prioriza, adapta, mejora pág 71



1. Introducción a las metodologías ágiles

Antes de sumergirnos en definiciones o en comparaciones con métodos tradicionales, es importante entender por qué hablamos de metodologías ágiles en una formación de desarrollo web. Quizá te preguntes: "¿No son estas metodologías solo para programadores senior o grandes equipos de software?" La respuesta es un rotundo no.

A medida que los proyectos digitales evolucionan, se hace evidente que no solo importa lo que sabes programar, sino **cómo trabajas con otros** para resolver problemas complejos en entornos cambiantes.

El desarrollo web no es un proceso completamente lineal y predecible: durante el camino, los requerimientos del cliente pueden cambiar, la tecnología puede actualizarse o surgir nuevas prioridades de negocio. En este contexto, aplicar un enfoque ágil no es un lujo, sino una necesidad.

Aprender metodologías ágiles te permitirá:

- **Responder rápidamente a cambios en los requisitos.**
- **Entregar valor al cliente de manera continua e incremental.**
- **Colaborar constantemente con desarrolladores, diseñadores, testers y stakeholders.**
- **Mejoren la calidad del producto mediante ciclos cortos de retroalimentación.**
- **Desarrollar una mentalidad orientada a la mejora continua**, fundamental tanto en la academia como en la vida profesional.

Esta guía no busca convertirte en un "experto en agilidad", sino darte una caja de herramientas prácticas para que entiendas cómo aplicar sus principios, marcos y dinámicas tanto en proyectos académicos como en tu futuro profesional, e incluso si lo deseas, en tu vida personal.



1.1 ¿Qué son las metodologías ágiles y por qué importan?

En el mundo actual, donde los cambios son constantes y la información fluye más rápido que nunca, planear todo desde el principio y seguir un camino fijo puede resultar ineficiente. Aquí es donde entran las metodologías ágiles: formas flexibles y adaptativas de trabajar en equipo, diseñadas para enfrentar la incertidumbre y los cambios de forma natural.

Imagina que vas a resolver un rompecabezas, pero alguien va cambiando las piezas mientras tú lo armas. ¿Cómo puedes completarlo? Lo lógico sería revisar con frecuencia lo que llevas, ajustar tu enfoque y colaborar con otros para encontrar nuevas soluciones. Eso, en esencia, es el corazón de la agilidad.

Las metodologías ágiles son un conjunto de principios y prácticas que permiten desarrollar productos, resolver problemas o avanzar en proyectos de manera iterativa e incremental. Es decir, se trabaja en pequeños pasos, entregando valor de forma constante, aprendiendo del proceso y adaptándose según lo que se descubre en el camino. Este enfoque resulta especialmente valioso en el campo del desarrollo web ya que los cambios son repentinos, actualizaciones que generan bugs o comunicación entre los integrantes de un proyecto robusto que necesito óptimos tiempos de entrega.

Además, las metodologías ágiles se nutren de filosofías orientales como:

- Kaizen: filosofía japonesa que significa "mejora continua". Implica hacer pequeños ajustes constantes para mejorar procesos y resultados. En agilidad, esto se traduce en retroalimentaciones continuas y optimización progresiva del trabajo.
- Kata: prácticas repetitivas orientadas a la mejora de habilidades. En agilidad se utiliza para formar hábitos de pensamiento y acción que promuevan la resolución de problemas y la innovación.
- Shuhari: un concepto que describe el proceso de aprendizaje en tres etapas: *Shu* (seguir las reglas), *Ha* (romper las reglas de forma controlada), y *Ri* (trascender las reglas). Esto refleja cómo una persona evoluciona dentro de un equipo ágil: desde aprender las bases hasta adaptar y crear nuevas formas de trabajar.



Comprender estos conceptos ayuda a ver que la agilidad no es solo una serie de herramientas, sino un enfoque profundo hacia la mejora continua, la práctica disciplinada y la evolución del conocimiento.

1.2 ¿Cómo se diferencian de los métodos tradicionales?

Para entender mejor lo que representa la agilidad, pensemos en el enfoque tradicional como una receta de cocina estricta: primero se hace la lista de ingredientes, luego se siguen los pasos uno a uno sin desviarse, y al final se prueba el platillo. Esto se llama el modelo "en cascada", y aunque funciona para ciertos tipos de proyectos, puede ser problemático cuando el objetivo o las condiciones cambian a mitad del proceso.

El modelo en cascada (waterfall) se basa en una secuencia rígida de fases:

- Recolección de requisitos: se define todo lo que debe hacerse, antes de comenzar el trabajo.
- Diseño: se elabora un plan detallado de la solución, sin desarrollarla aún.
- Implementación: se lleva a cabo el desarrollo o ejecución siguiendo lo planeado.
- Pruebas: se verifica si lo construido cumple con lo definido al inicio.
- Despliegue: se pone en marcha el resultado final.
- Mantenimiento: se atienden errores o mejoras luego de la entrega.

Este modelo parte del supuesto de que todo puede saberse desde el inicio, lo cual es poco realista en proyectos donde se aprende haciendo, como el desarrollo web.

Las metodologías ágiles, en cambio, se parecen más a preparar una comida con amigos usando lo que hay en la nevera. Se empieza con una idea general, se cocina por partes, se prueba constantemente, se ajusta el sabor, y todos participan activamente en decidir qué se necesita. Así, el resultado final es más dinámico, colaborativo y adaptable.



En desarrollo web, rara vez un proyecto arranca con todos los detalles cerrados. Al inicio, el cliente puede decir: "*Quiero una página para mostrar mis productos*", pero a mitad del trabajo surgen nuevas ideas como "*¿y si añadimos un carrito de compras?*", "*¿qué tal un login con redes sociales?*" o "*necesitamos un panel de administración*". Estos cambios son inevitables y, si se trabaja con un modelo rígido en cascada, pueden convertirse en retrasos y sobrecostos.

Con un enfoque ágil, en cambio, el equipo puede adaptar el proyecto paso a paso, entregando funcionalidades pequeñas (como el catálogo básico), recibiendo retroalimentación y luego sumando nuevas características sin tener que rehacer todo el sistema.

Comparación entre enfoque tradicional (cascada) y ágil

Aspecto	Modelo en Cascada (Tradicional)	Metodologías Ágiles
Filosofía base	El cambio es un riesgo que debe evitarse.	El cambio es una oportunidad de mejora continua.
Estructura de trabajo	Lineal y secuencial (fases rígidas).	Iterativa e incremental (por ciclos).
Planificación	Detallada desde el inicio; difícil de modificar.	Flexible y revisada constantemente.
Entrega de resultados	Producto final al final del proceso.	Entregas parciales y funcionales frecuentes.
Participación del cliente	Limitada a la etapa inicial de requisitos.	Constante a lo largo del proyecto.
Gestión del cambio	Costoso y difícil de implementar.	Se espera y se adapta rápidamente.
Control y seguimiento	Basado en documentación extensa.	Basado en retrospectivas y mejoras continuas.
Adecuado para...	Proyectos muy estructurados, con requisitos estables.	Proyectos complejos, con incertidumbre o descubrimiento progresivo.



1.3 Ventajas y desafíos de ser ágil en el desarrollo web

Adoptar una mentalidad ágil trae múltiples ventajas al trabajar en desarrollo web:

- Adaptación al cambio: si el cliente cambia de idea ("quiero un formulario de contacto", "mejor hagamos un chatbot"), el enfoque ágil permite ajustar sin esperar hasta el final.
- Colaboración constante: diseñadores, desarrolladores y testers trabajan juntos desde el principio, compartiendo avances y resolviendo problemas en equipo.
- Entrega continua de valor: en lugar de esperar meses para ver el producto terminado, el cliente recibe versiones funcionales parciales (como un login, un carrito de compras o un diseño responsive) que ya aportan valor real.

Sin embargo, también hay desafíos:

- Falsa sensación de improvisación: ser ágil no significa improvisar sin dirección. Requiere planificación, disciplina y prioridades claras.
- Dificultad al estimar tiempos: al construir aplicaciones web, las tareas pueden variar mucho en complejidad (una landing page no es lo mismo que una API completa), lo que hace difícil predecir tiempos si el equipo no tiene experiencia.
- Resistencia al cambio: tanto en lo académico como en lo profesional, hay entornos acostumbrados a planificar todo desde el inicio. Cambiar esa mentalidad requiere práctica ya que adoptar ciclos iterativos puede generar dudas o incomodidad (*"primero diseñamos TODO el frontend, luego pasamos al backend"*)

En resumen, la agilidad no es una técnica, es una actitud frente al trabajo: abierta, flexible, colaborativa y centrada en generar valor desde el primer momento.

1.4 Ágil más allá del software: aplicar agilidad en lo académico y personal

Aunque las metodologías ágiles nacieron en el mundo del desarrollo de software, sus principios se han extendido a muchas otras áreas, desde la educación hasta la vida personal.



- ¿Se puede usar Kanban para planificar tus estudios? Sí. Puedes usar un tablero visual para organizar tus tareas, priorizar lo más urgente y dar seguimiento a tu progreso.
- ¿Puedes aplicar Scrum para hacer un trabajo en grupo? Absolutamente. Dividan el proyecto en sprints, reúnanse cada cierto tiempo para compartir avances, distribuyan roles según fortalezas.
- ¿Y en tu vida diaria? Claro. Puedes organizar tus metas personales por ciclos, revisar tus avances semanalmente, ajustar lo que no esté funcionando y celebrar tus pequeños logros.

Ser ágil no es una moda ni una etiqueta: es una forma de pensar y trabajar que te prepara para responder al cambio, colaborar mejor y aprender más en menos tiempo. Es una competencia transversal que, como futura desarrolladora, te hará más valiosa tanto en la academia como en el entorno laboral o incluso en tus propios proyectos de emprendimiento.

Ahora que comprendemos qué es la agilidad y por qué es importante, en los siguientes capítulos exploraremos con más detalle cómo funcionan los distintos marcos de trabajo ágiles y cómo puedes aplicarlos de forma concreta en tus estudios y proyectos de desarrollo web.

2. El Manifiesto Ágil

Antes de hablar de metodologías como Scrum, Kanban o XP, es necesario ir a la raíz de todo: el Manifiesto Ágil. Este documento, sencillo pero poderoso, cambió la forma en la que se desarrollan proyectos en tecnología (y en muchos otros sectores) desde principios de los años 2000. Para entenderlo, primero debemos conocer su historia, sus valores, principios y, sobre todo, cómo podemos aplicarlo al contexto de desarrollo web.

Imagina que el Manifiesto Ágil es como una brújula moral o una carta de navegación: no te dice exactamente qué camino tomar, pero sí en qué dirección moverte cuando tomes decisiones en un proyecto. Veamos paso a paso cómo surgió, qué propone y cómo puede ayudarte a trabajar de forma más efectiva y humana con tu equipo.



2.1 Historia y contexto del manifiesto

A finales de los años 90 y principios de los 2000, muchas personas que trabajaban en desarrollo de software estaban cansadas. Cansadas de procesos largos, burocráticos y poco efectivos. Había muchos documentos, mucha planificación rígida, pero al final el cliente no quedaba satisfecho o el producto llegaba tarde y no cumplía las expectativas.

Un grupo de 17 expertos en desarrollo de software se reunió en 2001 en un lugar llamado Snowbird, en Utah (EE. UU.). No iban a crear una metodología nueva. Solo querían hablar sobre qué cosas les funcionaban mejor a la hora de trabajar. Lo que surgió de ese encuentro fue el Manifiesto Ágil: una declaración de valores y principios que buscaba poner a las personas, la colaboración y la flexibilidad por encima de los procesos rígidos.

2.2 Los 4 valores ágiles

Los valores ágiles son comparativos, lo que significa que no niegan el valor de los elementos a la derecha, sino que proponen que los elementos a la izquierda tienen más peso cuando se trata de tomar decisiones. Son la base ética y operativa de todo lo que vendrá después en metodologías ágiles. Veamos cada uno con detalle.

Individuos e interacciones sobre procesos y herramientas

Este valor nos invita a poner a las personas primero. Las herramientas son importantes, claro. Pero lo más valioso en un equipo ágil es la calidad de la comunicación entre las personas que lo integran. Ninguna herramienta reemplaza una conversación clara y efectiva.

Ejemplo: Imagina que estás en un equipo de desarrollo con una plataforma digital muy completa para gestionar tareas, pero el equipo no se comunica bien. Por más tecnología que tengas, si las personas no colaboran, el trabajo se tranca. En cambio, un grupo que se reúne 10 minutos diarios para hablar cara a cara puede avanzar mucho más.



