

Fundamentos de Ingeniería de Pruebas y Calidad de Software

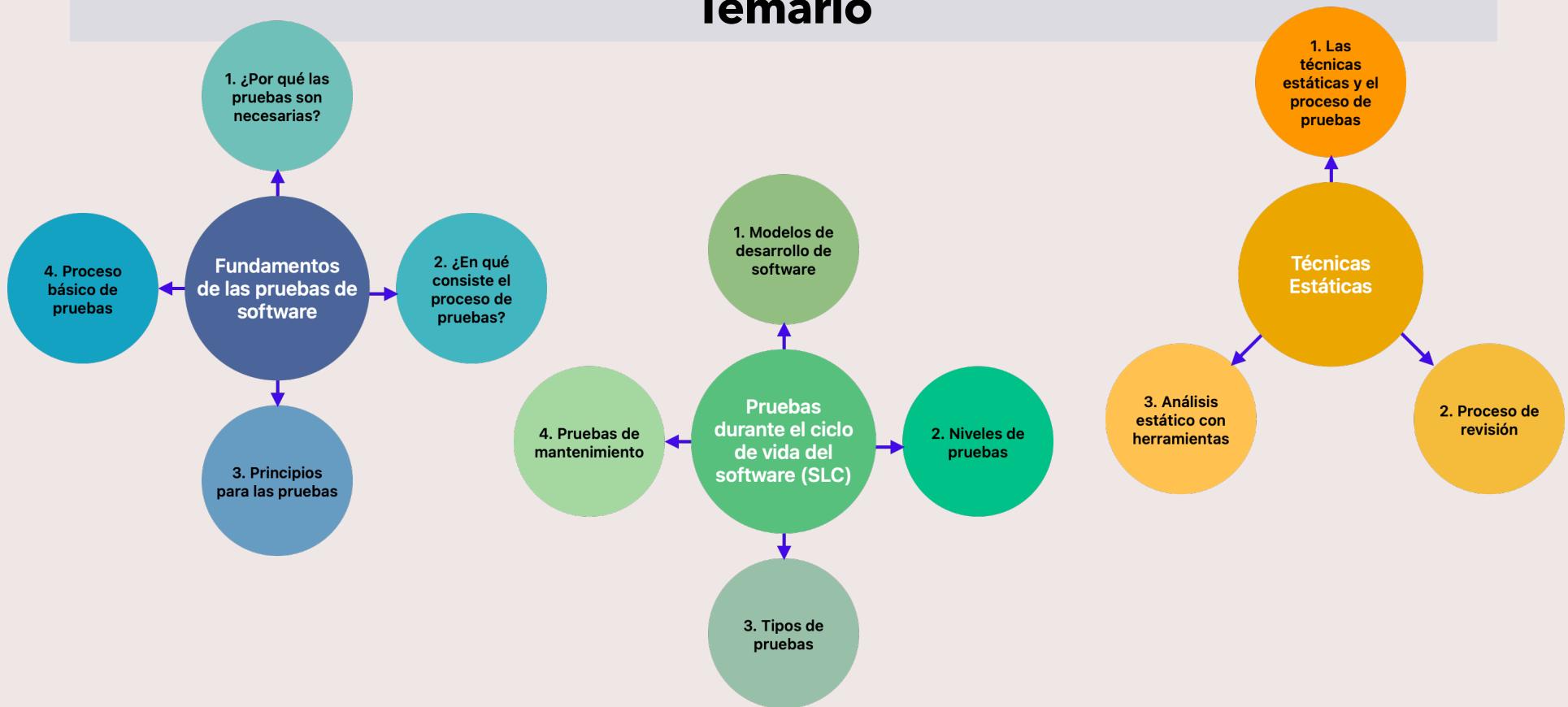
Presentación

- Nombre
- Carrera
- ¿Por qué elegiste esta especialidad?



Fundamentos de Ingeniería de Pruebas y Calidad de Software

Temario



Fundamentos de Ingeniería de Pruebas y Calidad de Software

Temario



Fundamentos de las pruebas de software

¿Qué es probar?

- Es una forma de evaluar la calidad del software y de reducir el riesgo de fallos en un entorno de operaciones o de producción
- Es un proceso que incluye muchas actividades diferentes como analizar, diseñar e implementar las pruebas, informar del avance y de los resultados y evaluar la calidad del objeto de prueba

¿Por qué es necesario probar?

- Para reducir el riesgo de que se puedan producirse fallos durante la operación
- Para identificar defectos y corregirlos
- Para probar si el software cumple con los requisitos del cliente

Objetivos Característicos de la Prueba

- Evaluar productos de trabajo como requisitos, historias de usuario, diseño y código
- Verificar el cumplimiento de todos los requisitos especificados
- Validar si el objeto de prueba esta completo y funciona como se espera
- Generar confianza en el nivel de calidad del objeto de prueba
- Prevenir defectos
- Encontrar fallos y defectos

Objetivos Característicos de la Prueba

- Ayudar a la toma de decisiones informadas, especialmente relacionadas a la calidad
- Reducir el nivel de riesgo de calidad inadecuada del software
- Cumplir con requisitos o normas

Proceso de prueba

- Modelo de ciclo de vida y desarrollo de software
- Niveles y tipos de prueba
- Riesgos de producto y proyecto
- Dominio del negocio

Proceso de prueba

- Restricciones operativas
 - Presupuestos y recursos
 - Plazos
 - Complejidad
- Políticas y Prácticas de la Organización
- Estándares internos y externos

Actividades y tareas de prueba

1. Planificación

2. Monitorización, seguimiento y control

3. Análisis

4. Diseño

5. Implementación

6. Ejecución

7. Compleción

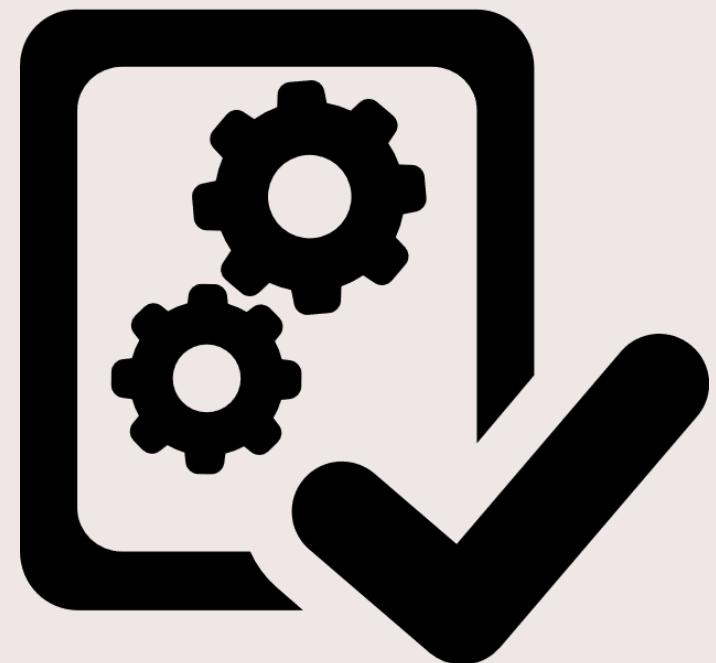
1. Planificación de la Prueba

- Son actividades que definen los objetivos de la prueba y el enfoque para cumplir con los objetivos de la prueba dentro de las restricciones impuestas
 - Especificaciones técnicas
 - Tareas de prueba
 - Calendario de pruebas



2. Monitorización y Control de la Prueba

- La monitorización es la comparación del estado real de la prueba contra como se va comparado al plan de pruebas
- El control, implica tomar medidas necesarias para cumplir con el plan de pruebas y sus objetivos
- La monitorización y el control de prueba se apoyan de la evaluación de los criterios de salida y la definición de hecho



2. Monitorización y Control de la Prueba

- Criterios de salida de la ejecución de prueba
 - Comprobar los resultados y los registros de la prueba
 - Evaluar el nivel de calidad de las pruebas de los componentes o sistemas en base a los resultados
 - Determinar si se necesitan más pruebas

3. Análisis de pruebas

- Se analiza la base de prueba para identificar que se puede probar
- “Que probar”
- El análisis de prueba incluye:
 - Analizar la base de prueba considerando:
 - Especificaciones de requisitos
 - Información de diseño e implementación
 - Implementación del componente
 - Informe de análisis de riesgos



3. Análisis de pruebas

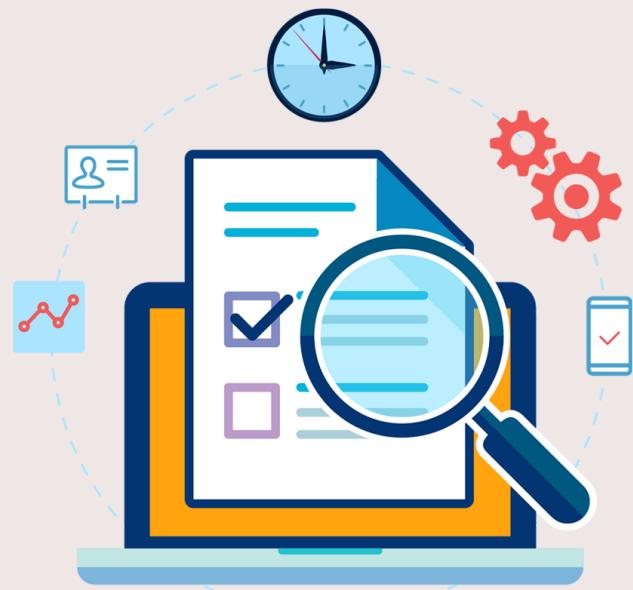
- Evaluar la base de prueba y los elementos de prueba para identificar distintos tipos de defectos:
 - Ambigüedades
 - Omisiones
 - Inconsistencias
 - Inexactitudes
 - Contradicciones

3. Análisis de pruebas

- Identificar lo que se va a probar y como se va a probar
- Definir y priorizar las pruebas
- Identificar la relación existente entre casos de prueba

4. Diseño de la prueba

- Se construyen los casos de prueba
- Después de “¿qué probar?”, “¿cómo probar?”



4. Diseño de la prueba

- El diseño de la prueba, incluye las siguientes actividades principales:
 - Diseñar y priorizar casos y conjuntos de casos de prueba
 - Identificar los datos de prueba necesarios
 - Diseñar el entorno de prueba así como infraestructura y herramientas necesarias
 - Encontrar la relación entre condiciones de prueba, casos de prueba y procedimientos de prueba

5. Implementación de la prueba

- Se crean y/o completan los productos de prueba necesarios para la ejecución de la prueba
- “¿Está todo preparado para realizar la prueba?”

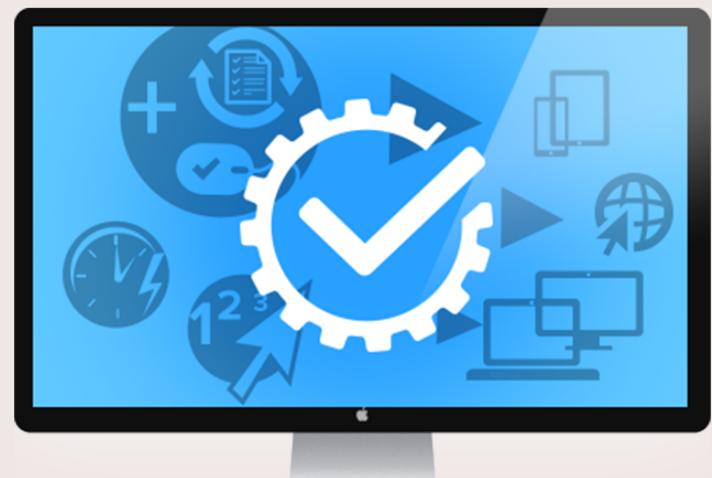


5. Implementación de la prueba

- La implementación de la prueba incluye las siguientes actividades:
 - Desarrollar y priorizar procedimientos de prueba
 - Crear juegos de pruebas y guiones automatizados
 - Organizar las pruebas en el calendario
 - Construir entorno de prueba
 - Preparar datos de prueba
 - Verificar trazabilidad de la prueba

6. Ejecución de prueba

- Se ejecutan los casos de prueba de acuerdo al plan de pruebas o calendario de pruebas



6. Ejecución de prueba

- La ejecución de la prueba incluye las siguientes actividades:
 - Registrar los identificadores y las pruebas de los elementos
 - Ejecutar las pruebas
 - Comparar resultados
 - Analizar anomalías
 - Informar defectos
 - Registrar resultado de ejecución
 - Repetir actividades de prueba
 - Verificar la relación entre los casos de prueba, su ejecución y resultados

7. Compleción de prueba

- Recopilan datos de las pruebas completadas
- Permiten hacer la liberación del software que fue probado
- Cuando finaliza la iteración de un proyecto ágil
- Cuando se completa un nivel de prueba
- Cuando se completa la liberación de un mantenimiento



Principios de las pruebas

Siete principios de la prueba

- Son directrices generales del proceso de prueba que son comunes para todo tipo de prueba



1. La prueba muestra la presencia de defectos, no su ausencia

- La prueba puede mostrar la presencia de defectos, pero no puede probar que no hay defectos
- La prueba reduce la probabilidad de que queden defectos no descubiertos en el software
- Si no se encuentran defectos, no quiere decir que este correcto



Ejemplo: una aplicación bancaria

- Se prueba a fondo y se somete a diferentes niveles de prueba y no se detectan defectos en el sistema
- En el entorno de producción, un cliente real prueba una funcionalidad que raramente se usa y los testers pasaron por alto esa funcionalidad → no se encontraron errores o no se ha tocado ese código



Ejemplo: Un anuncio de jabón antibacterial

- Un anuncio de jabón antibacterial dice que mata el 99% de los gérmenes
- El producto no deja 100% libre de gérmenes → ningún software está libre de defectos



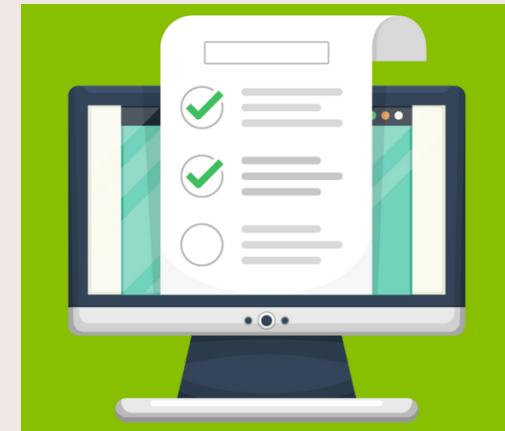
2. La prueba exhaustiva es imposible

- No es posible probar todas las funcionalidades con todas sus combinaciones válidas e invalidas
- En lugar de intentar realizar pruebas exhaustivas, se debería utilizar el análisis de riesgos, las técnicas de pruebas y las prioridades para centrar los esfuerzos de prueba



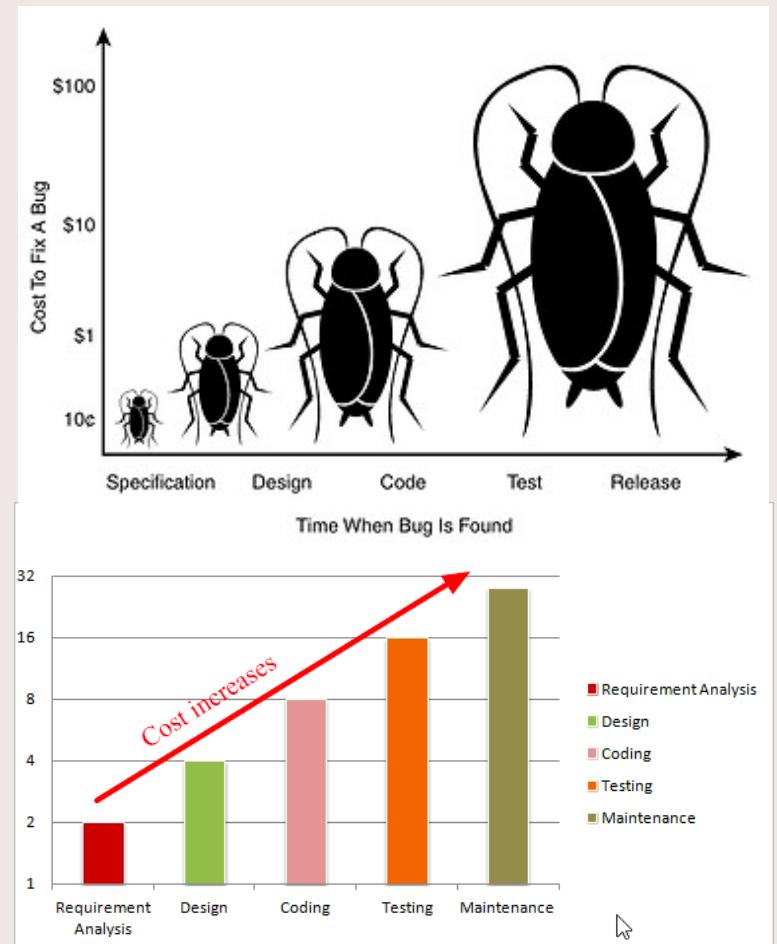
Ejemplo: Un campo en un formulario

- Se tiene un campo de entrada que acepta letras, caracteres especiales y números del 0 al 1000
- Es imposible probar todas las combinaciones para cada tipo de entrada
- No se pueden realizar pruebas exhaustivas



3. La prueba temprana ahorra tiempo y dinero

- Para detectar defectos en forma temprana, las pruebas deben iniciarse lo antes posible en el ciclo de vida de desarrollo de software
- La prueba temprana ayuda a reducir o eliminar cambios costosos



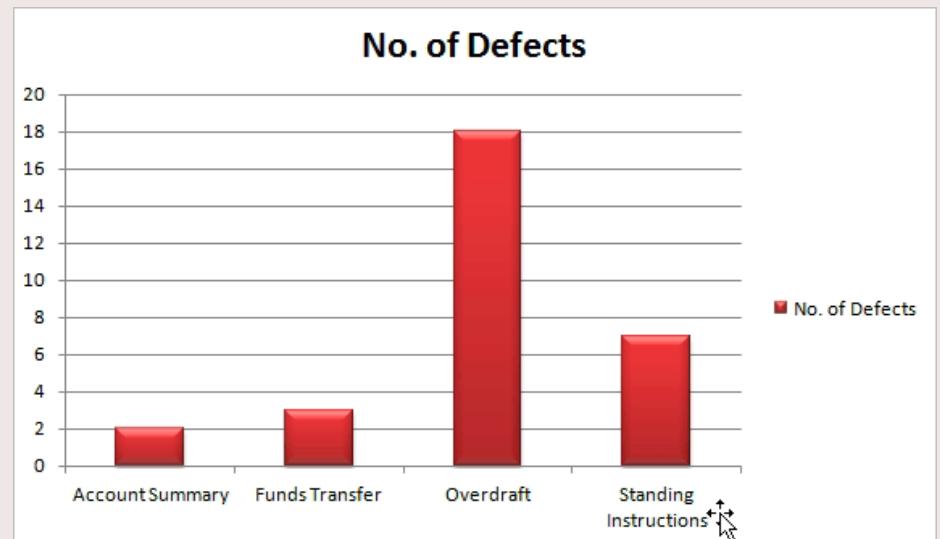
4. Los defectos se agrupan

- Un pequeño número de módulos contiene la mayoría de defectos descubiertos
- Las agrupaciones de defectos son importantes para hacer un análisis de riesgos y saber donde centrar esfuerzos
- *Principio de Pareto:* El 80% de los problemas, se encuentran en el 20% de los módulos



