



# CLASE 1 - Laboratorio

Análisis y Diseño de  
Sistemas 1 Sección A+ 1er  
Semestre 2021

CESAR SAZO [checha18964@gmail.com](mailto:checha18964@gmail.com)

# Temas A Cubrir

1. Acerca del Curso
2. Temas del Curso
3. Ponderación
4. Información Adicional
5. Clase 1 - Metodologías  
Robustas y Ágiles
6. Tareas/Hoja de trabajo

0

# Acerca 1 del Curso

De qué me sirve este curso?



02

## Temas del Curso

# Temas por ver en laboratorio

Metodologías  
de Desarrollo

Control de  
Versiones

Pruebas  
Unitarias

Pruebas  
Funcionales

Pruebas no  
Funcionales

Integración  
Continua

03

# Ponderación

Tareas	4*	6
Hoja de Trabajo	4*	6
Practicas	4	65
Cortos	2	10
Examen Final	1	10
Total	-	100

04

# Información Adicional

## Tareas

- No se permiten entregas tardes
- Tareas de investigación deben llevar conclusión/es

## Foro

- 1 Cada semana en UEDI
- No obligatorio



## Laboratorio

- No se guardan notas
- Reposicion es justificadas
- Asistencia?
- [AyD1]Asunto

## Prácticas

Es obligatorio aprobar TODAS las práctica con nota minima de 61.

05

# Metodologías Robustas y Ágiles

# La ingeniería de

## Software

Se trata de desarrollar productos o soluciones para un cliente o mercado en particular, teniendo en cuenta factores como costos, planificación, calidad y dificultades.

A esto se le denomina metodología de desarrollo de software.

# Etapas comunes

Análisis de la situación y la necesidad.

Bosquejo del

proyecto

Desarrollo del

software

Comprobación de su  
funcionamiento.

# Ciclo de vida del software

Consiste en cuatro etapas

- Concepción: Determina la repercusión del proyecto y diseño del modelo de negocio
- Elaboración: Planificación del proyecto, especificando características y arquitectura.
- Construcción: Elaboración del producto.
- Transición: Entrega del producto terminado a los usuarios.  
Al finalizar el ciclo de vida inicia el mantenimiento del software, en donde se corrigen errores detectados por los usuarios finales.

# METODOLOGÍAS DE DESARROLLO DE SOFTWARE

Proceso que se sigue a la hora de diseñar una solución o proyecto en específico.

Enfoque de carácter estructurado y estratégico que permite el desarrollo de programas con base a modelos de sistemas, reglas, sugerencias y guías.

Involucra:

- Comunicación
- Manipulación de modelos
- Intercambio de información
- Datos de partes involucradas

# Metodologías Robustas o Tradicionales

# Metodología Cascada

## Fases

Formales(Royce):

1. Requisitos de Sistema
2. Requisitos de Software
3. Análisis
4. Diseño
5. Implementacion
6. Prueba
7. Servicio

- Versión más utilizada:
1. Analisis
  2. Diseño
  3. Implementaci  
on
  4. Verificacion
  5. Mantenimient



## Ventajas

- Facil de entender e implementar
- Documentación excelente y total
- Seguimiento Sencillo
- Fechas estipuladas
- El cliente aprueba la documentación

● El cliente obtiene el proyecto hasta que esté terminado

- No es amigable a cambios
- Fallos detectados en las últimas fases

## Restricciones

- No se puede regresar al paso anterior
- No se puede avanzar a la otra fase si no se termina la presente

# Metodología de Prototipos

## Etapas

1. Requerimientos
2. Modelado
3. Construcción
4. Desarrollo
5. Refinamiento
6. Prototipo finalizado

- Desechable
- Evolucionario

## Ventajas

- Reduce riesgo
  - Aumenta probabilidad de éxito
  - Retroalimentación
  - Planeación rápida
- Desventaja

- Malentendido con el Cliente
- Tentación



# Metodología en Espiral

Respuesta a problemas del modelo de cascada

Se enfoca en la identificación y resolución de

riesgos Acepta cambio en requerimientos

Seguridad

económica

Retroalimentación



# RUP (Rational Unified Process)

Es un proceso de desarrollo de software que es el estándar más utilizado para el análisis, diseño, implementación y documentación de proyectos orientados a objetos.

Desarrollado por la empresa Rational Software (IBM) Unificado de Modelado UML.



No es un sistema con pasos firmemente establecidos sino un conjunto de metodologías

# Fases de RUP

- Requerimiento
- Análisis e implementación
- Prueba
- Evaluación

## Ventajas

- Evaluación de cada fase y cambios de objetivos.
- Proyectos de innovación
- Sencillo
- Seguimiento detallado en cada una de las fases

## Desventajas

- Evaluación de riesgos compleja
- Flexibilidad
- Posición incómoda para el cliente
- Necesidad del cliente para detallar las necesidades y alcance.

# Conclusiones Metodologías Robustas

## Ventajas

- Documentación exhaustiva
- Lleva un plan de proyecto
- Se definen claramente los roles y etapas

## Desventajas

- Altos costos para implementar un cambio
- No ofrece buenas soluciones en entornos volátiles
- Gasto de recurso
- Comunicación con el cliente

# Metodologías Ágiles

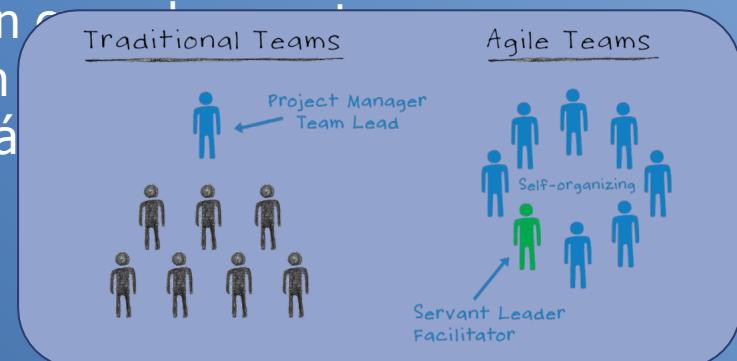
Manejo de un proyecto al dividirlo en varias fases , en donde se involucra constantemente al stakeholder(considerado parte del equipo), mejora continua y varias iteraciones en cada fase.

# Principios de Las Metodologías Ágiles

# Equipo

Individuos e interacciones encima de procesos y herramientas.

- Es más importante formar un buen equipo de trabajo, el recurso humano es el que consigue el éxito del proyecto. Si el equipo falla el proyecto fallará.
- No se debe crear un entorno y que el equipo se ajuste, se debe crear un buen equipo y entre ellos van a crecer.
- El equipo no tiene un leader pero si un facilitador.
- El equipo no sigue la estructura en pirámide.



# Software Funcional

Software Funcional en vez de documentación extensa.

- La documentación extensa absorben demasiado tiempo, por lo que si es necesario deberá ser corta.
- Entregar software funcional a través de las iteraciones nos permite obtener retroalimentación

Desarrolladores nuevos en el equipo?

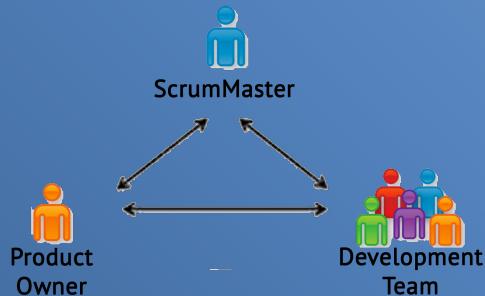
- Código Fuente
- Interacción con el Equipo

# Colaboración con el cliente

Colaboración con el cliente en vez de un contrato de negocio.

Interacción constante entre el cliente y el equipo de desarrollo.

El cliente proporcionará retroalimentación mientras se va desarrollando el sistema y logre analizar nuevas funcionalidades y objetivos. Esto asegura que el cliente estará totalmente satisfecho.



# Responder al Cambio

Responder a los cambios en vez de seguir un plan estricto.

Se permiten cambios en medio proyecto, gracias a que no se está siguiendo un plan estricto.

Si el cliente tiene la idea de incrementar objetivos, especificaciones o requerimientos, que sea un proceso sencillo y rápido.

Esto permite que el sistema sea flexible para un entorno volátil. El cliente quedará satisfecho ya que no se debe conformar con el primero que aceptó en los requerimientos.



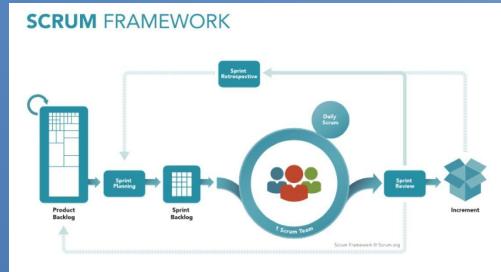
# Ventajas

- Mejoran la satisfacción del cliente
- Mejora de la motivación e implicación del equipo de desarrollo
- Ahorra tiempo y costos
- Mayor velocidad y eficiencia
- Eliminar cualquier característica innecesaria del producto
- Mejora la calidad del producto
- Alerta de forma rápida retrasos o errores
- Mayor flexibilidad

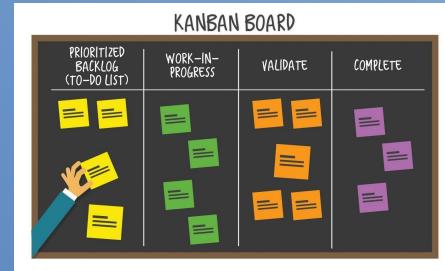
# Las que veremos



Extreme  
Programming



Scrum



Kanban

06

# Tareas/Hojas

# Tarea 1



Metodologías Ágiles vs Robustas



Entrega  
Individual  
Jueves



DUDAS  
Y  
COMENTARIOS



# CLASE 2 - LABORATORIO

Análisis y Diseño de  
Sistemas 1 Sección A+ 1er  
Semester 2022

Cesar Sazo checha18964@gmail.com

# TEMAS A CUBRIR

1. Información General
2. Retroalimentación T1
3. Clase 2
4. Tareas/Hoja de trabajo

0

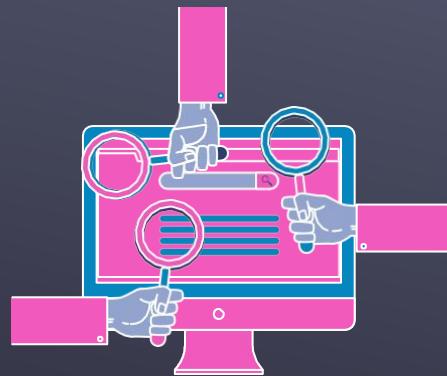
# INFORMACIÓN GENERAL

02

## RETROALIMENTACIÓN T1

# TAREA 1

- La metodología de desarrollo y los requerimientos funcionales y no funcionales.
- Agiles solo en proyectos pequeños.