Analogía: Es como un grupo de estudio. Puedes tener todos los libros y recursos online del mundo, pero si no te organizas bien con tus compañeros o no sabes cómo preguntar y escuchar, el aprendizaje se complica.

Software funcionando sobre documentación extensiva

Este valor hace énfasis en los resultados tangibles. La documentación es importante, pero lo fundamental es que el producto funcione. En un mundo cambiante, entregar valor rápido es más útil que tener 50 páginas explicando lo que aún no existe.

Ejemplo: Un equipo pasa un mes haciendo diagramas de flujo, requerimientos y documentación técnica para una aplicación que aún no han empezado a construir. Mientras tanto, otro equipo entrega una versión básica de la aplicación en dos semanas. ¿Cuál aporta más valor en ese momento? El que ya tiene algo funcional, aunque le falten detalles.

Analogía: Es como hacer una tesis con un marco teórico perfecto, pero sin datos, sin análisis y sin resultados. ¿Qué pesa más, la teoría o la evidencia?

Colaboración con el cliente sobre negociación contractual

La idea aquí es que el cliente o usuario forme parte activa del proceso. No se trata de que firme un contrato al inicio y luego desaparezca hasta la entrega final. Su feedback continuo permite que el producto se ajuste mejor a sus necesidades.

Ejemplo: Un cliente firma un contrato para una app con 10 funcionalidades. A la semana #3, se da cuenta de que necesita una nueva función y que otra no es tan necesaria. En vez de renegociar todo el contrato, el equipo ágil conversa, adapta y sigue adelante.

Analogía: Imagina que estás haciendo un proyecto final con un profesor como tutor. En lugar de entregarle el proyecto al final sin mostrarle nada, prefieres reunirte con él periódicamente para recibir retroalimentación. Así mejoras continuamente.

Respuesta ante el cambio sobre seguir un plan

Planificar es necesario, pero el plan nunca es sagrado. El cambio es inevitable, especialmente en entornos complejos. Los equipos ágiles abrazan el cambio y lo integran, en lugar de resistirse a él.



Ejemplo: Un equipo planea lanzar una app para un evento en dos meses. A mitad del desarrollo, el evento cambia de fecha. En lugar de frustrarse, ajustan sus entregas y se adaptan al nuevo calendario.

Analogía: Es como organizar un viaje con tus amigos: tienes un itinerario armado, pero cuando llegas descubres que un museo está cerrado o que hay un festival en otra ciudad. En vez de enojarte porque "ese no era el plan", ajustas la ruta y disfrutas la nueva experiencia.

2.3 Los 12 principios ágiles

Estos principios detallan cómo se puede vivir y aplicar la agilidad en el día a día. Cada uno representa una pequeña brújula para actuar con flexibilidad, colaboración y enfoque en el valor. Vamos a verlos uno a uno con ejemplos.

- Nuestra mayor prioridad es satisfacer al cliente mediante la entrega continua de software con valor.
    - Ejemplo: Entregar pequeñas mejoras útiles frecuentemente es mejor que esperar meses para entregar una solución gigante.
    - En Desarrollo Web: En lugar de esperar a tener todo el sitio terminado, se entrega primero el módulo de registro de usuarios, luego el panel de administración, y después nuevas funciones.
- Damos la bienvenida a los requisitos cambiantes, incluso en etapas tardías del desarrollo.
    - Ejemplo: El cliente puede descubrir nuevas necesidades mientras ve el producto crecer. Cambiar es parte natural del proceso.
    - En Desarrollo Web: A mitad del desarrollo de una plataforma, el cliente pide reemplazar el login tradicional con autenticación por Google. El equipo adapta el flujo sin problema.
- Entregamos software funcional con frecuencia, desde un par de semanas hasta un par de meses.



  - Ejemplo: Las entregas frecuentes generan confianza y permiten correcciones tempranas.
  - En Desarrollo Web: Primero se entrega una versión básica de la aplicación con la página principal funcional, luego se suman las secciones de productos, carrito de compras y pasarela de pago.
- Los responsables del negocio y los desarrolladores deben trabajar juntos a diario.
  - Ejemplo: Comunicación fluida entre áreas técnicas y de negocio evita malentendidos.
  - En Desarrollo Web: El equipo de desarrollo tiene reuniones frecuentes con el product owner o cliente para priorizar el backlog, aclarar requisitos y validar lo que ya está funcionando, asegurando que cada sprint entregue valor real.
- Los proyectos se desarrollan en torno a individuos motivados.
  - Ejemplo: La motivación y autonomía del equipo es clave para la calidad del resultado.
  - En Desarrollo Web: Un desarrollador propone optimizar la carga de imágenes con lazy loading, aunque no estaba en los requisitos iniciales, mejorando la velocidad del sitio y la experiencia del usuario.
- El método más eficiente y efectivo de comunicar información es la conversación cara a cara.
  - Ejemplo: Las herramientas ayudan, pero nada supera una conversación directa para resolver dudas.
  - En Desarrollo Web: En lugar de enviar un correo con 10 preguntas, se hace una videollamada de 10 minutos para aclarar todo.
- El software funcional es la principal medida de progreso.
  - Ejemplo: Lo que realmente importa es lo que ya funciona, no cuánto se ha documentado o planificado.



  - En Desarrollo Web: El avance se mide por las funcionalidades que ya están desplegadas en un entorno de pruebas (como un buscador o un checkout), no por la cantidad de diagramas o documentación generada.

- Los procesos ágiles promueven el desarrollo sostenible.
  - Ejemplo: El ritmo de trabajo debe poder mantenerse a largo plazo. No se trata de trabajar hasta el agotamiento.
  - En Desarrollo Web: Se evitan “crunch times” (sobrecarga de tareas) justo antes de una entrega. El equipo distribuye las tareas en sprints y mantiene un flujo de commits y revisiones constante para asegurar calidad sin desgaste.
- La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
  - Ejemplo: La calidad técnica facilita adaptarse a los cambios. Si el código es limpio, cambiarlo no da miedo.
  - En Desarrollo Web: Bases de datos bien estructuradas, scripts ordenados y código legible permiten iterar más rápido.
- La simplicidad - el arte de maximizar la cantidad de trabajo no realizado— es esencial.
  - Ejemplo: Hacer lo justo y necesario es mejor que sobrecomplicar con funciones que nadie usará.
  - En Desarrollo Web: En lugar de implementar desde el inicio un sistema complejo de filtros en la tienda online, se comienza con un buscador simple que ya soluciona la necesidad principal.
- Los mejores resultados emergen de equipos autoorganizados.
  - Ejemplo: Los equipos que toman sus propias decisiones suelen encontrar soluciones más efectivas.
  - En Desarrollo Web: El equipo decide por sí mismo cómo dividirse: un grupo trabaja en la API de usuarios, otro en la interfaz del usuario y otro en las pruebas automáticas, sin esperar que un jefe dicte cada detalle.



- El equipo reflexiona regularmente sobre cómo mejorar su forma de trabajar y ajusta su comportamiento en consecuencia.
    - Ejemplo: La mejora continua no es una opción, es parte del proceso.
    - En Desarrollo Web: Tras un sprint, el equipo identifica que los merge conflicts retrasaron entregas. En la retrospectiva acuerdan usar ramas más cortas y hacer revisiones de código más frecuentes para la próxima iteración.

2.4 Cómo se traduce el manifiesto ágil en proyectos de desarrollo web

Aunque el Manifiesto Ágil nació en el mundo del software, sus ideas son perfectamente aplicables al desarrollo web. Aquí te doy algunos ejemplos concretos de cómo traducir los valores y principios:

- Valor #1 (individuos e interacciones) → En un equipo de desarrollo web, esto significa que la comunicación entre diseñadores, frontend, backend y QA sea fluida y constante. No basta con depender de un gestor de tareas o un tablero, las conversaciones directas son clave para resolver bloqueos rápido.
- Valor #2 (software funcionando) → Para nosotros, el valor está en entregar funcionalidades web reales que el usuario pueda probar: una landing page navegable, un formulario que guarda datos correctamente o un login que funciona. El progreso se mide por lo que ya está online, no por la cantidad de documentación escrita.
- Valor #3 (colaboración con el cliente) → Involucrar al cliente o al usuario desde el inicio. Por ejemplo, si desarrollamos un e-commerce, el área comercial debería estar en contacto constante con el equipo para priorizar qué módulos son más importantes: carrito de compras, pasarela de pago o catálogo de productos (es muy común en el área Fintech o apps con alto impacto en los usuarios).
- Valor #4 (respuesta ante el cambio) → En desarrollo web, los cambios son inevitables: el cliente puede pedir un rediseño de la interfaz, la integración con otra API o ajustar la arquitectura por nuevas necesidades. La agilidad permite redirigir esfuerzos y adaptarse sin frustrarse por haber “fallado al plan inicial”.



Ejemplo: Imagina que estás desarrollando una aplicación web para gestionar reservas en un restaurante. En lugar de esperar dos meses para mostrar la versión completa, puedes hacer entregas semanales: primero la página para que los usuarios vean el menú, luego el formulario de reservas y después la integración con pasarelas de pago. Así recibes retroalimentación temprana y puedes ajustar el desarrollo según las necesidades reales. Eso es aplicar agilidad.

2.5 ¿Por qué hay tantas metodologías ágiles diferentes?

La agilidad no es una receta única, es un conjunto de principios. Eso significa que cada equipo y cada tipo de proyecto puede adaptar esos principios a su realidad. Por eso existen tantas metodologías ágiles diferentes como Scrum, Kanban, XP (Extreme Programming) o incluso marcos híbridos.

En el mundo del desarrollo web, no todos los proyectos son iguales:

- Una startup que está construyendo una aplicación web desde cero necesita moverse rápido, probar ideas y pivotar si es necesario.
- Una agencia digital que hace landing pages para clientes diferentes quizá prefiere un tablero Kanban, porque cada entrega es corta y muy visual.
- Un equipo corporativo que mantiene una aplicación bancaria probablemente necesita Scrum, con sprints claros y revisiones periódicas para asegurar calidad y seguridad.

En pocas palabras: las metodologías son como distintos estilos de cocina. Todas buscan alimentarte (entregar valor), pero cada una tiene su manera de preparar y organizar el trabajo.

¿Cómo elegir cuál usar según tu proyecto o contexto?

No existe “la mejor” metodología, sino la que mejor se adapta al proyecto y al equipo.

Algunas guías rápidas para desarrollo web:



- Scrum: Ideal si estás desarrollando una aplicación web con funcionalidades complejas (ejemplo: un marketplace, un CRM o un SaaS). Te permite organizar el trabajo en sprints, revisar avances y recibir feedback constante.
- Kanban: Perfecto si trabajas en un flujo continuo de tareas (ejemplo: mantener un sitio web, corregir bugs, implementar mejoras pequeñas en un portal corporativo). Visualizas todo en un tablero y vas moviendo las tarjetas a medida que avanzan.
- Lean/Startup ágil: Muy útil si el proyecto es experimental o aún no se sabe qué quiere el cliente (ejemplo: una startup que prueba un nuevo modelo de e-commerce). Aquí se trabaja entregando prototipos rápidos, midiendo resultados y ajustando.
- Extreme Programming (XP): Recomendado cuando el proyecto necesita un enfoque técnico fuerte, con buenas prácticas como integración continua, pruebas automatizadas y código limpio (ejemplo: una fintech que desarrolla un sistema de pagos web).

Ejemplo: Si estás desarrollando un e-commerce pequeño, probablemente te funcione Kanban para organizar las tareas (catálogo, carrito, pasarela de pago, SEO, etc.). Pero si el mismo e-commerce crece y se convierte en una plataforma con múltiples módulos (administración, usuarios, reportes, integración con apps externas), Scrum puede darte una mejor estructura de trabajo en equipo.

3. Kanban para visualizar y organizar procesos

En esta sección vamos a sumergirnos en una de las herramientas visuales más efectivas para la organización del trabajo: el método Kanban. Imagina que estás en una cocina industrial, donde cada chef sabe exactamente en qué etapa está cada platillo. Hay una pizarra grande con etiquetas: "Por hacer", "En preparación" y "Listo para servir". Así funciona Kanban: hace visible el flujo de trabajo y permite detectar cuellos de botella, priorizar tareas y avanzar de forma continua y clara.



3.1 ¿Qué es Kanban y cómo se diferencia de Scrum?

Kanban es un sistema de organización visual que se originó en Toyota, en la industria automotriz japonesa, para mejorar la eficiencia de la producción. Con el tiempo, fue adoptado por equipos de desarrollo de software y más adelante por muchas otras disciplinas. En lugar de dividir el tiempo en ciclos (como sprints en Scrum), Kanban se enfoca en visualizar el trabajo, limitar el trabajo en curso (WIP, por sus siglas en inglés) y mejorar continuamente el flujo.

