lo que creo es mas importante

Nexus (1)
Lean kanban (3)
Métricas (7)
Testing (11)

**(si encuentran algo mal o medio medio avisenme al wp)**

falta
esta
hace mención

Temas que se evalúan en el segundo parcial

**Unidad 2: Gestión Lean Agile de Productos de Software**

- Framework para escalar SCRUM / Nexus 71 del largo
- Filosofía Lean - Método Kanban en el contexto del desarrollo de software lean: 11 - 13 kanban: 14 - 17
- Métricas en los 3 enfoques (Tradicional / Lean / Agile) 21 - 25

**Unidad 3: Gestión de software como producto**

- Prácticas continuas 102 del largo

**Unidad 4: Aseguramiento de Calidad de Proceso y de Producto**

- Calidad de Producto: 26-33 7 3-8
  - Planificación de pruebas para el software- Niveles y tipos de pruebas para el software.
  - Técnicas y herramientas para probar software. 5-7
  - Testing agile 9-12
  - Técnicas y Herramientas para la realización de revisiones técnicas del software. 25-33

Temas que no van al parcial 2

1. Modelos de calidad
2. Auditorias de software
3. Proceso de auditorias
4. Aseguramiento de la calidad de software
5. Conceptos generales sobre calidad

# PARCIAL 2 ISW

## Unidad 2: Gestión Lean Agile de Productos de Software

### Framework para escalar SCRUM / Nexus

**Framework Nexus para escalar SCRUM:** Scrum necesita ser escalado para proyectos grandes. Nexus es un framework diseñado para coordinar entre **3 y 9 equipos Scrum** que trabajan en un mismo producto y generar un incremento integrado al final de cada sprint.

- Todos los equipos comparten un **único Product Backlog** y colaboran para entregar **un solo incremento integrado** al final del Sprint.
- Hay un **único Product Owner** que prioriza el trabajo y mantiene la visión del producto.

Tener varios equipos Scrum trabajando sobre un mismo Product Backlog genera algunos desafíos que Nexus resuelve:

- **Requerimientos compartidos:** a veces distintos equipos deben trabajar sobre partes similares del producto. Si no se coordinan bien, pueden pisarse o generar conflictos entre funcionalidades.

- **Conocimiento del negocio:** no todos los equipos conocen todo el sistema. Por eso, es importante distribuir ese conocimiento de forma inteligente para que cada equipo pueda avanzar sin depender constantemente de otros.

**Responsabilidades (Roles) de Nexus:**

1. **Nexus integration team:** equipo responsable de asegurar que **al menos una vez en cada Sprint, se produzca un incremento integrado del producto**. Se encargan de identificar y abordar restricciones técnicas y no técnicas que puedan limitar la capacidad de los equipos para entregar un incremento funcional. Se encarga de mantener la integración continua entre los equipos. Suele incluir de **3 a 9 miembros**.

2. **Product owner:** tiene la responsabilidad de **maximizar el valor del producto y del trabajo** realizado por los **diferentes equipos Scrum** que conforman el Nexus. Además, es quien gestiona eficazmente el **Product Backlog**, asegurando que esté priorizado, actualizado y alineado con los objetivos de negocio.

3. **Scrum Master:** líder de servicio que apoya al equipo, resuelve impedimentos y busca asegurar los recursos. No es un coordinador. **Es responsable de la efectividad del Scrum Team** al permitir que el equipo mejore sus prácticas de Scrum.

4. **Developers:** son las **personas técnicas** del equipo Scrum que se comprometen a crear cualquier aspecto de un incremento funcional en cada Sprint.

**Artefactos de Nexus:**

1. **Product Backlog:** es **único** para todo el Nexus y está compuesto por **elementos de producto (no tareas)**, conocidos como **PBI (Product Backlog Items)**. Entre ellos se incluyen: User Stories (US), Spikes, Themes, Epics, defectos, historias técnicas (Tech Stories) - que suelen representar RNF y no estimables, así como también deuda técnica, por ejemplo, el uso de hardcodeo.

2. **Nexus Sprint Backlog:** Es un **artefacto compartido** que se crea durante la **Nexus Sprint Planning**. Combina el **objetivo del Sprint del Nexus** con los **elementos del Product Backlog** seleccionados por cada **Scrum Team**. Se **actualiza continuamente** a lo largo del Sprint, reflejando el progreso y los ajustes realizados por los equipos.

3. **Integrated Increment:** es el resultado final del trabajo combinado de todos los Scrum Team dentro del Nexus. Representa la **suma de todos los incrementos integrados** y debe ser funcional, de valor y utilizable. **Se revisa en la Nexus Sprint Review**, aunque puede entregarse antes si ya está terminado. Debe estar asociado a una DoD.