03

CLASE 2

# REQUERIMIENTOS

# INGENIERÍA DE REQUERIMIENTOS

Proceso de recopilar, analizar y verificar las necesidades del cliente para un sistema de software.

Mejora la forma en que entendemos y definimos sistemas de software complejos.

Meta:

Entregar una especificación de software correcta y completa.



# ACTIVIDADES

## Extracción

- ▶ Análisis              Visión del usuario
- ▶ Equipo de desarrollo
- ▶        Problemas y soluciones Cliente
  - ▶ Discutir los requisitos

## Especificación

### Documentación

- ▶ Validación
- ▶ Verificación de requisitos y su descripción

# REQUERIMIENTO

Condición o capacidad que debe satisfacer o poseer un sistema o una componente de un sistema para satisfacer un contrato, un estándar, una especificación u otro documento formalmente impuesto.



# DIFICULTADES

- Traducción de ideas vagas de necesidades de software.
- Considerar el conjunto de funciones y restricciones.
- Abstracción a través de muchas personas.



# REQUERIMIENTOS NO FUNCIONALES

No son funciones directas del sistema o actividad del usuario.

Son propiedades del sistema.

- ~~QUE HACE?~~
- COMO LO HACE?



# POR QUE EXISTEN?



## Organización

Políticas de la  
organización



## Presupuesto

Límites  
presupuestarios



## Factores Externos

Ciertas leyes o políticas



## Interoperabilidad

Cómo interactúan con  
otros sistemas.

# TIPOS NO FUNCIONALES

Requisitos de producto

Comportamiento del producto

- Rendimiento
- Memoria
- Fiabilidad
- Portabilidad
- Seguridad
- Usabilidad
- Eficiencia

Requisitos organizacionales

Políticas y procedimientos  
● Estandares existentes en la  
● Patrón de organización diseño



Necesidades Externas

Factores externos al sistema y su desarrollo

- Comunicación con otros sistemas
- Se imponen para asegurar que el sistema será aceptado por el usuario
- Regulatorios
- Eticos
- Legislativos

# EJEMPLOS

- El sistema debe ser capaz de procesar N transacciones por segundo.
- La respuesta de peticiones es de menos de 5 segundos.
- Se debe desarrollar con patrón de diseño observer.
- El sistema debe contar con manuales de usuario.
- El sistema deberá ser desarrollado utilizando JPA.
- Las pruebas de software se ejecutarán con JUNIT.
- La metodología de desarrollo será TDD.
- El sistema no revelará a sus operadores los datos personales de los clientes.
- El sistema no guardará el número de tarjeta del cliente.
- Los procesos de negocio se desarrollarán en la base de datos.
- El sistema se desarrollará para PC y Android.



# REQUERIMIENTOS FUNCIONALES

Descripción de las actividades y servicios que un sistema debe proveer el cual están vinculados a entradas, datos almacenables y salidas.

Cómo se comportará el sistema en ciertos casos.

Esta descripción puede indicarnos también lo que el sistema no debe hacer.

# EJEMPLOS

- Login
- Registro
- Envío de correo electrónico en transacciones
- Permitir seleccionar varios productos de un mismo producto en el carrito
- Asignar Ids a X dato.
- Ingreso de eventos de una cuenta dependiendo de su estado.
- Autorizar automáticamente solicitudes si cumple con las reglas de negocio.
- Login tanto con correo como con usuario hacia la misma cuenta.
- El usuario podrá ver el catálogo en la página principal al iniciar sesión.

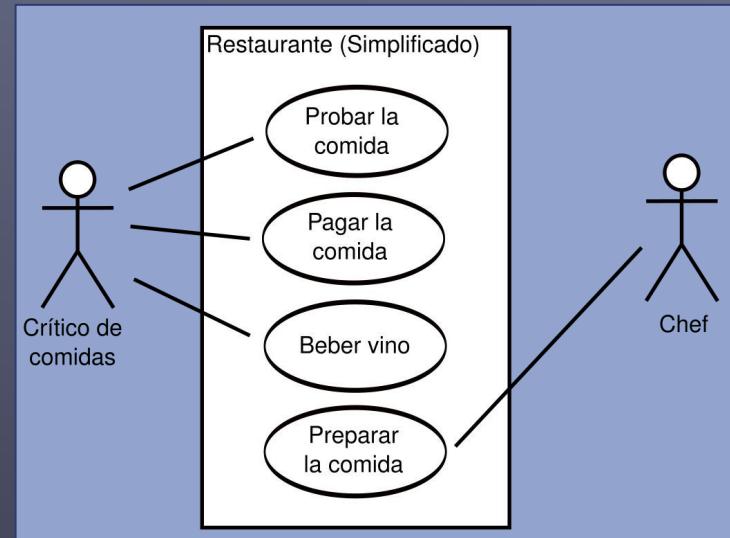


# CASOS DE USO

Es una técnica para la captura de requerimientos de un nuevo sistema o una actualización de software.

Proporciona escenarios que deben indicar cómo interactúa el sistema con el usuario u otros sistemas para realizar un objetivo en específico.

Cada caso se centra en describir una característica del sistema.



## CARACTERISTICAS

- Descriptivos
- Sencillos
- No debería mostrar funcionalidad interna ni como se implementará
- Solamente pasos
- Deberá mostrar actividades que le da valor de negocio

## Limitantes

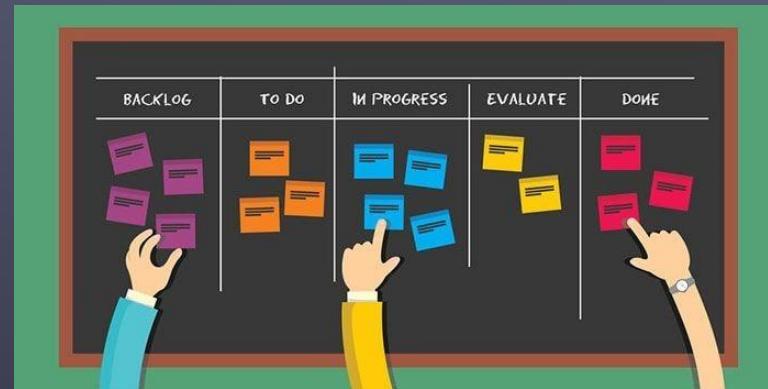
- Requisitos de comportamiento pero no completamente requerimientos funcionales.
- No permite determinar requerimientos no funcionales.
- Se deben complementar con lo que haga falta. Como reglas de negocio o requerimientos no funcionales.

# HISTORIAS DE USUARIO

Son parte de un enfoque ágil que ayuda a cambiar la forma obtenerlos pasando de escribir sobre los requisitos a hablar de ellos con los involucrados.

Son cortas y simples contadas desde la perspectiva de la persona a la cual va dirigida el desarrollo.

INPUT	REFINEMENT	RESOLVING	AWAITING ESTIMATION	BACKLOG
2	2	3	6	



# A DONDE VAN?

Cuando se  
que termina?

Criterio de aceptación

Quien las  
crea?

Cualquiera del  
equipo

Formato

Como [Usuario] quiero  
[objetivo] para [motivo].

# CONSIDERACIONES

Se deberá considerar el peso de cada historia a la hora de escogerlas en las iteraciones

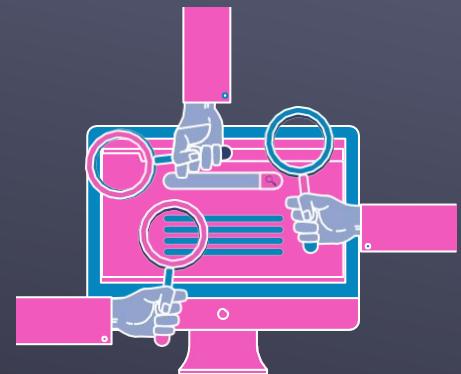
Relación con las otras historias en el backlog de cada iteración

Cada prioridad o valor de las historias debe ser definida por los stakeholders



# PREGUNTAS QUE AYUDAN A IDENTIFICAR LOS REQUERIMIENTOS

- Qué datos manipula el proceso?
- Cual es el proceso básico de la empresa?
- Qué pasos se siguen para realizar el proceso?
- Donde se utiliza?
- Quienes lo utilizan ?
- Con qué frecuencia lo utilizan?
- Quienes utilizan la información resultante?