Ejemplo: aplicación bancaria

- La mayoría de los defectos están relacionados al sobregiro
- El resto como resumen de cuenta, transferencia de fondos, instrucción permanente, etc. tienen un número limitado de defectos



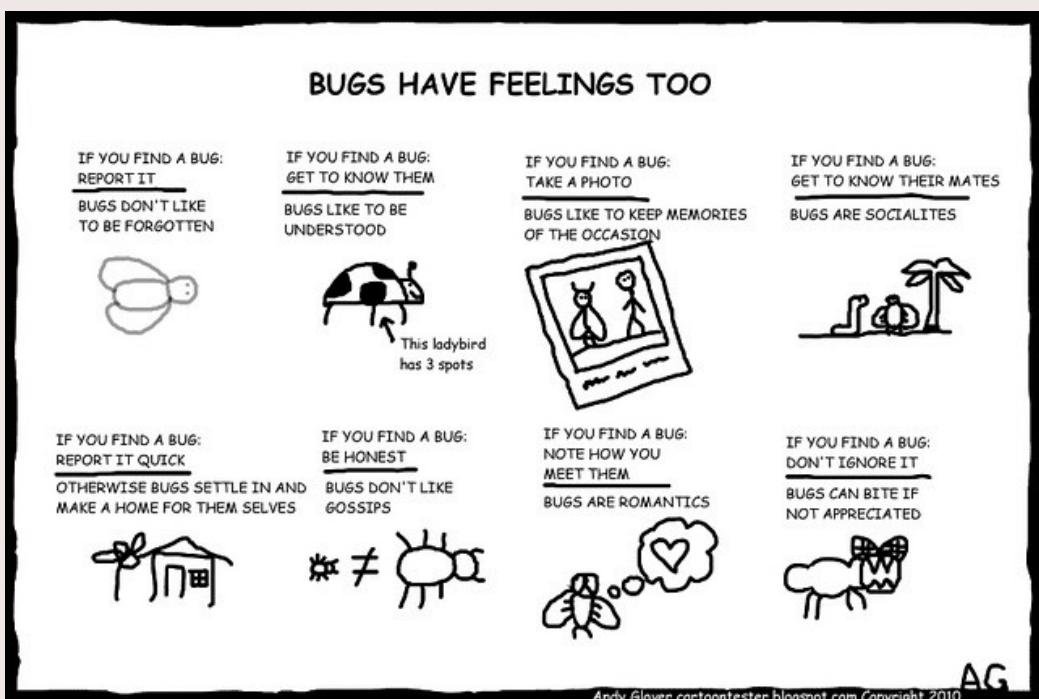
5. Cuidado con la paradoja del pesticida

- Si las mismas pruebas se repiten muchas veces, eventualmente no se encontrarán más errores
- Para detectar nuevos defectos, es necesario cambiar las pruebas y los datos de prueba



Ejemplo

- Hay 50 casos de prueba para un módulo en particular
- De esos 50 casos, 20 no detectaron ningún defecto en las últimas 5 iteraciones
- Esos 20 casos de prueba deben revisarse a fondo para ver si conservarlos, cambiarlos o eliminarlos



6. La prueba depende del contexto

- Las pruebas se realizan de manera diferente dependiendo del contexto
- Una prueba en un proyecto Ágil, no se realiza igual que una prueba en un proyecto waterfall



Ejemplo

- Hay aplicaciones de banco, seguros, viajes, publicidad, etc
- Cada aplicación es diferente en requisitos, funciones, propósitos de prueba, riesgos, técnicas etc.
- Al ser diferentes, se prueban de manera diferente, por lo que se basa en el contexto de la aplicación



7. La ausencia de errores es una falacia

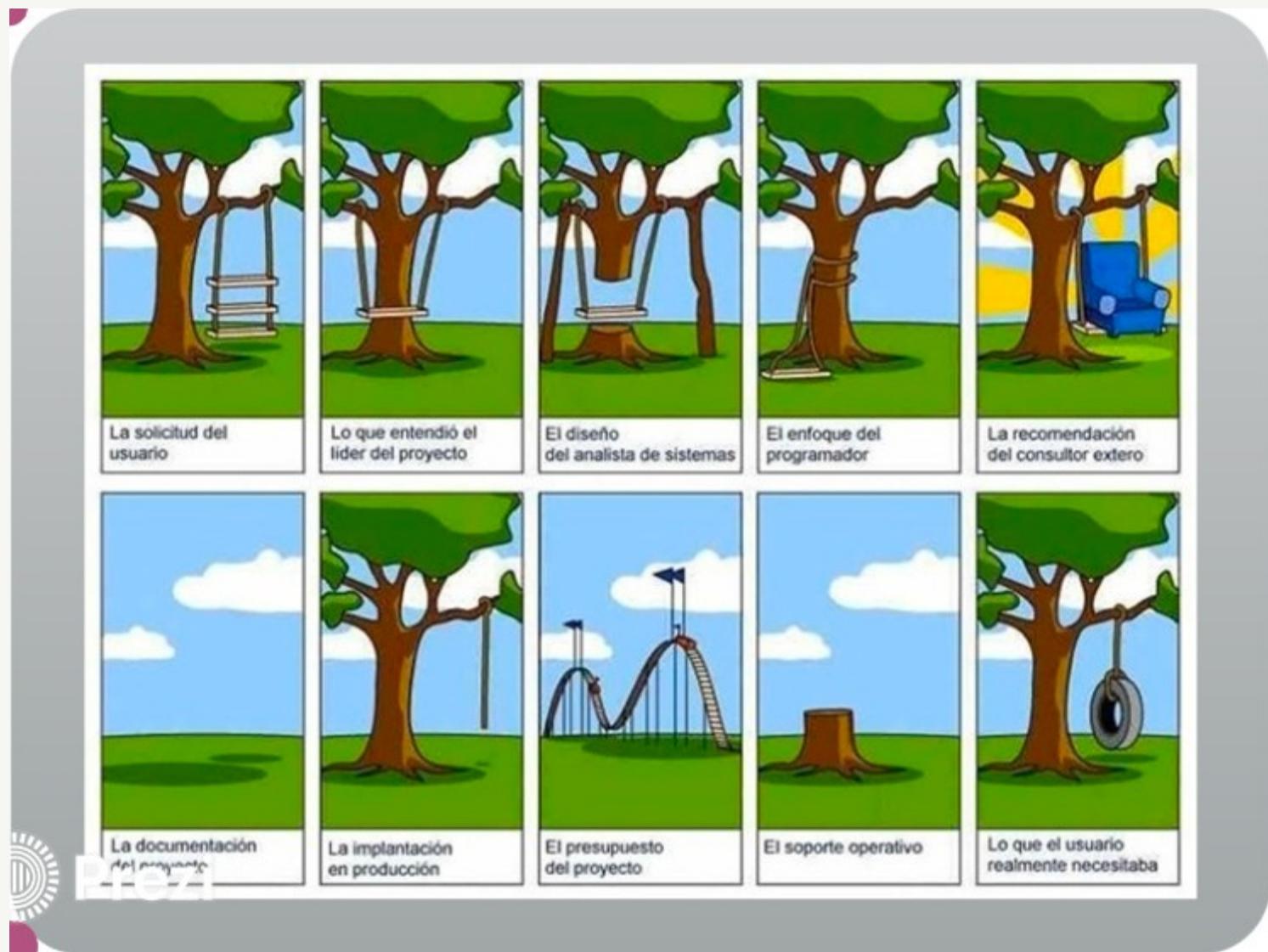
- Algunas organizaciones esperan que los testers puedan realizar todas las pruebas posibles y encontrar todos los defectos posibles
 - Eso es imposible
 - Si el software se prueba completamente y no se encuentran defectos, se podría decir que el software está libre de defectos un 99%
 - Si se prueba con datos incorrectos, no se ajustan a las necesidades del usuario final



Ejemplo: app e-commerce

- Los requisitos de la funcionalidad de carritos de compra se interpretan y se prueban incorrectamente
- Aún encontrar errores y arreglarlos no ayuda ya que las pruebas se realizan en requisitos incorrectos y no se ajustan a las necesidades del usuario final





Pruebas Durante el ciclo de vida del desarrollo de software

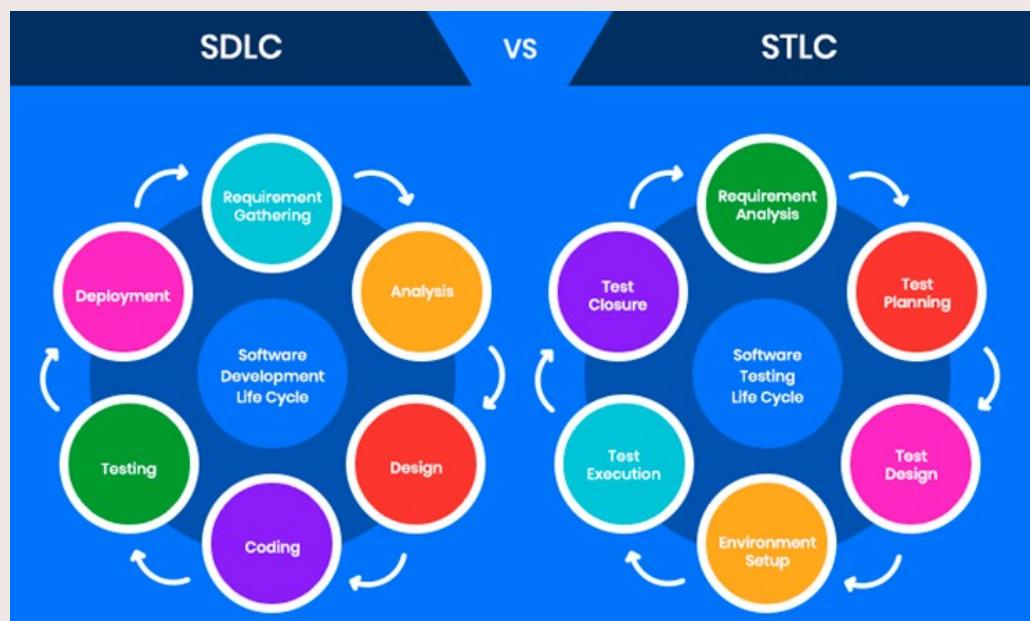
Modelos del ciclo de vida del desarrollo de software

- Un modelo de ciclo de vida de desarrollo de software describe los tipos de actividades que se realizan en cada etapa del proyecto de desarrollo de software y como las actividades se relacionan entre si de forma lógica y cronológica
- Hay diferentes modelos de desarrollo de ciclo de vida de software y cada uno requiere de diferentes enfoques de prueba

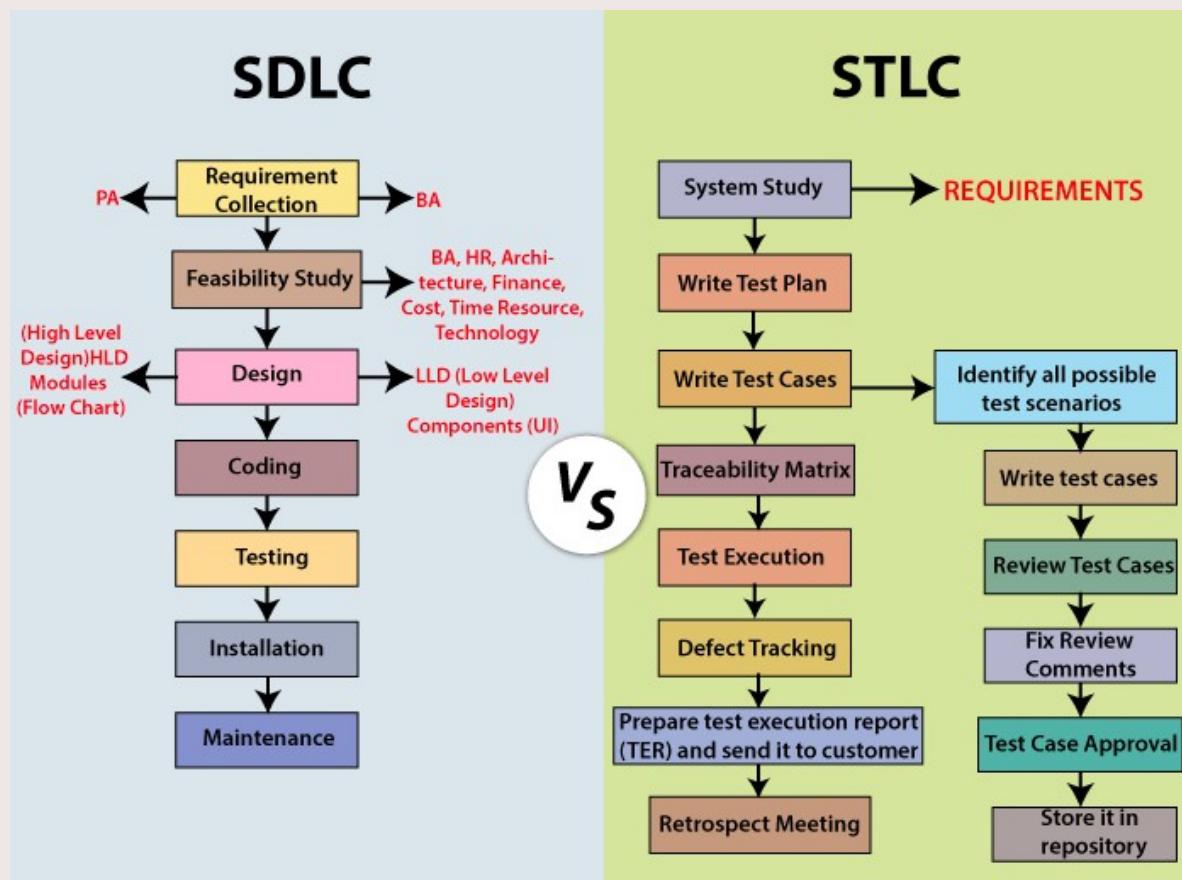


Desarrollo de software y Prueba de Software

- Es importante que los testers estén familiarizados con los modelos de ciclo de vida de desarrollo de software para que puedan realizar las actividades de prueba necesarias y adecuadas
- **SDLC:** Software Development Lifecycle
- **STLC:** Software Testing Lifecycle



Desarrollo de software y Prueba de Software



Desarrollo de software y Prueba de Software

- En cualquier modelo de ciclo de vida de desarrollo de software hay características que hacen que las pruebas sean adecuadas:
 - Para cada nivel de desarrollo, hay una actividad de prueba asociada
 - Cada nivel de prueba tiene objetivos de prueba específicos para ese nivel
 - El análisis y diseño de la prueba para un nivel de prueba comienza en la misma etapa de desarrollo
 - Los testers, participan en discusiones para definir y refinar los requisitos y diseño y tienen que estar involucrado en la revisión

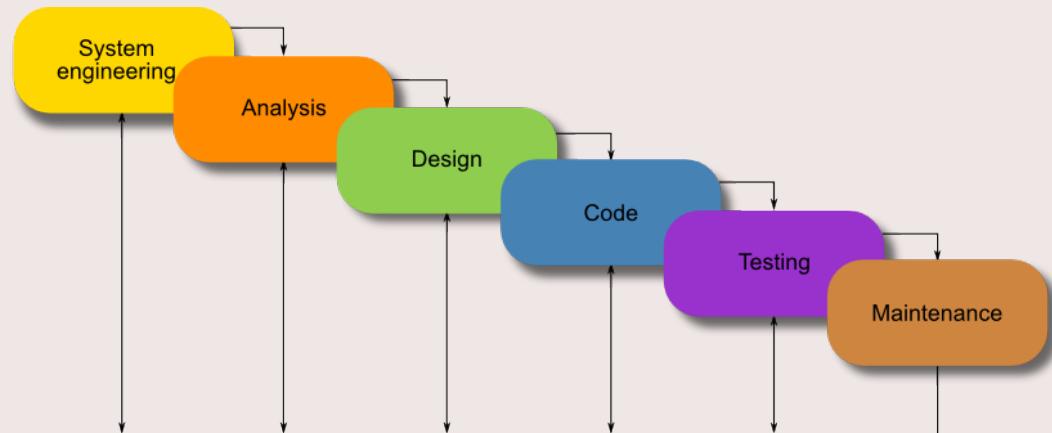
Desarrollo de software y Prueba de Software

- Independientemente del modelo de ciclo de vida de desarrollo de software, las actividades de prueba deben de comenzar en las etapas iniciales del ciclo de vida, adhiriéndose al principio de **prueba temprana**

Modelos de ciclo de vida del desarrollo de software

Modelos de ciclo de vida del desarrollo de software

- Un modelo de ciclo de vida de software es una vista de las actividades que ocurren durante el desarrollo de software e intenta determinar el orden de las etapas involucradas y los criterios de transición asociados a estas etapas

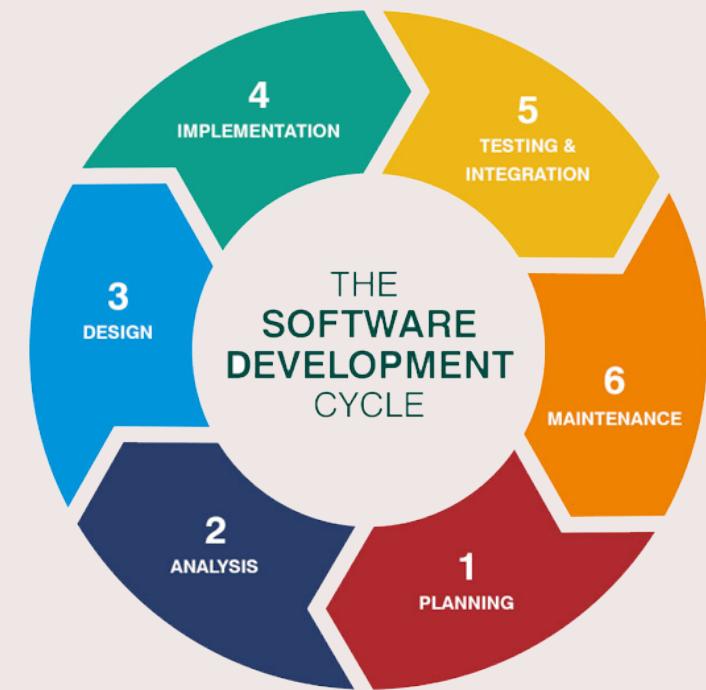


Modelos de ciclo de vida del desarrollo de software

- Un modelo de ciclo de vida del software:
 - Describe las fases principales del desarrollo del software
 - Define las actividades primarias esperadas a realizarse durante esas fases
 - Ayuda a administrar el progreso del desarrollo
 - Provee un espacio de trabajo para la definición de un proceso detallado del desarrollo del software

