APUNTE DE PRUEBAS

- Hay pruebas centradas en la **VERIFICACIÓN** y otras centradas en la **VALIDACIÓN**.

- La verificación es para controlar que el producto esté construido como pretendíamos.
  La validación es para controlar que hayamos construído el producto que el cliente quería.

- Hay pruebas centradas en probar funcionalidades (ej.: Si intento colocar una ficha en una celda ocupada, el programa debe impedirlo). Estas son **PRUEBAS FUNCIONALES**.
  Pero a veces hay que probar características del sistema que no son funcionales (ej.: El tiempo de respuesta del juego no puede ser superior a 1 segundo). Estas son **PRUEBAS DE ATRIBUTOS DE CALIDAD**.

- Algunos tipos de pruebas de atributos de calidad son:

  - Pruebas de compatibilidad: chequean diferentes configuraciones de hardware o de red y de plataformas de software que debe soportar el producto.

  - Pruebas de rendimiento (en condiciones de uso habitual)

  - Pruebas de resistencia o de estrés: comprueban comportamiento del sistema ante demandas extremas de recursos.

  - Pruebas de seguridad

  - Pruebas de recuperación (luego de una falla)

  - Pruebas de instalación

- Las pruebas de verificación se pueden clasificar en pruebas **unitarias** o de **integración**:

  - Las unitarias verifican pequeñas porciones de código. En POO, por ejemplo, verifican alguna responsabilidad única de un método.

  - Las de integración prueban que varias porciones de código en conjunto hagan lo que pretendíamos. En POO, por ejemplo, son pruebas que

involucran varios métodos, clases o subsistemas enteros.

  - Ambos tipos pueden ser automatizadas con herramientas disponibles para cada lenguaje/plataforma que suelen ser los frameworks xUnit.

- Las pruebas de verificación se suelen escribir antes del código que prueban.

- Hay ocasiones en que no es fácil hacer una prueba unitaria porque queremos probar código que necesita de otros objetos, métodos o funciones para funcionar. En estas situaciones se usan objetos ficticios para aislar el código a probar.

- Las pruebas de verificación pueden ser de **caja negra** o de **caja blanca**:

  - Se dice que es de caja negra cuando la ejecutamos sin mirar el código que estamos probando.

  - Cuando analizamos el código durante la prueba, decimos que es de caja blanca. (El debugging es una técnica de caja blanca)

  - Una de caja blanca que ya no se usa tanto es la llamada **prueba de escritorio**, en donde el programador da valores a ciertas variables y recorre mentalmente el código (ayudándose con anotaciones en papel), para ver que se comporte como él espera.

- Para las pruebas de validación, se suele trabajar con pruebas de aceptación diseñadas por usuarios, pero que ejecuta el equipo de desarrollo. Estas pruebas se suelen denominar **pruebas de aceptación de usuarios (UAT)** y se deberían ejecutar en un entorno lo más parecido posible al que usará el usuario.

- Si las pruebas las hacemos en un entorno controlado por el equipo de desarrollo, se llaman **pruebas alfa**. Si el producto se deja a disposición del cliente para que lo pruebe en su entorno, se llaman **pruebas beta**.

- Cuando se prueba solo comportamiento, sin interfaz de usuario, se llaman **pruebas de comportamiento**. Hay muchas herramientas para automatizar pruebas de comportamiento, con formatos diferentes, pero que pretenden resultar amigables para los usuarios no técnicos.

- Se puede también poner el sistema en producción y esperar a ver qué problemas encuentran los usuarios. A esto lo llamamos realizar las pruebas en producción.

- Hay distintos niveles de prueba, pero las que más deben abundar son las unitarias y las que menos las de aceptación a través de la interfaz de usuario, ya que las unitarias son las más veloces y menos costosas.
  No obstante, en todo sistema bien hecho, debería haber pruebas de todos los niveles (unitarias, de integración técnicas, de comportamiento, de interfaz de usuario)

- Dentro de la organización del desarrollo, la **visión tradicional** sostiene que debe haber personas con el rol de programadores y otras con el rol de testers. Los testers ejecutan las pruebas y velan porque el producto llegue sin errores al cliente.
  La calidad es introducida por los programadores, quienes en primera instancia son los responsables de que el programa no tenga errores.

VISION ÁGIL

- Varios métodos agiles de desarrollo de software plantearon que todo el equipo de desarrollo debe trabajar para entregar un producto de calidad y sin errores teniendo todos esta responsabilidad. (programadores y testers).
  Lo que se destaca en los métodos agiles es que, aún cuando haya roles diferentes de programadores y testers, deben trabajar juntos, al menos para que el programador entienda la perspectiva del tester, aprenda a programar código más fácil de probar y el tester pueda aprender a automatizar pruebas.
  Pero el concepto clave es que esta es materia que debe definir el equipo hacia adentro, no alguien externo.

PRUEBAS AUTOMATIZADAS: ¿Quién las desarrolla?

- Las pruebas unitarias y de integración técnicas son pruebas de programador.

- Las pruebas de comportamiento están a mitad de camino entre pruebas de programadores y pruebas de testers.