**Eventos de Nexus:**

1. **Refinamiento entre equipos (Cross-Team Refinement)**: es una actividad continua en donde el **Product Backlog se descompone y analiza**. Este refinamiento cumple dos funciones principales:

- Definir qué equipo desarrollará cada ítem del Product Backlog.
- Detectar y resolver dependencias entre equipos.

2. **Nexus Sprint Planning**: busca **coordinar** las actividades de **todos los equipos** Scrum dentro del **mismo Sprint**. En esta reunión participan el **Product Owner y representantes de cada equipo**, quienes colaboran para planificar de manera conjunta el trabajo a realizar. Los resultados son:

    - Un objetivo Sprint para cada team.
    - Un objetivo Nexus, que representa la meta compartida del Sprint.
    - Un Nexus Sprint Backlog.
    - Un Sprint backlog para cada equipo.

3. **Nexus Daily Scrum:** busca **revisar el progreso** hacia el objetivo del Nexus y detectar problemas de integración entre los equipos. Asisten **solo representantes de cada equipo Scrum** y se centra en los problemas de integración y coordinación.

4. **Nexus Sprint Review:** se realiza **al finalizar el Sprint** para inspeccionar el incremento integrado desarrollado y obtener **retroalimentación** del cliente o stakeholders. Este evento reemplaza las revisiones individuales de equipo.

5. **Nexus Sprint Retrospective:** el objetivo es **identificar y planificar mejoras** de todo el marco de trabajo. Se inspeccionan aspectos como colaboración entre equipos, Procesos, herramientas y flujos de trabajo o DoD.

## Filosofía Lean - Método Kanban en el contexto del desarrollo de software

**¿Qué es LEAN?:** es una filosofía que busca **maximizar el valor** generado al cliente con el **mínimo uso de recursos**, argumentando que todo el esfuerzo que no cree valor, se considera desperdicio.

**Principios LEAN:**

1. **Eliminar desperdicios:** se busca detectar y eliminar todo lo que **no aporta valor al cliente** como por ejemplo:

    - Producir funcionalidades que luego no se usan.
    - Retrabajo.
    - Producir artefactos que se vuelven obsoletos antes de terminarlos.

2. **Amplificar aprendizaje:** se basa en poder **aprender a partir del trabajo** y que ese conocimiento sea **accesible a toda** la organización

3. **Embeber la integridad conceptual:** mantener la **coherencia y calidad** del producto desde el inicio, asegurando que todas sus partes encajen de forma consistente. Implica diseñar una **estructura clara, coherente y fácilmente mantenible**, que preserve la uniformidad del sistema a lo largo de su desarrollo.

4. **Diferir compromisos:** posponer las decisiones importantes hasta el **último momento** responsable, con el fin de disponer de la **mayor cantidad de información** posible antes de comprometer recursos o definir una dirección.

5. **Dar poder al equipo:** confiar en los equipos de trabajo y darles autonomía para realizar las actividades.

6. **Ver el todo:** analizar el sistema completo para optimizar el flujo de valor de extremo a extremo. Tener una visión holística del conjunto (producto, valor agregado y el servicio del producto).

7. **Entregar lo antes posible:** liberar versiones tempranas del producto para obtener retroalimentación rápida por parte del cliente.

**¿Cuáles son los tipos de desperdicio?**

**Tipo 1 - Desperdicio necesario:** actividad que **no agrega valor directamente al cliente**, **pero es necesaria** en el proceso actual ya que son tareas que no se pueden evitar, aunque se busca minimizarlas lo máximo posible. Ejemplo: actividades de supervisión y de control.

**Tipo 2 - Puro desperdicio:** es una actividad que **no agrega valor y tampoco es necesaria**. Se busca eliminarlas por completo.

**¿Cuáles son los desperdicios según LEAN?**

1. **Defectos:** son errores que se **trasladan de una etapa a otra** posterior en la cual se introdujo, provocando retrabajo, debido a que Testing debe “devolver” la funcionalidad a desarrollo para la resolución.

2. **Talento no utilizado**: no aprovechar las ideas, conocimiento o capacidades del equipo de trabajo.

3. **Esperas:** tiempo muerto que una persona tiene que pasar para poder realizar su trabajo, puede ser causado por aprobaciones, dependencias o falta de insumos.

4. **Movimientos:** cambios constantes de contexto o de tareas que dispersan la atención de los equipos.

5. **Transporte:** transferencia de información entre equipos o herramientas que generan demoras o errores.

6. **Procesos extra:** pasos innecesarios que no agregan valor al producto, se busca identificarlos y eliminarlos.

7. **Sobreproducción:** desarrollar más funcionalidades de las que el cliente necesita o pidió.

8. **Inventario:** trabajo pendiente (a medias) o código no utilizado.