# IDENTIFICACIÓN DE ELEMENTOS

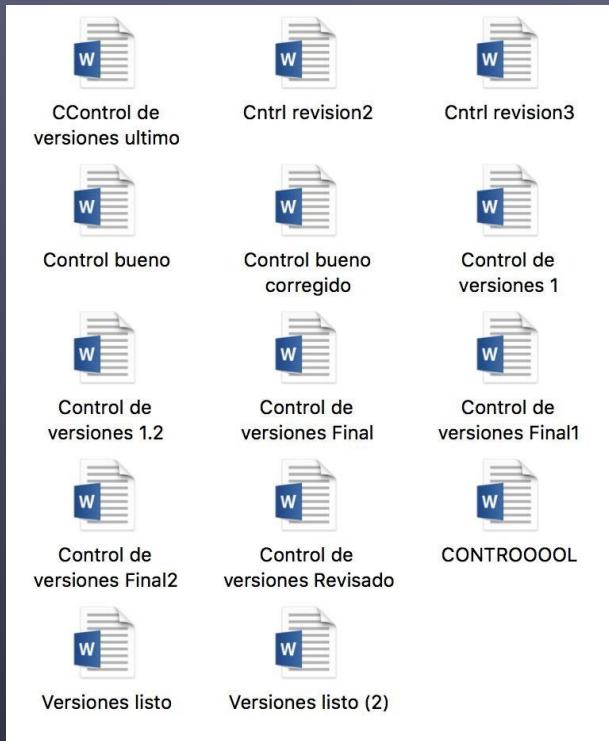
- Procesos
- Flujos de datos entre los procesos
- Datos de cada flujo
- Base de datos
- Datos de la base de datos



# CONTROL DE VERSIONES

Forma de organizar los elementos de algo,  
el cual es modificado a lo largo del tiempo.

# EL ANTES



Diferentes  
nombres

Muchas  
copias

Perdida de  
cambios útiles

# SISTEMAS DE VERSIÓN DE CONTROL

Es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos, para un mejor control de versiones cuando sea necesario.

Es una de las herramientas más importantes a la hora de trabajar en desarrollos en equipo en sistemas grandes que ya se encuentran en producción.



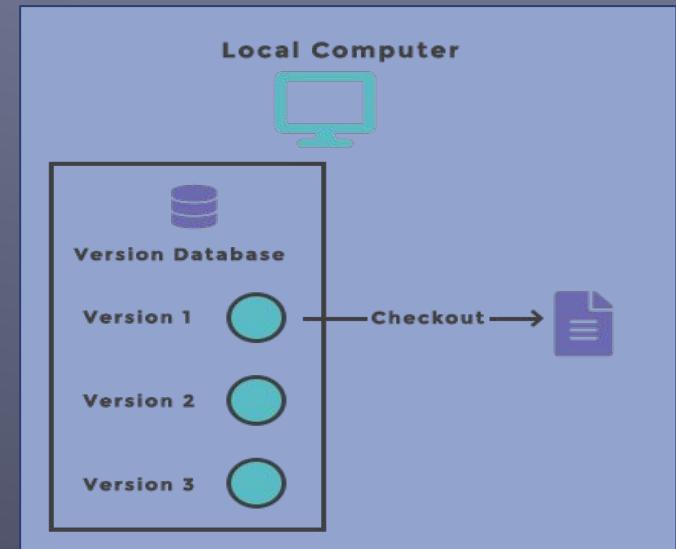
# LO QUE PERMITE

- Mejor manejo de versiones funcionales (Tags)
- Respaldo de cambios (Stash)
- Safepoint (Reset)
- Facilidad de trabajo en equipo (Versioning & Merge control)
- Historial de cambios completo (Commit)
- Insertar los cambios realizados(Merge)
- Desarrollo de requerimientos en ambientes separados(Branches)

# TIPOS DE SISTEMAS DE CONTROL DE VERSIONES

# SISTEMA LOCAL

- Se basa en copiar los archivos a otro directorio, con opcionalidad de color fecha y hora o algún numeral para indicar la versión.
- Se implementó una simple base de datos para llevar el registro.



## Características

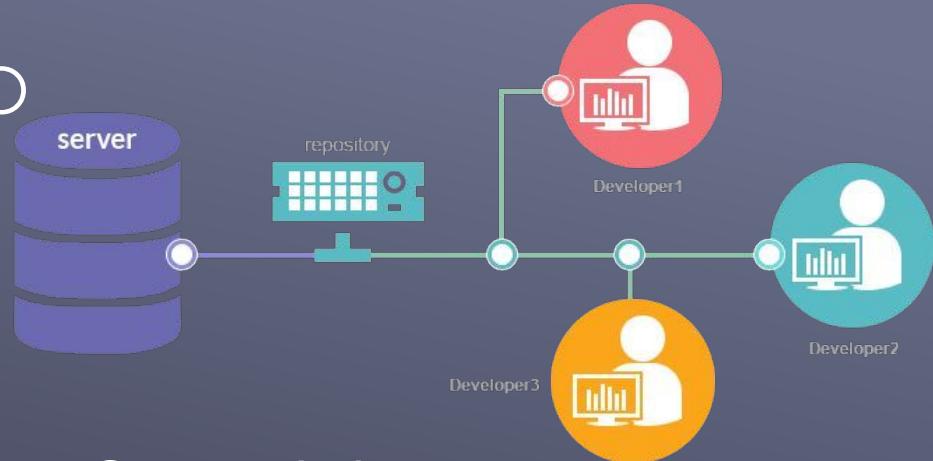
- Sencillo de implementar.
- Factor humano y crecimiento hace que tenga sus inconvenientes.
- Posibilidad de guardar en un directorio erróneo.

# Trabajar con otros desarrolladores

- ▶ Dificultad de trabajar con más personas en un proyecto en especial si tocan archivos en común.

# SISTEMA CENTRALIZADO

- Solucion a problematica de desarrollo en equipo.
- Más sencillo que administrar datos locales.
- Permite obtener las versiones del servidor y hacer commits hacia el repositorio.



## Características

- Un único repositorio en un único servidor.
- Todas las operaciones se realizan directamente en el repositorio.

NO HAY DISPONIBILIDAD  
LOCAL

TIENE QUE EXISTIR CONECCION A LA  
RED PARA REALIZAR CUALQUIER  
ACCIÓN.

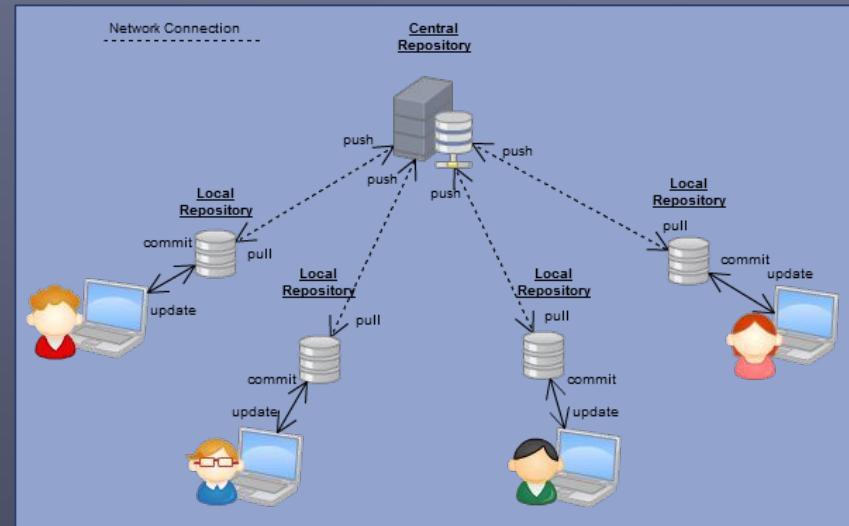
Demasiado vulnerable

Al ser un sistema centralizado, en caso de cualquier corrupción o crasheo resulta en pérdida total de datos del proyecto.

# SISTEMA DISTRIBUIDO

Todos los colaboradores tienen una copia propia del repositorio central.

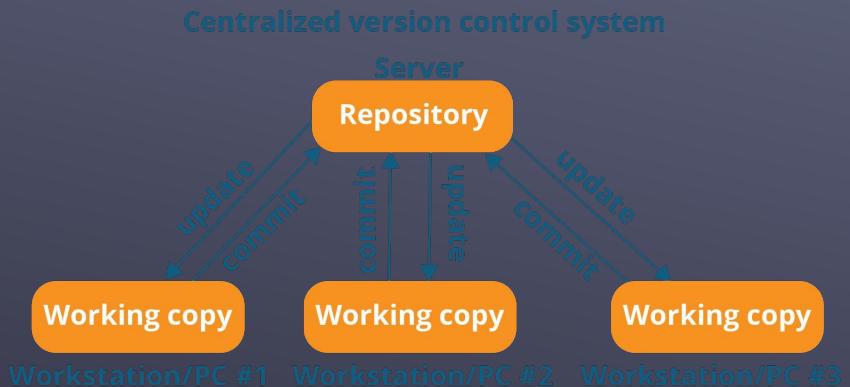
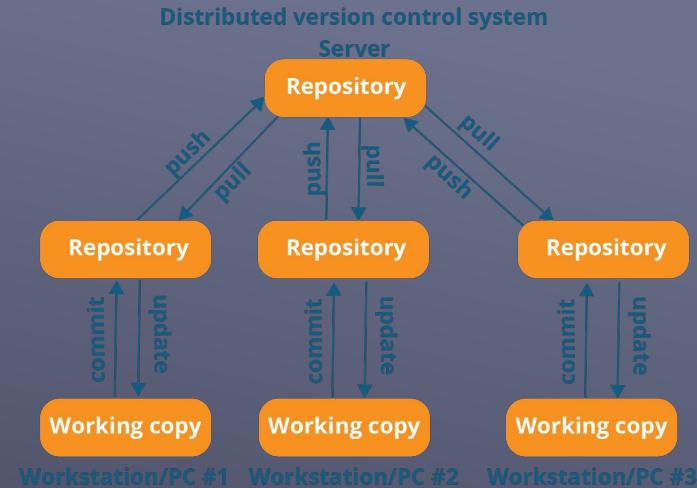
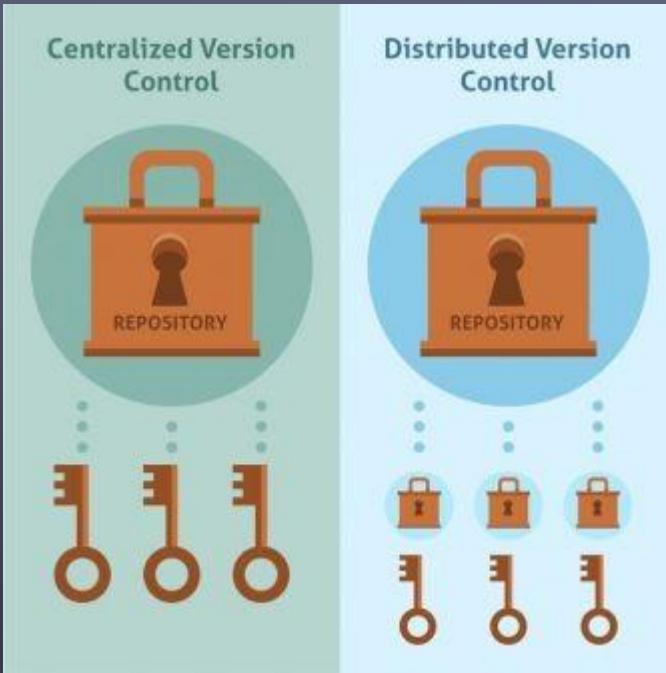
- Si el servidor central se crashea todos poseen una copia de respaldo para restaurar.
- Las acciones se realizan sin afectar al repositorio central.
- Commits locales