Diferencias clave con Scrum:

- Iteraciones vs Flujo continuo: Scrum se organiza en sprints (ciclos de 1 a 4 semanas), mientras que Kanban trabaja de forma continua, sin ciclos fijos.
- Roles definidos vs flexibilidad: Scrum tiene roles como Scrum Master, Product Owner, etc. Kanban no requiere roles específicos.
- Eventos obligatorios vs adaptabilidad: Scrum tiene eventos como la planificación del sprint, reuniones diarias, revisiones, retrospectivas. Kanban no impone eventos, pero promueve la mejora continua.
- Medición del progreso: Scrum mide la velocidad (número de puntos completados por sprint). Kanban usa el "Lead Time" (tiempo que tarda una tarea en completarse desde que se inicia).

Piensa en Scrum como una orquesta con partituras y tiempos marcados, y en Kanban como una banda de jazz improvisando sobre una estructura clara, pero flexible.

3.2 Creando un tablero Kanban para un proyecto de desarrollo web

Vamos a construir paso a paso un tablero Kanban para un proyecto de desarrollo web.

Imagina que tu equipo debe desarrollar una aplicación web para gestionar reservas en un restaurante.



- Define tus columnas principales:
  - Por hacer
  - En proceso
  - En revisión
  - Finalizado
- Lista las tareas principales del proyecto:
  - Configurar el entorno del proyecto
  - Diseñar la página de inicio (Home)
  - Implementar el sistema de registro e inicio de sesión
  - Crear la base de datos de usuarios y reservas
  - Desarrollar el formulario de reservas
  - Implementar notificaciones por correo de confirmación
- Coloca cada tarea en su columna según su estado actual.
  - Por ejemplo:

Por hacer	En proceso	En revisión	Finalizado
Implementar notificaciones	Crear base de datos de reservas	-	Configurar entorno

- Limita el WIP (Work In Progress): Si tienes 3 tareas en "En proceso", considera no empezar una nueva hasta terminar una. Esto evita la multitarea y mejora el enfoque.
- Revisa y ajusta: A medida que avanza el proyecto, mueve las tareas y agrega nuevas si es necesario. El tablero es un ente vivo.



3.3 Kanban personal: organiza tus tareas académicas o tu emprendimiento

El Kanban no solo se aplica a proyectos grupales. También puede ayudarte a gestionar tu vida académica o tu emprendimiento personal. ¿Alguna vez te ha pasado que tienes tantas cosas por hacer que no sabes por dónde empezar? Usar Kanban para ti misma te permite ver todo lo que tienes pendiente, priorizar y liberar tu mente.

Ejemplo para tareas académicas

- Columnas:
  - Por hacer (Tareas de la semana)
  - En curso (Tareas que ya empezaste)
  - En pausa (Por falta de datos o espera de respuesta)
  - Listo (Lo que ya entregaste o completaste)
- Tareas:
  - Estudiar fundamentos de HTML y CSS
  - Preparar presentación sobre POO
  - Revisar feedback del docente
  - Subir trabajo a GitHub

Ejemplo para un emprendimiento

- Diseño del logo
- Validación de idea con clientes
- Publicación en redes sociales
- Diseñar la landing page del negocio
- Definir planes de mantenimiento y soporte



Puedes llevar este tablero en una pizarra física, post-its, una hoja de cálculo, o apps especializadas (Trello, Jira, GitHub Projects). Lo importante es que lo revises y lo mantengas actualizado.

3.4 Ejemplos de Herramientas Colaborativas

Hoy en día, muchas herramientas digitales te permiten aplicar el enfoque Kanban de forma fácil, atractiva y colaborativa. A continuación, exploramos dos de las más populares:

Trello:

- Gratuito y muy visual.
- Puedes crear tableros, columnas y tarjetas.
- Las tarjetas se pueden etiquetar, asignar a personas, agregar fechas límite, checklists, comentarios y archivos.
- Ejemplo: crear un tablero para tu proyecto final de desarrollo, con columnas "Por Desarrollar", "En Desarrollo", "Testing", "Finalizado".

Notion:

- Más que Kanban, es una herramienta de organización todo-en-uno.
- Puedes crear bases de datos tipo Kanban que se filtran, ordenan, conectan con notas o documentos.
- Ejemplo: una base de datos con tus proyectos personales de portafolio, con columnas para estado, prioridad, fecha de entrega y enlaces directos al repositorio en GitHub o al despliegue en producción.

Consejo: Prueba ambas y elige la que mejor se adapte a tu estilo. Lo importante no es la herramienta, sino que te permita mantener la claridad sobre qué tienes que hacer, qué estás haciendo, y qué ya lograste.

En resumen, Kanban es una filosofía visual y flexible que, si se aplica con constancia, puede ayudarte a organizar cualquier proyecto, desde los más técnicos hasta los personales. Como estudiante o desarrolladora en formación, adoptar este enfoque puede marcar una gran diferencia en tu productividad y claridad mental.



4. Marco de trabajo Scrum

Scrum es uno de los marcos de trabajo ágiles más utilizados en el mundo del desarrollo de software, pero sus principios también se adaptan perfectamente a otros contextos. En esta sección exploraremos cómo Scrum puede ayudarte a organizar tu trabajo, colaborar efectivamente con otros y entregar valor de manera constante, incluso cuando tu proyecto no se trata de construir una aplicación, sino de responder a preguntas con datos.

4.1 Estructura de Scrum y sus roles clave

Las metodologías ágiles son un conjunto de enfoques de desarrollo que permiten adaptarse rápidamente a los cambios, mejorar la colaboración en equipo y entregar valor al cliente de forma incremental. Entre ellas, Scrum es una de las más utilizadas en el desarrollo de software y productos digitales.

Scrum se basa en un enfoque iterativo e incremental para abordar problemas complejos. Su estructura fomenta la colaboración constante, la entrega continua de valor y la mejora continua.

Los 3 pilares de Scrum:

- Transparencia: todos los aspectos del proceso deben ser visibles para los involucrados. Ejemplo: mantener el tablero de tareas actualizado y visible para todo el equipo.
- Inspección: el equipo revisa regularmente los progresos para detectar posibles desviaciones. Ejemplo: las reuniones diarias (Daily Scrum) permiten evaluar el avance y obstáculos.
- Adaptación: si se detectan desviaciones o problemas, se ajusta el proceso o el trabajo planificado. Ejemplo: modificar tareas o prioridades tras una Sprint Review si el producto no cumple expectativas.



Artefactos de Scrum:

- Product Backlog: lista priorizada de todas las funcionalidades, mejoras, correcciones y requisitos del producto.
- Sprint Backlog: subconjunto del Product Backlog seleccionado para trabajarse en un Sprint específico.
- Incremento: resultado funcional al final de un Sprint, que suma valor y puede ponerse en producción.

4.1.1 Scrum Master, Product Owner y equipo de desarrollo

Pensemos en un equipo como una banda musical:

- El Product Owner (PO) sería como el compositor: define la melodía (necesidades del negocio) y prioriza qué canciones tocar primero (backlog del producto).
- El Scrum Master es como el director de orquesta: no toca ningún instrumento directamente, pero se asegura de que todos los músicos estén coordinados, que no haya interrupciones y que sigan el ritmo de la obra (procesos ágiles).
- El Equipo de Desarrollo son los músicos que realmente tocan: desarrolladores, testers, diseñadores.

En un proyecto de desarrollo web:

- El PO define qué funcionalidades debe tener la aplicación o sitio (ejemplo: autenticación de usuarios, carrito de compras, panel de administración).
- El Scrum Master guía al equipo en la metodología ágil, resolviendo bloqueos y asegurando que se cumpla la dinámica de Scrum.
- El equipo de desarrollo programa las funcionalidades, integra APIs, diseña interfaces y valida que el sistema funcione correctamente.



4.1.2 ¿Cómo se adapta Scrum a un equipo de desarrollo web?

Aunque Scrum nació en el mundo del software, su aplicación en desarrollo web es muy natural porque también trabajamos en iteraciones, con entregables claros y colaborando de forma constante con otros perfiles (diseño, negocio, QA, stakeholders).

Por ejemplo, en lugar de entregar una nueva funcionalidad de software, un sprint puede enfocarse en implementar el sistema de autenticación, desarrollar el carrito de compras, o mejorar el rendimiento de la página de inicio.

4.1.3 Casos reales: rol de un desarrollador web dentro de un equipo ágil

Supongamos que trabajas en una startup que quiere lanzar un e-commerce. El Product Owner plantea como objetivo mejorar la conversión de usuarios (que más visitantes completen una compra).

Como desarrolladora web, en un sprint puedes:

- Implementar un checkout simplificado para reducir pasos en la compra.
- Optimizar el tiempo de carga de la página de productos.
- Integrar un sistema de recomendaciones que sugiera artículos relacionados.
- Colaborar con diseño para mejorar la experiencia en dispositivos móviles.

4.1.4 El rol del desarrollador en proyectos académicos y personales

Aunque no estés en una empresa, puedes aplicar Scrum para organizar tus propios proyectos. Por ejemplo, si estás haciendo un proyecto final de análisis de datos:

- Tú puedes asumir el rol de PO y definir qué quieres analizar.
- También puedes ser el Scrum Master y marcar los tiempos y reuniones.
- Y por supuesto, ser el analista que ejecuta.



Scrum te ayuda a dividir un gran proyecto en partes pequeñas y alcanzables, lo que evita la parálisis por análisis o la frustración por no avanzar.

4.2 Sprints: ¿Qué son y cómo organizar un sprint?

Un **sprint** es un ciclo de trabajo de duración fija (por lo general entre 1 y 4 semanas) en el que el equipo se compromete a entregar una parte del producto. En desarrollo web, el producto puede ser un módulo del sitio, una nueva funcionalidad o una mejora técnica visible para el usuario o el negocio.

Para organizar un sprint:

- Se define un **objetivo del sprint**: por ejemplo, "Implementar el módulo de autenticación con login y registro de usuarios".
- Se seleccionan tareas del backlog que ayudarán a cumplir ese objetivo.
- Se ejecutan las tareas durante el sprint.
- Se revisa el resultado y se aprende del proceso.

Es clave que el sprint tenga un resultado concreto, medible y útil para los interesados.

4.3 Backlog de producto y backlog del sprint aplicado a desarrollo web

El **backlog de producto** es una lista dinámica de todo el trabajo conocido necesario para desarrollar el producto. Es gestionado por el Product Owner y puede cambiar según se descubran nuevas necesidades. Los elementos están ordenados por prioridad y pueden incluir funcionalidades, errores por corregir, mejoras técnicas, etc. Es como una lista de deseos.

Ejemplo:

- Registro de usuario
- Envío de notificaciones
- Mejorar rendimiento de búsqueda



En un proyecto de desarrollo, puede incluir:

- Diseñar y maquetar la página de registro
- Programar el backend de autenticación
- Conectar el frontend con la API de login
- Probar y depurar el flujo de registro/login

El backlog del sprint es el conjunto de elementos del Product Backlog seleccionados para trabajarse en un Sprint, junto con un plan para entregarlos. Es gestionado por el equipo de desarrollo. A diferencia del Product Backlog (que es de todo el producto), el Sprint Backlog solo contiene lo que se trabajará en el Sprint actual. Es el plan para ese ciclo.

Diferencias clave entre Product Backlog y Sprint Backlog:

Característica	Product Backlog	Sprint Backlog
Propietario	Product Owner	Equipo de desarrollo
Contenido	Todo lo necesario para el producto final	Solo lo que se trabajará en el Sprint actual
Dinamismo	Se actualiza continuamente	Se congela durante el Sprint (en lo posible)
Horizonte temporal	Largo plazo (visión general del producto)	Corto plazo (solo este Sprint)
Prioridad	Priorizado por valor al negocio	Se ordena según capacidad y compromiso

4.4 Ceremonia Scrum - Eventos clave

Scrum incluye una serie de reuniones o *ceremonias* que marcan el ritmo del equipo y fomentan la colaboración.



4.4.1 Sprint Planning

Se realiza al inicio del sprint. Aquí también se estima el esfuerzo necesario.

- **Duración:** Máximo 8 horas para un Sprint de 1 mes (proporcionalmente menos si el Sprint es más corto).
- **Objetivo:** definir qué trabajo del Product Backlog se incluirá en el Sprint y cómo se realizará.
- **Participantes:** todo el equipo Scrum.
- **Ejemplo:** el equipo decide trabajar en el desarrollo del login, registro y pruebas unitarias asociadas en los próximos 10 días.