**¿Qué es Kanban?:** es un **método de gestión visual** que permite organizar, controlar y optimizar el **flujo de trabajo**, especialmente en servicios profesionales o actividades creativas y de diseño.

**Principios de Kanban**

1. **Gestión de cambios:** Kanban promueve la **evolución gradual** en lugar de cambios drásticos. El proceso se mejora de forma continua, limitando el trabajo en curso y ajustando el flujo según la capacidad del equipo. WIP

2. **Entrega de servicios:** se centra en **cumplir compromisos** de forma **predecible y continua**, gestionando el flujo de trabajo como un servicio que agrega valor al cliente. Se busca optimizar el tiempo de entrega y mantener la calidad.

**Prácticas de Kanban y principios LEAN asociados:**

1. **Limitar el trabajo en curso:** evitar que haya demasiadas tareas a medio hacer o en progreso para evitar cuellos de botellas permitiendo **eliminar desperdicios** ya que se evita la sobrecarga, multitarea y espera.

2. **Hacer políticas explícitas:** definir claramente las reglas como DoD, DoR, límite de WIP, o criterios de pull. Esto permite **embeber la integridad conceptual y dar poder al equipo**.

3. **Visualizar el trabajo:** usar el tablero para visualizar el flujo de trabajo y los riesgos. Permite **eliminar desperdicios** identificando fácilmente los cuellos de botella y **ver el todo** (sistema completo).

4. **Establecer ciclos de retroalimentación:** fomentando la mejora y el conocimiento, permite **amplificar el aprendizaje**.

5. **Mejorar colaborativamente, evolucionar experimentalmente:** ejecutar experimentos para avanzar o aprender mediante el uso del método científico. Esto permite **amplificar el aprendizaje.**

6. **Gestionar el flujo**: observar cuán rápido se mueven las tareas y eliminar cuellos de botellas. De esta manera **eliminamos los desperdicios.**

Ejemplo:
1. Análisis: el analista funcional entiende qué necesita el cliente y escribe los requisitos.
2. Diseño: el arquitecto define cómo se va a hacer (dónde va el botón, qué librería usar).
3. Desarrollo: los programadores escriben el código del botón.
4. Revisión técnica: otro desarrollador revisa el código para asegurar calidad y buenas prácticas.
5. Testing: los analistas de prueba verifican que el botón funcione correctamente.
6. Evaluación con el cliente: el cliente prueba la nueva función y confirma que cumple lo que pidió.
7. Despliegue / En producción: se publica la versión final en el sistema que usa el cliente.

**Diseño del tablero Kanban**

EP: En progreso
H: Hecho
L: listo

US
Defecto
Mantenimiento
Petición de cambio

El equipo:
1 Analista Funcional
1 Arquitecto
2 Desarrolladores
2 Analistas de Prueba
1 Responsable de Despliegue

Acuerdos:
- El cliente valida funcionalidades antes de producción
- Revisión técnica al código es parte del "criterio de hecho"

| Cola de tareas | Analisis 1 | | Diseño 2 | | Desarrollo 3 | | Revisión tecnica | | Listo para build | Testing 4 | | Evaluación con el cliente | | Despliegue 5 | | En producción |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | EP | H | EP | H | EP | H | EP | H | | EP | H | EP | L | EP | L | |

**Defina los tipos de trabajo que utilizará, justificando la respuesta y al menos 5 políticas de calidad (pregunta parcial)**

1. **User stories:** se centran en el usuario, son independientes, testeables, estimables, etc.

2. **Defectos:** asegura que el trabajo no planificado se vuelva visible, permitiendo analizar el impacto de los mismos en el flujo de trabajo.

3. **Mantenimiento:** este tipo de trabajo previene fallos o ralentizaciones que pueden resultar muy costosas en el futuro.

4. **Peticiones de cambio:** generalmente son externas por lo que tienen valor directo para el cliente. Su visibilidad permite priorizarlas adecuadamente.

**Ejemplos de Políticas de calidad (pregunta parcial)**

- Las tarjetas solo pueden ser añadidas por el Product Owner o líder de equipo, para asegurar relevancia.

- No se incorporan nuevas tareas al tablero sin descripción o criterios de aceptación.

- Se realiza cada día una TEAM Kanban Meeting de 15 minutos máximo frente al tablero para observar el flujo de trabajo e identificar bloqueos o cuellos de botella y decidir cómo proceder.

- La fase de testing de un producto de trabajo termina al pasar el 80% de tests sin fallas y no hallar defectos críticos, bloqueantes o graves.

- Hay un límite para el WIP, cada columna cuenta con un límite máximo de tareas, y si se supera, el equipo debe priorizar la liberación de los bloqueos.

- Ejemplos que dió en clase: Definition of Ready y Definition of Done.