## Características

- Mayor rapidez ya que se accede a los datos locales y no al servidor.
- No existe tanta dependencia a la red.
- Copia total del repositorio.

# CENTRAL - DISTRIBUIDO



04

## TAREA/HOJA DE TRABAJO

# TAREA 2



Sistemas de control de versiones



Entrega  
Individual  
Github/Gitlab



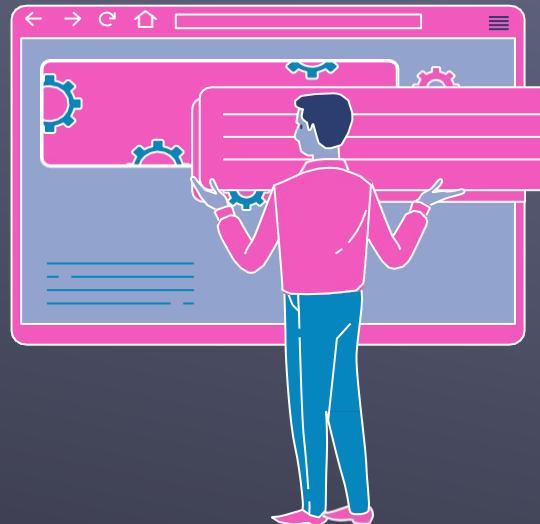
# HOJA DE TRABAJO 1



Requerimientos funcionales y no  
funcionales

27/01/2022 9:50 A.M

10:10 A.M



# DUDAS Y COMENTARIOS



# CLASE 3 - Laboratorio

Análisis y Diseño de  
Sistemas 1 Sección A+  
1er Semestre 2022

Cesar Sazo

# Temas A Cubrir

1. Información General
2. Clase 3 - Scrum, Git  
y Gitflow
3. Tareas/Hoja de trabajo
4. Practica 1

0

Retro Alimentación<sup>1</sup>

# Tarea 2/ Hoja 1



02

## Clase 3

# SCRUM

# SCRUM

Es un framework de trabajo donde se pueden manejar problemas complejos y adaptativos mientras se incrementa el valor del producto



# Características

Obtiene  
Resultados  
Pronto

Requisitos  
cambiantes

Innovación

Competitividad

Flexibilidad

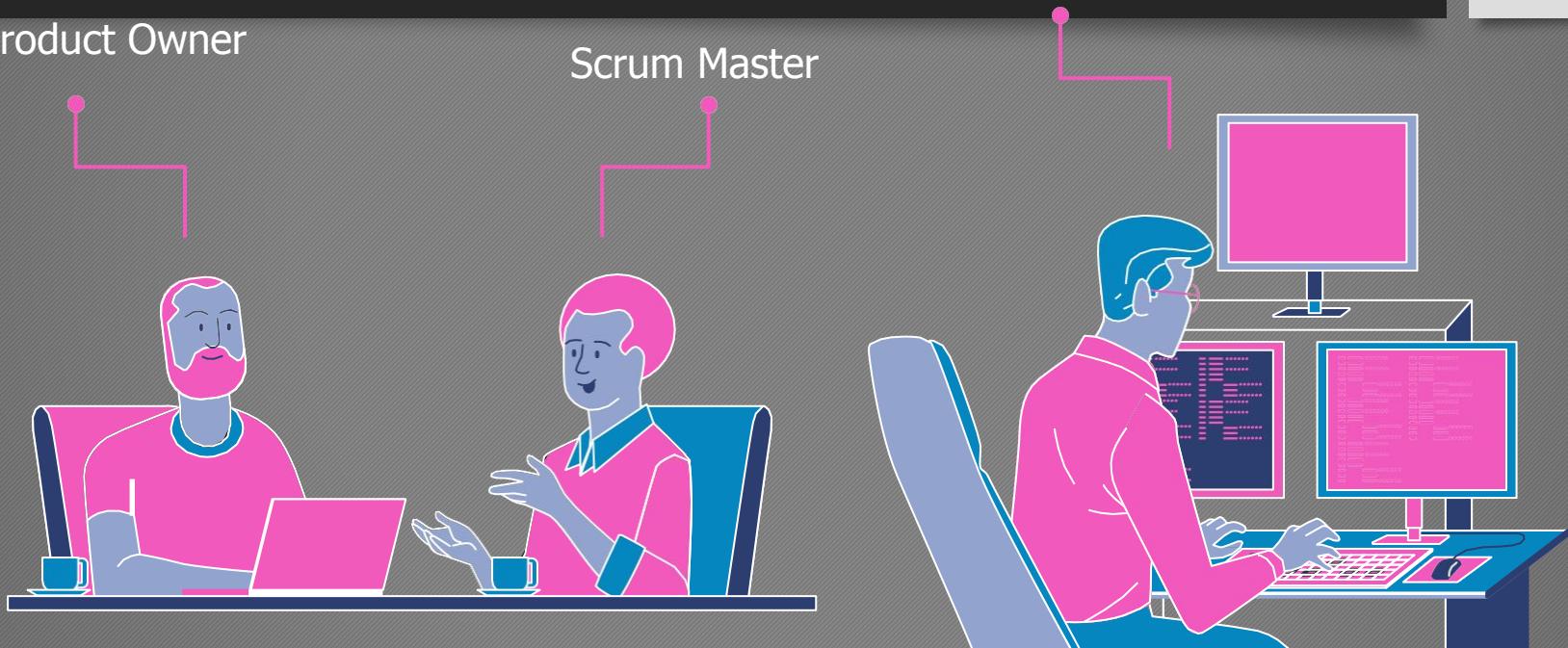
Productividad

# Roles

Product Owner

Scrum Master

Development Team



# Product Owner

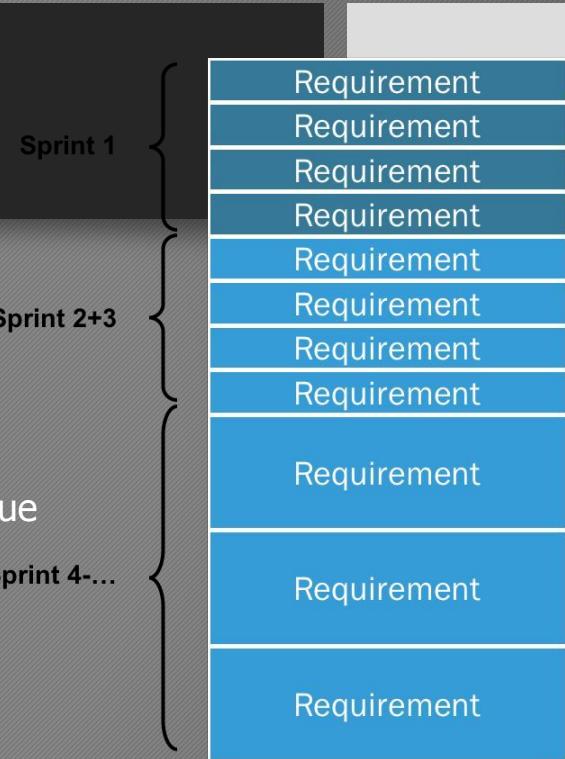
Maximizar el valor del producto

Maneja el product backlog

\*Lo realiza o designa pero siempre responsable

- Ordena los ítems.
- Escoge segun valor.
- Asegura que el backlog sea transparente y visible a todos, y en lo que se trabajara

Una Persona no un comite



# Scrum Master

## Responsabilidades con el product owner

- Se asegura que los objetivos, alcance y dominio del producto sean comprendidos por el equipo de desarrollo.
- Encontrar técnicas para el manejo eficiente del backlog.
- Entender y practicar agilidad.
- Facilitar eventos de scrum.

## Responsabilidades con el equipo de desarrollo

- Guía al equipo de desarrollo para auto organizarse.
- Ayuda al equipo de desarrollo a crear productos de alto valor.
- Remover impedimentos que atrasen al equipo de desarrollo.



# Equipo de Desarrollo

Nadie posee título aunque sus tareas sean diferentes.

No hay sub equipos.

Miembros con habilidades específica y área de enfoque.

Menos de 3 miembros resulta en menos productividad( Reduce interacción ) y puede existir poca habilidad para cumplir la meta.