4.4.2 Daily Scrum

Una reunión diaria de 15 minutos tipo seguimiento de carácter informativo sobre el estado del proyecto:

- **Duración:** 15 minutos todos los días del Sprint.
- **Objetivo:** sincronizar al equipo, identificar impedimentos y planificar el trabajo del día.
- **Formato habitual:**
  - ¿Qué hice ayer?
  - ¿Qué haré hoy?
  - ¿Hay algo que me bloquee?
- **Ejemplo:** un desarrollador informa que terminó el módulo de registro, otro menciona que necesita ayuda con la autenticación.

Esto mantiene la transparencia y permite identificar problemas a tiempo.



4.4.3 Sprint Review

Se realiza al final del sprint. El equipo muestra lo que ha logrado.

- **Duración:** Máximo 4 horas para un Sprint de 1 mes.
- **Objetivo:** revisar el incremento del producto con los interesados, recibir retroalimentación y adaptar el Product Backlog.
- **Participantes:** equipo Scrum y stakeholders (clientes, usuarios, etc.).
- **Ejemplo:** se presenta el nuevo sistema de notificaciones, los usuarios sugieren agregar filtros por tipo de mensaje.
- Se recogen comentarios para el siguiente sprint.

4.4.4 Sprint Retrospective

Una reunión para reflexionar sobre el proceso: ¿Qué funcionó bien?, ¿Qué se puede mejorar?, ¿Qué acciones tomaremos?

- **Duración:** Máximo 3 horas para un Sprint de 1 mes.
- **Objetivo:** reflexionar sobre el proceso, identificar mejoras y planificar acciones para el siguiente Sprint.
- **Participantes:** equipo Scrum.
- **Ejemplo:** el equipo detecta que las tareas estaban mal divididas y acuerdan definir criterios de aceptación más claros para el próximo Sprint.

Estos eventos aseguran que Scrum funcione como un ciclo constante de planeación, ejecución, revisión y mejora. Gracias a ellos, se promueve la transparencia, se inspecciona el progreso frecuentemente y se adapta el trabajo según las necesidades reales del negocio y del equipo. mejora continua es clave para equipos que aprenden juntos.



4.5 Herramientas digitales para implementar Scrum

Hoy en día hay muchas herramientas para aplicar Scrum digitalmente. Algunas de las más usadas por su facilidad son:

- Trello: tableros estilo Kanban donde puedes simular un backlog y los sprints.
- Jira: más completo y robusto, permite manejar tareas, sprints, métricas y flujos de trabajo.
- Notion: muy flexible para personalizar tu tablero, ideal para proyectos personales o académicos.

Ejemplo:

- En Notion puedes crear una tabla con tareas del backlog, agregar campos como prioridad, sprint asignado y estado.
- Puedes filtrar solo por tareas del sprint actual y ver qué falta por hacer.

Scrum no depende de la herramienta, pero estas aplicaciones ayudan mucho a organizar y visualizar el trabajo.

Scrum es un marco potente para organizar proyectos de desarrollo web, ya sean personales, académicos o profesionales. No se trata de seguir reglas rígidas, sino de adoptar una forma de trabajo colaborativa, iterativa y centrada en entregar valor continuamente.

5. Planificación ágil en desarrollo web

Cuando hablamos de desarrollo web, solemos imaginar código, frameworks, servidores y pantallas llenas de diseño. Pero hay una parte fundamental que a veces se olvida y que es igual de importante: saber cómo organizar el trabajo de forma eficiente. Especialmente cuando trabajamos en equipo o dentro de un proyecto con fechas de entrega claras, necesitamos una forma de planificarnos que sea flexible y adaptable. Ahí es donde entra la planificación ágil.



No necesitas ser experto en metodologías como Scrum o Kanban para comenzar. En esta sección vamos a hablar de herramientas sencillas y prácticas para que puedas organizar tu trabajo como desarrolladora web desde una perspectiva ágil. Lo veremos paso a paso, con ejemplos reales y explicaciones claras, desde cómo estimar lo que vas a hacer hasta cómo dividir tareas específicas del análisis de datos.

5.1 ¿Cómo estimar tareas y esfuerzo cuando aún estás aprendiendo?

Cuando estás empezando en desarrollo web, es normal que te cueste saber cuánto tiempo te va a tomar una tarea. Quizá te piden "implementar un login con autenticación" y tú piensas: "¿Eso cuánto se demora? ¿Una hora, un día, una semana?". Y no es por falta de ganas: es porque aún estás conociendo las herramientas, los pasos, y te estás enfrentando a retos nuevos.

La estimación es un arte que se mejora con la práctica, pero incluso desde el inicio puedes hacerlo con sentido común y algunos trucos.

Empecemos con algo simple: el "desglose"

Una buena estrategia para estimar mejor es dividir la tarea grande en tareas más pequeñas. Por ejemplo:

Tarea grande: Construir el módulo de autenticación de usuarios.

- Diseñar la interfaz de login y registro.
- Implementar validaciones en el frontend (correo válido, contraseña segura).
- Crear endpoints en el backend para registrar e iniciar sesión.
- Conectar la base de datos para almacenar usuarios.
- Probar el flujo completo.

Ahora sí puedes preguntarte: ¿cuánto me demoraría en diseñar la interfaz? ¿Y en conectar la base de datos? Así, puedes ir sumando pedacitos y tendrás una mejor idea del total.



Consejo:

Cuando no sabes cuánto vas a tardar, usa rangos. Por ejemplo: “Creo que esto me tomará entre 1 y 2 horas”. O mejor aún, habla en términos de dificultad: “Esto lo veo fácil”, “esto va a requerir investigar”, “esto nunca lo he hecho”. Así tu equipo, tu líder técnico o tu tutor también puede ayudarte a ajustar expectativas.

5.2 ¿Qué es una historia de usuario y cómo escribirla en desarrollo web?

En metodologías ágiles usamos un recurso llamado **historia de usuario**. Una historia de usuario es una frase sencilla que explica qué necesita una persona (o usuario) del sistema o del análisis que estás haciendo.

Estructura típica de una historia de usuario:

Como [rol del usuario], quiero [acción o necesidad], para [beneficio o resultado].

Ejemplo en desarrollo web:

“Como usuario registrado, quiero recuperar mi contraseña por correo electrónico, para no perder acceso a mi cuenta si la olvido.”

¿Ves qué claro? No estamos hablando de código ni de frameworks. Estamos entendiendo qué necesita el usuario y por qué. Esto te guía a ti como desarrolladora para saber qué hacer y cómo priorizarlo.

Consejo:

Cuando hagas un proyecto de desarrollo web para otros (jefes, profesores, clientes), escríbelo como una historia. Te ayudará a mantener el foco en lo que realmente importa: **el valor que entregas al usuario final.**

5.3 División de tareas técnicas

Cuando se trabaja en un equipo pequeño de desarrollo web, **cada parte del proyecto puede ser abordada por diferentes personas**, y la manera en que se reparten estas tareas es fundamental para que el trabajo avance de forma coordinada, efectiva y con buena calidad. No se trata solo de saber programar: se trata de **colaborar**, de saber



cuándo intervenir, de escuchar lo que el otro hizo y de alinearse con los objetivos comunes del proyecto.

Antes de dividir tareas... ¿Hay alguien que coordine?

En equipos pequeños (3 a 6 personas), puede que no haya un "jefe" formal, pero sí es muy útil definir ciertos roles, aunque sea de manera rotativa. Uno de esos roles es la figura de un líder técnico.

Este toma el papel de líder, alguien que:

- Da contexto sobre lo que se espera del sprint o entrega.
- Ayuda a repartir tareas según habilidades.
- Resuelve dudas técnicas o guía la implementación.
- Se asegura de que todos estén alineados en la dirección del trabajo.

Esta figura puede rotar entre los miembros en cada sprint o entrega. Incluso en un entorno académico, fomenta la responsabilidad y la comunicación.

División de tareas técnicas en un equipo de desarrollo web (con roles y habilidades)

Ahora sí, veamos cómo podríamos dividir las tareas técnicas en un equipo real. Para cada etapa, indicamos qué perfil puede encargarse, qué habilidades requiere y cómo se relaciona con el resto del equipo.

Ejemplo: Desarrollo del módulo Login

Diseño UX

Responsable sugerido: Diseñador UX/UI o frontend con perfil creativo.

Responsabilidades:

- Crear el prototipo del formulario de inicio de sesión (desktop y mobile).
- Definir la experiencia de usuario (mensajes de error claros, recordar sesión, accesibilidad).

Fortalezas:

- Manejo de herramientas como Figma, Sketch o Adobe XD.



- Conocimiento en usabilidad y diseño responsivo.
- Sensibilidad para crear interfaces claras y amigables.

Coordinación:

- Se reúne con el PO o cliente para definir campos necesarios (correo, contraseña, recordar sesión).
- Entrega prototipo a desarrollo frontend y valida que sea fiel al diseño.

Ejemplo:

- Diseña en Figma la vista de login con dos campos (correo y contraseña) más botón de acceso, en versión mobile y desktop, compartiendo el archivo editable.

Base de datos

Responsable sugerido: Desarrollador backend con perfil de base de datos.

Responsabilidades:

- Crear la tabla users con campos clave: id, email, password_hash, rol, fecha_creación.
- Definir índices para mejorar rendimiento (ejemplo: índice en email).

Fortalezas:

- Conocimiento en modelado relacional (MySQL, PostgreSQL) o NoSQL (MongoDB).
- Buen criterio de seguridad (hash de contraseñas, no guardar texto plano).
- Habilidad para estructurar datos pensando en escalabilidad.

Coordinación:

- Requiere información del desarrollo backend sobre qué datos exactos debe guardar y deben ser obligatorios (ej. ¿roles?, ¿tokens de recuperación?).



- Confirma con desarrollo frontend qué campos se capturan en el login y registro.

Ejemplo:

- Crea el esquema en SQL, documenta estructura y lo documenta en el repositorio.

Desarrollo Backend

Responsable sugerido: Desarrollador backend.

Responsabilidades:

- Implementar endpoint /login que reciba email y contraseña.
- Validar credenciales contra la base de datos.
- Generar token JWT para sesiones seguras.

Fortalezas:

- Manejo de frameworks backend (Node.js con Express, Django, Spring Boot).
- Conocimientos de autenticación y seguridad (hash, JWT, OAuth).
- Capacidad para escribir código modular y reutilizable.

Coordinación:

- Recibe esquema de BD y lo usa en las consultas.
- Confirma con desarrollo frontend cómo debe responder la API (mensaje de error, token JWT, datos básicos del usuario).

Ejemplo:

- Implementa un POST /login que devuelve un token JWT si las credenciales son correctas.



Desarrollo Frontend

Responsable sugerido: Desarrollador frontend.

Responsabilidades:

- Construir el formulario de login con validación básica en cliente.
- Conectar el formulario al endpoint /login.
- Mostrar mensajes de error (usuario inexistente, contraseña incorrecta).

Fortalezas:

- Dominio de frameworks como React, Angular o Vue.
- Conocimientos de HTML5, CSS3 y JavaScript moderno.
- Experiencia en consumo de APIs REST y manejo de estados.
- Conocimientos de herramientas para diseño responsive

Coordinación:

- Usa prototipo de UX/UI como guía.
- Se comunica con desarrollo backend para validar qué datos enviar/recibir.

Ejemplo:

- Implementa la vista con React, conecta el formulario al endpoint /login y muestra mensajes de error.

QA / Testing

Responsable sugerido: QA Tester o dev con rol rotativo de pruebas.

Responsabilidades:

- Probar flujo de login en diferentes escenarios: usuario válido, usuario inexistente, contraseña incorrecta, campos vacíos.
- Validar que token JWT se genere y guarde correctamente en cliente.

Fortalezas:



- Conocimientos de pruebas funcionales y unitarias.
- Manejo de herramientas como Jest, Cypress o Postman.
- Pensamiento crítico para cubrir casos límite.

Coordinación:

- Reporta errores a desarrollo frontend (por ejemplo, no aparece mensaje de error).
- Informa a desarrollo backend si hay problemas con validación de credenciales.

Ejemplo:

- Detecta que el login acepta espacios en blanco como contraseña y levanta un ticket en Jira o Trello.

Documentación

Responsable sugerido: Rol rotativo dentro del equipo.

Responsabilidades: Documentar uso de la API y el flujo del login.

- Documentar uso del endpoint /login, estructura de la BD y vista de frontend.
- Actualizar README del proyecto con instrucciones claras para devs nuevos.

Fortalezas:

- Redacción técnica clara.
- Conocimiento general del flujo del proyecto (frontend, backend, BD).
- Organización y uso de herramientas como Markdown, Git.
- capacidad de sintetizar y mantener documentación actualizada.