**Métricas en los 3 enfoques (Tradicional / Lean / Agile)**

**¿Qué son las métricas?** las métricas **son resultados no procesos**, se debe medir lo necesario y nada más.


**1. Métricas en Agile:**

**Principios ágiles que guían la elección de métricas:**

- Nuestra mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuas de software valioso.

- El Software funcionando es la principal medida de progreso

**Métricas:**

1. **Velocidad:** mide la **cantidad de story points** completados y aceptados por el **Product Owner** en un Sprint. Solo se cuentan las historias terminadas. Permite evaluar la estabilidad del equipo y ajustar futuras estimaciones.

2. **Capacidad:** indica cuánto trabajo **puede asumir** el equipo en un Sprint. Se estima durante el **Sprint Planning** y puede medirse en **horas ideales o story points**. Sirve para planificar cuántas historias se tomarán del Product Backlog.

3. **RTF:** mide la **cantidad** de funcionalidades testeadas y funcionando. No considera la complejidad de cada una, solo cuántas están operativas. Si el valor no crece o baja, puede indicar problemas en el avance del proyecto.

**2. Métricas en Lean:**

**Métricas:**

1. **Lead Time (vista del cliente):** mide el tiempo total desde que el cliente **solicita** una tarea hasta que se **entrega**. Refleja la percepción del cliente sobre la rapidez del equipo y es la métrica más importante desde su punto de vista.

2. **Cycle Time (vista interna):** mide el tiempo desde que el equipo **comienza a trabajar** en una tarea hasta que la finaliza, sin contar el tiempo que estuvo en espera dentro del backlog. Sirve para analizar la eficiencia interna del equipo.

3. **Touch Time:** indica el tiempo efectivo en que una tarea está siendo **trabajada activamente**, excluyendo los períodos de espera o inactividad. Ayuda a detectar cuellos de botella y mejorar el flujo de trabajo.

4. **Eficiencia del ciclo de proceso:** se calcula como "Touch time / Lead time". Si el resultado es cercano a 1, significa que la mayor parte del tiempo la tarea estuvo siendo trabajada activamente. Si el valor es bajo, una pequeña parte del tiempo se trabajó realmente la tarea.

**3. Métricas en Tradicional:**

**Métricas principales:**

1. **Esfuerzo**
2. **Tiempo**
3. **Costo**
4. **Riesgos/Defectos**

**Se agrupan en:**

- **Métricas de proceso:** permite conocer el rendimiento en **términos de la organización** independientemente de un proyecto en específico. Ejemplo: porcentaje de proyectos terminados con éxito, porcentaje de proyectos cancelados.

- **Métricas de proyecto:** permiten saber si un proyecto de SW en ejecución está **cumpliendo o no** de acuerdo a lo **planificado**. Se consolidan para crear métricas de proceso. Ejemplo: variación del calendario real vs planificado.

- **Métricas de producto:** tienen una relación directa con el **producto de software** que estamos construyendo. Ejemplo: tamaño del Producto, cantidad de errores encontrados.

**Suponiendo un desarrollo ágil, analice el siguiente gráfico sobre el comportamiento del proyecto manifestado y explique detalladamente la interpretación que hace luego del análisis (pregunta parcial)**


El burndown Chart permite visualizar el progreso del trabajo restante en un Sprint.

**Línea Azul (Ideal):** representa la trayectoria lineal que debería seguir el equipo para completar exactamente 15 Puntos de Historia para el final del Día 16, a un ritmo constante.

**Línea Naranja (Real):** representa los Story Points que realmente quedan pendiente al final de cada día.

El Burndown Chart del Sprint 7 fue exitoso en su resultado final (cumplimiento del 100% del alcance), pero ineficiente en su flujo de trabajo.

**Dado el siguiente gráfico explique la razón por la cual todo lo que está fuera de coincidencia es considerado un desperdicio (pregunta parcial)**


● **Calidad programada:** alcances de producto planificados.

● **Calidad realizada:** lo que realmente se desarrolló del producto.

● **Calidad necesaria:** mínimas características que el producto debe tener, para satisfacer los requerimientos.

Todo lo que está fuera de la zona central se considera desperdicio porque representa un **recurso, esfuerzo o tiempo** que no se tradujo en el **valor real** que el cliente esperaba, o no se ajustó a lo planificado.