Modelos de ciclo de vida del desarrollo de software

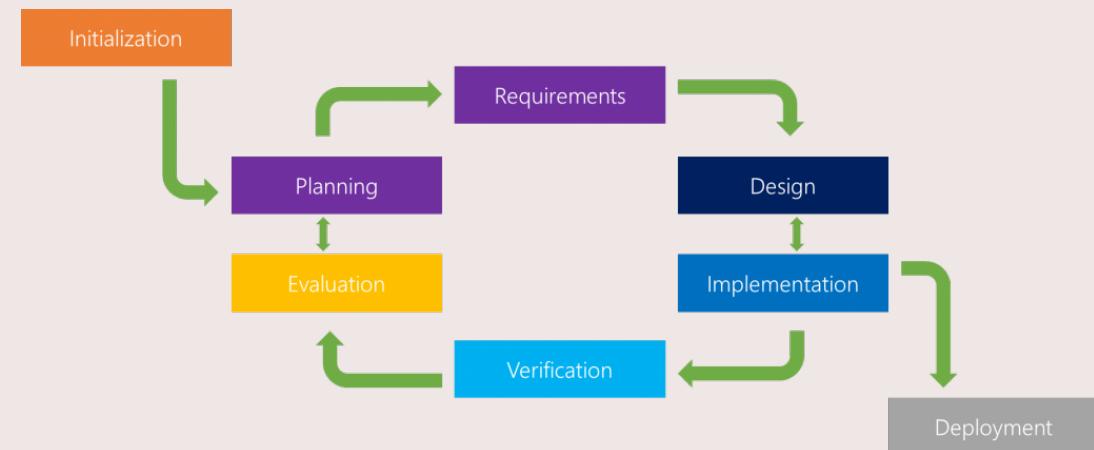
- Las principales diferencias entre distintos modelos del ciclo de vida están en:
 - El alcance del ciclo dependiendo hasta donde llegue el proyecto
 - Las características de las fases en que dividen el ciclo
 - La estructura y la sucesión de las etapas



Modelos de ciclo de vida del desarrollo de software

- Los principales modelos de ciclo del vida del software se dividen en:

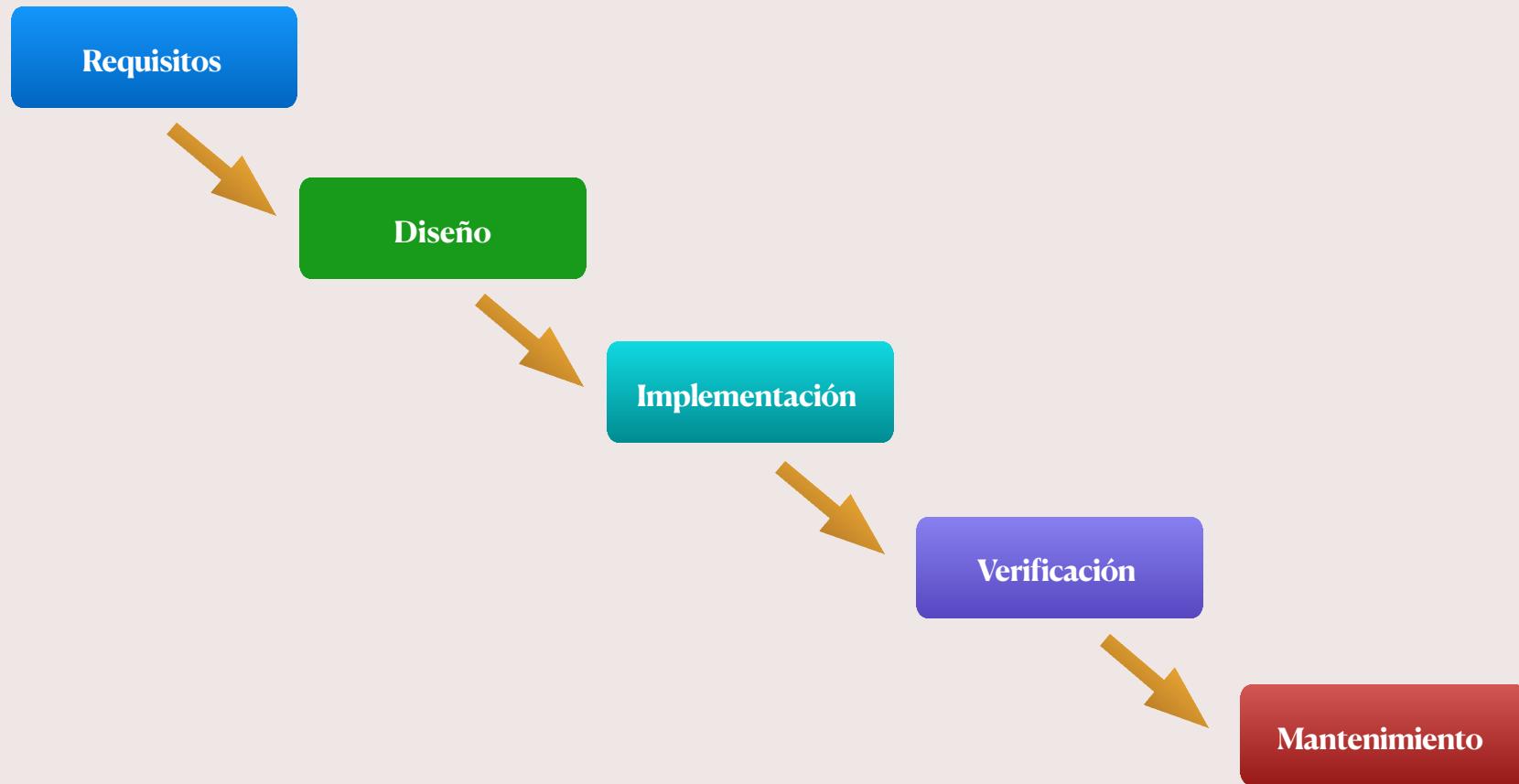
- **Modelo de desarrollo secuencial**
- **Modelo de desarrollo iterativo e incremental**



Modelo de desarrollo secuencial

- También es conocido como en cascada
- Describe el proceso de desarrollo de software como un flujo lineal y secuencial de actividades
- Cualquier fase del proceso de desarrollo, debe comenzar cuando se haya completado la fase anterior
- Las fases se completan una tras otra
- En este modelo, las actividades de prueba solo ocurren después de que todas las demás actividades de desarrollo hayan sido completadas

Modelo de desarrollo secuencial



Modelo de desarrollo secuencial

Desventajas

- Los proyectos raramente siguen el paradigma secuencial que propone el proyecto
- Los responsables del desarrollo, regularmente se retrasan, muchas veces, innecesariamente
- No siempre se sigue el flujo secuencial
- Es difícil tener un 100% de los requisitos al inicio
- El cliente debe tener paciencia, los primeros resultados serán hasta que ya este operando el sistema

Modelos de desarrollo iterativos e incrementales

- El **desarrollo incremental** implica establecer requisitos, diseñar, construir y probar un sistema en fragmentos, lo que significa que las funcionalidades del software crecen de forma incremental
- El tamaño de los incrementos, varia, ya que algunos tienen tiempos más grandes y otros más pequeños
- Los incrementos en funcionalidades pueden ser tan pequeños como un camino en la interfaz o una nueva opción en una consulta

Modelos de desarrollo iterativos e incrementales

- El **desarrollo iterativo** se produce cuando se especifican, diseñan, construyen y prueban conjuntamente grupos de funciones en una serie de ciclos, usualmente de una duración fija
- Las iteraciones pueden implicar cambios en las funciones desarrolladas en las iteraciones anteriores junto con cambios en el alcance del proyecto
- Cada iteración proporciona software operativo, hasta que se entregue el software final se detiene el desarrollo

Modelos de desarrollo iterativos e incrementales

Ejemplos

- Algunos ejemplos son:
 - Rational Unified Process
 - Scrum
 - Kanban
 - Espiral o prototipado

Rational Unified Process

- Fue creada por Rational Software de IBM
- Cada iteración tiende a ser relativamente larga (2-3 meses)
- Los incrementos en la funcionalidad son proporcionalmente grandes
- No son pasos establecidos, son un conjunto de metodologías adaptables al contexto y necesidades de cada organización

Rational Unified Process

Principios de Desarrollo

- Está basado en 6 principios clave que son:

- Adaptar el proceso**

- El proceso deberá adaptarse a las características propias del proyecto u organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico.
 - También se debe tener en cuenta el alcance del proyecto

Rational Unified Process

Principios de Desarrollo

- **Equilibrar prioridades**

- Los requerimientos de los diversos participante, pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe de encontrarse un equilibrio que satisfaga los deseos de todos.
- Gracias a este equilibrio se podrán corregir desacuerdos que surjan en el futuro

Rational Unified Process

Principios de Desarrollo

- **Demostrar valor iterativamente**

- Los proyectos se entregan, aunque sea de un modo interno, en **etapas iteradas**. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto y se refina la dirección del proyecto así como los riesgos involucrados

Rational Unified Process

Principios de Desarrollo

- **Colaboración entre equipos**

- El desarrollo de software no lo hace una sola persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados, etc

- **Enfocarse en la calidad**

- El control de la calidad no debe realizarse al final de cada iteración, si no en todos los aspectos de la producción
- El aseguramiento de calidad forma parte del proceso de desarrollo y

Rational Unified Process

Principios de Desarrollo

- **Elevar el nivel de abstracción**

- Este principio motiva el uso de conceptos reutilizables como el uso de patrones de diseño de software o frameworks. Esto evita que los ingenieros de software vayan directamente de los requisitos a la codificación del software sin saber con certeza qué codificar para satisfacer de la mejor manera los requerimientos y sin comenzar pensando desde un inicio en la reutilización del código

Rational Unified Process

Características

- Iterativo e incremental
- Esta compuesto por fases, cada una de sus fases dividida en iteraciones
- Las iteraciones, tienen como resultado incremento en el producto desarrollado o mejoras en la funcionalidad del sistema
- Dirigido por los casos de uso
- Los caso de uso se utilizan para capturar los requisitos funcionales y definir los contenidos de las iteraciones
- La idea es que cada iteración tome un conjunto de casos de uso o escenarios

Rational Unified Process

Características

- Centrado en la arquitectura
 - Asume que no existe un modelo único que cubra todos los aspectos del sistema
 - Existen multiples modelos y vistas que definen la arquitectura de software de un sistema
- Enfocado en los riesgos
 - Requiere que el equipo del proyecto se centre en identificar los riesgos críticos en una etapa temprana del ciclo de vida
 - Los resultado de cada iteración, deben ser seleccionados en un orden que asegure que los riesgos principales son considerados primero

Rational Unified Process

Fases

- **Inicio**

- Se define el alcance del proyecto.
- Se hace mayor énfasis en actividades de modelado del negocio y de requerimientos
- Las primeras iteraciones se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de riesgos críticos y al establecimiento de una línea base de la arquitectura

Rational Unified Process

Fases

- **Elaboración**

- Se obtiene la visión refinada del proyecto a realizar, la implementación iterativa del núcleo de la aplicación, la resolución de riesgos altos, nuevos requisitos y se ajustan estimaciones
- Las iteraciones se orientan al desarrollo de la linea base de la arquitectura, abarcan más flujos de trabajo de requerimientos, modelos de negocios, análisis, diseño y una parte de implementación orientada a la linea base de la arquitectura

Rational Unified Process

Fases

- **Construcción**

- Se lleva a cabo la construcción del producto por medio de una serie de iteraciones
- La evolución del desarrollo hasta convertirse en un producto listo
- Por cada iteración se selecciona algunos casos de uso, se refina su análisis y diseño y se procede a su implementación y pruebas

Rational Unified Process

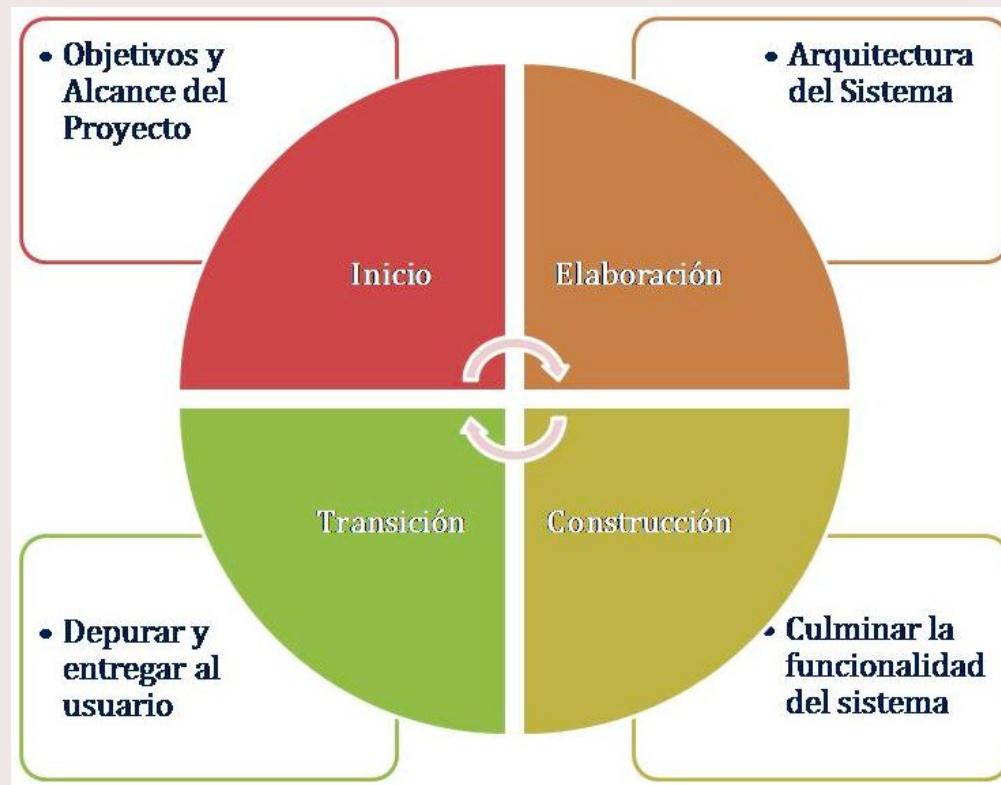
Fases

- **Transición**

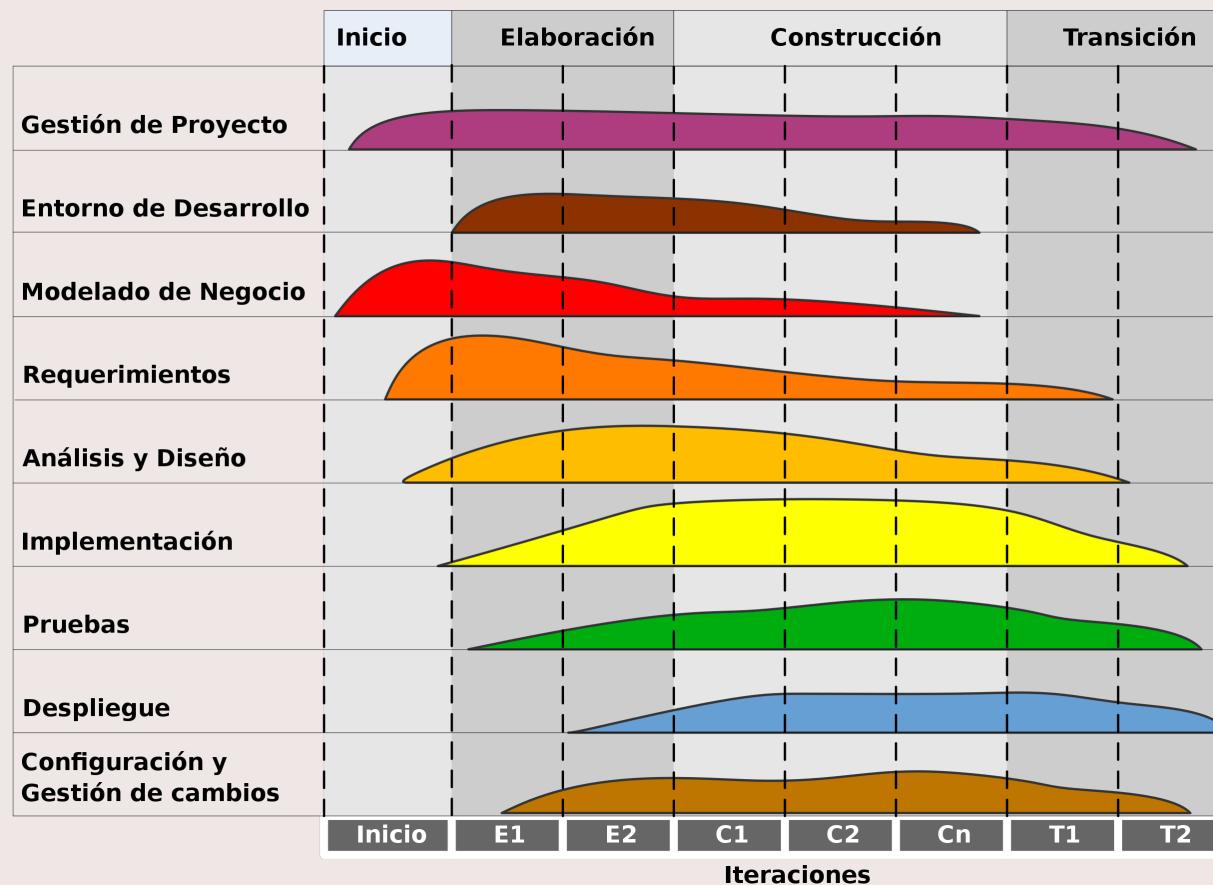
- Se pretende garantizar que se tiene un producto preparado para su entrega a los usuarios
- Es la fase final, el producto debe de estar listo para ser probado, instalado y utilizado por el cliente sin ningún problema
- Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto
- En cada fase participan todas las disciplinas pero dependiendo de a fase el esfuerzo dedicado a cada una varia

Rational Unified Process

Fases



Rational Unified Process Proceso



Scrum

- Cada iteración tiende a ser relativamente corta (horas, días o pocas semanas)
- Los incrementos de las funcionalidades son proporcionalmente pequeñas como pocas mejoras o pocas funcionalidades nuevas
- Scrum es un framework dentro del cual las personas pueden abordar problemas complejos de adaptación al mismo tiempo que entregan productos de manera productiva y creativa con el mayor valor posible

Scrum

- Se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al proyecto
- Es especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales

Scrum

También se utiliza para resolver las siguientes situaciones:

- Cuando las entregas se alargan demasiado
- Cuando no se está entregando al cliente lo que necesita
- Cuando los costos se disparan
- Cuando la calidad no es aceptable
- Cuando se necesita capacidad de reacción ante la competencia
- Cuando la moral del equipo es baja y la rotación alta
- Cuando es necesario identificar y solucionar ineficiencias sistemáticas
- Cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de un producto

Scrum

- Scrum requiere un **Scrum Master** para fomentar un entorno donde:
 - Un **Product Owner** ordena el trabajo de un problema complejo en un **Product Backlog**
 - El **Scrum Team** que convierte una selección del trabajo en un incremento de valor durante un **Sprint**
 - El **Scrum Team** y sus partes interesadas inspeccionan los resultados y se ajustan para el próximo **Sprint**
 - Repetir

Scrum Glosario

- **Product Owner:** Rol en Scrum responsable de maximizar el valor de un producto, principalmente al administrar y expresar de manera incremental las expectativas comerciales y funcionales de un producto a los desarrolladores
- **Scrum Master:** Rol dentro de un Scrum Team responsable de guiar, entrenar, enseñar y ayudar a un Scrum Team y sus entornos en una comprensión y uso adecuados de Scrum

Scrum Glosario

- **Scrum Team:** Un equipo de autogestión que consta de Scrum Master, un Product Owner y desarrolladores. El tamaño recomendado es de 3 a 9 personas
- **Sprint:** Evento de Scrum que dura un mes o menos, regularmente, dos semanas, que sirve como contenedor para los otros eventos y actividades de Scrum. Los Sprints se realizan de forma consecutiva sin espacios intermedios. Durante el Sprint, los elementos del Backlog se convierten en un producto funcionando

Scrum Glosario

- **Stakeholder:** Una persona ajena al Scrum Team que está interesada en el producto, representada por el Product Owner y comprometida con el Scrum Team en el Sprint Review
- **Definition of Done:** Es una descripción formal del estado del incremento cuando cumple con las medidas de calidad requeridas. En el momento que un elemento del Backlog cumple con el Definition of Done, se convierte en un incremento. Los responsables de establecer el Definition of Done son los Developers

Ceremonias Scrum

- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective
- Sprint Grooming o Refinement



Sprint Planning

- Se realiza al comienzo del Sprint
- Sirve para que el Scrum Team inspeccione el trabajo del Product Backlog que sea más valioso realizar en ese Sprint y se diseñe ese trabajo

Sprint Planning

- Esta reunión se realiza el primer día del Sprint y consta de dos partes:
 - Selección de Requisitos
 - El cliente o Product Owner, presenta al Scrum Team la lista de requisitos priorizada
 - El Scrum Team pregunta las dudas que surgen y selecciona los requisitos más prioritarios que cree se podrán completar en la iteración

Sprint Planning

- **Planificación de la iteración**

- El Scrum Team elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos seleccionados
- Se estima el esfuerzo de manera conjunta y el Scrum Team se autoasigna las tareas y se auto organizan para trabajar

Daily Scrum

- Es una reunión que debe de durar 15 minutos en la que participan los Developers
- Se lleva a cabo todos los días del Sprint, ahí los Developers planean su trabajo para las siguientes 24 horas
- Optimiza la colaboración y desempeño inspeccionando el trabajo desde el primero hasta el último día del Sprint

Daily Scrum

- Se lleva a cabo a la misma hora y en el mismo lugar todos los días
- ¿Qué hice ayer?
- ¿Qué voy a hacer hoy?
- ¿Qué impedimentos hay?