Coordinación:

- Recibe insumos de desarrollo backend (endpoints), BD (estructura) y desarrollo frontend (pantallas).

Ejemplo:



- Actualiza README con: “Para iniciar sesión, enviar POST a /login con email y password. Devuelve token JWT válido por 24h.”

¿Cómo se organiza todo esto en un tablero ágil?

Podrías usar una herramienta como Trello, Notion, ClickUp o Jira, donde cada tarjeta corresponde a una tarea clara, y se asigna a un miembro del equipo con etiquetas como:

- Diseño UX
- Base de Datos
- Desarrollo Backend
- Desarrollo Frontend
- QA/Testing
- Documentación

Así todos saben qué está haciendo quién, en qué orden, y qué entregables se esperan.

En resumen: así se trabaja en equipo en desarrollo.

Etapa	Perfil sugerido	Habilidades clave	Producto esperado
Diseño/UX	UX/UI designer	Figma, usabilidad, diseño responsivo	Prototipo de login
Base de datos	Backend/DB dev	SQL/NoSQL, seguridad, modelado	Tabla users lista
Backend	Backend dev	API REST, seguridad, JWT	Endpoint /login funcional
Frontend	Frontend dev	HTML, CSS, JS, consumo APIs	Formulario conectado
Documentación	Rol rotativo	Redacción técnica, síntesis	README actualizado



Importancia de Git en un equipo de desarrollo web con tareas distribuidas

En un equipo de desarrollo web donde las tareas están divididas entre diseño, frontend, backend, base de datos, pruebas y documentación, Git es una herramienta esencial para garantizar el trabajo colaborativo, organizado y trazable. Su uso permite que cada miembro del equipo trabaje de forma independiente en ramas separadas, sin sobrescribir el trabajo de otros.

A través del versionamiento, es posible rastrear los cambios realizados, entender el porqué de las decisiones, y recuperar versiones anteriores en caso de errores. Además, Git fomenta la transparencia y la colaboración, ya que cada avance queda registrado y puede ser revisado por el equipo. En proyectos bajo metodologías ágiles como Scrum, Git facilita la integración continua, el control de calidad y el cumplimiento de entregables dentro de los sprints.

5.4 Velocidad del equipo y gestión de avances

Una de las preguntas más comunes en un equipo es: ¿vamos bien o nos estamos quedando atrás?

En metodologías ágiles usamos el concepto de velocidad del equipo, que simplemente es una forma de decir: *¿cuántas tareas o cuántos puntos de trabajo podemos hacer por semana?*.

Analogía:

Imagina que tú y tu equipo van en bicicleta por una ruta. La velocidad no es cuán rápido va una sola persona, sino cuánto puede avanzar todo el grupo en una semana. Si un miembro va más lento, todos ajustan. Si uno avanza rápido pero solo, no sirve de mucho. Lo importante es avanzar juntos.

¿Cómo gestionar avances?

- Algunas estrategias:
    - Tableros Kanban: Usa herramientas como Trello, Notion o Jira para ver en qué está cada tarea.
    - Revisiones semanales: Evalúa qué hiciste, qué faltó y por qué.



- Retroalimentación constante: Comparte tus avances, dudas y bloqueos con el grupo.

Así, aunque estés aprendiendo, puedes tener una idea clara de cómo se avanza y cómo mejorar poco a poco.

6. Casos de uso reales: desarrollo web con metodologías ágiles

Hasta ahora hemos explorado cómo trabajar en equipo, cómo dividir tareas, cómo estimar tiempos y cómo aplicar la agilidad a proyectos académicos. Sin embargo, el desarrollo web no se limita a un salón de clases. Lo que estás aprendiendo también es útil para el mundo laboral, para proyectos personales e incluso para emprender.

En esta sección queremos mostrarte cómo aplicar estas herramientas a distintos contextos reales: en un entorno académico, si eres freelance, si trabajas en una empresa, o si simplemente quieres usar tus habilidades para organizar mejor tu vida. La clave aquí es comprender que el desarrollo web y la planificación ágil no son solo técnicas profesionales: son formas de pensar, de estructurar tu trabajo, de resolver problemas y tomar decisiones informadas.

Vamos a construir una simulación realista, paso a paso, como si estuviéramos dentro de un equipo de desarrollo web trabajando en un entorno ágil, educativo pero alineado con prácticas profesionales. Esto nos ayudará a entender cómo fluye un proyecto en equipo, qué tipo de conversaciones se dan y cómo cada rol aporta a una entrega final.

6.1 Proyecto académico ágil: desde el planteamiento hasta la entrega

Un pequeño equipo de estudiantes del curso debe desarrollar un módulo de login para una aplicación web. El objetivo es que el usuario pueda registrarse, iniciar sesión de forma segura y acceder a un panel básico.

Tienen 1 semana para presentar un prototipo funcional ante el docente.



Rol asignado	Fortalezas principales
líder técnico	Organización, pensamiento lógico, visión general
Frontend	Creatividad, diseño responsivo, experiencia en UI
Backend	Lógica de negocio, manejo de APIs, seguridad
Base de datos	Modelado de datos, integridad y consistencia
QA / pruebas	Detalle, pensamiento crítico, capacidad de encontrar errores
Presentación	Síntesis, comunicación con el público

Día 1 – Reunión inicial de planificación

Líder Técnico:

Sugiere comenzar con la creación del formulario de login y definir desde el inicio cómo será el flujo de autenticación. Menciona que el módulo debe incluir registro, login y validación de credenciales con base de datos, además de preparar un entorno de pruebas.

Distribución de tareas inicial:

- Frontend: diseña el formulario de login con campos de email y contraseña.
- Backend: configura el endpoint /login que recibirá las credenciales.
- Base de datos: diseña la tabla users con los campos básicos (id, email, password, created_at).
- QA: pendiente para validar una vez se integre frontend y backend.
- Documentación: abre el README compartido, registra las decisiones y estructura la bitácora del proyecto.
- Presentación: comienza un documento compartido para recoger ideas y capturas del proceso



Día 2 – Primeros avances

Desarrollo Frontend:

Desarrolla el formulario inicial: campos de correo y contraseña + botón de "Iniciar sesión". Sube un commit a GitHub, pero en la revisión descubren que el campo contraseña no enmascaraba los caracteres. El líder técnico abre un issue para que lo corrija antes de integrar.

Desarrollo Backend:

Crea la estructura del proyecto en Node.js y define la ruta /login. Usa express.json() para recibir datos en formato JSON. Descubre que aún no puede probar la validación porque la base de datos no está lista. Deja un endpoint que responde con {status: "API funcionando"}.

Base de Datos:

Crea la tabla users e inserta un usuario de prueba. Al intentar conectarse desde Node.js, detecta que olvidó habilitar las credenciales de red en su PostgreSQL local. Corrige la configuración y documenta el string de conexión.

QA:

Prueba el formulario visual y detecta el bug del campo contraseña en texto plano. Reporta el error en Trello y confirma la corrección tras el fix.

Documentación:
Actualiza el README con instrucciones de instalación y el esquema de la tabla users.

Día 3 – Integración Inicial

Desarrollo Frontend:

Conecta el formulario al backend usando fetch("/login"). Sin embargo, en las pruebas obtiene un error CORS. Reporta el problema en Trello.

Desarrollo Backend:

Resuelve el error agregando cors() a la configuración de Express. Ahora recibe correctamente el email y password desde el frontend.



Prueba con el usuario de prueba de la BD y descubre que el login siempre falla. Tras investigar, encuentra que olvidó normalizar los correos (comparaba Test@example.com con test@example.com). Agrega una función toLowerCase() antes de verificar.

Base de Datos:

Sugiere poner una restricción en la tabla para guardar todos los correos en minúscula, evitando duplicados con mayúsculas/minúsculas.

QA:

Hace la primera prueba de integración. Envía credenciales correctas e incorrectas, detecta que la validación fallaba con correos en mayúscula y confirma la solución tras la corrección.

Documentación:
Registra el bug de CORS y su solución. Actualiza las instrucciones de la API en el README.

Día 4 – Validación real de usuarios

Desarrollo Backend:

Implementa la validación de contraseñas con bcrypt.compare(). Al inicio siempre devolvía "usuario o contraseña incorrectos", porque estaba comparando la contraseña en texto plano contra el hash equivocado (tomaba la contraseña como email). Corrige el error y logra que el login funcione con el usuario de prueba.

Desarrollo Frontend:

Añade validación en el cliente: si el usuario deja el campo vacío, muestra mensajes de error antes de enviar al backend. Integra alertas dinámicas: "Correo o contraseña inválidos" en caso de error.

Base de Datos:

Inserta 3 usuarios adicionales para pruebas. Reporta que el login con pepito@example.com funciona correctamente.

QA:

Ejecuta pruebas funcionales completas:



- *Credenciales correctas → login exitoso.*
- *Contraseña incorrecta → mensaje claro.*
- *Usuario inexistente → mensaje correcto.*
- *Campos vacíos → validación en frontend activa.*

Documentación:
Registra la estructura final de la tabla y anota los endpoints del backend con ejemplos de request/response.

Presentación:

Empieza a armar capturas de pantalla: formulario en frontend, error en consola por CORS y su solución, login exitoso.

Día 5 – Mejoras de seguridad

Desarrollo Backend:

Implementa JWT para devolver un token al iniciar sesión. Documenta cómo incluirlo en el header Authorization. Reporta que al principio olvidó firmar el token con una clave secreta única (usaba "1234"), lo cual fue corregido para mayor seguridad.

Desarrollo Frontend:

Ajusta el código para guardar el token en localStorage. Descubre que olvidó limpiar el campo contraseña después de enviar el formulario. Lo corrige y sube un nuevo commit.

Base de datos:

Agrega un índice único en el campo email para evitar duplicados. Reporta que al probar con un registro duplicado, la API devolvía un error 500. El backend maneja el caso y devuelve un error claro: "El correo ya está registrado".

QA:

Valida que el token funcione correctamente, que caduque como corresponde y que los mensajes de error sean entendibles. Detecta un bug donde el backend seguía aceptando tokens expirados → backend lo corrige.



Documentación:

Incluye un diagrama de flujo simple mostrando:

Formulario → API /login → Validación → JWT.

Día 6 – Ensayo de presentación

Todo el equipo se reúne por videollamada y revisan:

- ¿El login funciona correctamente en escritorio y móvil?
- ¿Los mensajes de error son claros para el usuario final?
- ¿Los tokens generados son válidos y expiran como corresponde?
- ¿La base de datos refleja correctamente los registros y evita duplicados?
- ¿La documentación incluye bugs detectados y sus soluciones?

Desarrollo Frontend:

Muestra el formulario en acción. Reporta que en dispositivos móviles el botón se veía desalineado, así que añade un ajuste en CSS flex.

Desarrollo Backend:

Explica cómo funciona la ruta /login y demuestra que se devuelve un JWT válido.

Base de datos:

Muestra la tabla users y los registros de prueba.

QA:

Hace una regresión completa: prueba login correcto, incorrecto, duplicado, token válido y token expirado. Todo pasa tras las correcciones.

Documentación:
Presenta el README actualizado con instrucciones claras para correr el proyecto: instalación, dependencias, pruebas locales.



Presentación:

Organiza la demo final en diapositivas:

- Problema inicial (necesidad de login)
- Flujo de trabajo por roles
- Errores encontrados y soluciones
- Resultado final (login funcionando con token)

Día 7 – Entrega final

El equipo presenta ante el docente y compañeros:

- Explican cómo dividieron las tareas.
- Muestran el formulario funcionando, con login exitoso y errores controlados.
- Discuten problemas encontrados (CORS, validación de contraseñas, duplicados en BD).
- Destacan que Git y Trello permitieron organizar y documentar todo el proceso.
- QA presenta un resumen de los bugs encontrados (5 en total, 2 críticos y 3 menores) y cómo fueron resueltos.

Aprendizajes clave de esta simulación

- El líder guía, no manda. Se organizó el trabajo, pero cada decisión se discutió en grupo.
- QA no es un adorno: su intervención en los momentos justos permitió detectar fallos críticos (CORS, token expirado, contraseñas mal validadas) antes de la entrega final.
- La coordinación fue real: cuando Backend olvidó normalizar correos, BD y QA empujaron la corrección; cuando Frontend olvidó limpiar contraseñas, QA lo expuso y se resolvió.



- La documentación fue un hilo conductor: cada bug y cada decisión quedó registrado en el README, asegurando trazabilidad.
- La reunión del Día 6 no fue un repaso superficial, fue un filtro riguroso: QA guió la revisión final asegurando que ningún error pasara a producción.
- El equipo aprendió que en desarrollo web los problemas no son teóricos: aparecen en pruebas reales y se resuelven colaborando en el momento.
- Git y Trello no fueron simples herramientas, fueron el soporte que evitó caos y pérdidas de trabajo.