# Unidad 3: Gestión de software como producto

## Prácticas continuas

- **Continuous Integration:** Automatizamos el **nivel de integración**
- **Continuous Delivery:** Automatizamos el **nivel de sistema**
- **Continuous Deployment:** Automatizamos el **nivel de aceptación**

**Prácticas continuas**: gestionar el software como producto implica mantenerlo en constante evolución y mejora, no como algo que se desarrolla una vez y se entrega terminado.

integracion constante

1. **Continuous integration (CI):** tiene el objetivo de mantener el software siempre estable y listo para pruebas de aceptación, reduciendo errores y costos.

automatizacion del proceso de integracion

- Los desarrolladores integran su código varias veces al día en un repositorio compartido.
- Cada integración se verifica automáticamente con pruebas (idealmente automatizadas).
- Se usa TDD para asegurar que el código cumple con el comportamiento esperado.

entrega/puesta en produccion constante

2. **Continuous delivery (CD):** su objetivo es que cada versión sea "entregable" sin contratiempos.

automatizacion del proceso de control despliegue

(con intervencion human

- El software se construye, prueba y valida continuamente para estar siempre listo para producción.
- Se automatizan las pruebas de aceptación y el proceso de despliegue.
- El Product Owner decide cuándo liberar, equilibrando agilidad técnica y necesidades del negocio.
- Requiere tener CI implementado previamente.

3. **Continuous development:**

automatizacion del proceso de despliegue

(sin intervencion humana)

- Las nuevas versiones se despliegan automáticamente a producción sin intervención humana.
- Requiere un pipeline de automatización sólido y confiable.
- Ventaja: si hay errores, el desarrollador puede corregirlos rápidamente porque tiene fresco el contexto.

**Estrategias de Continuous development:**

1. **Canary Deployment:** se lanza la nueva versión a un grupo pequeño de usuarios para detectar fallos antes de expandirla.

2. **Blue-green Deployment:** utiliza dos entornos iguales: uno activo (blue) y otro con la nueva versión lista (green). Cuando el entorno green se prueba y funciona correctamente, se redirige el tráfico hacia él. Si ocurre algún problema, se vuelve al entorno blue de forma inmediata, evitando que los usuarios se vean afectados.

## Unidad 4: Aseguramiento de Calidad de Proceso y de Producto

### Planificación de pruebas para el software - Niveles y tipos de pruebas para el software

**¿Qué son los casos de prueba?:** son datos necesarios de seteo para determinar si se cumplen uno o más criterios de aceptación, se basa en las US, los casos de uso y los requerimientos.

**¿Cuánto consume la fase de implementación (Desarrollo) del sw?:** entre el 33% y 35% del costo del proyecto.

**¿De qué pruebas se encarga el PO y de qué pruebas se encarga el equipo de desarrollo?:** el PO de las UAT (Pruebas de adaptación de usuario) y el equipo de desarrollo de las demás.

**¿Cuál es la diferencia entre error y defecto?:** la diferencia es el **momento** en el cual se detectan y solucionan.

- Un **error** es detectado y corregido en una **misma etapa** mientras que un **defecto** es un error que se **traslada** de una etapa a otra etapa posterior.

- El **testing encuentra defectos**, ya que son errores que surgieron en etapas anteriores, pero se detectan en la etapa de prueba.

- Ambos conceptos pueden o no generar fallas en el sistema. Por ejemplo, un error en los colores de una interfaz no es una falla, ya que no conllevan a un mal funcionamiento del sistema.

**¿Qué es la "severidad" de un defecto, la "prioridad" y cómo se clasifica?:** la **severidad** refleja el **impacto** que genera el defecto en el sistema y no el costo de corregirlo. Mientras que la **prioridad** la define el cliente según la urgencia de la corrección (Alta/Media/Baja).

- **Bloqueante o invalidante:** impide el uso del sistema o de una funcionalidad crítica.
- **Grave:** afecta funciones principales, pero existe alguna alternativa temporal.
- **Leve:** causa errores menores que no interfieren con el funcionamiento general.
- **Cosmética:** afecta solo la apariencia o aspectos visuales de la interfaz.

**¿Qué son los ciclos de prueba del testing?** es la ejecución de un **conjunto de casos de prueba** en una versión determinada del producto.

- Generalmente se tienen 2 ciclos, el primero es conocido como **ciclo 0** y siempre es manual, es donde se configura todo inicialmente y a partir del **ciclo 1** ya se pueden automatizar las pruebas.

- En caso de detectar defectos, el producto vuelve a desarrollo para su corrección, se debe evitar grandes cantidades de ciclos, ya que generan retrabajo.

**¿Qué estrategias de ciclos de prueba existen?**

1. **Con regresión:** se ejecutan exhaustivamente todos los casos de prueba incluyendo nuevos y anteriores para asegurar que las modificaciones no afecten a funcionalidades que ya fueron corregidas.

2. **Sin regresión:** se enfoca únicamente en probar las nuevas funcionalidades y no ejecutar las pruebas anteriores del sistema.

**¿Cuánto testing es suficiente?:** el testing exhaustivo es **imposible** por la cantidad de tiempo que requiere. El momento en que se deja de hacer testing depende del **nivel de riesgo o costo del proyecto**.