Sprint Review

- Ocurre al final del Sprint
- El Product Owner y el Scrum Team presentan a los stakeholders el incremento terminado para su inspección y adaptación
- Se revisa el incremento terminado, se muestra el software funcionando y los stakeholders tienen la oportunidad de hacer preguntas
- Si no existe software terminado, el Sprint Review carece de sentido

Sprint Review

- Los Developers comentan que ocurrió durante el Sprint, impedimentos encontrados y soluciones tomadas y actualizan a los stakeholders con la situación del equipo
- Es una reunión de trabajo que sirve para marcar la estrategia de negocio

Sprint Retrospective

- Es una reunión que se realiza al finalizar el Sprint, después del Sprint Review
- Sirve para que el Scrum Team inspeccione qué pasó en el Sprint pasado y planifique mejoras a implementar durante los Sprints futuros

Sprint Retrospective

- Se analiza:
 - ¿Qué fue bien durante el Sprint?
 - ¿Que falló en el Sprint?
 - ¿Qué se puede mejorar?

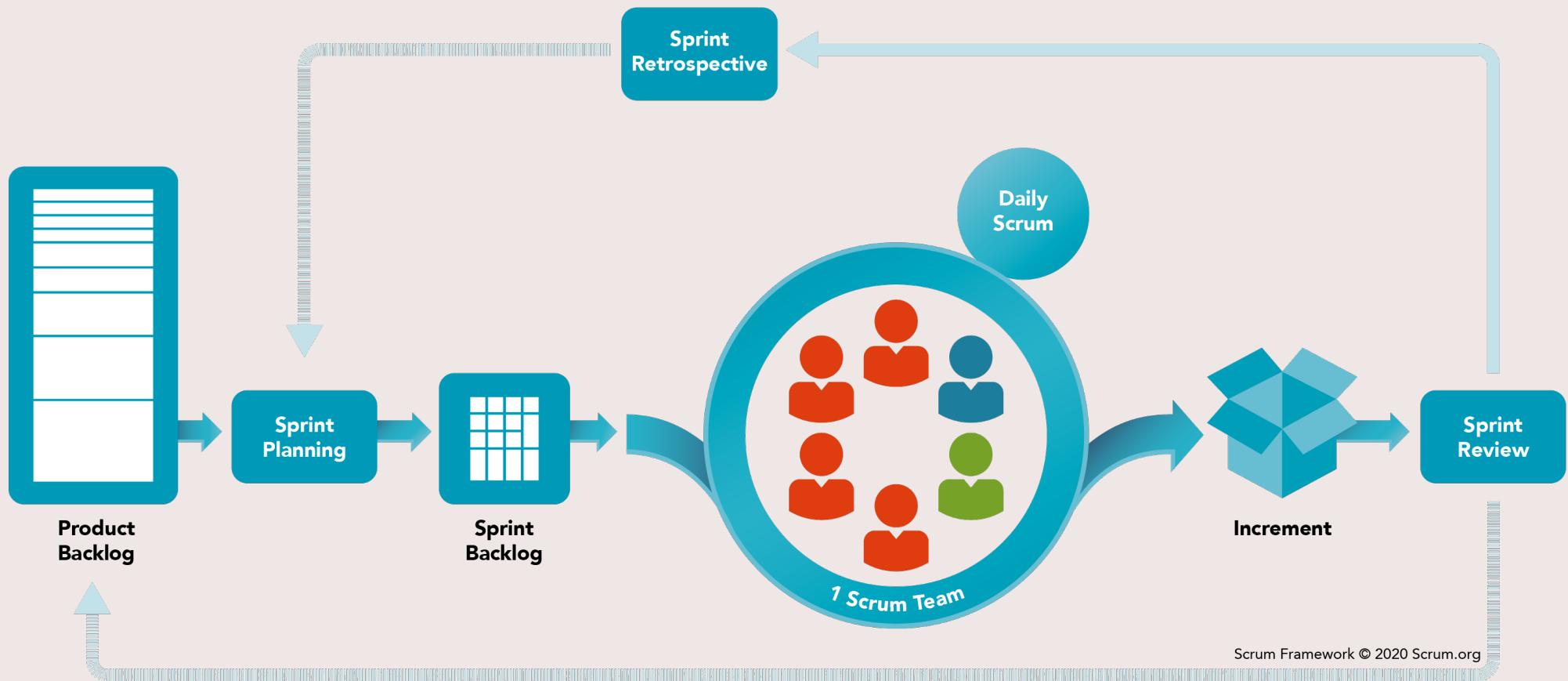
Sprint Grooming o Refinement

- El refinamiento del Product Backlog es una práctica recomendada para asegurar que éste siempre esté preparado
- Es responsabilidad del Product Owner agendar, gestionar y dirigir esta reunión
- Participa todo el Scrum Team y cualquier recurso adicional que el Product Owner considere que puede aclarar dudas o requerimientos

Sprint Grooming o Refinement

- Es necesario que antes de esta reunión todos conozcan los requerimientos y las historias de usuario que van a ser tratadas

Scrum



Scrum Framework © 2020 Scrum.org

Product Backlog

Scrum

- El **Product Backlog** es un listado de todas las tareas que se pretenden hacer durante el desarrollo de un proyecto.
- Es una lista de características que han sido priorizadas, y contiene descripciones breves sobre todo lo que se desea para el producto que se va a desarrollar.
- Todas las tareas deben listarse en el **Product Backlog** para que estén visibles ante todo el equipo y se pueda tener una visión panorámica de todo lo que se espera realizar

Product Backlog

Scrum

- No tiene una longitud definida, depende del proyecto le asignan prioridades a cada uno de los elementos del product backlog en base a las necesidades del cliente y la complejidad de cada elemento
- Cada sprint, se pueden agregar nuevos elementos para satisfacer las necesidades ya sea de diseño, funcionalidad o verificación de actividades

Product Backlog

Scrum

- Se le asignan prioridades a cada uno de los elementos del product backlog en base a las necesidades del cliente y la complejidad de cada elemento
- Cada sprint, se pueden agregar nuevos elementos para satisfacer las necesidades ya sea de diseño, funcionalidad o verificación de actividades

Product Backlog

Scrum

- Se le asignan prioridades a cada uno de los elementos del product backlog en base a las necesidades del cliente y la complejidad de cada elemento
- Cada sprint, se pueden agregar nuevos elementos para satisfacer las necesidades ya sea de diseño, funcionalidad o verificación de actividades

Product Backlog

Scrum

- Al aplicar Scrum, no es necesario definir todos los requisitos al inicio de un proyecto. Típicamente, el product owner en conjunto con el equipo empiezan escribiendo todo lo que consideran importante en el product backlog.
- Este product backlog es casi siempre suficiente para iniciar con el primer sprint. Y este product backlog tiene permitido crecer y cambiar tanto como sea necesario, en función a lo que se va aprendiendo sobre el producto y los clientes.

Product Backlog

Scrum

- El Product Backlog, puede incluir distintos tipos de trabajo o elementos:
 1. Features
 2. Bugs
 3. Technical work
 4. Knowledge acquisition

Product Backlog

Scrum

- La forma en la que regularmente escriben los elementos del backlog es expresando las características en forma de historias de usuario que son breves descripciones de la funcionalidad que se desea, contadas desde la perspectiva de un usuario

Ejemplo: “*Como comprador, yo puedo revisar los productos que están en mi carrito de compras antes de confirmar mi compra, y así estar seguro de lo que he seleccionado*”

Product Backlog

Scrum

- Los bugs se pueden incluir en el product backlog a medida que se detectan.
- El trabajo técnico y las actividades para adquirir nuevo conocimiento también se consideran en el product backlog.
Ejemplo: “Actualizar el equipo de todos los desarrolladores a Windows 10”; “Investigar y comparar librerías de JavaScript y hacer una elección”

Product Backlog → Sprint Backlog

Scrum

- En el **Sprint Planning**, el product owner se presenta en el con el product backlog priorizado y describe los items principales al equipo.
- El **Scrum Team**, determina qué items pueden completar durante el sprint que está por iniciar. El equipo mueve los items desde el product backlog al sprint backlog correspondiente.

Product Backlog → Sprint Backlog

Scrum

- Al mover los items, el **Scrum Team**, expande cada item del product backlog en una o más tareas (sobre el sprint backlog). Así el equipo puede compartir el trabajo de forma más efectiva durante el sprint

Backlog Grooming / Refinement

Scrum

- Es la reunión en la que el **Product Owner** y el **Scrum Team** se reúnen para revisar el **Product Backlog** y añadir, retirar o reestimar historias de usuario.

Backlog Grooming / Refinement

¿Para qué sirve?

Scrum

- Para que el **Product Owner** tenga las conversaciones necesarias con el equipo antes de introducir una historia de usuario en el **Sprint Backlog**.
- Para que el **Product Owner** pueda estimar las historias con el feedback proporcionado por el equipo.
- Para evitar que el equipo se quede sin historias en las que trabajar.
- Para incorporar al **Product Backlog** el feedback proveniente de las demos y del producto en producción.

Backlog Grooming / Refinement

¿Cómo se hace?

Scrum

- No tiene por qué ser una única reunión, pero debe atenderse al espíritu de la misma y cuidar el **Product Backlog** frecuentemente.
- El **Scrum Team** y el dueño de producto acuerdan una reunión periódica (una por iteración). También se puede publicar en el tablón junto a la hora de la reunión diaria.
- No debe ser una reunión demasiado larga (1 hora)

Backlog Grooming / Refinement

¿Cómo se hace?

Scrum

- El **Product Owner** trae escritas las historias de usuario, pero se pueden reescribir o matizar en esta reunión con ayuda del equipo.
- El **Product Owner** explica cada historia en orden de importancia para él. Es muy importante que el equipo aclare para qué quiere el **Product Owner** esa historia de usuario.
- Se estima cada historia atendiendo a su complejidad, cantidad de esfuerzo requerido o incertidumbre.
- Es el momento de distinguir entre un “defecto” (*bug*) o una funcionalidad nueva y mejorar la manera de describir las historias de usuario.

Backlog Grooming / Refinement

Malas Prácticas

Scrum

- Que no esté presente el **Product Owner**
- Salir de la reunión sin haber estimado y priorizado todas las historias de usuario o sin que cada historia tenga su criterio de aceptación.
- Dejar historias de usuario con estimaciones muy altas demasiado arriba en el **Backlog**. Una historia muy grande suele indicar que necesita ser redefinida y posiblemente se puede separar en más de una.

Backlog Grooming / Refinement Estimación de Historias de Usuario Scrum

- La estimación es el proceso de predecir la cantidad más realista de esfuerzo, a menudo expresada en términos de horas-persona, necesaria para realizar un elemento de trabajo.
- La estimación de puntos de historia puede ayudar a medir el tamaño de la historia y aprovechar los beneficios de la estimación ágil mientras se minimizan los riesgos planteados por factores desconocidos.
- La estimación de puntos de historias de usuario es un componente clave de la gestión ágil de proyectos ya que ayuda a mejorar la comprensión de los miembros del equipo sobre los requisitos del proyecto.

Backlog Grooming / Refinement Estimación de Historias de Usuario Scrum

- Una estimación precisa ayuda a los equipos ágiles a planificar la carga de trabajo y comprender las dependencias.
- Los equipos pueden usarlo como herramienta para medir el progreso, hacer que todos sean responsables, introducir disciplina en el proceso de desarrollo y establecer un cronograma de lanzamiento.
- Al comparar el esfuerzo estimado con la cantidad de tiempo real invertido, se pueden obtener mejores conocimientos sobre los procesos para analizarlo en las retrospectivas y realizar mejoras.

Backlog Grooming / Refinement

Estimación de Historias de Usuario

Scrum

The Benefits of Agile Estimation



Improve
decision-making
and sprint planning



Achieve better
coordination
among teams



Support risk
management for
on-time, quality
delivery

Backlog Grooming / Refinement

Estimación de Historias de Usuario

Scrum

Para estimar **Historias de Usuario** en un proyecto **Scrum** se deben tener en cuenta 4 elementos:

- **Complejidad de la historia.** Junto con el equipo se debe definir qué tan difícil es implementar la historia.
- **Cantidad de trabajo requerido.** Representa el esfuerzo que debe invertir el equipo para llevar a cabo la historia.
- **Conocimientos necesarios.** Habilidades técnicas, de diseño y de negocio, entre otras, para completar la historia.
- **Incertidumbre.** Acciones que se requieren para ejecutar la Historia de Usuario, aunque no se tenga claro el tiempo o esfuerzo real.

Backlog Grooming / Refinement Estimación de Historias de Usuario Scrum

- El proceso de estimación de las Historias de Usuario se hace a través de **puntos**. Estos puntos no están relacionadas con una escala de medición, no representan horas o días de trabajo, sino un estimado empírico con base en la experiencia del equipo.

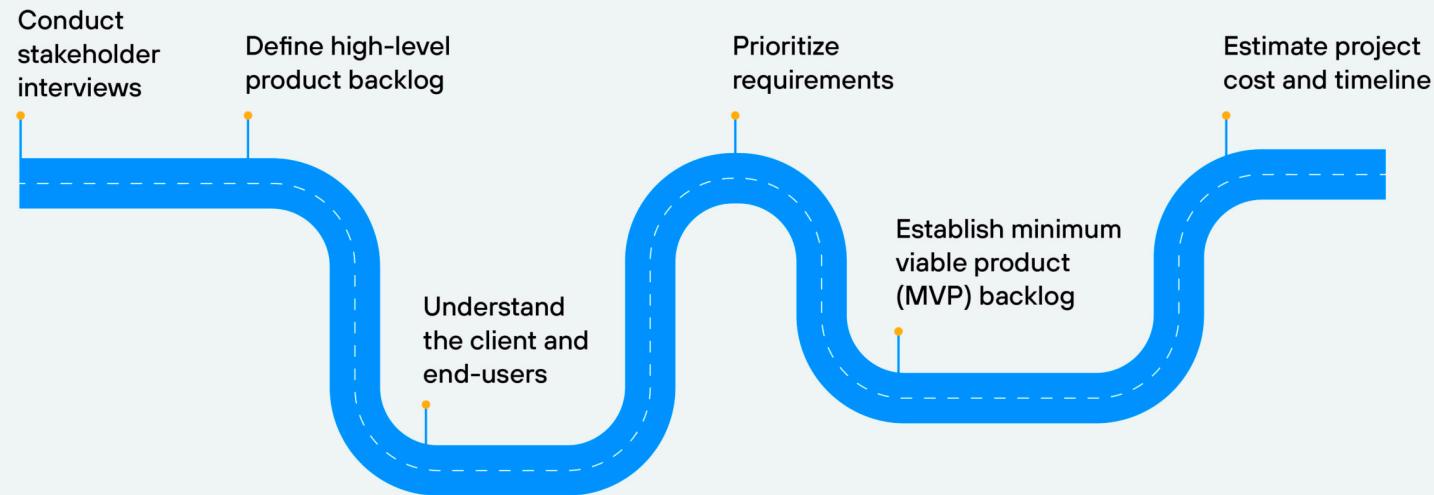
Backlog Grooming / Refinement

Estimación de Historias de Usuario

Scrum

Set the Stage for Successful User Story Estimation

Plan a product discovery phase to help the team understand project requirements:



Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario

Scrum

- Las historias de usuario son la base del proyecto. Al dividir un proyecto grande en partes más pequeñas, se puede obtener información granular sobre cada componente para realizar estimaciones precisas y planificar los sprints de manera productiva.
- La secuencia de Fibonacci (0, 1, 2, 3, 5, 8, 13, 20, 40, 100...) se utiliza a menudo en la estimación de puntos de una historia para representar el esfuerzo. La naturaleza exponencial permite a los equipos ser más realistas al estimar tareas grandes y complejas, que tienen más margen de error.

Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario

Scrum

Existen muchas técnicas de estimación ágiles, las más comunes son:

- Planning Poker
- T-shirt size
- Dot voting
- Bucket system
- Affinity mapping system

Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario - Planning Poker

Scrum

- Este método utiliza la secuencia de Fibonacci donde los valores de puntos de la historia del usuario se presentan como 0, 1, 2, 3, 5, 8, 13, 20, 40 y 100 en tarjetas, asociados con diferentes niveles de complejidad.
- El **Product Owner** o **Scrum Master** describe la historia del usuario y cada miembro del equipo selecciona en secreto un número de tarjeta para la estimación. Luego, todos revelan sus cartas. El número que obtenga más votos será la estimación final de la tarea.
- El planning poker ayuda a los equipos de desarrollo a establecer un entendimiento mutuo del proyecto y es particularmente útil para estimar una pequeña cantidad de elementos de trabajo.
- [Plan it poker](#)

Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario - T-shirt size

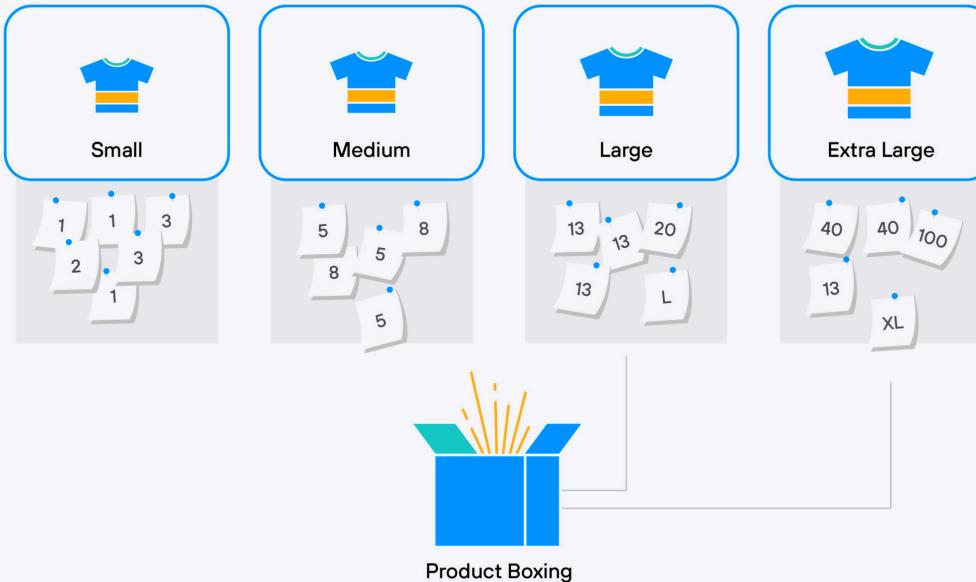
Scrum

- Esta técnica ágil de estimación implica asignar a cada historia de usuario una talla de camiseta (S, M, L, XL).
- El proceso ayuda a los miembros del equipo a lograr una comprensión general de los requisitos.
- Esta técnica produce una estimación rápida y aproximada del tiempo y el trabajo necesarios y es particularmente útil para gestionar un gran trabajo pendiente. También es útil para la estimación en las primeras etapas para obtener una vista panorámica rápidamente.

Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario - T-shirt size Scrum

T-shirt Size Method For Story Point Estimation



Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario - Dot voting

Scrum

- Utilizando esta metodología de estimación, los equipos ágiles organizan los elementos de trabajo de mayor a menor prioridad para decidir dónde concentrar su tiempo y esfuerzos.
- Se pone la descripción de cada historia en una tarjeta y a cada miembro se le dan x cantidad de puntos para que los coloque en las historias que consideren de mayor prioridad
- Esta técnica funciona en equipos scrum bien establecidos para poder priorizar cantidades grandes de historias

Backlog Grooming / Refinement

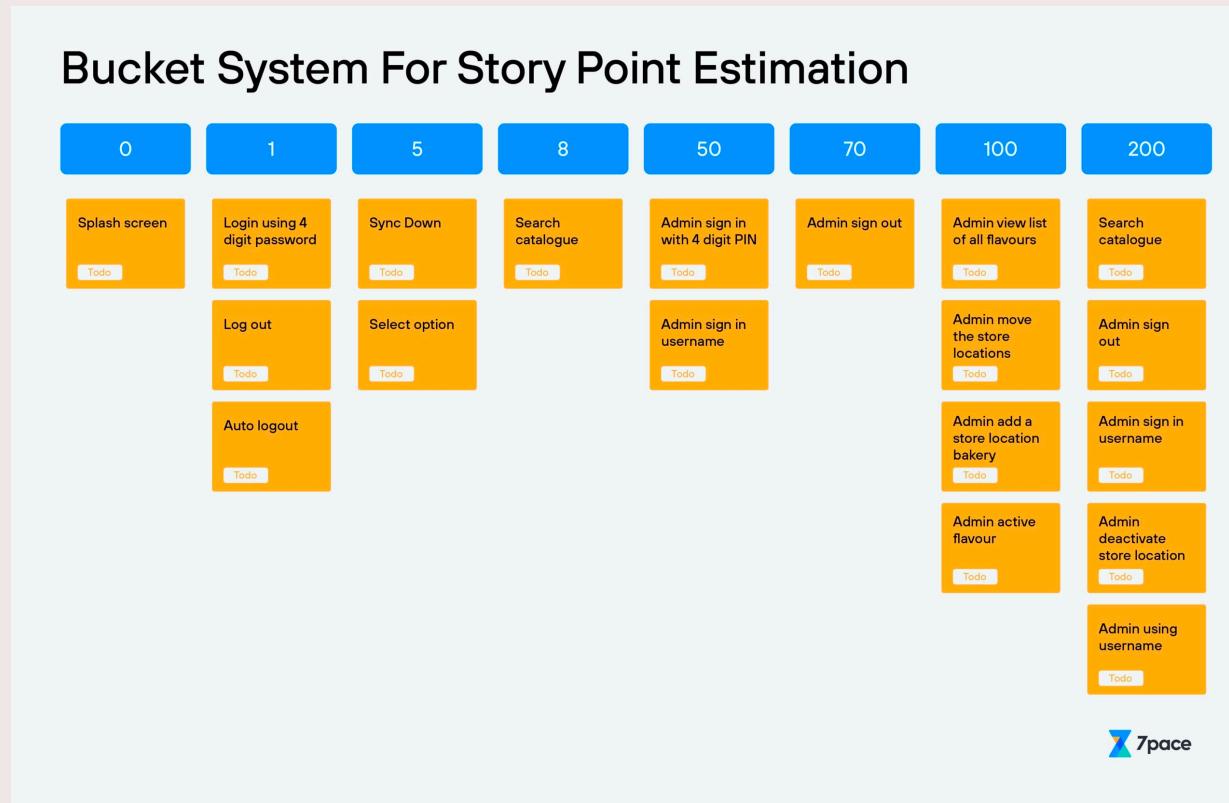
Técnicas de Estimación de Historias de Usuario - Bucket system

Scrum

- Se comienza el proceso de estimación configurando una fila de tarjetas (cubos) con valores en la secuencia de Fibonacci (0, 1, 2, 3, 4, 5, 8, 13, 20, 30, 50, 100, 200).
- Los miembros del equipo discuten un elemento de trabajo y colocan la historia del usuario en un depósito apropiado.
- Esta técnica ágil de estimación es adecuada para estimar una gran cantidad de artículos o proyectos a largo plazo.
- Los equipos de desarrollo pueden hacer estimaciones rápidas, mientras que el método es sencillo para quienes son nuevos en el mundo ágil.

Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario - Bucket system Scrum



Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario - Affinity mapping system Scrum

- Este método no utiliza un tamaño como tal.
- Se comienza colocando dos tarjetas en extremos opuestos de una pared y brindándoles a los miembros del equipo una lista de historias de usuarios
- El equipo organiza los elementos según el esfuerzo estimado.
- Luego, el **scrum master** posiciona los elementos en el **Product Backlog**
- Este enfoque ayuda a elaborar un plan a largo plazo para un proyecto con grandes retrasos. Es más adecuado para estimaciones en etapas iniciales y ayuda al equipo a obtener una comprensión mutua de los requisitos.

Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario

Scrum

Best User Story Estimation Techniques For:



A small number of work items: **Planning poker**



Getting a bird's-eye view: **T-shirt size estimation**



Prioritizing backlogs: **Dot voting**



A large number of work items: **Bucket system**



Early-stage estimations: **Affinity mapping system**

Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario

Scrum

- Sin importar la técnica usada para estimar, la parte más importantes son los **datos**
- Se pueden refinar las estimaciones tomando en cuenta información histórica y comparando las funcionalidades y requerimientos con cosas similares en proyectos pasados para reducir la cantidad de datos desconocidos
- Debido a que muchas decisiones son basadas en la linea del tiempo del proyecto, se necesita información confiable para traducir los **story points** a la cantidad de tiempo necesaria para completar una tarea

Backlog Grooming / Refinement

Técnicas de Estimación de Historias de Usuario

Scrum

- Existe un término denominado **Pace** que ayuda a correlacionar los **story points** con el tiempo utilizado.
- Después de analizar información de ejecuciones pasadas, se puede calcular el **Pace** (ritmo) para calcular cuanto tiempo se va a necesitar para una nueva historia

$$\text{Pace} = \frac{\text{Tracked Time}}{\text{Estimation of Effort or Story Points}}$$

Actividad 2

Revista - Scrum

- Hacer equipos de 5-6, cada equipo debe tener:
 - Un Scrum Master (elegido por el equipo)
 - Un Product Owner (elegido por el equipo)
 - Revistas
 - Tijeras
 - Pegamento
 - Hojas
 - Plumones

Actividad 2

Revista - Scrum

- Objetivo:**

- Construir una revista basada en recorte de imágenes y textos que estos sean pegados en las hojas tamaño carta.

- Requisitos:**

- La revista debe contar con un índice de artículos y numeración para cada hoja.
 - Todo a excepción del índice de artículos debe ser cortado y pegado

Actividad 2

Revista - Scrum

- Para construir las hojas se emplearán fotos de diferente tipo y renglones de texto cortados en forma de rectángulo.
- Se emplearán los siguientes tipos de fotos
 - A: Foto de una mujer
 - B: Fotos de un carro
 - C: Foto de un hombre
 - D: Foto de algo que sea diferente a A, B y C

Actividad 2

Revista - Scrum

- Las hojas de la revista pueden de las siguientes tipos:
 - Hoja Tipo 1: 1 foto + 1 descripción
 - Hoja Tipo 2: 2 fotos + 2 descripciones
 - Hoja Tipo 3: 3 fotos
 - Hoja Tipo 4: 5 fotos + 1 descripciones
 - Hoja Tipo 5: Índice
 - Hoja Tipo 6: 2 fotos
 - Hoja Tipo 7: 3 fotos + 3 descripciones

Actividad 2

Revista - Scrum

- Por lo tanto si se pide una Hoja Tipo 7 con fotos A, significa que la hoja debe contener:
 - 3 fotos de mujeres recortadas
 - 3 descripciones de texto recortadas

Actividad 2

Revista - Scrum

•Backlog:

- | | |
|---------------------------|----------------------------|
| 1.Tipo 1- 1A | 9.Tipo 2 - 1D, 1C |
| 2.Tipo 2 - 1B, 1D | 10.Tipo 3 - 1D, 1C, 1B |
| 3.Tipo 3 - 1A, 1B, 1C | 11.Tipo 4 - 2D, 2A, 1B |
| 4.Tipo 4 - 1A, 1B, 1C, 2D | 12.Tipo 6 - 1A, 1C |
| 5.Tipo 5 | 13.Tipo 7 - 1B, 1A, 1D |
| 6.Tipo 6 - 1C, 1D | 14.Tipo 4 - 2D, 1C, 1A, 1B |
| 7.Tipo 7 - 2A, 1D | 15.Tipo 7 - 1B, 1A, 1C |
| 8.Tipo 1- 1D | |

Actividad 2

Revista - Scrum

- **3 Sprints - 3 días cada sprint**
 - **Planning = 3 minutos**
 - **Duración del día 1= 7 minutos**
 - **Daily Stand up 1 = 2 minutos**
 - **Duración del día 2 = 7 minutos**
 - **Daily Stand up 2 = 2 minutos**
 - **Duración del día 3 = 7 minutos**
- **Daily Stand up 3 = 2 minutos**
- **Backlog Grooming = 2 minutos**
- **Sprint Review = 3 minutos**
- **Retrospective = 3 minutos**

Kanban

- “Kanban” es la palabra japonesa para “señal visual”
- Es un framework popular utilizado para implementar el desarrollo de software ágil y DevOps
- Requiere comunicación en tiempo real de la capacidad y total transparencia de trabajo
- Los elementos de trabajo se representan visualmente en un tablero Kanban (Kanban Board), lo que permite a los miembros del equipo ver el estado de trabajo en cualquier momento

Kanban

- Sirve para limitar el trabajo en curso o Work in Progress (WIP)
- Es una aproximación al proceso gradual y evolutivo
- Es un sistema de gestión de progreso visual que indica qué producir, cuándo producirlo y cuánto producir
- Es una aproximación al proceso gradual, evolutivo y al cambio de sistemas para las organizaciones

Kanban

- Se enfoca en cubrir un proyecto de manera global
- Es un sistema basado en la metodología ágil que busca conseguir un proceso organizado, productivo y eficiente al momento de realizar las diferentes tareas en un área de la empresa
- Es un método efectivo para la gestión de proyectos profesionales
- Produce solo lo necesario y toma el material requerido de la operación anterior, evita procesar material innecesario

Kanban

- Trabaja en pequeños bloques llamados tarjetas, que permite resolver todo de una manera más rápida
- Este método asegura un buen resultado y de calidad gracias a las diferentes fases con las que cuenta.

Kanban

- Para poder agilizar las tareas, la metodología sigue cinco principios básicos:
 - Visualización
 - Priorización
 - Mejora Continua
 - Liderazgo en todos los niveles
 - Calidad Garantizada

Kanban

Visualización

- Kanban es completamente transparente y permite comprender en que momento del desarrollo se encuentra el proyecto
- Tener acceso a todas las tareas en cualquier momento ayuda a organizar y hacer modificaciones para el correcto funcionamiento del equipo

Kanban

Priorización

- Cuando se checa el bloque de tareas pendientes, ya se tiene claro cual va a ser el siguiente tema a tratar
- La transparencia hace que sea posible una buena gestión del tiempo y colocar las tareas en orden coherente para facilitar el trabajo

Kanban

Mejora Continua

- Kanban fomenta el cambio continuo, ya que se trata de un sistema de trabajo inmediato, compuesto por tareas de corta duración
- El cambio evolutivo es incremental, no radical, para no dar a los equipos motivo de alarma o resistencia

Kanban

Liderazgo en todos los niveles

- La implementación exitosa de Kanban es un esfuerzo colectivo de todos los miembros del equipo y no depende de un jefe o un gerente
- Al respetar los roles y responsabilidades actuales, permite que los equipos identifiquen e implementen en colaboración todos los cambios

Kanban

Calidad Garantizada

- El principio de mejora continua, cuando está profundamente arraigado en la mentalidad del equipo, trae consigo mejores resultados de manera natural.
- Todo esto se traduce en una mejor calidad en los productos y servicios.

Kanban

Ventajas

- La utilización de Kanban en la gestión de proyectos ofrece una serie de ventajas para las organizaciones:
 - Permite un mejor seguimiento de los proyectos y tareas relacionadas
 - Evita los procesos innecesarios
 - Disminuye los tiempos de entrega
 - Mantiene al equipo motivado y les brinda las pautas para enfrentarse a los cambios y ser más eficaces.

Kanban Board

- Es una herramienta de gestión de proyectos ágil diseñada para ayudar a visualizar el trabajo, limitar el trabajo en curso y maximizar la eficiencia
- Ayuda a establecer el orden en el trabajo diario
- Usa tarjetas, columnas y mejora continua para ayudar a los equipos a comprometerse con la cantidad correcta de trabajo y hacerlo

Kanban Board Elementos

- Los tableros de Kanban se pueden dividir en 5 elementos:
 - Señales visuales
 - Columnas
 - Límites de trabajo en progreso
 - Punto de compromiso
 - Punto de entrega

Kanban Board Señales Visuales

- Se usan tarjetas visuales (adhesivos, boletos u otros)
- Los equipos Kanban escriben todos sus proyectos y elementos de trabajo en tarjetas, generalmente una por tarjeta.
- Para equipos ágiles, cada tarjeta podría encapsular una historia de usuario
- Una vez en el tablero, estas señales visuales ayudan a los compañeros de equipo y las partes interesadas a comprender rápidamente en qué está trabajando el equipo.

Kanban Board Columnas

- Cada columna representa una actividad específica que juntas componen un "flujo de trabajo".
- Las tarjetas fluyen por el flujo de trabajo hasta su finalización.
- Los flujos de trabajo pueden ser tan simples como "Pendiente", "En curso", "Completo" o mucho más complejos.

Kanban Board

Límites de trabajo en curso (WIP)

- Son el número máximo de tarjetas que pueden estar en una columna en un momento dado.
- Una columna con un límite de WIP de tres no puede tener más de tres tarjetas.
- Cuando la columna está “al máximo”, el equipo necesita pulir esas tarjetas y hacerlas avanzar antes de que las tarjetas nuevas puedan pasar a esa etapa del flujo de trabajo.

Kanban Board

Límites de trabajo en curso (WIP)

- Estos límites de WIP son fundamentales para exponer los cuellos de botella en el flujo de trabajo y maximizar el flujo.
- Los límites de WIP le dan una señal de advertencia temprana de que se comprometió a trabajar demasiado.

Kanban Board

Punto de Compromiso

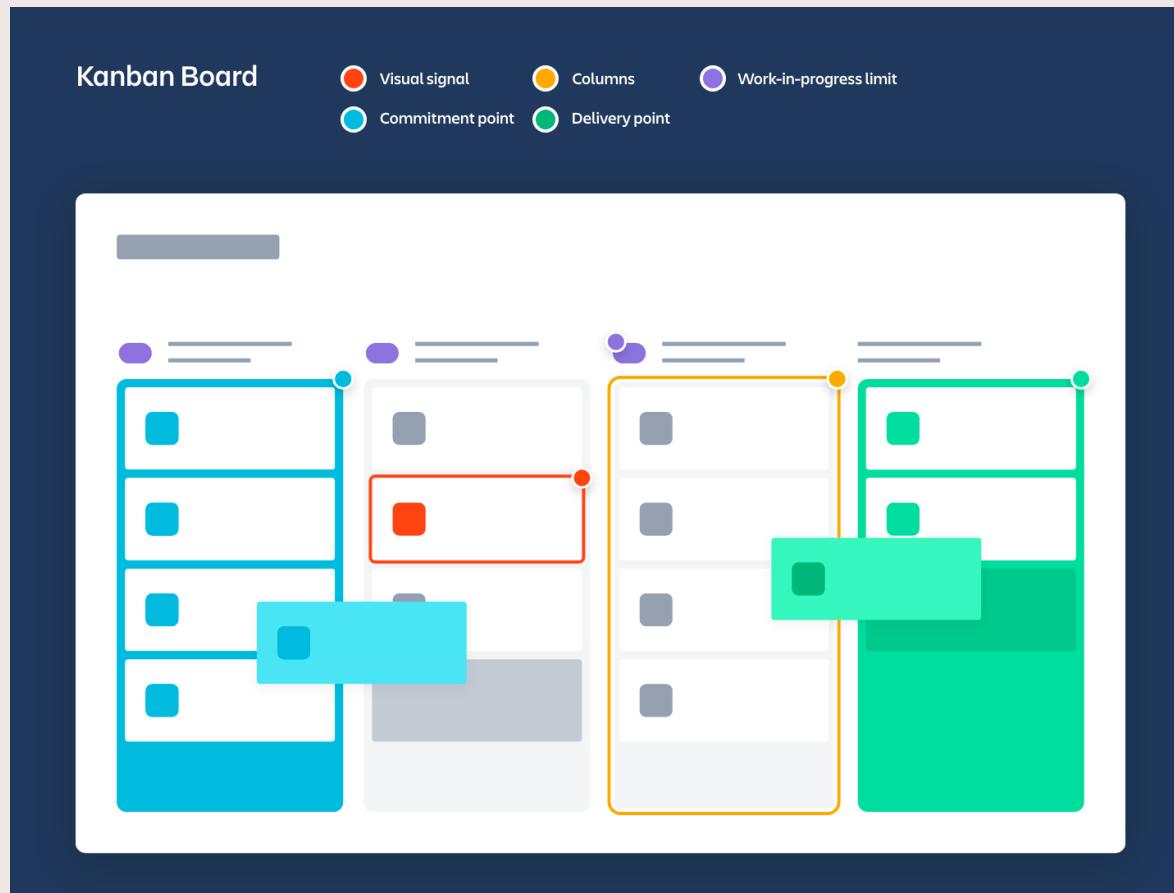
- Los equipos Kanban a menudo tienen un retraso en su tablero.
- Aquí es donde los clientes y compañeros de equipo ponen ideas para proyectos que el equipo puede recoger cuando estén listos.
- El punto de compromiso es el momento en que el equipo recoge una idea y comienza el trabajo en el proyecto.