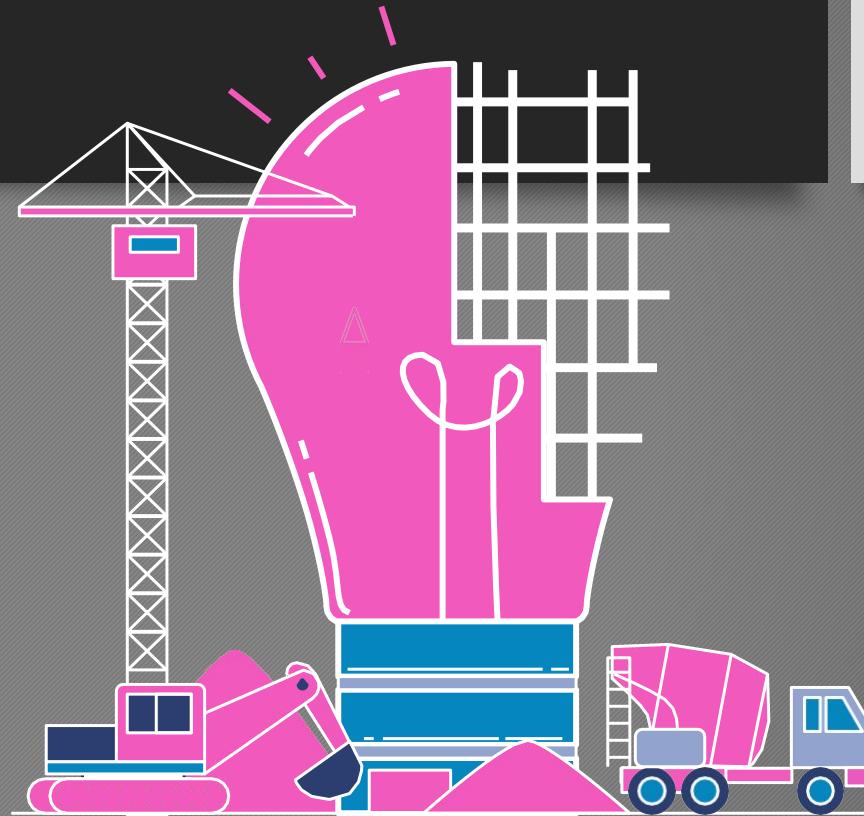
Mayor a 9 integrantes requiere mucha coordinación



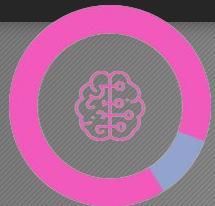
# Product Backlog

Lista ordenada de todo lo que se necesita del producto

- Dinámico
- El product backlog evoluciona y crece mientras el ambiente y producto cambian.
- Lo más importante de scrum.



# SPRINT



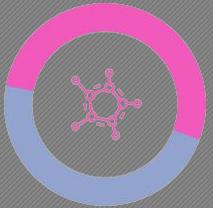
Sprint  
Planning  
Meeting



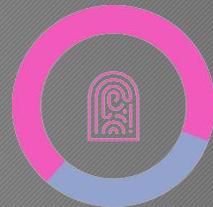
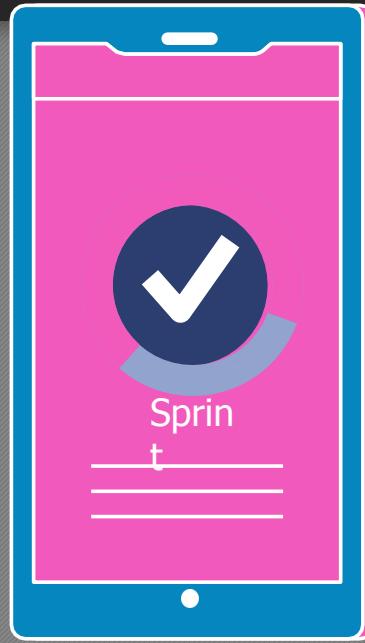
Sprint  
Review



Scrum  
Diario



Sprint  
Retrospecti  
ve

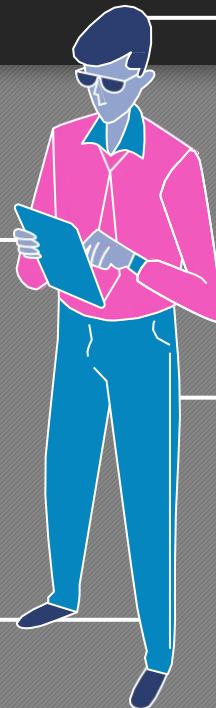


Sprint  
backlog

# Sprint planning Meeting

Que se espera lograr al final del sprint y como

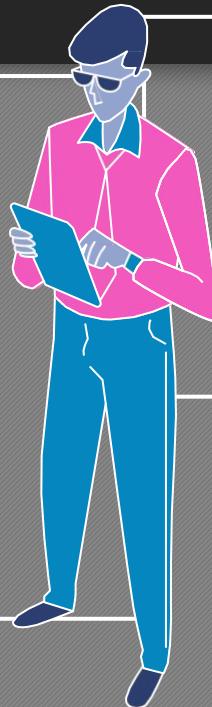
En este evento se inicia oficialmente el sprint.



# Scrum Diario

Responde

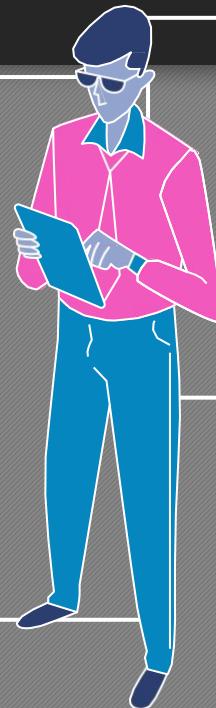
1. Qué hice ayer?
2. Qué haré hoy?
3. Tengo algún impedimento?



Inicio del dia, 15 minutos maximos.

# Sprint Review

Se presenta el incremento del producto y se da retroalimentación así como las tareas completadas.

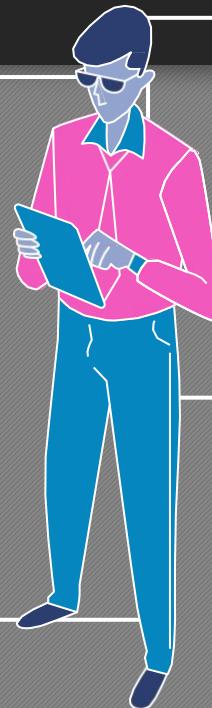


Que se logró en el sprint?

Se cumplio la meta?

# Sprint Retrospective

Que está sirviendo?  
Que se puede mejorar  
Como puede ser más  
productivo?



Se discute cómo  
mejorar el proceso de  
desarrollo.

# Review vs Retrospective

	Review	Retrospective
Enfoque	Que se hizo?	Como se hizo?
Que mejora?	Sprints y entregables	Proceso de desarrollo

# Sprint Review

## Sprint Review

Meeting at the end of the sprint to check the increment



# Sprint Retrospective

## Sprint Retrospective

Meeting after Sprint Review to review processes

- What went well?
- What could be improved?
- How can we improve it?



30 min - 3h



Product owner + Scrum team

Self-analysis on  
how to work



Problem analysis and  
improved aspects

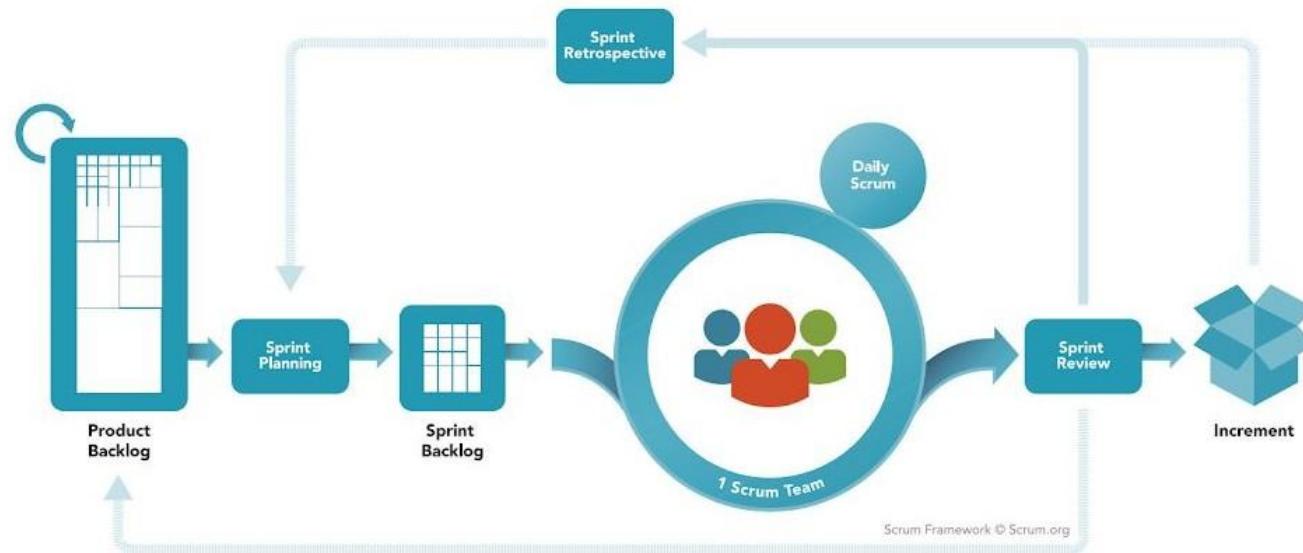


Framework improvements



# Framework

## SCRUM FRAMEWORK

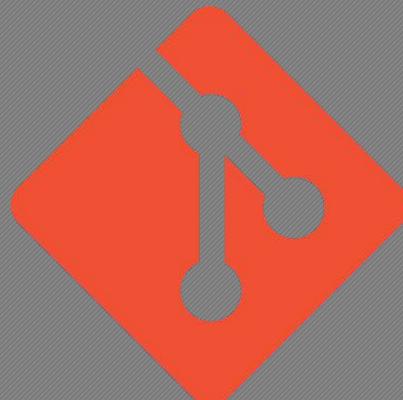


# Git y Gitflow

# GI T

Git es un software de control de versiones para sistemas distribuidos muy popular que nos otorga los diferentes comandos para realizar las actividades de un VCS.