6.2 Proyecto freelance o emprendimiento de desarrollo

Imagina que un pequeño negocio local —una cafetería, una tienda de ropa o una librería independiente— te contrata como freelance para desarrollar su página web. Quieren un sitio sencillo donde mostrar productos, horarios de atención, ubicación y un formulario de contacto.

Como freelance, tú no tienes un equipo grande. Pero eso no significa que no puedas aplicar una metodología ágil. Lo que cambia es la escala, no el proceso.

¿Cómo aplicar ágil a tu proyecto freelance?

- Plantea historias de usuario reales

  En lugar de empezar con frases técnicas como “hacer frontend con React”, ponte en los zapatos del cliente.

  “Como dueño de la cafetería, quiero que los clientes vean el menú actualizado en la web, para que sepan qué opciones tienen antes de visitarnos.”

  Esto te permite centrarte en el valor para el cliente, no en los detalles técnicos.



- Divide en sprints semanales

  Un sprint no necesita ser de dos semanas si estás solo. Puedes trabajar en entregas semanales pequeñas:

  - Semana 1: diseño inicial y estructura de navegación.
  - Semana 2: maquetación de páginas principales (home, menú/productos, contacto).
  - Semana 3: integración del formulario de contacto con backend o servicio de correo.
  - Semana 4: pruebas finales, ajustes de diseño responsivo y entrega del sitio publicado.

- Crea tu propio tablero de tareas

  Puedes usar Trello, Notion, o incluso una hoja de cálculo. Lo importante es que visualices lo pendiente, lo que estás haciendo, y lo que ya terminaste. Esto evita que te disperses y ayuda a mantener el enfoque.

- Entrega valor desde el inicio

  Si tienes un avance útil, compártelo de inmediato. Por ejemplo:

  "Ya tengo la sección del menú lista. ¿Quieres que además agreguemos fotos de los productos en la primera entrega?"

  Esto genera confianza y muestra profesionalismo, porque el cliente siente que está participando en la construcción.

Analogía:

Un proyecto freelance en desarrollo web es como hacer un traje a la medida. No estás produciendo ropa en masa para todas las tallas y gustos, como lo haría una gran fábrica, sino que escuchas a una persona específica, tomas medidas, eliges la tela con ella, haces pruebas de ajuste y entregas algo que encaja perfecto con lo que necesita.



6.3 Cómo usar ágil si trabajas como desarrolladora en una empresa

Ahora imagina que ya conseguiste un trabajo como desarrolladora web en una empresa: puede ser una fintech, una ONG, una startup educativa o un hospital. Allí no estás sola, y hay estructuras, prioridades, jefes y equipos técnicos o no técnicos.

Aquí, la agilidad se convierte en tu **forma de coordinar con otros, de comunicarte bien, y de entregar resultados en tiempos cortos sin perder calidad**. No se trata solo de programar o graficar; se trata de ser parte de un flujo de trabajo más amplio.

¿Qué cambia con respecto al entorno académico o freelance?

Aspecto	Proyecto académico	Proyecto freelance	Empresa como analista
Tiempos	Fijos (calendario)	Flexibles	Dependientes del negocio
Comunicación	Interna (entre pares)	Con cliente	Con múltiples áreas
Objetivo principal	Aprender / practicar	Resolver un problema	Impactar decisiones reales
Feedback	Docente	Cliente	Jefe, usuarios, otros equipos

¿Cómo organizarte como desarrolladora ágil en una empresa?

- Participa en reuniones de planificación

  Suelen ser los lunes. Ahí te asignan tareas para la semana (sprint). Si no sabes algo, dilo. Pedir ayuda no es una debilidad, es parte del trabajo ágil.

- Trabaja en tareas claras y pequeñas

  Evita objetivos ambiguos como “hacer el módulo de login”. En su lugar, pide aclaraciones:



¿Debo crear solo el formulario en frontend? ¿O también la validación de credenciales en backend? ¿El flujo incluye recuperación de contraseña?

- Entrega incrementos, no soluciones finales

  Por ejemplo, primero muestra un prototipo estático del formulario antes de conectarlo a la base de datos. Así recibes feedback temprano y evitas tener que rehacer gran parte del código más adelante.

- Documenta todo para que tu equipo entienda

  En empresas, la comunicación escrita es clave. Anota en el README o en la herramienta de gestión qué hiciste y por qué lo hiciste de esa manera: "Agregué validación en frontend para evitar llamadas innecesarias al backend". Esto le ahorra tiempo a todos y evita malentendidos.

Analogía:

Ser desarrolladora web en una empresa es como ser arquitecta de una casa que muchas personas van a habitar. No solo diseñas paredes y puertas (código), sino que debes asegurarte de que cada cambio tenga sentido para quienes usarán el espacio (usuarios) y que encaje con lo que construyen los demás (equipo). Si trabajas aislada, la casa queda desordenada; si construyes en coordinación, el resultado es habitable y útil para todos.

6.4 Adaptar lo aprendido a proyectos de vida personal

Hasta ahora hemos visto aplicaciones "serias" de las metodologías ágiles: trabajos, estudios, emprendimientos. Pero ¿y si te dijera que también puedes aplicar Scrum o Kanban a tu vida diaria? No necesitas un equipo de desarrollo ni un cliente para aprovechar la lógica ágil. Puedes usarla para organizar tus estudios, tus finanzas, tus hábitos o tus metas personales, con mayor claridad, constancia y enfoque.

Ejemplos:

Planificación de estudios (con Scrum)



- Historia de usuario: *"Como estudiante, quiero repasar tres veces cada tema antes del examen para sentirme preparada."*
- Sprint semanal: seleccionas 2 temas clave para abordar cada semana.
- Backlog de tareas: leer teoría, hacer ejercicios, explicar el tema a alguien.
- Revisión del Sprint: ¿qué lograste?, ¿qué quedó pendiente?, ¿qué puedes mejorar para la próxima semana?
- Retrospectiva personal: ajustar técnicas de estudio o tiempo dedicado según lo que funcionó o no.

Organización financiera (con Kanban)

- Columnas del tablero: *"Por hacer"*, *"En curso"*, *"Hecho"*.
- Tarjetas: registrar gastos, analizar presupuestos, buscar formas de ahorrar.
- Flujo continuo: ir moviendo las tareas según las vayas completando, manteniendo el enfoque en evitar el multitasking excesivo.
- Revisión semanal o mensual: detectar cuellos de botella (por ejemplo, tareas que nunca avanzan) y tomar decisiones informadas.

Rutina saludable (con Scrum o Kanban)

- Historias personales: *"Como persona que quiere mejorar su salud, quiero mantener una rutina diaria de ejercicio y descanso para sentirme con más energía."*
- Sprint: fijar metas semanales (por ejemplo, caminar 5 días, dormir mínimo 7 horas, tomar 2 litros de agua).
- Tareas concretas: preparar ropa de ejercicio, dejar agua visible, establecer alarmas para dormir a tiempo.
- Daily personal (reflexión diaria rápida): ¿qué hice hoy?, ¿qué podría mejorar mañana?
- Tablero visual o app de seguimiento: mantener tu progreso visible.



Analogía:

Aplicar metodologías ágiles a tu vida personal es como convertirte en el Product Owner de tu propio bienestar. Puedes priorizar lo que más valor tiene para ti, ajustar lo que no funciona y celebrar avances medibles. Es una forma de trabajar contigo mismo de manera flexible, enfocada y realista, adaptándote a los cambios y manteniendo la motivación.

7. Herramientas digitales para la gestión ágil

En la vida real, trabajar con metodologías ágiles no significa pegar post-its de colores en una pared y moverlos con la mano (aunque eso también se puede hacer si estás en un espacio físico). En entornos digitales —como clases virtuales, proyectos freelance o trabajos remotos— necesitamos **herramientas que nos ayuden a organizar, colaborar y visualizar el progreso del trabajo**.

Estas herramientas no reemplazan la comunicación ni la disciplina, pero **sí te dan un soporte visual y estructural** para trabajar de forma ordenada y efectiva. En esta sección exploraremos algunas de las más populares: Trello, Notion, Miro y FigJam. Además, te mostraremos cómo usarlas en proyectos reales de análisis de datos, con ejemplos paso a paso y recomendaciones prácticas.

7.1 Trello: estructura básica y tableros ejemplo para proyectos de desarrollo

Trello es una de las herramientas más sencillas y visuales para empezar a trabajar con gestión ágil. Usa el formato de **tablero Kanban**, que organiza las tareas en columnas que representan el estado del trabajo. Es como tener una pizarra virtual donde cada tarjeta es una tarea, y tú puedes moverla de una columna a otra según su avance.

¿Cómo funciona Trello?

Un tablero de Trello se compone de tres elementos básicos:

- **Listas** (columnas): por ejemplo, *Por hacer*, *En progreso*, *Hecho*.
- **Tarjetas** (tareas): por ejemplo, *Diseñar formulario de login*, *Crear endpoint /login*, *Configurar base de datos*.



- **Etiquetas y checklists**: para categorizar o dividir una tarjeta en subtareas.

Ejemplo: proyecto de desarrollo del módulo de login

Imaginemos que tú y tu equipo están construyendo un módulo de login para una aplicación web. Así podría verse el tablero:

Estructura del tablero

Lista (columna)	Ejemplo de tarjeta
Por hacer	"Diseñar formulario de login (frontend)"
En progreso	"Implementar validación de contraseñas con bcrypt (backend)"
En revisión	"Prueba de login con base de datos real"
Hecho	"Flujo de login completo con JWT"
Ideas y backlog	"Agregar recuperación de contraseña con email"

Ejemplo de tarjeta real en Trello

Título de la tarjeta:	
	Diseñar formulario de login
Descripción:	
	Crear un formulario responsive con campos de correo y contraseña. Debe incluir validación básica en frontend y mensajes de error en caso de campos vacíos.
Checklist de tareas:	
☐	Crear campo de correo con validación de formato.
☐	Crear campo de contraseña enmascarado.
☐	Agregar botón de "Iniciar sesión".
☐	Manejar errores: campos vacíos - alerta visual.
☐	Probar en versión móvil.



Etiquetas (categorías):	
	Frontend
	Responsivo
	UI/UX
Miembro asignado:	
	Fulano

Analogía:

Piensa en Trello como una cocina donde cada receta (tarea) se mueve desde "pendiente", a "en preparación", hasta que finalmente está "servida". Tú decides el orden, los pasos y quién cocina qué.

7.2 Notion: gestión personal y de equipo

Notion es una herramienta mucho más flexible y poderosa. Es como un cuaderno digital infinito donde puedes escribir, planear, conectar información, insertar bases de datos, hojas de cálculo, checklists, y hasta incrustar código. Puedes usarlo tú sola o compartirlo con tu equipo.

¿Por qué usar Notion en proyectos de datos?

- Documentar pasos técnicos y decisiones: por ejemplo, explicar por qué usaste JWT y no cookies de sesión.
- Registrar acuerdos de equipo: definir tareas, responsables y fechas.
- Crear dashboards de avance: para ver cómo progresa cada módulo (frontend, backend, base de datos, QA)..

Ejemplo: diario de desarrollo personal

Imagina que estás trabajando sola en un módulo de registro y login. Tu página de Notion podría verse así:



Proyecto: Módulo de Login – Aplicación Web

Fase 1: Frontend

- Creé formulario responsive con campos de correo y contraseña.

- Validación en cliente: campos obligatorios + formato de email.

Fase 2: Base de Datos

- Tabla users con campos: id, email, password_hash, created_at.

- Índice único en email para evitar duplicados.

Fase 3: Backend

- Endpoint /login con Express.

- Implementación de bcrypt para validar contraseñas.

- Error inicial: comparaba contraseña en texto plano con hash - corregido.

Fase 4: Seguridad

- Implementé JWT para manejar sesiones.

- Token guardado en localStorage del navegador.

También puedes insertar tablas de Notion, diagramas de arquitectura o fragmentos de código para tener todo documentado en un solo lugar, incluso hacer un “to-do” con casillas:

- [X] Diseñar formulario de login
- [X] Crear endpoint /login
- [X] Conectar BD y validar usuarios
- [X] Implementar hash con bcrypt
- [X] Generar JWT y guardar en localStorage
- [ ] Agregar recuperación de contraseña por correo

Analogía:

Usar Notion en desarrollo web es como tener un taller digital todo en uno: la libreta donde apuntas ideas, el tablero donde organizas tareas, el pizarrón donde dibujas



esquemas y hasta la estantería donde guardas el código. Todo queda en un mismo lugar, ordenado y accesible para ti o para tu equipo.