Kanban Board

Punto de Entrega

- Es el final del flujo de trabajo de un equipo kanban.
- Es cuando el producto o servicio está en manos del cliente.
- El objetivo del equipo es llevar las cartas desde el punto de compromiso hasta el punto de entrega lo más rápido posible.
- El tiempo transcurrido entre los dos se denomina Lead Time. Los equipos Kanban mejoran continuamente para disminuir su tiempo de entrega tanto como sea posible.

Kanban Board



Actividad

Pizza Delivery

- 5 Equipos
- Kanban board
- Ordenes - Establecer compromiso
- Iteraciones de tiempo sin establecer
- Hacer rebanadas de pizza dependiendo el pedido

- **Pizza hawaiana**

- 3 pedazos de piña
- 3 pedazos de jamón
- Base con salsa de jitomate

- **Pizza rucula**

- 5 pedazos de Arugula
- 3 pedazos de jitomate
- Base con salsa de jitomate
- **Nota:** Arugula se pone después de haber estado en el horno, si no, se quema

Actividad

Pizza Delivery

- Un pedazo de pizza - Papel
- Salsa - rojo (colorear)
- Piña - amarillo (papel)
- Jamón - rosa (papel)
- Arugula - verde (papel)
- Jitomate - rojo (papel)
- Horno - 30 segundos, máximo 3 rebanadas de pizza, no se puede agregar ni quitar dentro de esos 30 segundos
- Dejar orilla como pizza - máximo 1 cm

Actividad

Pizza Delivery - Puntuación

- Rebanada de pizza terminada = 10 puntos
- Pizza base = -4 puntos
- Toppings = -1 punto c/u

Kanban vs Scrum

- Kanban se trata de visualizar su trabajo, limitar el trabajo en progreso y maximizar la eficiencia.
- Los equipos Kanban se centran en reducir el tiempo que tarda un proyecto o la historia de usuario de principio a fin. Para ello, utilizan un tablero kanban y mejoran continuamente su flujo de trabajo.

Kanban vs Scrum

- Los equipos de Scrum se comprometen a enviar software de trabajo a través de intervalos establecidos llamados sprints.
- Su objetivo es crear ciclos de aprendizaje para recopilar e integrar rápidamente los comentarios de los clientes. Los equipos de Scrum adoptan roles específicos, crean artefactos especiales y celebran ceremonias regulares para que las cosas sigan avanzando.

Kanban vs Scrum

	Scrum	Kanban
Cadencia	Sprints regulares	Flujo continuo
Metodología de lanzamiento	Al final de cada sprint	Entrega continua
Roles	Product Owner, Scrum Master, Scrum Team	No se requieren roles
Métricas	Velocidad	Plazo de ejecución, tiempo del ciclo, WIP
Cambio en la filosofía	Los equipos no deben realizar cambios durante el sprint	El cambio puede ocurrir en cualquier momento

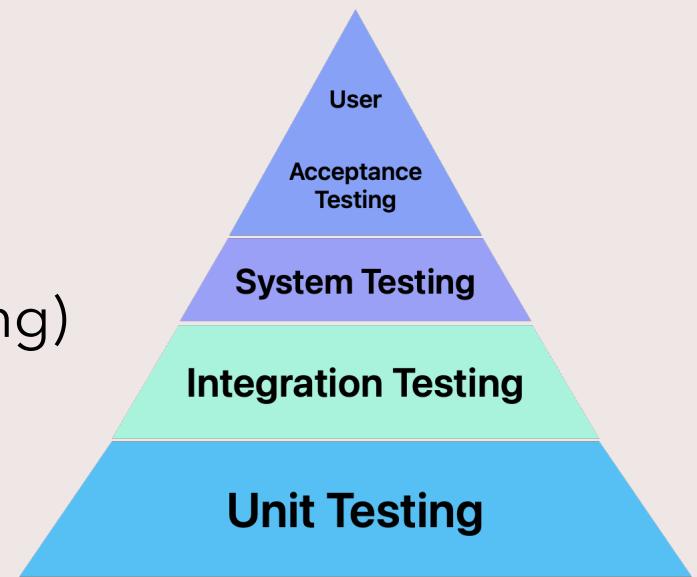
Niveles de Pruebas de Software

Niveles de Pruebas

- Un nivel de prueba es un grupo de actividades que están organizadas y gestionadas de forma conjunta
- Un nivel de prueba está vinculado a las responsabilidades en un proyecto
- Los niveles de prueba están relacionados con otras actividades del ciclo de vida de desarrollo de software

Niveles de Pruebas

- Los niveles de prueba son:
 - Prueba de Componente (Unit Testing)
 - Prueba de Integración (Integration Testing)
 - Prueba de Sistema (System Testing)
 - Prueba de Aceptación de Usuario (User Acceptance Testing)



Niveles de Pruebas

- Los niveles de prueba se caracterizan por los siguientes atributos:
 - Objetivos específicos
 - Bases de pruebas, referenciadas para generar casos de prueba
 - Objeto de prueba, que es lo que se está probando
 - Defectos y fallos característicos
 - Enfoques y responsabilidades específicos

Prueba de Componente

Unit Testing

- Se centra en los componentes que se pueden probar por separado
- Objetivos de la Prueba de Componente
 - Reducir el riesgo
 - Verificar que los comportamientos funcionales y no funcionales del componente son los diseñados y especificados
 - Generar confianza en la calidad del componente
 - Prevenir la propagación de defectos a niveles de pruebas superiores

Prueba de Componente

Unit Testing

- Bases de prueba
- Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba para la prueba de componente son los siguientes:
 - Diseño detallado
 - Código
 - Modelos de datos
 - Especificaciones de los componentes

Prueba de Componente

Unit Testing

- Objetos de prueba
- Los objetos de prueba característicos para la prueba de componente incluyen:
 - Componentes, unidades o módulos
 - Código y estructuras de datos
 - Clases
 - Módulos de base de datos

Prueba de Componente

Unit Testing

- Defectos y fallos característicos
- Ejemplos de defectos y fallos característicos de la prueba de componente incluyen:
 - Funcionamiento incorrecto
 - Problemas de flujo de datos
 - Código y lógica incorrectos
- Por lo general los defectos se corrigen tan pronto como se detectan

Prueba de Componente

Unit Testing

- Enfoques y responsabilidades específicos
- El desarrollador que escribió el código, realiza la prueba de componente
- Los desarrolladores pueden alternar el desarrollo de componentes con la búsqueda y corrección de defectos
- Los desarrolladores escriben y ejecutan pruebas después de haber escrito el código de ese componente

Prueba de Integración

Integration Testing

- La prueba de integración se centra en las interacciones entre componentes o sistemas
- Objetivos de la Prueba de Integración
- Reducir el riesgo
- Verificar que los comportamientos funcionales y no funcionales de las interfaces sean los diseñados y especificados
- Generar confianza en la calidad de las interfaces
- Encontrar defectos (interfaces o componentes)
- Prevenir la propagación de defectos a niveles de prueba superiores

Prueba de Integración

Integration Testing

- Hay dos tipos de pruebas integración diferentes:
 - **Prueba de integración de componentes**
 - **Prueba de integración de sistemas**

Prueba de Integración de componentes

- Se centra en las interacciones e interfaces entre los componentes integrados
- Se realiza después de las pruebas de componentes y en general esta automatizada
- En el desarrollo iterativo e incremental, suele formar parte del proceso de integración continua

Prueba de Integración de sistemas

- Se centra en las interacciones e interfaces entre sistemas, paquetes y microservicios
- Puede incluir interacciones con interfaces o sistemas proporcionadas por organizaciones externas
- Pueden realizarse después de la prueba de sistema o en paralelo con las actividades de prueba de sistema en curso

Prueba de Integración

Integration Testing

- Bases de prueba
- Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba para la prueba de integración son los siguientes:
 - Diseño de software y sistemas
 - Diagramas de secuencia
 - Especificaciones de interfaz y protocolos de comunicación
 - Casos de uso
 - Arquitectura a nivel de componente o sistema
 - Flujos de trabajo
 - Definiciones de interfaces externas

Prueba de Integración

Integration Testing

- Objetos de prueba
- Los objetos de prueba característicos para la prueba de integración incluyen:
 - Subsistemas
 - Bases de datos
 - Infraestructura
 - Interfaces
 - Interfaces de programación de aplicaciones (API)
 - Microservicios

Prueba de Integración

Integration Testing

- Defectos y fallos característicos
- Ejemplos de defectos y fallos característicos de la prueba de integración de componentes incluyen:
 - Datos incorrectos, datos faltantes o codificación incorrecta de datos
 - Secuenciación o sincronización incorrecta de las llamadas a la interfaz
 - Incompatibilidad de la interfaz
 - Fallos en la comunicación entre componentes
 - Fallos de comunicación entre componentes no tratados o tratados de forma incorrecta
 - Suposiciones incorrectas sobre el significado, las unidades o los límites de los datos que se transmiten entre componentes

Prueba de Integración

Integration Testing

- Defectos y fallos característicos
- Ejemplos de defectos y fallos característicos de la prueba de integración de sistemas incluyen:
 - Estructuras de mensajes inconsistentes entre sistemas
 - Datos incorrectos, datos faltantes o codificación incorrecta de datos
 - Incompatibilidad de la interfaz
 - Fallos en la comunicación entre sistemas
 - Fallos de comunicación entre componentes no tratados o tratados de forma incorrecta
 - Suposiciones incorrectas sobre el significado, las unidades o los límites de los datos que se transmiten entre sistemas
 - Incumplimiento de las normas de seguridad obligatorias

Prueba de Integración

Integration Testing

- Enfoques y Responsabilidades Específicos
- Deben concentrarse en la integración hecha
- La prueba debe centrarse en la comunicación entre los módulos, no en la funcionalidad de los módulos individuales
- Se puede utilizar los tipos de prueba funcional, no funcional y estructural
- La prueba de integración de componentes suele ser responsabilidad de los desarrolladores
- La prueba de integración de sistemas es responsabilidad de los testers

Prueba de Integración

Integration Testing

- Los testers que realizan la prueba de integración de sistemas deberían entender la arquitectura del sistema y deberían haber influido en la planificación de la integración
- Para simplificar el aislamiento de defectos y detectar defectos en forma temprana, la integración debe ser normalmente incremental
- Un análisis de riesgo de las interfaces más complejas puede ayudar a centrar la prueba de integración

Prueba de Sistema

System Testing

- Se centra en el comportamiento y las capacidades de todo un sistema o producto teniendo en cuenta las tareas de extremo a extremo que el sistema puede realizar y los comportamientos no funcionales que exhibe mientras se realizan esas tareas
- Objetivos de la Prueba de Sistema
 - Reducir el riesgo
 - Verificar que los comportamientos funcionales y no funcionales del sistema son los diseñados y especificados
 - Validar que el sistema está completo y que funcionará como se espera
 - Generar confianza en la calidad del sistema considerado como un todo
 - Encontrar defectos
 - Prevenir la propagación de defectos a nivel de pruebas superiores o a producción

Prueba de Sistema System Testing

- Produce información que es utilizada para tomar decisiones con respecto a lanzamiento
- Esta prueba, puede satisfacer requisitos o estándares legales o regulatorios
- El entorno de prueba debe corresponder, en condiciones ideales al entorno objetivo final o entorno de producción

Prueba de Sistema System Testing

- Bases de prueba
- Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba para la prueba de sistema son los siguientes:
 - Especificaciones de requisitos del sistema y del software (funcionales y no funcionales)
 - Informe de análisis de riesgos
 - Casos de uso
 - Épicas e historias de usuario
 - Diagramas de estado
 - Manuales del sistema y del usuario

Prueba de Sistema System Testing

- Objetos de prueba
- Los objetos de prueba característicos para la prueba de sistema incluyen:
 - Aplicaciones
 - Sistemas hardware / software
 - Sistemas operativos
 - Sistema sujeto a prueba
 - Configuración del sistema y datos de la configuración

Prueba de Sistema System Testing

- Defectos y fallos característicos
- Ejemplos de defectos y fallos característicos de la prueba de sistema incluyen:
 - Cálculos incorrectos
 - Comportamiento funcional o no funcional del sistema incorrecto o inesperado
 - Control y/o flujos de datos incorrectos dentro del sistema
 - Incapacidad para llevar a cabo de forma adecuada y completa las tareas funcionales de extremo a extremo
 - Fallo del sistema para operar correctamente en el entorno de producción
 - Fallo en el sistema para funcionar como se describe en los manuales del sistema y de usuario

Prueba de Sistema System Testing

- Enfoques y Responsabilidades Específicos
- Deben concentrarse en el comportamiento global y extremo a extremo del sistema en su conjunto, tanto funcional como no funcional
- Se debe identificar la técnica más adecuada para testear
- Los testers llevan a cabo esta prueba
- Los defectos en las especificaciones pueden llevar a una falta de comprensión o desacuerdos sobre el comportamiento esperado del sistema

Prueba de Aceptación

Acceptance Testing

- Al igual que la prueba de sistema, se centra en el comportamiento y las capacidades de todo un sistema o producto
- Objetivos de la Prueba de Aceptación
 - Establecer confianza en la calidad del sistema en su conjunto
 - Validar que el sistema está completo y que funcionará como se espera
 - Verificar que los comportamientos funcionales y no funcionales del sistema sean los especificados

Prueba de Aceptación

Acceptance Testing

- Puede producir información para evaluar el grado de preparación del sistema para su despliegue y uso por parte del cliente
- Los defectos pueden encontrarse durante la prueba de aceptación, pero encontrar defectos no suele ser un objetivo
- Encontrar muchos defectos en una prueba de aceptación puede considerarse un riesgo importante para el proyecto
- La prueba de aceptación también puede satisfacer requisitos o normas legales o reglamentarias

Prueba de Aceptación

Acceptance Testing

- Las formas comunes de pruebas de aceptación, son las siguientes:
 - Prueba de aceptación de usuario
 - Prueba de aceptación operativa
 - Prueba de aceptación contractual y de regulación
 - Prueba alfa y beta

Prueba de aceptación de usuario

UAT Testing

- Se centra normalmente en la validación de la identidad para el uso de sistema por parte de los usuarios
- El objetivo principal es crear confianza en que los usuarios pueden utilizar el sistema para satisfacer sus necesidades, cumplir con los requisitos y realizar los procesos de negocio con la mínima dificultad, coste y riesgo

Prueba de aceptación operativa

- Es por parte del personal de operaciones o de la administración del sistema se realiza en un entorno de producción (simulado)
- La prueba se centra en los aspectos operativos y puede incluir:
 - Prueba de copia de seguridad y restauración
 - Instalación, desinstalación y actualización
 - Recuperación ante desastres

Prueba de aceptación operativa

- Gestión de usuarios
- Tareas de mantenimiento
- Carga de datos y tareas de migración
- Comprobación de vulnerabilidades de seguridad
- Prueba de rendimiento

Prueba de aceptación operativa

- El objetivo de la prueba de aceptación operativa es generar confianza de que los operadores o administradores del sistema pueden mantener el sistema funcionando correctamente para los usuarios en el entorno operativo, incluso en condiciones excepcionales o difíciles

Prueba de aceptación contractual y normativa

- Se realiza en función de los criterios de aceptación del contrato para el desarrollo de software a medida
- Los criterios de aceptación deben definirse cuando las partes acuerdan el contrato
- Suele ser realizada por usuarios o probadores independientes
- Se lleva a cabo con respecto a cualquier norma que deba cumplirse como las gubernamentales, legales o de seguridad física

Prueba de aceptación contractual y normativa

- En ocasiones los resultados son presenciados o auditados por agencias reguladoras
- El principal objetivo de la prueba de aceptación contractual y normativa es crear confianza que se ha logrado la conformidad contractual o normativa

Pruebas alfa y beta

- Suelen ser utilizadas por los desarrolladores de software comercial de distribución masiva que desean obtener retroalimentación de los usuarios, clientes y/u operadores potenciales o existentes antes de que el producto de software sea puesto en el mercado

Pruebas alfa y beta

- La prueba alfa se realiza en las instalaciones de la organización que desarrolla
- La prueba beta es realizada en las instalaciones de los probadores
- La prueba beta puede tener lugar después de la alfa o puede ocurrir sin que se haya realizado una alfa previamente

Prueba de Aceptación

Acceptance Testing

- Bases de prueba
- Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba para la prueba de aceptación son los siguientes:
 - Proceso de negocio
 - Requisitos de usuario o de negocio
 - Normativas, contratos legales y estándares
 - Casos de uso
 - Requisitos del sistema
 - Documentación del sistema o del usuario
 - Procedimientos de instalación
 - Informes de análisis de riesgos

Prueba de Aceptación Acceptance Testing

- Bases de prueba
- Además, como base de prueba para derivar casos de prueba para la prueba de aceptación operativa, se pueden utilizar los siguientes:
 - Procedimiento de copia de seguridad y recuperación
 - Procedimientos de recuperación de desastres
 - Requisitos no funcionales
 - Documentación de operaciones
 - Instrucciones de despliegue e instalación
 - Objetivos de rendimiento
 - Paquetes de base de datos
 - Estándares o reglamentos de seguridad

Prueba de Aceptación

Acceptance Testing

- Objetos de prueba
- Los objetos de prueba característicos para cualquier forma de prueba de aceptación incluyen:
 - Sistema sujeto a prueba
 - Configuración del sistema y datos de configuración
 - Procesos de negocio para un sistema totalmente integrado
 - Sistemas de recuperación y sitios críticos
 - Procesos operativos y mantenimiento
 - Formularios
 - Informes
 - Datos de producción existentes y transformados

Prueba de Aceptación

Acceptance Testing

- Defectos y fallos característicos
- Ejemplos de defectos y fallos característicos de la prueba de aceptación incluyen:
 - Los flujos de trabajo del sistema no cumplen con los requisitos del negocio o de usuario
 - Las reglas de negocio no se implementan de forma correcta
 - El sistema no satisface los requisitos contractuales reglamentarios
 - Fallos no funcionales como vulnerabilidades de seguridad, eficiencia de rendimiento inadecuada bajo cargas elevadas o funcionamiento inadecuado en una plataforma soportada

Prueba de Aceptación

Acceptance Testing

- Enfoques y Responsabilidades Específicos
- Es responsabilidad de los clientes, usuarios de negocio, propietarios de producto u operadores de un sistema
- Se considera como el último nivel de prueba en un ciclo de vida de desarrollo secuencial pero puede ocurrir en otros momentos, como:
 - La prueba de aceptación de un producto de software comercial de distribución masiva puede tener lugar cuando se instala o integra
 - La prueba de aceptación de una mejora funcional nueva puede tener lugar antes de la prueba de sistema

Prueba de Aceptación

Acceptance Testing

- En el desarrollo iterativo los equipos pueden emplear varias formas de pruebas de aceptación durante y al final de cada iteración
 - Validar una nueva característica en relación con sus criterios de aceptación
 - Validar que una nueva característica satisface las necesidades de los usuarios

Tipos de Prueba

Tipos de Prueba

- Un tipo de prueba es un grupo de actividades de prueba destinadas a probar características específicas de un sistema de software o de una parte de sistema, basadas en objetivos de prueba específicos

Tipos de Prueba

- Los objetivos de prueba pueden incluir:
 - Evaluar las características de calidad funcional, como la compleción, corrección y pertinencia
 - Evaluar las características no funcionales de calidad, tales como fiabilidad, eficiencia de desempeño, seguridad, compatibilidad y usabilidad
 - Evaluar si la estructura o arquitectura del componente o sistema es correcta, completa y según lo especificado
 - Evaluar los efectos de los cambios, tales como confirmar que los defectos han sido corregidos (prueba de confirmación) y buscar cambios no deseados en el comportamiento que resulten de cambios en el software o entorno de prueba (prueba de regresión)