- Con las UAT automatizadas se pretende que las escriban usuarios o analistas de negocio. Si no se consiguiera esto, las podrían escribir los testers.

PRUEBAS MANUALES: ¿Quién diseña la prueba?

- Hoy en día se suelen dejar como pruebas manuales algunos tipos de UAT que luego ejecutarán los testers y clientes. Estas pruebas no son pruebas de

programadores y no deberían escribirlas ellos mismos. En la visión tradicional podrían ser desarrolladas por analistas o los mismos testers. En la ágil, habría que dar la mayor participación posible a los usuarios.

¿Cuándo probamos?

- El primer ciclo de vida del desarrollo de software suponía que las pruebas se ejecutaban al final de un desarrollo de proyecto, antes de ponerlo en producción. Pero en la práctica no funcionaba así ya que se podía probar una funcionalidad a medida que se la terminaba de desarrollar.

- En 1990, se incorporó la idea de probar al final de cada iteración.

- En los 2000, los métodos ágiles, abogaron por la prueba más o menos continua, automatizando todo lo posible.

- Hoy en día las pruebas son continuas a lo largo de todo el desarrollo.

- Para evitar que al agregar una nueva funcionalidad, se rompan cosas que ya habían sido probadas (**regresión**), se ejecutan cada tanto, **pruebas de regresión**, que ejecuta las pruebas de todo el sistema a intervalos regulares.

Ventajas de la automatización

- Nos independizamos del factor humano.

- Es más fácil repetir las mismas pruebas a un costo ínfimo comparado con las pruebas realizadas por una persona.

- Las pruebas en código sirven como herramienta de comunicación, minimizando las ambigüedades.

TDD

- Incluye tres sub-prácticas:

    - **Automatización**: las pruebas deben ser hechas en código, y con solo ejecutar el código de las pruebas debe saber si lo probado funciona bien

o mal.

    - **Test-First**: las pruebas se escriben antes del código a probar.

    - **Refactorización posterior**: para mantener la calidad del código, se lo cambia sin cambiar la funcionalidad, manteniendo las pruebas como reaseguro.

- Las ventajas del TDD son:

    - Las pruebas en código sirven como documentación del uso esperado de lo que se prueba.

    - Las pruebas escritas con anterioridad ayudan a entender mejor lo que se va a desarrollar.

    - Las pruebas escritas con anterioridad suelen incluir más casos de pruebas negativas que las que escribimos a posteriori.

    - Escribir las pruebas antes del código a probar minimiza el condicionamiento del autor por lo ya construido.

    - Escribir las pruebas antes del código a probar permite especificar el comportamiento sin restringirse a una única implementación.

    - La automatización permite independizarse del factor humano y facilita la repetición de las pruebas a menor costo.

    - La refactorización constante facilita el mantenimiento de un buen diseño a pesar de los cambios que, en caso de no hacerla, lo degradarían.

Integración y entregas continuas

- La idea es facilitar y automatizar al máximo las integraciones antes de liberar el producto. Consiste en compilar, costruir y probar el producto en forma sucesiva y automática como parte de la integración.

- Como una derivación de la **integración continua (CI**) surgió la práctica de **entrega continua (CD)**, que pretende que el código siempre esté en condiciones de ser desplegado en el ambiente productivo.

- Finalmente, hay una técnica denominada **despliegue continuo** que pretende que cada pequeño cambio se despliegue en el ambiente de producción.

- Con estas prácticas se disminuye la aparición de errores en las pruebas, porque cualquier problema que surja es atribuible al último tramo de código desarrollado e integrado.

Diseño de pruebas unitarias y técnicas en general

- Una primera forma de probar es ponernos en el rol de quien usa el módulo a probar. Lo que podríamos hacer es seguir las nociones del diseño por contrato: establecer pre y postcondiciones. Con las pre podemos obtener casos de excepción y con las post las posibles salidas. Cada post debería al menos definir una prueba.

Diseño de pruebas de cliente

- Si la prueba debe chequear un requisito del cliente es bueno usar la técnica de especificar con ejemplos, en donde el rol del programador suele ser el de quien puede prever las clases de equivalencia y valores especiales, y pide ejemplos para cada caso. En términos de requerimientos de software, al menos tendría que haber una prueba para el escenario típico, una para cada flujo de excepción y una para cada flujo alternativo.

COBERTURA

- Llamamos cobertura al grado en que los casos de prueba de un programa llegan a recorrer dicho programa al ejecutar las pruebas.

- A mayor cobertura, las pruebas son más exhaustivas, y por lo tanto existen menos situaciones no probadas.

- El nivel de cobertura no debe guiar el desarrollo. Solo mide la calidad de las pruebas.

- Que una línea esté siendo cubierta por un conjunto de pruebas no implica que se estén analizando todos los casos posibles.

- Una buena idea es usar las métricas de cobertura para observar tendencias.

- Un objetivo razonable es llegar a un 80% o 90% de cobertura en métricas tales como las de sentencias, ramas y combinada de ramas y condiciones.

- Un análisis serio de cobertura debería partir por hallar partes que no están siendo recorridas, analizar la criticidad de esas partes y, si lo amerita, agregar pruebas para cubrirlas.