7.3 Miro y FigJam para colaborar y mapear ideas

A veces lo que necesitas no es una lista de tareas, sino una manera de pensar en voz alta con tu equipo, mapear ideas, esquematizar procesos o visualizar la arquitectura de tu aplicación. Para eso están Miro y FigJam, dos herramientas de pizarra colaborativa online.

¿Cuándo usar Miro o FigJam?

- Al inicio del proyecto, para organizar ideas.
- Para hacer lluvia de ideas (brainstorming).
- Para construir mapas mentales o diagramas de flujo.
- Para dibujar la arquitectura del sistema (frontend, backend, base de datos, APIs).

Ejemplo: lluvia de ideas para un módulo de login

Imagina que el proyecto es diseñar un sistema de autenticación para una aplicación web. Usando Miro, el equipo arma este esquema inicial:

```
                Módulo Login / Registro
                    /       |       \
            Horarios    Métodos   Distracciones
               |           |           |
            Mañana      Lectura     TikTok
            Noche       Resúmenes   Series
```

Luego conectan cada rama con preguntas:

- ¿Qué validaciones debe tener el formulario?
- ¿Cómo vamos a guardar las contraseñas (hash, salt)?



- ¿Qué pasa si el usuario olvida la contraseña?
- ¿Dónde se almacenará el token: cookies o localStorage?

Puedes mover, agrupar, conectar y reorganizar visualmente, como si tuvieras una pizarra gigante que todos pueden editar al mismo tiempo.

Analogía:

Miro y FigJam son como pizarras mágicas. No importa si tu equipo está en distintos países: todos pueden escribir, dibujar y conectar ideas juntos en tiempo real.

7.4 Plantillas útiles para proyectos de desarrollo web

A veces no queremos empezar desde cero. Por eso existen **plantillas predeterminadas**: estructuras que puedes copiar, adaptar y usar como base para tus propios proyectos.

Aquí te comparto ejemplos prácticos que puedes replicar o ajustar según el tipo de proyecto:

Plantilla básica de tablero ágil para Trello

- **Listas**:
  - Backlog (ideas por hacer)
  - Por hacer
  - En proceso
  - En revisión
  - Hecho
- **Tarjetas ejemplo**:
  - *Crear estructura base con React / Angular / Vue*
  - *Implementar endpoint POST /auth/login en Node.js o Spring*
  - *Configurar Dockerfile para el proyecto*
  - *Implementar pruebas unitarias con Jest*



Plantilla de Notion para gestión personal de proyectos

Página principal

Proyecto: Plataforma Web de Gestión de Usuarios

Tareas

- ¨ Configurar entorno (Node, Angular, dependencias)
- ¨ Diseñar wireframes iniciales
- ¨ Crear módulo de autenticación
- ¨ Implementar CRUD de usuarios

Base de datos de tareas

Tarea	Responsable	Fecha límite	Estado
Configurar base de datos	Fulano	07/11	En proceso
Pruebas de API con Postman	Pepa	12/11	Pendiente

Plantilla Miro o FigJam para planificación

- Mapa mental con las siguientes ramas:
  - Requerimientos del producto
  - Arquitectura del sistema
  - Diseño de interfaz (UI/UX)
  - Dependencias y herramientas
  - Preguntas del cliente o del Product Owner
- Cajas con colores para tareas, personas, dudas, decisiones.



Conclusión

Trabajar en equipo o de forma autónoma con metodologías ágiles es mucho más sencillo si cuentas con herramientas digitales que **te permitan visualizar, organizar, colaborar y documentar**. No importa si prefieres lo visual (Miro), lo estructurado (Trello) o lo flexible (Notion), lo importante es que puedas llevar a la práctica lo que planificas. Y estas herramientas están diseñadas justamente para eso: ayudarte a que tus ideas se conviertan en resultados.

8. Buenas prácticas y errores comunes al aplicar agilidad

Uno de los aprendizajes más importantes al comenzar a trabajar con metodologías ágiles no está en saber usar Trello o escribir una historia de usuario perfecta, sino en **aprender a pensar y colaborar de manera flexible y adaptativa**. Esto suena muy bonito, pero en la práctica, lo que más ocurre al principio son confusiones, sobreesfuerzo, parálisis por perfeccionismo o frustración porque el equipo no avanza. Y eso está bien. La agilidad se entrena haciendo, errando, corrigiendo.

En esta sección hablaremos de **los errores más comunes y las buenas prácticas que pueden ayudarte a evitarlos o al menos, a recuperarte con más facilidad si ocurren**. No son reglas estrictas ni fórmulas mágicas, sino consejos construidos a partir de lo que muchos equipos (incluso profesionales) suelen enfrentar en sus primeros pasos.

8.1 Evitar la sobreplanificación y aprender a iterar

Cuando un equipo está empezando, especialmente en entornos educativos o proyectos de formación, hay una tendencia natural a querer planearlo todo desde el inicio: cada paso, cada tarea, cada variable del análisis de datos. Esto parece "organizado", pero irónicamente, puede volverse una trampa. La **sobreplanificación** se parece a querer escribir todo un libro antes de escribir la primera página.

En agilidad, en vez de tratar de predecirlo todo, aprendemos a **iterar**: trabajar en ciclos cortos, probar cosas pequeñas, y mejorar paso a paso. Piensa en cómo aprendiste a andar en bicicleta: no fue leyendo un manual completo ni planeando una ruta larga. Fue intentando, tambaleándote, corrigiendo el equilibrio, y avanzando de a poco.



Ejemplo aplicado al desarrollo web

En vez de decir: “Vamos a construir toda la plataforma con sistema de autenticación, dashboard, roles, notificaciones, pasarela de pago y panel administrativo. ", una iteración podría enfocarse solo en:

- “Crear la estructura base del proyecto (frontend y backend).”
Luego:
- “Implementar el módulo de autenticación básico (login y registro).”
Después:
- “Construir la primera vista funcional del usuario (ej: perfil).”

Buena práctica:

Planifica lo justo para comenzar, pero no más. Luego, revisa lo que hiciste, aprende, y adapta tu siguiente paso.

8.2 Perfeccionismo vs. entregas incrementales

Hay una frase muy usada en agilidad que dice: *“Hecho es mejor que perfecto.”*

En un entorno académico o de aprendizaje, muchas veces el perfeccionismo nace del miedo a “hacerlo mal” o a “no saber lo suficiente”. Pero en proyectos ágiles, lo que buscamos es **hacer avances visibles y útiles, aunque no sean perfectos**, para poder corregir rápido.

Analogía:

Imagina que estás armando una escultura de plastilina. Si tratas de dejar perfecta la cabeza antes de tener el cuerpo, podrías darte cuenta tarde de que el cuerpo no soporta el peso, o que las proporciones están mal. En cambio, si haces una versión simple de todo, aunque sea tosca, podes mejorar las partes poco a poco. Eso es una **entrega incremental**.

Ejemplo aplicado al desarrollo web:

En vez de pasar una semana tratando de dejar el diseño “impecable”, los estilos milimétricos y la arquitectura completa de la app lista desde el inicio, es mejor:



- Crear una versión funcional del home (aunque sea con estilos básicos).
- Implementar un formulario sin validaciones avanzadas todavía.
- Conectar una API con datos de prueba.
- Mostrar al equipo lo que encontraste.
- Iterar con mejoras.

Así, si algo no tiene sentido o hay que cambiar de enfoque, lo sabes temprano.

Buena práctica:

Entrega versiones funcionales, aunque incompletas, y mejóralas en equipo.

8.3 Cuando el equipo no colabora: cómo adaptarse y seguir avanzando

Este es uno de los desafíos más reales y menos hablados. En entornos reales (y académicos también), **a veces hay miembros del equipo que no cumplen, se desconectan o no colaboran como se esperaba**. La metodología ágil no es una varita mágica que soluciona eso, pero sí ofrece herramientas para adaptarse sin que el proyecto se detenga.

Analogía

Piensa en una banda de música. Si el bajista no ensaya, el grupo puede hacer dos cosas:

- Quedarse esperando, sin tocar nada.
- Ajustar el repertorio, cubrir esa parte entre todos, y seguir practicando.

En agilidad, se elige la segunda opción. No es ideal, pero es **realista y resiliente**.

Estrategias prácticas para estudiantes o proyectos académicos

- Redefinir el alcance si el equipo se reduce.
- Dividir tareas de forma flexible: si falta alguien, otro puede asumir una parte mínima de su rol.



- Documentar todo en espacios compartidos (Notion, Drive, Trello) para que cualquier persona pueda retomar donde otro dejó.
- Hacer reuniones cortas y frecuentes para mantener claridad.

Buena práctica:

No esperes a que todo funcione perfecto para avanzar. Aprende a adaptarte, y seguí el movimiento con lo que tengas.

8.4 Ágil no significa "sin estructura"

Uno de los malentendidos más comunes entre quienes recién empiezan con metodologías ágiles es pensar que "ágil" es sinónimo de "hacerlo como salga" o "sin planificación". Y esto es peligroso. La agilidad **sí tiene estructura**, pero es flexible. No te dice exactamente qué hacer, sino que te da un marco para organizarte y adaptarte rápido.

Analogía:
Imagina que estás armando una tienda de campaña. Puedes moverla, orientarla distinto, plegarla rápido si llueve. Pero si no tienes varillas ni puntos de anclaje, no tienes tienda. La agilidad funciona igual:

- Las **reuniones breves** (como dailies o retrospectivas) son puntos de anclaje.
- Los **tableros de tareas** son la estructura.
- Las **iteraciones** son la forma de moverte sin perder control.

Errores comunes por falta de estructura:

- No saber quién está haciendo qué.
- Acumular tareas sin prioridades.
- Terminar sin producto claro.

Buena práctica:

Define tus herramientas, tus espacios de trabajo y tus tiempos. Después, dale libertad al equipo para moverse dentro de esa estructura.



Buena práctica	Error común	Recomendación
Planificar lo necesario para empezar, y ajustar en el camino.	Esperar tener el plan perfecto antes de comenzar.	Sin acción, no hay nada que ajustar ni mejorar.
Entregar versiones simples e ir mejorando.	Buscar la perfección desde el inicio sin mostrar avances.	Avances pequeños permiten retroalimentación útil.
Redistribuir tareas si alguien no cumple.	Detenerse esperando que todos trabajen igual.	La flexibilidad es clave, pero con responsabilidad.
Usar herramientas visuales para organizarse.	Confiar solo en la memoria o el chat.	Visualizar el trabajo mejora la coordinación.
Reuniones cortas y periódicas para revisar.	Dejar pasar mucho tiempo sin revisar el proyecto.	Las reuniones bien usadas impulsan el equipo.
Definir roles o acuerdos sobre tareas.	Falta de claridad: todos hacen todo o nadie asume.	La claridad evita conflictos y retrasos.
Avanzar en pasos: explorar, modelar, presentar.	Querer modelar desde el primer día sin preparación.	Un buen proyecto crece paso a paso.
Documentar avances y decisiones.	No registrar nada y olvidar los motivos.	Documentar facilita aprender y mostrar resultados.
Saber que lo ágil también tiene estructura.	Pensar que ágil es improvisar sin control.	La estructura bien usada da libertad real.



9. Otras metodologías ágiles que debes conocer

Hasta ahora hemos hablado principalmente de Scrum porque es el marco de trabajo ágil más conocido y aplicado. Sin embargo, en el mundo real no todo es Scrum. Existen otras metodologías y enfoques ágiles que vale la pena conocer, porque te pueden ayudar a ajustar tu forma de trabajar según el tipo de proyecto, el tamaño de tu equipo o incluso el tipo de producto que estás desarrollando.

Aprender sobre estas otras metodologías te da más herramientas para adaptar lo aprendido a contextos variados. Aquí no se trata de elegir una sola "metodología perfecta", sino de tener un abanico de opciones que puedas combinar, adaptar y usar con sentido común. Vamos a explorar cuatro de ellas que son especialmente útiles y que incluso pueden mezclarse con Scrum: XP, Lean, Crystal y Scrumban.

9.1 XP (Extreme Programming): calidad técnica y colaboración

Extreme Programming, o simplemente XP, no tiene que ver con videojuegos ni con nada "extremo" en el sentido deportivo. Se le dice "extreme" porque lleva a un nivel más alto algunas buenas prácticas de programación y trabajo en equipo que ya eran conocidas, pero que a veces se aplicaban a medias.

¿En qué se enfoca XP?