Prueba Funcional

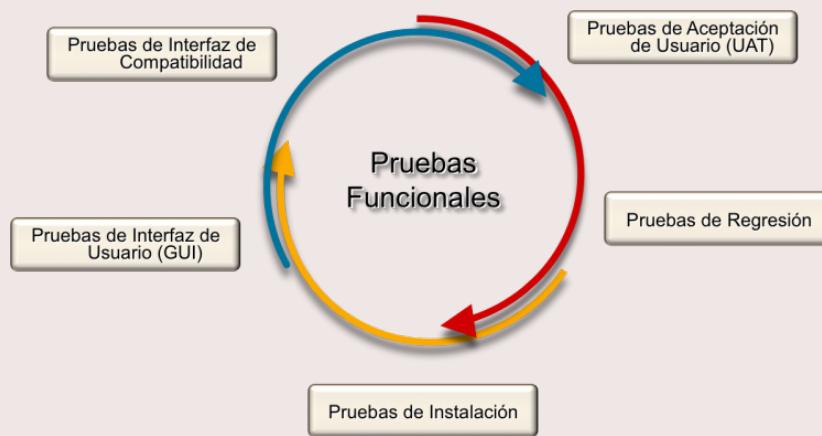
- Incluye pruebas que evalúan las funciones que el sistema debe realizar
- Los requisitos funcionales pueden estar descritos en productos de trabajo tales como especificaciones de requisitos de negocio, épicos, historias de usuario, casos de uso o especificaciones funcionales
- Las funciones describen “que” debe hacer el sistema

Prueba Funcional

- Se deben realizar pruebas funcionales en todos los niveles de prueba, aunque el enfoque es diferente en cada nivel
- La prueba funcional observa el comportamiento del software, por lo que pueden utilizar técnicas de caja negra para obtener las condiciones y los casos de prueba para la funcionalidad del componente o sistema

Prueba Funcional

- El diseño y la ejecución de pruebas funcionales pueden implicar competencias o conocimientos especiales como el conocimiento del problema de negocio específico que resuelve el software o el papel particular que desempeña el software



Prueba No Funcional

- Evalua las características de sistemas y software como la usabilidad, la eficiencia del desempeño o la seguridad
- La prueba no funcional prueba “que tan bien” se comporta el sistema

Prueba No Funcional

- Se deben realizar pruebas no funcionales en todos los niveles de prueba y se deben realizar tan pronto como sea posible
- El descubrimiento tardío de defectos no funcionales puede ser extremadamente peligroso para el éxito de un sistema
- Se pueden utilizar técnicas de caja negra para obtener condiciones de prueba y casos de prueba para pruebas no funcionales

Prueba No Funcional

- El diseño y la ejecución de la prueba no funcional pueden implicar competencias o conocimientos especiales, como el conocimiento de las debilidades inherentes a un diseño o tecnología o la base de usuarios concreta



Prueba de caja blanca

- La prueba de caja blanca obtiene pruebas basadas en la estructura interna del sistema o en su implementación
- La estructura interna puede incluir código, arquitectura, flujos de trabajo y/o flujos de datos dentro del sistema

Prueba de caja blanca

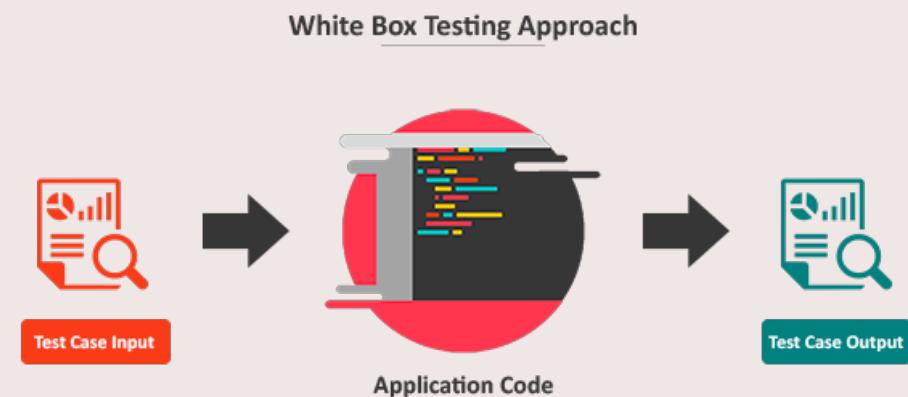
- En el nivel de prueba de componente, la cobertura del código se basa en el porcentaje de código del componente que ha sido probado y puede medirse en términos de diferentes aspectos del código, tales como el porcentaje de sentencias ejecutables probadas en el componente o el porcentaje de resultados de decisión probados

Prueba de caja blanca

- En el nivel de prueba de integración la prueba de caja blanca puede basarse en la arquitectura del sistema, como las interfaces entre componentes y la cobertura estructural puede medirse en términos del porcentaje de interfaces practicadas por las pruebas

Prueba de caja blanca

- El diseño y la ejecución de pruebas de caja blanca pueden implicar competencias o conocimientos especiales, como la forma en que se construye el código, como se almacenan los datos y como utilizar las herramientas de cobertura e interpretar correctamente sus resultados



Prueba Asociada al Cambio

- Cuando se realizan cambios en un sistema, ya sea para corregir un defecto o debido a una funcionalidad nueva o modificada se debe probar para confirmar que los cambios han corregido el defecto o implementado la funcionalidad correctamente y no han causado ninguna consecuencia adversa imprevista

Prueba Asociada al Cambio

- Existen dos tipos de pruebas asociadas al cambio:
 - Prueba de confirmación
 - Prueba de regresión

Prueba de confirmación

- Una vez corregido un defecto, el software se prueba con todos los casos de prueba que fallaron debido al defecto
- Se puede probar con nuevos casos de prueba
- Los mínimos pasos a seguir, es los pasos con los que se reproduciría el error
- El objetivo de una prueba de confirmación es confirmar que el defecto original se ha solucionado de forma satisfactoria

Prueba de regresión

- Es posible que un cambio hecho en una parte del código, ya sea una corrección u otro tipo de cambio, pueda afectar accidentalmente el comportamiento de otras partes del código, ya sea dentro del mismo componente, en otros componentes del mismo sistema o incluso en otros sistemas.

Prueba de regresión

- Los cambios pueden incluir modificaciones en el entorno, tales como una nueva versión de sistema operativo o de una gestión de bases de datos
- La prueba de regresión implica la realización de pruebas para detectar estos efectos secundarios no deseados

Prueba Asociada al Cambio

- La prueba de confirmación y la prueba de regresión se realizan en todos los niveles de prueba
- Especialmente en los ciclos de vida de desarrollo iterativos e incrementar, las nuevas características, los cambios en las características existentes y la refactorización del código dan como resultado cambios frecuentes en el código, lo que requiere pruebas asociadas al cambio

Prueba Asociada al Cambio

- La prueba de confirmación y la prueba de regresión son muy importantes sobre todo en los sistemas de internet de las cosas, donde los objetos individuales se actualizan o reemplazan con frecuencia
- Los juegos de prueba de regresión se ejecutan muchas veces y generalmente evolucionan lentamente, por lo que la prueba de regresión es un fuerte candidato para automatización

Tipos de Prueba y Niveles de Prueba

- Es posible realizar cualquiera de los tipos de prueba, en cualquier nivel de prueba
- Ejemplo: Una aplicación bancaria
- No es necesario que cada tipo de prueba este presente en todos los niveles

Aplicación Bancaria

Pruebas Funcionales

- *Prueba de Componente:*
 - Las pruebas se diseñan en base a la forma en que un componente debe calcular el interés compuesto
- *Prueba de Integración de Componentes:*
 - Las pruebas se diseñan en función de como la información de la cuenta capturada en la interfaz de usuario se transfiere a la lógica de negocio
- *Prueba de Sistema:*
 - Las pruebas se diseñan en base a como los titulares de cuentas pueden solicitar una linea de crédito sobre sus cuentas corrientes

Aplicación Bancaria

Pruebas Funcionales

- *Prueba de Integración de Sistemas:*

- Las pruebas se diseñan en función de cómo el sistema utiliza un microservicio externo para comprobar la calificación crediticia del titular de una cuenta

- *Prueba de Aceptación:*

- Las pruebas se diseñan en base a la forma en que el empleado de banco tramita la aprobación o rechazo de una solicitud de crédito

Aplicación Bancaria

Pruebas No Funcionales

- *Prueba de Componente:*
 - Las pruebas de rendimiento están diseñadas para evaluar el número de ciclos de CPU necesarios para realizar un cálculo de intereses totales complejo
- *Prueba de Integración de Componentes:*
 - Las pruebas de seguridad están diseñadas para vulnerabilidades de desbordamiento de memoria intermedia debido a que los datos pasan de la interfaz de usuario a la lógica de negocio
- *Prueba de Sistema:*
 - Las pruebas de portabilidad están diseñadas para comprobar si la capa de presentación funciona en todos los navegadores y dispositivos móviles soportados

Aplicación Bancaria

Pruebas No Funcionales

- *Prueba de Integración de Sistemas:*
 - Las pruebas de fiabilidad están diseñadas para evaluar la solidez del sistema en caso de que el microservicio de calificación crediticia falle en responder
- *Prueba de Aceptación:*
 - Las pruebas de usabilidad están diseñadas para evaluar la accesibilidad de la interfaz de procesamiento de crédito bancario para personas con discapacidades

Aplicación Bancaria

Pruebas De Caja Blanca

- *Prueba de Componente:*
 - Las pruebas de rendimiento están diseñadas para lograr una cobertura completa de sentencia y decisión para todos los componentes que realizan cálculos financieros
- *Prueba de Integración de Componentes:*
 - Las pruebas están diseñadas para practicar como cada pantalla de la interfaz del navegador pasa datos a la siguiente pantalla y a la lógica de negocio
- *Prueba de Sistema:*
 - Las pruebas de portabilidad están diseñadas para cubrir las secuencias de páginas web que pueden ocurrir durante una solicitud de línea de crédito

Aplicación Bancaria

Pruebas De Caja Blanca

- *Prueba de Integración de Sistemas:*

- Las pruebas están diseñadas para practicar todos los tipos de consulta posibles que se envían al microservicio de calificación crediticia

- *Prueba de Aceptación:*

- Las pruebas están diseñadas para cubrir todas las estructuras de archivos de datos financieros soportados y rangos de valores para transferencias de banco a banco

Aplicación Bancaria

Pruebas Asociadas al Cambio

- *Prueba de Componente:*
 - Se construyen pruebas de regresión automatizadas para cada componente y se incluyen dentro del marco de integración continua
- *Prueba de Integración de Componentes:*
 - Las pruebas están diseñadas para confirmar la corrección de defectos relacionados con la interfaz a medida que las correcciones se registran en el repositorio de código
- *Prueba de Sistema:*
 - Todas las pruebas de un flujo de trabajo dado se ejecutan de nuevo si cambia alguna pantalla de ese flujo de trabajo

Aplicación Bancaria

Pruebas Asociadas al Cambio

- *Prueba de Integración de Sistemas:*
 - Las pruebas de la aplicación que interactúa con el microservicio de calificación de crédito se vuelven a ejecutar diariamente como parte del despliegue continuo de ese microservicio
- *Prueba de Aceptación:*
 - Todas las pruebas que han fallado previamente se vuelven a ejecutar después de que se haya corregido un defecto encontrado en la prueba de aceptación

Prueba de Mantenimiento

- Ya desplegado en producción, es necesario mantener el software y los sistemas
- Los cambios de cualquier tipo son casi inevitables, ya sea para corregir defectos, agregar, eliminar o alterar funcionalidades
- El mantenimiento, también es necesario para preservar o mejorar las características de calidad no funcionales a lo largo de su vida útil (desempeño, fiabilidad, seguridad, compatibilidad y portabilidad)

Prueba de Mantenimiento

- Cuando se realizan cambios como parte de mantenimiento, se debe realizar una prueba de mantenimiento para:
 - Evaluar el éxito con el que se realizaron los cambios
 - Comprobar los posibles efectos secundarios en las partes inalteradas

Prueba de Mantenimiento

- La prueba de mantenimiento se centra en probar los cambios en el sistema, así como en probar las piezas no modificadas que podrían haberse visto afectadas por los cambios
- El mantenimiento puede incluir lanzamientos planificados y no planificados

Prueba de Mantenimiento

- El lanzamiento de un mantenimiento, puede requerir probar el mantenimiento en múltiples niveles de prueba según su alcance
- El alcance de la prueba de mantenimiento depende de:
 - El grado de riesgo
 - El tamaño del sistema existente
 - El tamaño del cambio

Activadores para el Mantenimiento

- Existen diferentes razones por las que el mantenimiento del software y su pruebas se pueden llevar a cabo, tanto en modificaciones planeadas, como no planeadas

Activadores para el Mantenimiento

- Se pueden clasificar los activadores de la siguiente manera:
 - Modificación: Mejoras planificadas, cambios correctivos y de emergencia, cambios en el entorno operativo, actualizaciones de software comercial de distribución masiva y parches para los defectos y las vulnerabilidades
 - Migración: Moverlo de una plataforma a otra puede requerir pruebas operativas del nuevo entorno y del software modificado o pruebas de conversión de datos cuando los datos de otra aplicación se migren al sistema en mantenimiento
 - Retirada: Cuando una aplicación llega al final de su vida útil

Análisis de Impacto para el Mantenimiento

- Evalúa los cambios que se hicieron para el lanzamiento de un mantenimiento con el objetivo de identificar las consecuencias previstas, así como los efectos secundarios esperados y posibles de un cambio y para identificar las áreas del sistema que se verán afectadas por el cambio
- También puede ayudar a identificar el impacto de un cambio en las pruebas existentes

Análisis de Impacto para el Mantenimiento

- Se puede realizar un análisis de impacto antes de realizar un cambio, para decidir si se debe realizar el cambio, basándose en las consecuencias potenciales en otras áreas del sistema

Análisis de Impacto para el Mantenimiento

- El análisis de impacto, puede ser difícil si:
 - Las especificaciones están desactualizadas o no existen
 - Los casos de prueba no existen o están desactualizados
 - No sea ha mantenido la trazabilidad bidireccional entre las pruebas y la base de prueba
 - El soporte de herramientas es débil o inexistente
 - Las personas involucradas no tienen conocimiento de dominio y/o sistema
 - No se ha prestado suficiente atención a la mantenibilidad del software durante el desarrollo

Prueba Estática

Prueba Estática

- *Prueba dinámica:* Requiere la ejecución del software
- *Prueba estática:*
 - Se basa en la evaluación manual de los productos de trabajo (revisión) o en la evaluación basada en herramientas del código u otro producto de trabajo que se este probando sin ejecutar el código o el producto que se este desarrollando

Prueba Estática

Productos de trabajo que pueden ser evaluados

- Especificaciones, requisitos de negocio, requisitos funcionales y requisitos de seguridad
- Épicos, historias de usuario y criterios de aceptación
- Especificaciones de arquitectura y diseño
- Código

Prueba Estática

Productos de trabajo que pueden ser evaluados

- Planes de prueba, casos de prueba, procedimientos de prueba y scripts de prueba automatizados
- Guías de usuario
- Páginas web
- Contratos, planes de proyecto, calendarios y presupuestos

Prueba Estática

Ventajas

- Permite la detección temprana de defectos antes de que se realicen pruebas dinámicas
- Detección y corrección de defectos de forma más eficiente y antes de la ejecución de pruebas dinámicas
- Identificar defectos que no se encuentran en las pruebas dinámicas
- Prevenir defectos en el diseño, o la codificación descubriendo inconsistencias, ambigüedades, contradicciones, omisiones, inexactitudes y redundancias en los requisitos

Prueba Estática

Ventajas

- Incrementar la productividad de desarrollo
- Reducir el costo y tiempo de desarrollo
- Reducir el costo y el tiempo de la prueba
- Reducir el costo total de la calidad durante la vida útil del software debido a la reducción de los fallos en etapas posteriores del ciclo de vida o después de la entrega en operación
- Mejorar la comunicación entre los miembros del equipo al participar en las revisiones

Diferencias entre Prueba Estática y Prueba Dinámica

- Ambas pruebas pueden tener los mismos objetivos
- Se complementan entre si al encontrar diferentes tipos de defectos

Diferencias entre Prueba Estática y Prueba Dinámica

Prueba Estática	Prueba Dinámica
Detecta defectos en los productos de trabajo	Detecta fallos causados por defectos cuando se ejecuta el software
Un defecto puede residir en un producto de trabajo mucho tiempo sin provocar un fallo	Si un defecto se encuentra en una parte de la aplicación poco usada, no será fácil construir y ejecutar una prueba dinámica que lo detecte
Se puede utilizar para mejorar la consistencia y calidad interna de los productos de trabajo	Se concentra en los comportamientos visibles desde el exterior

Defectos típicos identificados en la prueba estática

- Defectos en los requisitos
 - Inconsistencias, ambigüedades, contradicciones, omisiones, inexactitudes y redundancias
- Defectos de diseño
 - Algoritmos o estructuras de base de datos ineficientes, alto acoplamiento, baja cohesión
- Defectos de codificación

Defectos típicos identificados en la prueba estática

- Defectos de codificación
 - Variables con valores no definidos, variables que nunca se utilizan, código inalcanzable, código duplicado
- Desviaciones con respecto a estándares
 - Falta de adhesión a los estándares de codificación
- Especificaciones de interfaz incorrectas

Defectos típicos identificados en la prueba estática

- Especificaciones de interfaz incorrectas
- Vulnerabilidades de seguridad
 - Susceptibilidad a desbordamientos de la memoria intermedia
- Deficiencias o inexactitudes en la trazabilidad o la cobertura de la base de prueba
 - La falta de pruebas para un criterio de aceptación

Proceso de Revisión

Proceso de Revisión

- *Revisiones informales:* No siguen un proceso definido y no tienen una salida documentada formal
- *Revisiones formales:* Cuentan con la participación de un equipo, resultados de la revisión documentados y procedimientos documentados para llevar a cabo la revisión

Proceso de Revisión

- El grado de formalidad depende de factores como el modelo de ciclo de vida de desarrollo de software, la madurez del proceso de desarrollo, la complejidad del producto de trabajo que se debe revisar, cualquier requisito legal o reglamentario
- El foco de atención de una revisión depende de los objetivos acordados

Proceso de Revisión de Productos de Trabajo

- Planificar
- Iniciar Revisión
- Revisión Individual
- Comunicar y Analizar Cuestiones
- Corregir e Informar

Proceso de Revisión de Productos de Trabajo

Planificar

- Definir el alcance, que incluye el objetivo de la revisión, que documentos o partes de documentos se deben revisar y las características de calidad que se deben evaluar
- Estimar el esfuerzo y plazos
- Identificar las características de la revisión, como el tipo de revisión con los roles, actividades y listas de comprobación

Proceso de Revisión de Productos de Trabajo

Planificar

- Seleccionar las personas que participaran en la revisión y asignación de roles
- Definir los criterios de entrada y salida para los tipos de revisión más formales
- Comprobar el cumplimiento de los criterio de entrada

Proceso de Revisión de Productos de Trabajo

Iniciar Revisión

- Distribuir el producto de trabajo y productos de trabajo relacionados
- Explicar a los participantes el alcance, los objetivos, el proceso, las funciones y los productos de trabajo
- Responder cualquier pregunta que los participantes pudieran tener

Proceso de Revisión de Productos de Trabajo

Revisión Individual

- Revisar todo o parte del producto de trabajo
- Notificar posibles defectos, recomendaciones y preguntas