- Permite ramas
- Rápido
- Software Libre
- Historial completo
- Multiplataforma



git

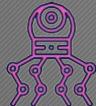
# Tipo de implementación



## Hosting local

Tipo de hosting que permite implementar un sistema de versión de controles en un servidor local.

Ej: Gitolite, Propio git



## Hosting en la nube

Proveedores de la nube nos proveen VCS sin que tengamos que setear el servidor por cuenta propia.

Ej: Github, Gitlab, Gitbucket.



GitHub



Bitbucket



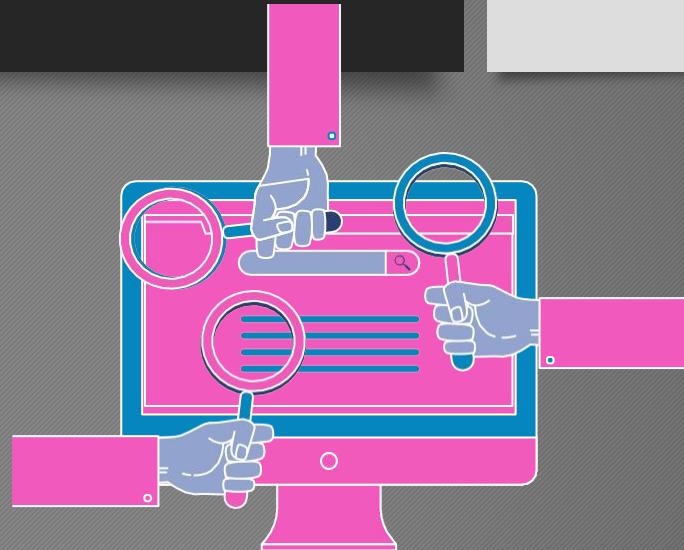
# Buenas practicas

- Cada desarrollador debe utilizar su propio usuario.
- No realizar push a master.
- No colocar información sensible, llaves por ejemplo.
- Realizar un buen gitignore.
- Realizar commits frecuentes.
- Mensaje de commit en tercera persona y descriptivos.
- No arreglar commits(ammend).



# TAGS

- Referencias que apuntan a puntos específicos en la historia de Git o algún repositorio para manejar las versiones del producto/repositorio.
- No existe norma estricta de identificar tags.



# VERSIONAMIENTO SEMÁNTICO

- Comúnmente existe un formato de versionamiento, llamado versionamiento semántico el cual indica el siguiente formato.

- Referencias

- <https://semver.org/>
- <https://github.com/semver/semver/blob/master/semver.md#is-v1-23-a-semantic-version>
- <https://stackoverflow.com/questions/2006265/is-there-a-standard>



# Comandos Git

`git clone URL`

Clona el repositorio  
en el directorio.

`git add -A`

Agrega todos  
los cambios  
para el  
commit

`git branch -avv`

Lista todas las  
ramas del  
repositorio

`git add Archivo`

Agrega  
archivo a  
commit

`git init`

Inicia el  
repositorio

`git commit -m  
"String"`

Crea el commit  
con el mensaje  
indicado

# Comandos Git

## git diff branch

Muestra diferencias con la branch actual con la indicada  
**git push -u**  
remota local

Agrega la branch local al repositorio remoto

## git status

Muestra estado actual de la branch, push o commits

## git merge rama

Hace merge de la rama en la que se este perado

## git push

Realiza un push al remote

## git checkout branch

Permite mover de branch en branch

# Comandos Git

**git push**  
**--set-upstream**  
origin branch  
Permite push de un  
branch creado local  
al origin

**git reset --hard**  
Reinicia la rama  
al último  
commit local

**git merge rama**

**--no-commit**  
**--no-ff**

Permite hacer un  
merge sin realizar  
fast forward y sin  
commit automatico  
**origin/ref**

Hace referencia  
a repositorio  
remoto

**git stash**  
**branch**

Permite guardar  
cambios y  
regresar al último

**git checkout -b**  
name

Crea una branch  
en base a la que  
se está parado

**git fetch**

Sincronizar  
repositorio local  
con remoto

# Buena practica / Situacional

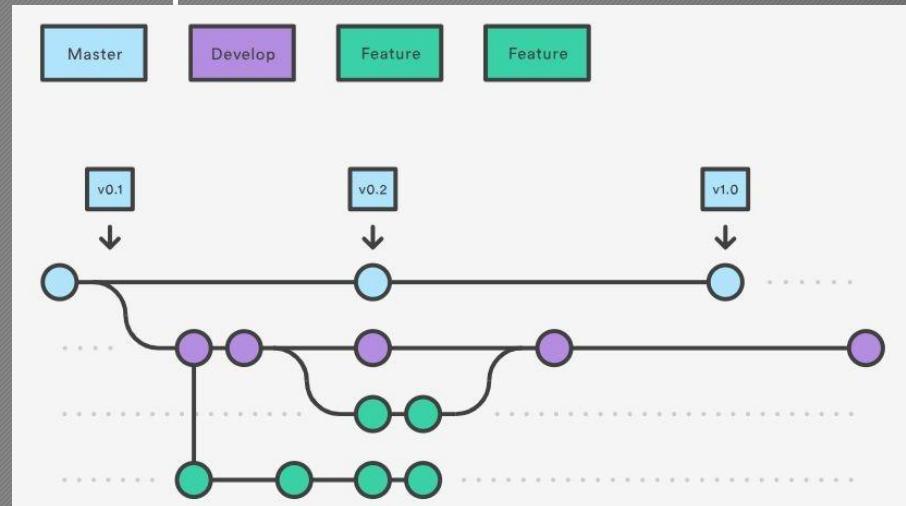
```
git merge rama  
  --no-commit  
  --no-ff
```

Permite hacer un  
merge sin realizar  
fast forward y sin  
commit automatico

# GIT FLOW

# Git Flow

- Es una recomendación sobre cómo usar Git para lograr trabajo de manera consistente y productiva. No son reglas forzosas.
- Permite integrar código rápidamente.
- Rastrea errores y permite tomar acción rápidamente.
- Procesos sencillos y definidos.
- Identificación rápida de versiones.



# RAMAS

01

master

02

develop

03

feature/name

04

release

05

hotfix

# master



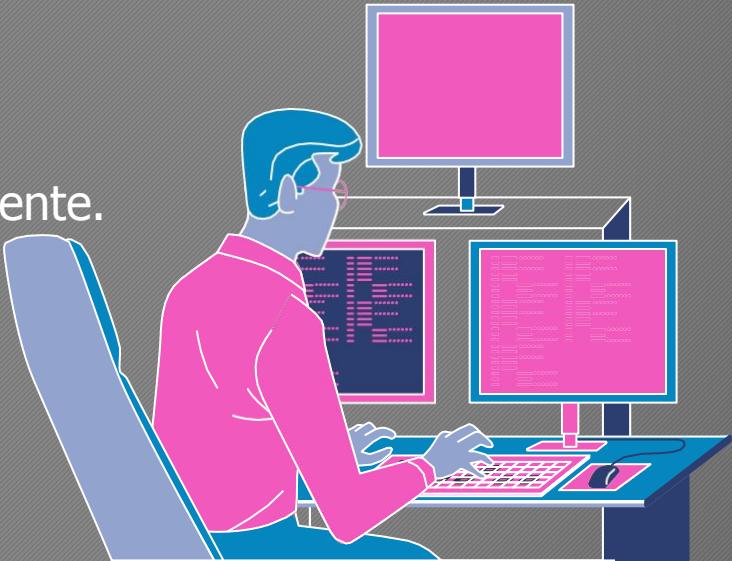
Versión en producción.



Se debe generar tag de la version.



De aqui nace hotfix y develop inicialmente.



# develop



Nace de versiones de master.

De esta rama nacen los feature para nuevos desarrollos



De esta rama nacen los releases.



# feature



Su formato es: feature/nombre



Nace de develop y aquí es donde se deberá crear las nuevas funcionalidades relacionadas a historias de usuarios.



Se debe crear de la última versión de develop.



Generalmente se eliminan luego de hacer merge a develop.



Permite n commits de código inestable/funcional.



# release



Formato release/version Ej release/1.0.0



Rama en la que se realizar pruebas.



Acepta commmits de bugfix

Se debe hacer merge hacia master.



Se elimina luego de su merge.