- Los riesgos permiten definir prioridades de qué se debe testear primero y con qué esfuerzo.

- El criterio de aceptación para decidir si una determinada fase de testing ha finalizado puede ser definido en términos de:

    - Costos.
    - % de tests corridos sin fallas.
    - Inexistencia de defectos de una determinada severidad.
    - Pasa exitosamente el conjunto de pruebas.
    - Good Enough: cantidad aceptable de fallas no críticas.
    - Defectos detectados es similar a la cantidad de defectos estimados.

**¿Cuáles son los ambientes de testing?:** son todos los recursos de Hw y Sw que se requieren para poder trabajar con el producto. Existen distintos ambientes según la etapa de desarrollo en la que se encuentre el software y las necesidades de la misma.

DPPP

1. **Desarrollo:** implica hardware y software necesario para desarrollar y desplegar el producto. En este ambiente se realizan las **pruebas unitarias**.

2. **Pruebas:** es el ambiente que utilizan los testers para llevar a cabo las pruebas. En este se realizan las **pruebas de sistema y de integración**.

3. **Preproducción:** similar al ambiente de producción, Sirve para poder comprobar que el producto funcionará una vez desplegado en producción de manera correcta. En este ambiente se realizan las **pruebas de aceptación**.

4. **Producción:** es la configuración de software y hardware que tienen los usuarios finales, y que utilizan para sus actividades. En este entorno **no se realizan pruebas.**

**Niveles de prueba:**

1. **Pruebas unitarias (Nivel 1):** verifican **componentes individuales** del sistema de forma aislada, usualmente realizadas por el desarrollador. Son fáciles de automatizar y detectan errores puntuales.

2. **Pruebas de integración (Nivel 2):** evalúan el **funcionamiento conjunto** de varios componentes previamente probados. Se realizan de forma incremental para detectar fallas en la interacción entre módulos.

3. **Pruebas de sistema (Nivel 3):** prueban el **sistema completo**, tanto en sus funciones como en aspectos no funcionales. Se ejecutan en un entorno similar al de producción.

4. **Pruebas de aceptación (Nivel 4):** realizadas por el **cliente o usuario final**. Buscan generar confianza en el sistema más que encontrar defectos, en un entorno lo más real posible.

**Tipos de prueba:**

1. **Smoke test:** verifica que las **funciones básicas** del sistema operen correctamente antes de realizar pruebas más profundas.

2. **Sanity test:** comprueba que los errores corregidos **no generen nuevos problemas** y que las funciones principales sigan operativas.

3. **Pruebas funcionales**: validan que el software cumpla con los **requerimientos** y **procesos de negocio** definidos.

4. **Pruebas no funcionales:** evalúan aspectos como **rendimiento, seguridad, usabilidad o escalabilidad**, asegurando que el sistema funcione correctamente bajo diferentes condiciones.

## Testing agile / Técnicas y herramientas para probar software

**Manifiesto de testing:**

- Pruebas **durante** sobre pruebas al final.
- **Prevenir** defectos sobre encontrar defectos.
- **Entender** lo que se está probando sobre verificar la funcionalidad.
- **Construir** un mejor sistema sobre romperlo.
- **Responsabilidad de todo el equipo** sobre el tester es responsable de la calidad.

**Prácticas concretas de testing agile:**

1. **Pruebas unitarias y de integración automatizadas:** se ejecutan mediante scripts, lo que permite una ejecución rápida y repetible.

2. **Pruebas de Regresión de sistema automatizadas:** se ejecutan para asegurar que los cambios recientes en el código no han **roto** la funcionalidad **ya probada** del software.

3. **Pruebas exploratorias:** pruebas **manuales** que el tester diseña y ejecuta de forma simultánea, sin un plan de prueba formal.

4. **TDD (Test Driven Development):** metodología de desarrollo de software que consiste en **escribir primero las pruebas unitarias antes de implementar el código**. De esta forma, el desarrollo se guía por los casos de prueba, lo que ayuda a entender mejor los requerimientos y a reducir errores.

   <u>El proceso sigue el ciclo Red-Green-Refactor:</u>

   - **Red:** se escribe una prueba que inicialmente **falla**.
   - **Green:** se desarrolla el código mínimo necesario para que la prueba **pase**.
   - **Refactor:** se mejora el código **sin alterar su funcionamiento**.

   <u>Funcionamiento:</u>

   A. **Escribir un test:** si un cliente tiene $1000 y quiere retirar $200, el saldo debe quedar en $800.

   B. **Hacer que el test compile:** creamos una función *retirar_dinero()* que no hace nada.

   C. H**acer que el test falle (red):** como no hay implementación, el test falla.

   D. **Hacer el cambio mínimo para que el test pase:** implementamos la lógica más simple.

   E. **Correr el test para que pase (Green)**

F. **Quitar duplicaciones mediante refactoring:** es una forma disciplinada de introducir cambios reduciendo la posibilidad de introducir defectos sin modificar la funcionalidad del componente. Mejora la calidad y elimina duplicaciones.

