Tema 3: Principios del desarrollo dirigido por pruebas.

- Probar todo lo que puede llegar a fallar, utilizando pruebas automatizadas.
- **Principios básicos:**
  - El código se comparte y se puede modificar rápidamente, para ello hay que asegurarse de que no falle. Hay que probar todas las clases.
  - Escribir las pruebas antes que el código. Prueba un poco, codifica un poco. Las pruebas deben mantenerse, no se usar y tirara, y almacenarse con el código fuente.
  - Todo el código que esta en el repositorio debe estar probado, y debe funcionar cuando nos lo descargamos y en cada paso que realicemos debemos ejecutar las pruebas. Y cuando hemos terminado y todo funciona las subimos junto al código fuente.
  - Solo se publica código que ha superado todas las pruebas. Eso aumenta la percepción de seguridad.
- La unidad básica para probar es el **método** y se llaman **Pruebas unitarias.**
- **Niveles de pruebas de software:**
  - **Pruebas unitarias:** Prueban las clases y métodos. Verifican la unidad mas pequeña de software, el método. XUNIT
    - El nombre debe recordar a la clase que se va a probar.
    - Primero escribir la prueba, y si el código falla, corregir el código fuente y repetir la prueba. Cuando se superan las pruebas se puede registrar el código y las pruebas.
    - **Tipo de pruebas**:
      - **Funcionales o Caja negra**: No se conoce la estructura que quiere probar. Se centra en las entradas y salidas.
      - **Estructurales o Caja blanca**: Se conocen la estructura y se pueden probar todos los caminos. Se centra en la estructura interna.
  - **Pruebas de integración:** Probar la relación entre las clases. XUNIT y Maven.
    - Primero se escribe los casos de prueba de nuevas funcionalidades a desarrollar.
    - El código no la supera todavía, porque no esta escrito, y debemos escribirlo teniendo una idea precisa del código funcional. El código debe ser el mas simple posible que permita superar las pruebas codificadas.
    - El código se refactoriza para que cumpla las reglas y recomendaciones del estándar.
    - Tras comprobar que todo funciona se puede publicar el código con las pruebas.
    - **Estrategias**:
      - **Top-Down**: Se empieza por el mas complejo y se continua con las que dependen de el. Se desciende por la jerarquía.
      - **Bottom-Up**: Empieza por la clase base, y va subiendo en las que dependen de el.
  - **Pruebas de sistema:** Formalización y automatización de casos de pruebas.
  - **Pruebas de aceptación:** Las que se llevan al cliente a aceptar el código.
- **Dificultades y recomendaciones:**
  - Cuesta el cambio de cultura, cuando no se esta acostumbrado ha escribir primero las pruebas ente que el código.
  - Necesidad de cambio en las rutina del equipo.
  - Trabajar el código en pequeños incrementos, que pueden resolverse en poco tiempo.
- **Beneficios**:
  - Permite centrarse en los requisitos que se debe satisfacer antes de empezar a escribir el código.

        - Mantener el código simple y fácil de probar entender y modificar, ya que esta dividido en pequeños pasos con sus propias pruebas.
        - Proporciona documentación acerca de como funciona el sistema que estamos intenso desarrollar y que se encuentra registrado en el código fuente.
    - LEER PREGUNTAS FRECUENTES EN TEMA 3.

    - **Hay herramientas de grabación y reproducción** que graban una secuencia de pasos en la interfaz de usuario y determina los resultados que se deben conseguir después de cada paso. Para cada paso hay que definir lo que debe encontrar, y cuando en un paso no se cumple ha fallado la prueba.
    - **Hay herramientas para ejecutar pruebas de sistema** como JUnit, que tiene un entorno que permite automatizarlas, pero también se pueden hacer mediante ficheros o hojas de cálculos con los resultados esperados y un programa que los lea y haga la prueba para cada entrada/