Proceso de Revisión de Productos de Trabajo

Comunicar y Analizar Cuestiones

- Comunicar los defectos potenciales identificados
- Analizar los posibles defectos, asignar su propiedad y estado
- Evaluar y documentar las características de calidad
- Evaluar los hallazgos de la revisión con respecto a los criterios de salida para tomar una decisión de la revisión

Proceso de Revisión de Productos de Trabajo

Corregir e Informar

- Elaborar informes de defecto para aquellos hallazgos que requieren modificaciones
- Corregir los defectos detectados
- Comunicar defectos a la persona o equipo adecuado
- Registrar el estado actualizado de los defectos (en revisiones formales)

Proceso de Revisión de Productos de Trabajo

Corregir e Informar

- Recopilar métricas (revisiones formales)
- Comprobar el cumplimiento de los criterios de salida (revisiones formales)
- Aceptar el producto de trabajo cuando se alcanzan los criterios de salida

Roles y Responsabilidades de una Revisión Formal

- Una revisión formal tiene los siguientes roles:
 - Autor
 - Dirección
 - Facilitador o Moderador
 - Líder de Revisión
 - Revisores
 - Escriba o grabador

Roles y Responsabilidades de una Revisión Formal

Autor

- Crea el producto de trabajo bajo revisión
- Corrige los defectos en el producto de trabajo

Roles y Responsabilidades de una Revisión Formal Dirección

- Es el responsable de la planificación de la revisión
- Decide acerca de la ejecución de las revisiones
- Asigna personal, presupuesto y tiempo
- Supervisa la rentabilidad en curso
- Ejecuta las decisiones de control en caso de resultados inadecuados

Roles y Responsabilidades de una Revisión Formal Facilitador o Moderador

- Asegura el funcionamiento efectivo de las reuniones de revisión
- Realiza una mediación entre los distintos puntos de vista
- Muchas veces es la persona de la que depende el éxito de la revisión

Roles y Responsabilidades de una Revisión Formal

Líder de Revisión

- Asume la responsabilidad general de la revisión
- Decide quienes estarán involucrados y organiza cuando y donde se llevará a cabo

Roles y Responsabilidades de una Revisión Formal Revisores

- Pueden ser expertos en la materia, personas que trabajan en el proyecto, implicados con un interés en el producto de trabajo y/o con antecedentes técnicos o de negocios en específico
- Identifican posibles defectos
- Pueden representar diferentes perspectivas

Roles y Responsabilidades de una Revisión Formal Escriba o Grabador

- Recopila los posibles defectos encontrados durante la actividad de revisión individual
- Registra nuevos defectos potenciales, puntos abiertos y decisiones de la reunión de revisión

Tipos de Revisión

Tipos de Revisión

- Uno de los principales objetivos de la revisión es descubrir defectos
- Todos los tipos de revisión pueden ayudar en la detección de defectos y el tipo de revisión seleccionado debe basarse en las necesidades del proyecto, los recursos disponibles, el tipo de producto y los riesgos, el dominio del negocio y la cultura de la empresa, etc.
- Las revisiones se pueden clasificar de acuerdo a diferentes características

Tipos de Revisión

Revisión Informal

- *Objetivo Principal:* Detectar defectos potenciales
- *Posibles objetivos adicionales:* generar nuevas ideas o soluciones, resolver de forma rápida problemas menores
- No se basa en un proceso formal
- Puede no implicar una reunión de revisión
- Puede ser realizado por un compañero de trabajo del autor o por más personas

Tipos de Revisión

Revisión Informal

- Se pueden documentar los resultados
- Varía en la utilidad dependiendo de los revisores
- El uso de listas de comprobación es opcional
- Utilizado con mucha frecuencia en el desarrollo ágil

Tipos de Revisión

Revisión Guiada

- *Objetivos Principales:* detectar defectos, mejorar el producto de software, considerar implementaciones alternativas, evaluar la conformidad con estándares y especificaciones
- *Posibles objetivos adicionales:* intercambio de ideas sobre técnicas o variaciones de estilo, formación de los participantes, alcanzar un consenso
- La preparación individual antes de la reunión de revisión es opcional

Tipos de Revisión

Revisión Guiada

- Normalmente, la reunión de revisión está dirigida por el autor
- El escriba es obligatorio
- El uso de listas de comprobación es opcional
- Puede tomar forma escenarios, ensayos o simulaciones
- Se pueden elaborar registros de posibles defectos e informes de revisión
- En la práctica puede variar desde muy informal a muy formal

Tipos de Revisión

Revisión Técnica

- *Objetivos Principales:* lograr un consenso, detectar defectos potenciales
- *Posibles objetivos adicionales:* evaluar la calidad y generar confianza en el producto de trabajo, generar nuevas ideas, motivar y capacitar a los autores para mejorar los futuros productos de trabajo, considerar implementaciones alternativas

Tipos de Revisión

Revisión Técnica

- Los revisores deben ser pares técnicos del autor y expertos técnicos en la misma u otras disciplinas
- Es necesario que haya una preparación individual antes de la reunión de revisión
- La reunión de revisión es opcional, lo ideal es que la dirija un facilitador capacitado, que no sea el autor

Tipos de Revisión

Revisión Técnica

- El escriba es obligatorio, idealmente que no sea el autor
- El uso de listas de comprobación es opcional
- Normalmente se elaboran registros de defectos potenciales e informes de revisión

Tipos de Revisión

Inspección

- *Objetivos Principales:* detectar defectos potenciales, evaluar la calidad, generar confianza en el producto de trabajo, prevenir futuros defectos similares mediante el aprendizaje del autor y el análisis de la causa raíz
- *Posibles objetivos adicionales:* motivar y capacitar a los autores para que mejoren los futuros productos de trabajo y el proceso de desarrollo de software, alcanzar un consenso

Tipos de Revisión

Inspección

- Sigue un proceso definido
- Los revisores deben ser pares técnicos del autor y expertos técnicos en la misma u otras disciplinas
- El escribe es obligatorio
- La reunión de revisión es dirigida por un facilitador capacitado
- Utiliza roles definidos

Actividad

- De su proyecto final, identificar un producto de trabajo que puedan revisar y definir que técnica de revisión van a usar

Técnicas de Prueba

Técnicas de Prueba

- Una técnica de prueba ayuda a identificar las condiciones de prueba, casos de prueba y datos de prueba

Elección de Técnicas de Prueba

- La elección de técnicas de prueba depende de una serie de factores:
 - Tipo de componente o sistema
 - Complejidad del componente o sistema
 - Estándares de regulación
 - Requisitos del cliente o contractuales
 - Niveles de riesgo
 - Clases de riesgo

Elección de Técnicas de Prueba

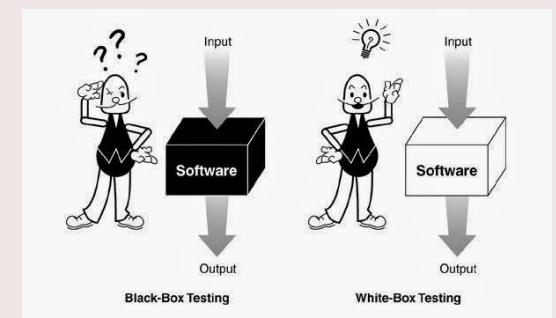
- Objetivos de prueba
- Documentación disponible
- Conocimientos y competencias del probador
- Herramientas disponibles
- Tiempo y presupuesto
- Modelo de ciclo de vida de desarrollo de software

Elección de Técnicas de Prueba

- Uso previsto del software
- Experiencia prueba del uso de las técnicas de prueba en el sistema o componente que se va a probar
- Tipos de defectos esperados en el componente o sistema

Categorías de técnicas de prueba

- Las técnicas de prueba se clasifican en
 - Técnicas de prueba de caja blanca
 - Técnicas de prueba de caja negra
 - Técnicas basadas en la experiencia



Técnicas de prueba de caja negra

- También llamadas conductuales o basadas en el comportamiento
- Se basan en un análisis de la prueba adecuada
- Son aplicables tanto a la prueba funcional como no funcional
- Se concentran en las entradas y salidas del objeto de prueba sin referencia a su estructura interna

Técnicas de prueba de caja negra

- Entre sus características comunes están:
 - Las condiciones de prueba, casos de prueba y datos de prueba se deducen de una base de prueba, que puede incluir requisitos de software, especificaciones, casos de uso e historias de usuario
 - Los casos de prueba se pueden usar para detectar diferencias entre los requisitos y la implementación de los requisitos, así como desviaciones respecto a los requisitos

Técnicas de prueba de caja negra

- La cobertura se mide en función de los elementos probados en la base de prueba y de la técnica aplicada a la base de prueba

Técnicas de prueba de caja blanca

- También llamadas estructurales o basadas en la estructura
- Se basan en un análisis de la arquitectura, el diseño detallado, la estructura interna o el código del objeto de la prueba

Técnicas de prueba de caja blanca

- Entre sus características comunes están:
 - Las condiciones de prueba, casos de prueba y datos de prueba se deducen de una base de prueba, que puede incluir el código, la arquitectura del software, el diseño detallado o cualquier otra fuente de información relacionada con la estructura del software
 - La cobertura se mide en base a los elementos probados dentro de una estructura seleccionada (código, interfaces)

Técnicas de prueba de caja blanca

- Las especificaciones se utilizan comúnmente como una fuente adicional de información para determinar el resultado esperado de los casos de prueba

Técnicas de prueba basadas en la experiencia

- Aprovechan la experiencia de los desarrolladores, probadores y usuarios para diseñar, implementar y ejecutar pruebas
- Regularmente se combinan con técnicas de caja blanca y de caja negra

Técnicas de prueba basadas en la experiencia

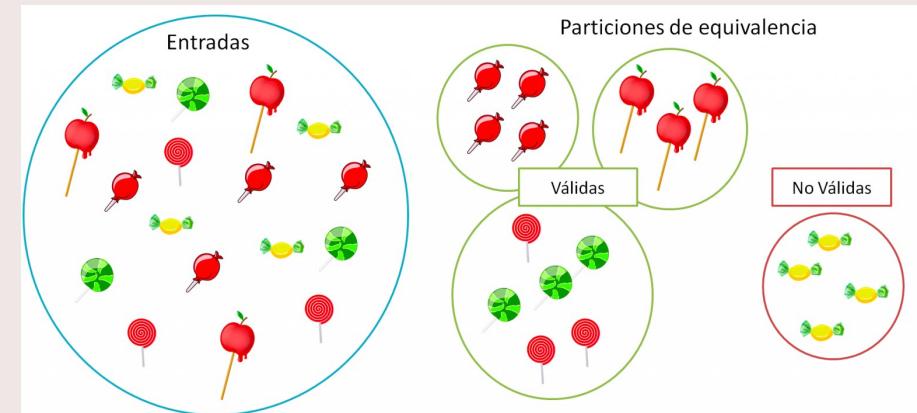
- Entre sus características comunes están:
 - Las condiciones de prueba, casos de prueba y datos de prueba se deducen de una base de prueba, que puede incluir el conocimiento y la experiencia de los probadores, desarrolladores, usuarios y otros implicados
 - Este conocimiento y experiencia incluye el uso esperado del software, su entorno, los posibles defectos y su distribución

Técnicas de Prueba de Caja Negra

Técnicas de Prueba de Caja Negra

Partición de Equivalencia

- Divide los datos en particiones o clases de equivalencia del manera que se espera que todos los miembros de una partición dada sean tratados de la misma manera
- Existen particiones de equivalencia tanto para valores válidos como no válidos



Técnicas de Prueba de Caja Negra

Partición de Equivalencia

- Los valores válidos son los valores que debe aceptar el componente o sistema
- Una partición de equivalencia que contiene valores válidos se llama “partición de equivalencia válida”

Técnicas de Prueba de Caja Negra

Partición de Equivalencia

- Los valores no válidos son los valores que deben ser rechazados por el componente o sistema
- Una partición de equivalencia que contiene valores no válidos se llama "partición de equivalencia no válida"
- Las particiones pueden identificarse para cualquier elemento de datos relacionados con el objeto de prueba, incluyendo entradas, salidas, valores internos, valores relacionados con el tiempo y parámetros de interfaz

Técnicas de Prueba de Caja Negra

Partición de Equivalencia

- Cualquier partición se puede dividir en subparticiones de si fuera necesario
- Cada valor debe pertenecer a una y sólo una partición de equivalencia
- Cuando se utilizan particiones de equivalencia no válidas en casos de prueba, deben probarse individualmente, no combinarse con otra partición de equivalencias no válidas, para garantizar que no se produzca enmascaramiento de los fallos

Técnicas de Prueba de Caja Negra

Partición de Equivalencia

- Los fallos se pueden enmascarar cuando se producen varios fallos al mismo tiempo, pero solo uno de ellos es visible, lo que hace que los otros fallos queden sin detectar
- Para lograr una cobertura del 100% con esta técnica, los casos deben de cubrir todas las particiones identificadas, utilizando al menos un valor de cada partición

Técnicas de Prueba de Caja Negra

Partición de Equivalencia

- La cobertura se mide como el número de particiones de equivalencia probadas por al menos un valor, dividido por el número total de particiones de equivalencia identificadas, normalmente expresado en porcentaje
- La partición de equivalencia es aplicable a todos los niveles de prueba

Técnicas de Prueba de Caja Negra

Partición de Equivalencia

- La cobertura se mide como el número de particiones de equivalencia probadas por al menos un valor, dividido por el número total de particiones de equivalencia identificadas, normalmente expresado en porcentaje
- La partición de equivalencia es aplicable a todos los niveles de prueba

Técnicas de Prueba de Caja Negra

Definir Partición de Equivalencia

- Las clases de equivalencia se definen según una serie de directrices:
 - **Rangos:** Si una entrada está condicionada a un rango de valores, se define una clase de equivalencia válida para todos los valores pertenecientes al rango y dos no válidas, una para los valores menores al límite inferior al rango y otras para los mayores al límite superior al rango
 - Ejemplo: Edades comprendidas entre 18 y 65 años

Técnicas de Prueba de Caja Negra

Definir Partición de Equivalencia

- **Valor específico:** El rango queda restringido a un valor único. Se siguen creando igualmente una clase de equivalencia válida y dos no válidas
 - Ejemplo: 18 años

Técnicas de Prueba de Caja Negra

Definir Partición de Equivalencia

- **Número de Valores:** Igual que las anteriores pero teniendo en cuenta el número de valores introducidos
 - Ejemplo:
 - 1 a 5 pasajeros sería la clase válida
 - 0 y más de 5 pasajeros las no válidas

Técnicas de Prueba de Caja Negra

Definir Partición de Equivalencia

- **Perteneciente a un conjunto:** Cuando un valor es válido si pertenece a un conjunto. Se define una clase de equivalencia válida para cada uno de los valores del conjunto, además, de una no válida, para aquellos valores que no pertenecen a él
 - Ejemplo:
 - Días de la semana en minúsculas

Técnicas de Prueba de Caja Negra

Definir Partición de Equivalencia

- **Valor lógico:** Se corresponde a la evaluación de una condición. Se establece una clase de equivalencia válida (se cumple la condición) y otra no válida (no se cumple la condición)
 - Ejemplo:
 - La edad debe ser mayor o igual a 18

Técnicas de Prueba de Caja Negra

Definir Partición de Equivalencia

- ***Ejemplo 1: Días de la semana en formato numérico***

- La condición de entrada reflejaría que sólo se podrían introducir números del 1 al 7, ambos inclusive. Identificaríamos 2 clases:
 - Clase válida: $1 \leq \text{día} \leq 7$
 - 2 clases no válidas: una cuando $\text{día} < 1$ y otra para $\text{día} > 7$

Técnicas de Prueba de Caja Negra Definir Partición de Equivalencia

- **Ejemplo 2: Colores RGB en formato String minúscula**

- El valor de entrada sólo puede corresponder a uno de los colores RGB escrito en minúscula: «red», «green», «blue». Se supone que cada una de esas entradas se debería manejar de formas distintas en el programa.
- 3 clases de equivalencia válidas, una para cada uno de los valores de entrada «red», «green» y «blue».
- Una clase inválida que incluiría aquellos colores no especificados en la condición.

Técnicas de Prueba de Caja Negra

Definir Partición de Equivalencia

- ***Ejemplo 3: Nombre con primera letra mayúscula***

- Clase válida para cadenas cuya primera letra es una mayúscula (se cumple la condición)
- Clase no válida para valores que no cumplen la condición: cadenas que comienzan en minúscula

Técnicas de Prueba de Caja Negra

Pruebas Mediante Partición de Equivalencia

- Definidas las clases hay que definir casos de prueba que cubran todas las particiones establecidas.
- No podemos combinar clases inválidas en casos de prueba individuales, es decir, hay que probarlas partición a partición.
- Esto es debido a que una condición inválida para una partición podría acarrear enmascarando otra condición posterior (el programa acaba, se lanza una excepción, etc. y no se alcanza el código que llega a usar la otra partición)

Técnicas de Prueba de Caja Negra

Pruebas Mediante Partición de Equivalencia

- No serviría de nada evaluar en el mismo caso de prueba a la vez como entrada el color «yellow» y que el nombre de usuario empiece por minúscula, porque la que fallara antes haría que la otra no se ejecutara
- Es por esto que a veces el generar casos de prueba correctos implica realizar múltiples combinaciones de particiones de equivalencia para las distintas entradas

Técnicas de Prueba de Caja Negra

Pruebas Mediante Partición de Equivalencia

- En esos casos puede resultar útil seguir los siguientes consejos:
 - Nombrar cada clase de equivalencia creada con un identificador único.
 - Por ejemplo: establecer una nomenclatura, y suponiendo los casos anteriores nombrar las respectivas clases de la siguiente forma: v_dia, nv_dia_menor, nv_dia_mayor, v_color_g, v_color_r, v_color_b, nv_color, v_mayúsculas, nv_mayúsculas.
 - Elaborar una tabla de clases de equivalencia

Técnicas de Prueba de Caja Negra

Ejemplo de Tabla de Clases de Equivalencia

Entrada	Tipo	Clases Válidas	ID	Clases no Válidas	ID
Día de la semana	Rango	$1 \leq \text{día} \leq 7$	v_dia	día < 1	nv_dia_menor
				día > 7	nv_dia_mayor
Colores	Conjunto	color = "red"	v_color_r	color distinto de los válidos	nv_color
		color = "green"	v_color_g		
		color = "blue"	v_color_b		
Nombre de usuario	Condición lógica	Empieza por mayúscula	v_mayúscula	No empieza por mayúscula	nv_mayúscula

Técnicas de Prueba de Caja Negra

Ejemplo de Casos de Prueba

Caso de Prueba	Clases de equivalencia	Condiciones de Entrada			Resultado esperado
		Día	Color	Nombre	
CP1	v_dia, v_color_r, v_mayúscula	3	red	User	R1
CP2	v_dia, v_color_g, v_mayúscula	1	green	USER	R2
CP3	v_dia, v_color_b, v_mayúscula	7	blue	UseR	R3
CP4	nv_dia_menor, v_color_r, v_mayúscula	0	red	User	E1
CP5	nv_dia_mayor, v_color_r, v_mayúscula	8	red	User	E2
CP6	nv_dia_menor, v_color_g, v_mayúscula	-1	green	USeR	E3
...

Ejercicio 1

- **Un campo de texto que solo acepta caracteres alfabéticos. La longitud del valor ingresado debe estar entre 6 y 10 caracteres**
- **Técnica de pruebas: Partición de equivalencias**