5. **ATDD:** práctica similar a TDD, pero centrada en las pruebas de aceptación. El equipo define los **criterios de aceptación** como pruebas antes de la codificación, asegurando que el desarrollo cumpla con las expectativas y necesidades del cliente.

6. **Control de versión de las pruebas con el código:** consiste en almacenar los **scripts de las pruebas junto con el código** fuente de la aplicación en el mismo repositorio.

**Testing en ambiente ágil vs tradicional cascada**

- **En enfoque tradicional:** las pruebas se realizan al **final** del proceso de implementación, una vez que el producto completo ha sido desarrollado. Esto implica que los **defectos** suelen detectarse en **etapas tardías**, lo que puede aumentar costos y tiempos de corregirlos.

- **En un enfoque ágil:** el testing se lleva a cabo en cada **iteración**. Permite detectar y corregir **defectos** de manera **temprana**. Sin embargo, esto puede introducir **nuevos defectos**, lo que exige verificación constante y pruebas de regresión.

**Artefactos del testing:**

1. **Plan de pruebas:** define los **datos, recursos y herramientas** necesarios para llevar a cabo las pruebas. Para la confección de este plan se necesitan los requerimientos. Es la salida de la **etapa de planificación** de testing.

se obtienen de la etapa de planificacion

2. **Casos de prueba:** describen una **secuencia de pasos de entrada** detallados que deben seguirse para comprobar que el software funciona según lo esperado, buscando obtener un resultado concreto y predecible. Una característica esencial es que **deben ser reproducibles**. Son la salida de la **etapa de diseño** de testing.

se obtienen de la etapa de diseño

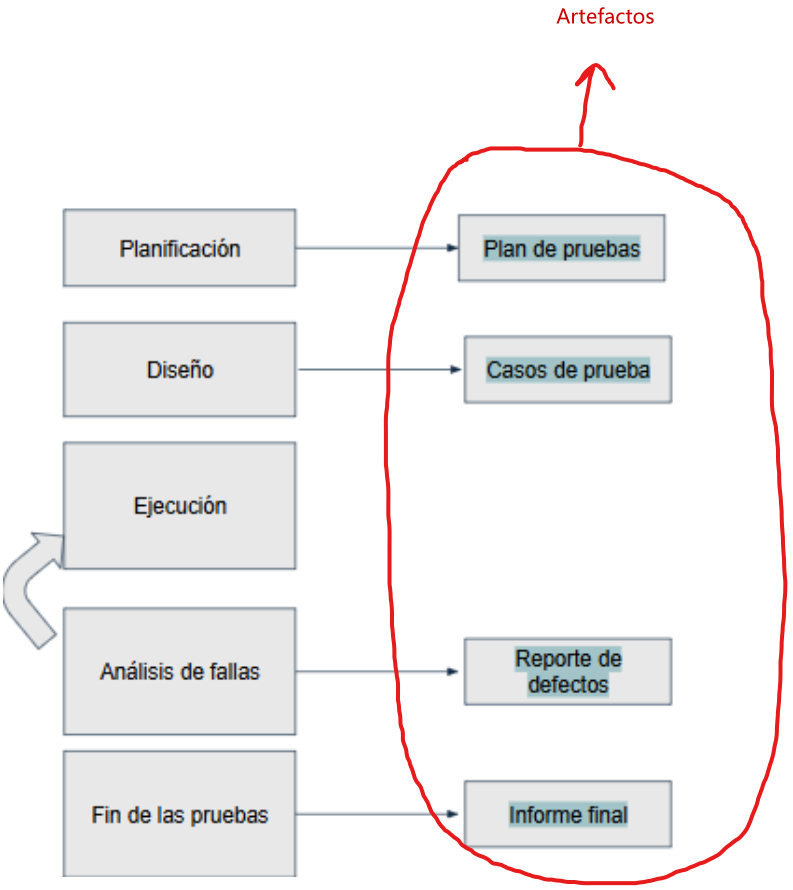
3. **Reporte de defectos:** documento que se elabora una vez finalizada la ejecución de los casos de prueba, detalla qué casos de prueba fueron exitosos y cuáles no, describiendo los defectos y los pasos necesarios para reproducirlos. Este documento es esencial para el equipo de desarrollo. Se genera como salida de la **etapa de ejecución** de testing

se obtienen de la etapa de ejecucion

4. **Informe final:** recopila la información relevante al cierre del ciclo de testing. Incluye métricas e indicadores del incremento evaluado. Ejemplo: número de defectos detectados o tipos de pruebas realizadas, **mide la efectividad del testing**.

se obtiene al fin de las pruebas

**Ciclo del testing:**




**Diferencia entre niveles de testing y niveles de prueba:** nivel de testing abarca un conjunto de niveles de prueba. Cada nivel de prueba corresponde a una unidad específica dentro del testing general. Ejemplo:

- Testing unitario → Prueba unitaria
- Testing de sistema → Prueba de sistema, y así sucesivamente.