# hotfix



Formato hotfix/1.0.1



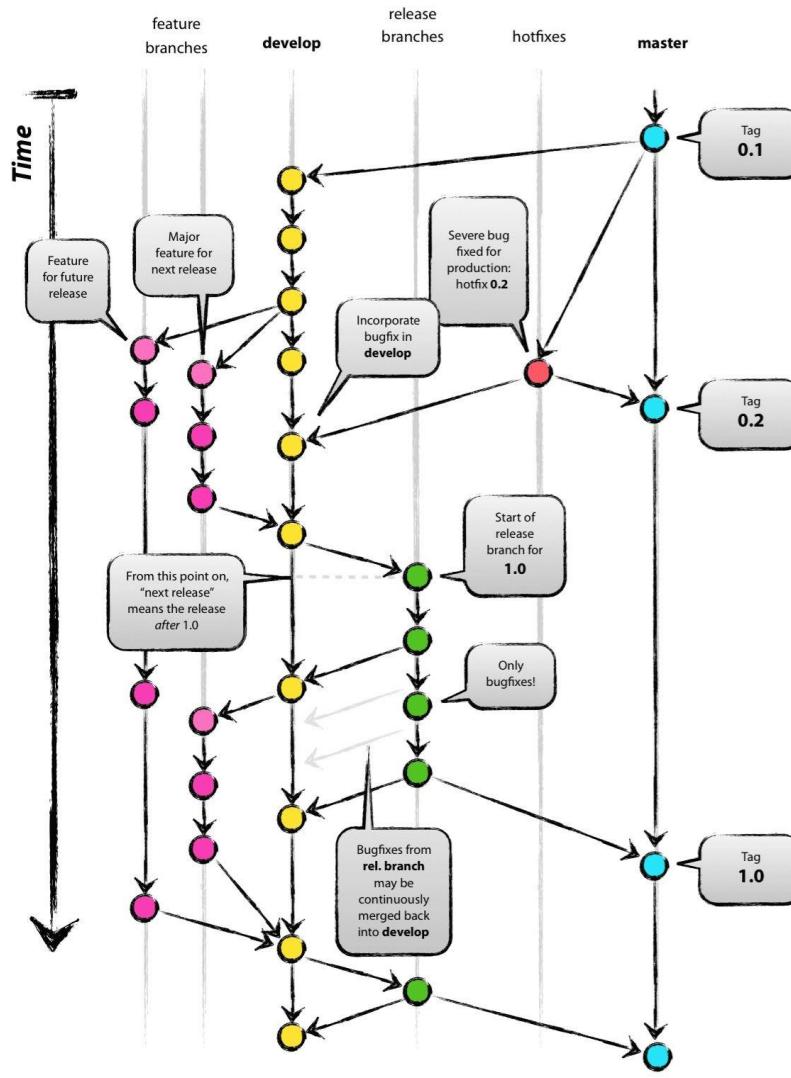
Se crean desde master.



Tienen prioridad y se hacen merge hacia la versión en producción.

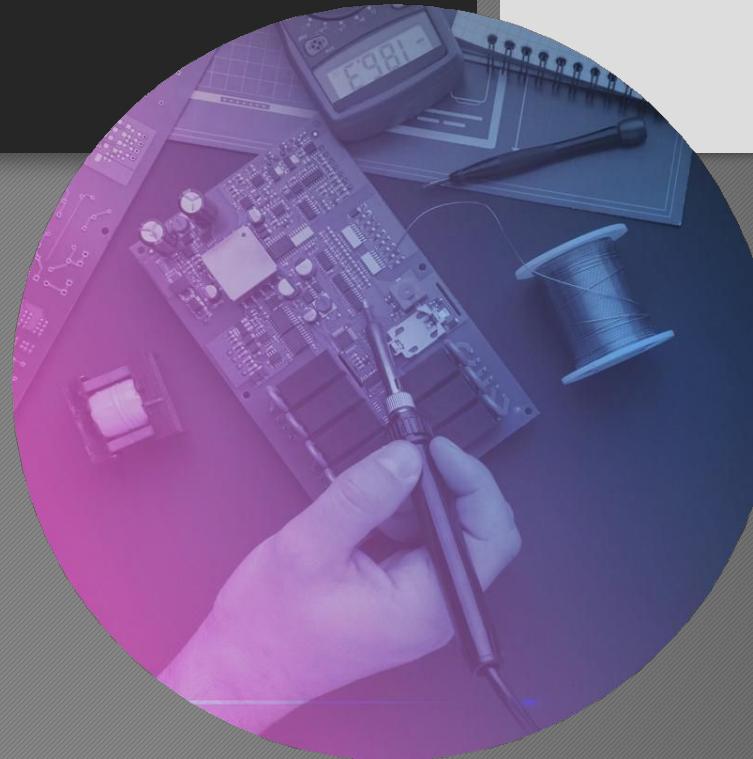


# Imagen Guia



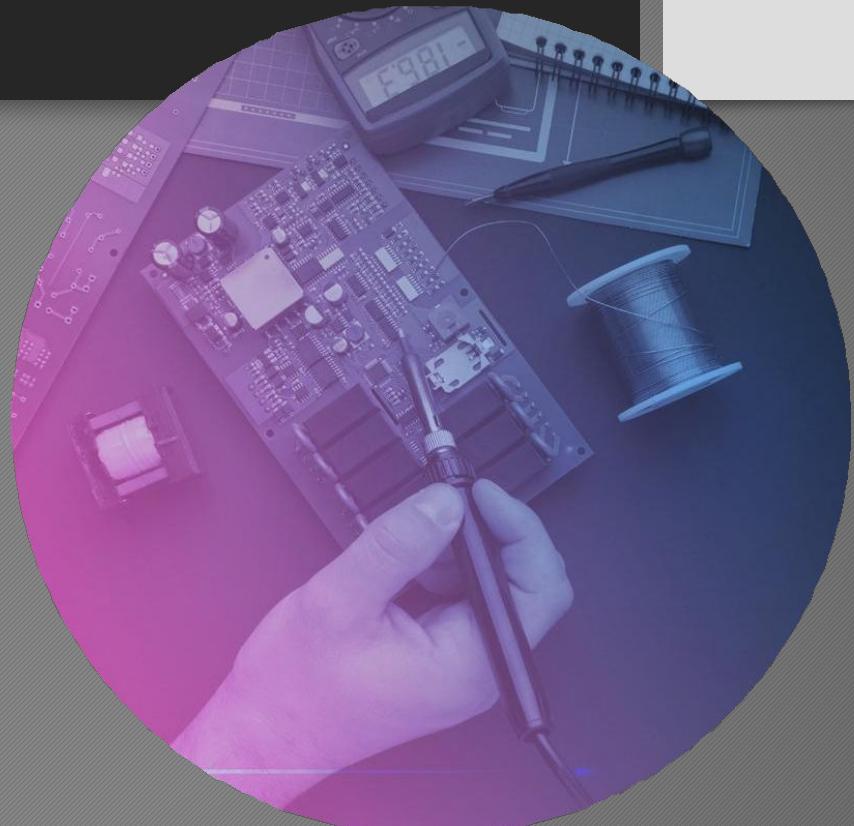
# Comandos utilizados propios de git

- Habilitar gitflow:
  - git flow init
- Iniciar feature:
  - git flow feature start <nombre>
- Iniciar feature desde una rama
  - git flow feature start <nueva> <existente>
- Finalizar feature:



# Comandos utilizados para gitflow

- Iniciar release:
  - `git flow release start <version>`
- Finalizar release:
  - `git flow release finish <version>`
- Iniciar hotfix:
  - `git flow hotfix start <version>`
- Finalizar hotfix:



03

# Tarea/Hoja de trabajo

# Tarea 3



Gitflow/Scrum



Entrega  
Individual  
Miércoles  
17/02/2021



04

# Practica 1

DUDAS  
Y  
COMENTARIOS



# CLASE 4 - Laboratorio

Análisis y Diseño de  
Sistemas 1 Sección A+ 1er  
Semestre 2022

Cesar Sazo

# Temas A Cubrir

1. Información General
2. Clase 4 – RUP , QA y Actividades Scrum
3. Tareas/Hoja de trabajo
4. Practica 1 dudas

01

# Información General

02

# CLASE - RUP



# RUP - QUE ES?

“RATIONAL UNIFIED PROCESS” es un proceso de ingeniería de software que otorga una forma disciplinada de asignar tareas y responsabilidades dentro del equipo de desarrollo. Tiene como objetivo asegurar que se produzca un software de alta calidad que cumpla con los requerimientos de los usuarios finales dentro de un tiempo y presupuesto predeterminado.

# RUP -

## Caracteristicas

- Es iterativo e incremental
- Proceso dirigido por los casos de uso
- Proceso centrado en la arquitectura
- Posee 4 fases principales
- El enfoque es la calidad del producto

## RUP - Proceso dirigido por los casos de uso

Se utilizan los casos de uso para el desenvolvimiento y desarrollo de las disciplinas con los artefactos, roles y actividades necesarias.

Los casos de uso son la base para la implementación de las fases y disciplinas del RUP

## RUP - Iterativo e incremental

El modelo de RUP plantea la implementación del proyecto a realizar en iteraciones, esto permite definir objetivos por cumplir en cada iteración para finalmente completar el proyecto iteración por iteración.

## RUP - Centrado en la arquitectura

Define la arquitectura de un sistema y una arquitectura ejecutable construida como un prototipo evolutivo.

Arquitectura de sistema es la organización o estructura de sus partes más relevantes.

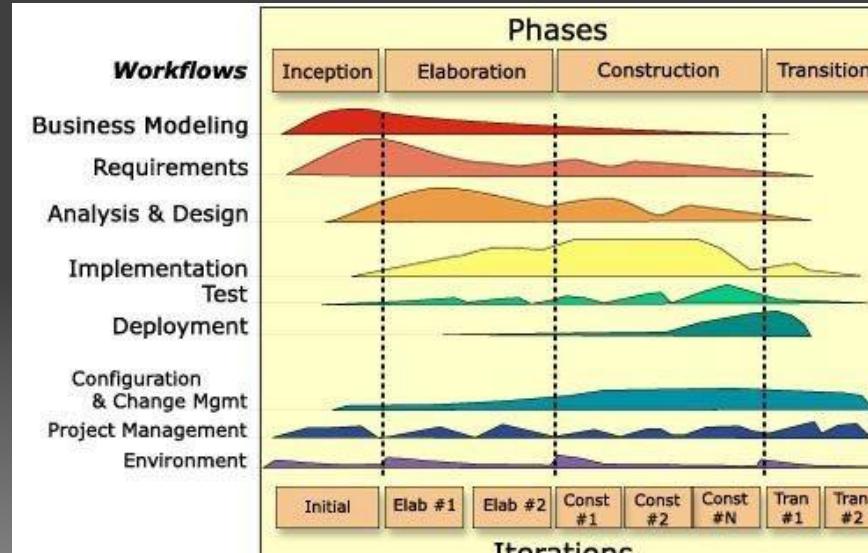
Arquitectura ejecutable es una implementación parcial del sistema para demostrar funciones y propiedades. RUP establece que esta se va refinando.