XP está muy centrado en el desarrollo de software, especialmente cuando el equipo está construyendo soluciones técnicas complejas. Sus pilares son:

- Comunicación constante entre equipo y cliente
- Retroalimentación rápida
- Mejora continua
- Simplicidad en el diseño
- Coraje para cambiar cosas si algo no está funcionando

Buenas prácticas de XP que puedes aplicar incluso en análisis de datos:

Aunque XP nació para equipos de software profesional, muchas de sus prácticas se pueden aplicar de inmediato en proyectos web educativos o de formación:



Práctica XP	Adaptación en análisis de datos
Programación en pareja	Dos personas trabajan juntas: una escribe el código y la otra revisa en tiempo real. Útil para frontend, refactorización, diseño de APIs o revisar seguridad.
Pruebas automáticas	Crear pruebas unitarias o de integración (Jest, PHPUnit, JUnit) que verifiquen componentes, endpoints o reglas de negocio.
Integración continua	Subir cambios constantemente al repositorio (GitHub) y ejecutar pruebas automáticamente con GitHub Actions, GitLab CI o Jenkins.
Cliente presente	Validar con frecuencia: mostrar prototipos, endpoints, avances del frontend o flujos de usuario al Product Owner o al cliente.
Simplicidad	Diseñar lo mínimo necesario: no crear 5 microservicios si un solo backend modular lo resuelve, ni agregar 20 dependencias cuando basta con 3.

XP también pone mucha atención en la calidad del código, y en desarrollo web eso significa escribir componentes, servicios y módulos que sean claros, ordenados y fáciles de mantener. No es lo mismo una funcionalidad que solo funciona "una vez y de suerte", que una que cualquier otro desarrollador puede leer, entender, reutilizar o ajustar sin romper el proyecto.

9.2 Lean: elimina el desperdicio, enfócate en lo esencial

Imagina que estás construyendo una plataforma web y dedicas una semana a crear un sistema completo de filtros avanzados… pero al final el cliente solo necesita un buscador simple.
O que inviertes horas configurando una base de datos súper compleja que luego ni siquiera se usa. Eso es desperdicio. Lean es una filosofía que busca precisamente evitar esto: hacer solo lo que agrega valor real.



¿Qué propone Lean?

Lean viene del mundo de la manufactura, pero se ha adaptado a la tecnología con muy buenos resultados. Sus principios clave son:

- Elimina lo que no genera valor
- Aprende rápido a través de ciclos cortos
- Empodera al equipo
- Decide lo más tarde posible, pero actúa lo más rápido posible
- Optimiza el todo, no solo una parte

Lean en un proyecto de desarrollo web:

- Validar primero si las funcionalidades solicitadas son realmente necesarias para el MVP.
- Evitar construir módulos gigantes "por si acaso" sin tener pruebas de que el usuario los necesita.
- Hacer prototipos rápidos (wireframes, mockups o versiones iniciales del frontend) para confirmar si el flujo funciona antes de programarlo completo.
- Tomar decisiones cuando tengas suficiente información técnica o de usuario, no antes (para evitar rehacer) ni después (para no frenar al equipo).

Analogía:
Piensa en Lean como empacar una maleta para un viaje corto. No llevas todo tu clóset "por si acaso". Llevas solo lo que sabes que vas a usar. Lo mismo aplica al análisis: menos cosas, pero bien elegidas.

9.3 Crystal: agilidad adaptativa según tu equipo

Crystal no es una sola metodología, es más bien una familia de métodos. Su creador, Alistair Cockburn, notó algo interesante: cada equipo es diferente, así que no tiene sentido aplicar exactamente el mismo enfoque ágil en todos los contextos.



¿Cómo funciona Crystal?

Crystal propone que el proceso ágil se adapte según:

- El tamaño del equipo
- La criticidad del proyecto (qué tan grave sería un error)
- La comunicación interna
- El nivel de madurez técnica

Hay varios "criterios" de Crystal, por colores. Por ejemplo:

- Crystal Clear: para equipos pequeños (2 a 6 personas), donde se prioriza la conversación directa, la entrega frecuente y poca documentación.
- Crystal Orange: para equipos medianos, donde ya se incorporan más roles y estructuras.

¿Cómo aplicar Crystal en tus proyectos?

En desarrollo web, podrías usar un enfoque Crystal Clear si estás en un equipo pequeño, como el de una hackathon o bootcamp:

- Trabajo cercano, diario.
- Reuniones breves pero frecuentes.
- Entregas pequeñas pero regulares del frontend, backend o prototipos.
- Documentación mínima pero útil (README, instrucciones de instalación, diagramas simples).
- Flujo de trabajo flexible que adaptes según cómo trabaja tu equipo real, no según un manual rígido.

Importante: Crystal te invita a evaluar tu propio equipo y crear tu forma de trabajar en lugar de copiarla de otro lado.



9.4 Combinación de metodologías

¿Se pueden combinar las metodologías? La respuesta es **sí**. En la práctica, muchos equipos combinan lo mejor de cada metodología. Este enfoque híbrido es muy común porque ningún método es perfecto por sí solo.

Algunos enfoques combinados útiles:

Enfoque combinado	¿Qué mezcla?	¿Para qué sirve?
Scrumban	Scrum + Kanban	Cuando necesitas la estructura de Scrum (roles, tiempos) pero la flexibilidad visual de Kanban
Lean Startup	Lean + experimentación + validación de negocio	Ideal para proyectos web tipo MVP o startups. Permite construir solo lo esencial, lanzar rápido, medir uso real y ajustar funcionalidades según el comportamiento del usuario.
Design Thinking + Scrum	Investigación centrada en usuarios + ejecución ágil	Te ayuda a entender primero el problema y luego construir la solución en ciclos rápidos
XP + Scrum	Calidad técnica + estructura de trabajo	Se usa cuando quieres sprints y ceremonias de Scrum, pero necesitas agregar prácticas técnicas fuertes: refactorización continua, pair programming, TDD o integración continua.
Kanban + DevOps	Flujo visual + automatización	Útil cuando el equipo gestiona despliegues constantes, hotfixes y microservicios. Kanban controla el flujo y DevOps automatiza pruebas, builds y deploys.



10. Cómo integrar la agilidad a tu formación y carrera como desarrolladora web

Hasta ahora hemos explorado herramientas, metodologías, prácticas y errores comunes. Pero ser ágil no es solo una forma de trabajar: es también una **forma de aprender, crecer y relacionarse con los desafíos del día a día**.

En este último módulo vamos a reflexionar y practicar cómo aplicar los principios de la agilidad en tu formación como estudiante, en la construcción de tu portafolio, en tu forma de mejorar proyectos pasados y, sobre todo, en cómo ver la agilidad como una cultura que puedes adoptar de manera personal y profesional.

10.1 Aprender con agilidad: autogestión del conocimiento

En la agilidad, uno de los valores más importantes es la capacidad de **aprender constantemente y adaptarse**. En lugar de memorizar todos los temas de un solo golpe, se trata de aprender lo esencial, aplicarlo, recibir retroalimentación y seguir aprendiendo.

¿Qué significa aprender de forma ágil?

Imagina que tu aprendizaje es como un experimento continuo: tú haces una hipótesis (por ejemplo, “creo que si separo este componente en Angular/React el rendimiento mejora”), pruebas, ves si funciona, y luego decides si lo mejoras o si necesitas aprender algo más.

Esto es muy distinto a intentar aprender todo “de una sola sentada” como si fuera un examen final. La agilidad te propone:

- Aprender **por ciclos**: pequeños bloques de conocimiento, aplicados y mejorados.
- Aprender **en función de tus proyectos**: estudiar lo que necesitas resolver ahora.
- Aprender **con curiosidad**, no solo por obligación.



¿Cómo se ve esto en la práctica?

Supongamos que estás aprendiendo a construir interfaces responsivas con CSS o Tailwind. En lugar de ver todas las clases, propiedades o breakpoints en un solo día, podrías:

- Empezar creando un layout simple, por ejemplo, una tarjeta con imagen y texto.
- Usar ese componente dentro de una página real de tu proyecto.
- Mostrarlo a otra persona o a tu instructor y recibir comentarios (muy ancho, poco contraste, mala jerarquía visual).
- Mejorarlo agregando responsive, ajustando tamaños, aplicando mejor semántica o extrayéndolo a un componente reutilizable.

Este proceso es ágil: aprendes, aplicas, recibes retroalimentación, mejoras, vuelves a aplicar.

10.2 Crear tu portafolio de desarrollo de forma ágil

Tu portafolio es la carta de presentación que mostrará lo que sabes hacer. Pero muchas veces, por perfeccionismo, las personas no lo terminan nunca. ¿La solución? Construirlo de forma incremental e iterativa, como un proyecto ágil.

¿Cómo construir un portafolio ágil?

- Crea tu primer caso simple. No necesita ser un sistema enorme. Puedes comenzar con un API REST básica, un servicio con autenticación simple, un microservicio CRUD, o un módulo de integración con una base de datos. Pequeño, funcional y claro.
- Publícalo. Sube tu repositorio a GitHub, agrega un README claro que explique cómo correr el servicio y cuáles son los endpoints y compártelo con tu red o profesores.
- Recibe retroalimentación. ¿Te dijeron que tu manejo de errores puede mejorar? Ajusta. ¿Notaron que la estructura de carpetas es confusa? Refactoriza. ¿Faltaba documentación? Añádela.



- Itera. Puedes actualizar ese mismo proyecto o empezar uno nuevo aplicando lo que aprendiste.
- Documenta tu proceso. Eso también es valor. No muestres solo el resultado final, sino cómo llegaste allí.

Analogía:
Construir tu portafolio ágilmente es como armar una colección de Lego. Puedes empezar con piezas básicas y luego irle agregando módulos, detalles y colores. Lo importante es que esté en evolución constante, no que sea una escultura perfecta desde el principio.

10.3 Iterar tus proyectos para mejorar con retroalimentación

Una de las claves de la agilidad es que nada está terminado del todo: siempre se puede mejorar. Esto no significa que debas estar reescribiendo tus proyectos sin parar, sino que puedes:

- Mejorar la organización de tu código.
- Refactorizar un endpoint que quedó poco claro.
- Agregar documentación más completa en el README de GitHub.
- Corregir errores que notaste después.
- incluso reutilizar partes de un proyecto en otro.

¿Cómo iterar tus proyectos?

Supongamos que construiste una API para la gestión de usuarios en un sistema interno.

- Primera versión (V1):
  - Implementaste un CRUD básico.
  - Conectaste la API a una base de datos.
  - Agregaste validaciones mínimas.
  - Lo subiste a GitHub.



- Retroalimentación recibida:
  - "No hay manejo de errores consistente."
  - "Sería útil agregar paginación para las consultas."
  - "El código del controlador está muy cargado."
- Versión mejorada (V2):
  - Agregaste un middleware o interceptor para manejar errores globales.
  - Implementaste paginación y filtrado en el endpoint de usuarios.
  - Refactorizaste el controlador separando servicios, repositorios y DTOs.
  - Documentaste el cambio en el README.

Ese proceso de **mejorar por ciclos cortos y con sentido** es lo que se conoce como iterar. No necesitas empezar de cero; solo retomar lo que ya hiciste y **hacerlo mejor** con lo aprendido.

10.4 Ágil como cultura de vida: prioriza, adapta, mejora

Más allá de las herramientas y los marcos de trabajo, **la agilidad es una mentalidad**. Y esa mentalidad se puede llevar a tu día a día, incluso fuera del mundo laboral o académico.

¿Qué significa vivir con agilidad?

- **Priorizar:** ¿Qué es lo más importante ahora? ¿En qué vale la pena invertir tiempo?
- **Adaptarte:** Si algo no salió como esperabas, ¿qué puedes cambiar? ¿Qué puedes aprender?
- **Mejorar:** No se trata de ser perfecto, sino de estar en constante crecimiento.

Ejemplo:
Imagina que planeaste estudiar 3 horas al día esta semana. Pero el miércoles te surgió un compromiso familiar. En vez de frustrarte, piensas de forma ágil: ¿Qué puedo



estudiar hoy en menos tiempo? ¿Puedo reprogramar lo que quedó pendiente? ¿Qué es lo más esencial de esta semana?

La agilidad no es rigidez, es ***flexibilidad con propósito****.*

La agilidad no es un tema más en tu formación. Es una forma de avanzar, de crear, de colaborar y de aprender. Te permite **hacer más con lo que tienes**, con menos estrés y más enfoque. No necesitas ser experto en Scrum ni dominar todas las herramientas desde el principio. Solo necesitas empezar con lo esencial, probarlo, mejorar con el tiempo y mantener la mente abierta.

Tu camino como analista de datos no se define únicamente por cuántas herramientas sabes usar, sino por cómo aprendes, cómo comunicas tus resultados y cómo evolucionas con cada proyecto. Y en eso, la agilidad puede ser tu mejor aliada.