**¿Cuál es la diferencia entre validación y verificación?**

¿Satisface/Cumple con las necesidades?

- **Validar:** consiste en comprobar que el SW **satisface las necesidades** de los usuarios, es lo **más caro** de corregir.

¿Se desarrollo el software segun lo planificado/especificado?

- **Verificar:** consiste en comprobar que el software **cumple las especificaciones** y se hizo según lo planificado.

**¿Cuál es la diferencia entre ser ágil y hacer ágil? (pregunta parcial)**

**Hacer ágil:** significa usar frameworks ágiles como Scrum o XP, aplicándolos de forma externa con sus artefactos y reuniones pero sin comprender realmente por qué existen, solo siguiendo las guías.

**Ser ágil:** implica interiorizar los valores y principios del Manifiesto Ágil. No se trata solo de seguir un framework, sino de tener una mentalidad y entender que las prácticas son un medio y no un fin. Cuando un equipo domina la esencia del agilismo puede incluso modificar los frameworks y crear sus propias reglas.

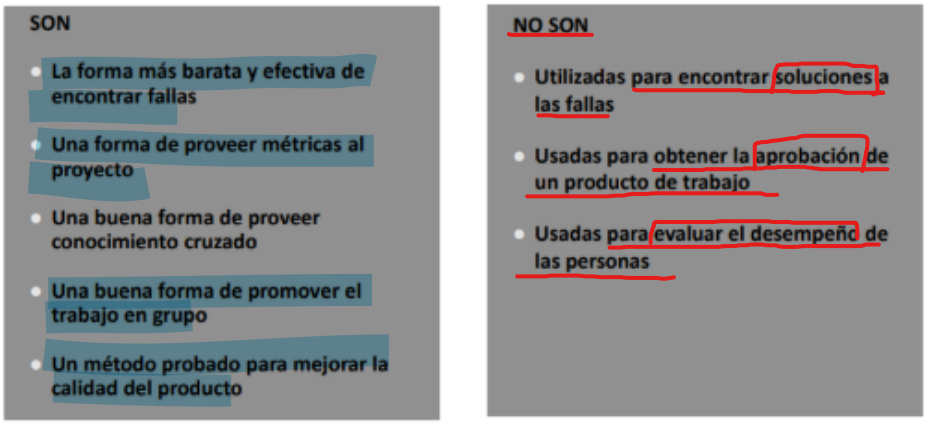
## Técnicas y Herramientas para la realización de revisiones técnicas del software

**Tipos de revisiones técnicas:**

1. **Inspecciones (formales):** son revisiones **estructuradas y planificadas** del software, con **roles** definidos (autor, moderador, lector, anotador, inspector). Se utiliza un **checklist** y se documentan los resultados para detectar defectos y garantizar la calidad del producto.

   Objetivos: detectar errores tempranos, verificar requerimientos, asegurar cumplimiento de estándares y uniformidad en el desarrollo.

| Rol | Responsabilidad |
|---|---|
| Autor | •Creador o encargado de mantener el producto que va a ser inspeccionado.<br>•Inicia el proceso asignando un moderador y designa junto al moderador el resto de los roles<br>•Entrega el producto a ser inspeccionado al moderador.<br>•Reporta el tiempo de retrabajo y el nro. total de defectos al moderador. |
| Moderador | •Planifica y lidera la revisión.<br>•Trabaja junto al autor para seleccionar el resto de los roles.<br>•Entrega el producto a inspeccionar a los inspectores con tiempo (48hs) antes de la reunión.<br>•Coordina la reunión asegurándose que no hay conductas inapropiadas<br>• Hacer seguimiento de los defectos reportados. |
| Lector | Lee el producto a ser inspeccionado. |
| Anotador | Registra los hallazgos de la revisión |
| Inspector | Examina el producto antes de la reunión para encontrar defectos. Registra sus tiempos de preparación. |




2. **Walkthrough (recorrido):** técnica **informal y colaborativa** para revisar código, diseño o documentación. El autor presenta su trabajo y el equipo comenta y sugiere mejoras. **No usa checklist** ni documentación formal.

   Objetivos: mínima sobrecarga, capacitación de desarrolladores y rápido retorno. Común en enfoques ágiles por su simplicidad y trabajo en equipo.