# RUP -

## Dimensiones

RUP posee 2 dimensiones

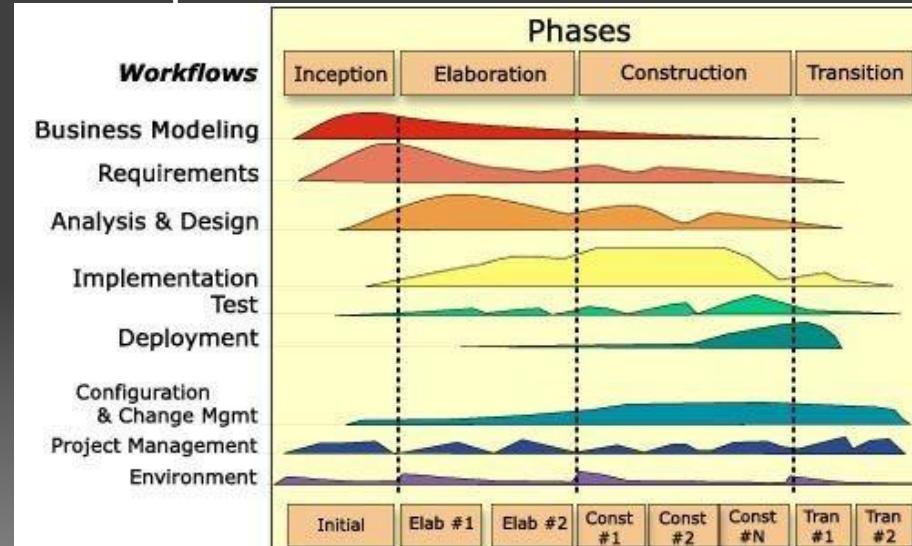
- El eje horizontal (tiempo / aspectos del ciclo).
- El eje vertical (disciplinas / actividades)



# RUP - Eje Horizontal

Representa el aspecto dinámico del proceso y se expresa en fases, iteraciones y finalización de las fases.

EJ. Se puede observar cómo se invierte más tiempo en requerimientos en las fases tempranas.

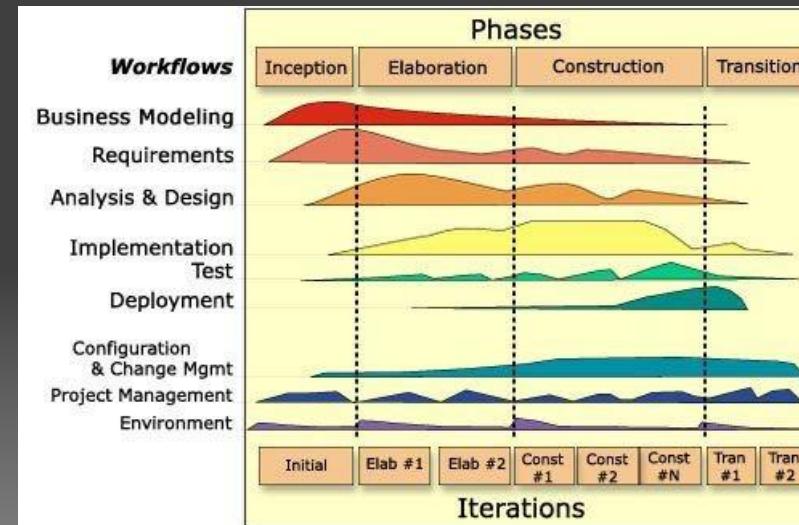


# RUP - Eje

## Vertical

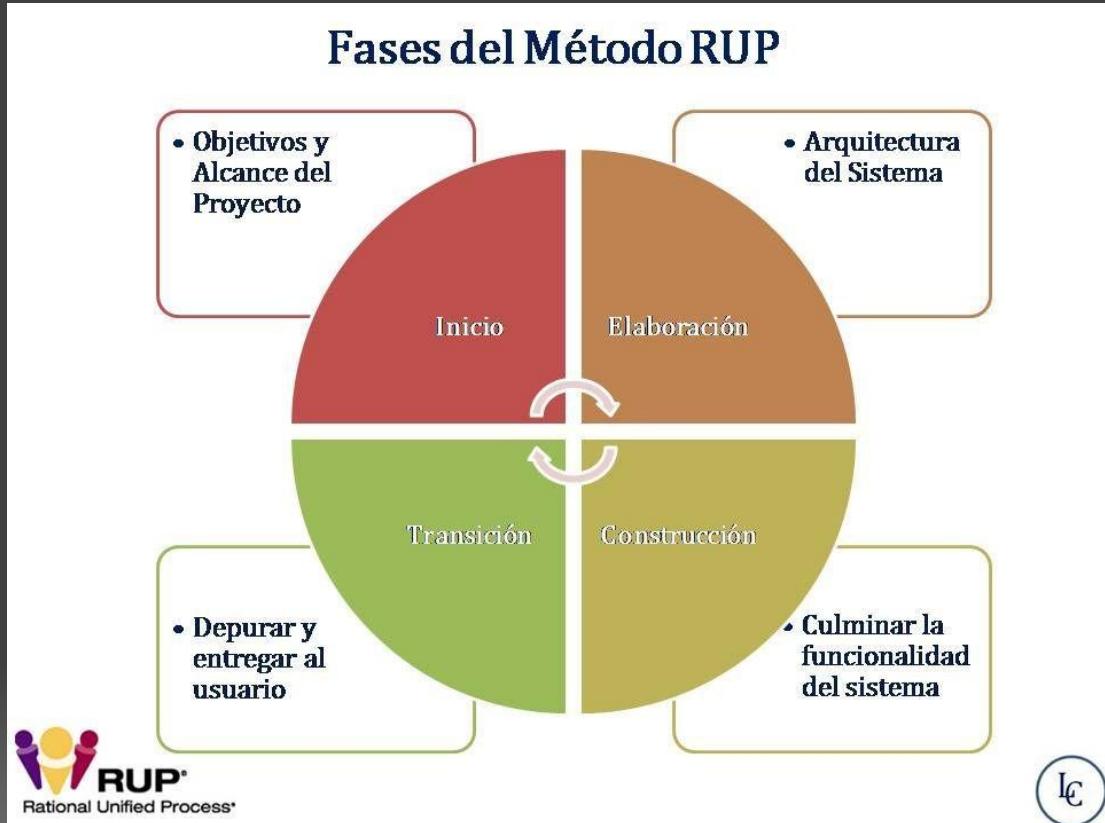
Representa el aspecto estático del proceso, se describe en términos de componentes de proceso, disciplinas, actividades, flujo de trabajo, artefactos y roles.

EJ. Actividades como requerimientos y análisis y diseño.



# RUP - FASES

- Inicio o Inception
- Elaboracion
- Construccion
- Transicion



## RUP - FASES - GENERAL

- Se descompone en cuatro fases secuenciales. En cada extremo de una fase se realiza una evaluación (Revisión para finalización de fase).

Con esto se determina si los objetivos de la fase se han cumplido y permite que el proyecto continúe a la siguiente fase.

- El ciclo de vida consiste en una serie de ciclos, cada uno de los cuales produce una nueva versión del producto, cada ciclo está compuesto por fases y cada fase está compuesta por iteraciones.
- Las tareas no son idénticas en tiempo o esfuerzo.

# RUP - FASES - INCEPCION

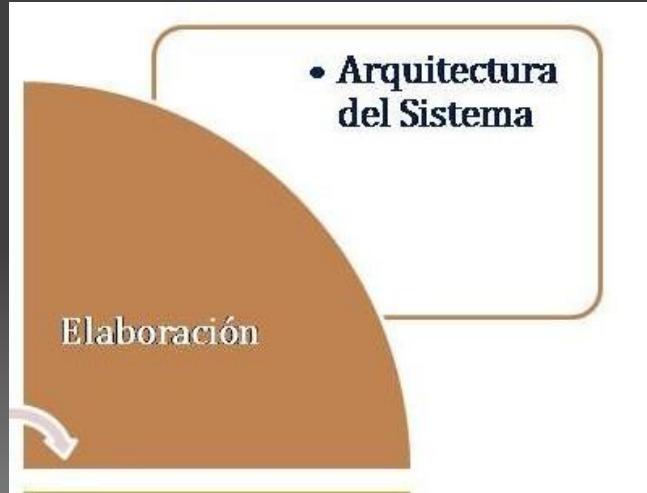
Inicio / Inception / Concepción / Estudio de oportunidad

- Define el ámbito , alcance y objetivos del proyecto
- Se define la funcionalidad y capacidades del producto.



# RUP - FASES - ELABORACION

- La funcionalidad y el dominio del problema se estudian en profundidad
- Se define una arquitectura básica
- Se planifica el proyecto considerando recursos disponibles.



# RUP - FASES - CONSTRUCCION

1. El producto se desarrolla a través de iteraciones donde cada iteración involucra tareas de análisis, diseño e implementación.
2. Las fases de estudio y análisis solo dan arquitectura básica y es en esta fase donde es refinada de manera incremental mientras se construye, permitiendo cambios.
3. Programación y pruebas.
4. Se documenta tanto el sistema como el manejo.
5. Proporciona producto construido junto con documentación



# RUP - FASES - INCEPCION

1. Se libera el producto y se entrega al usuario para uso real.
2. Se incluyen tareas de marketing, empaquetado atractivo, instalación, configuración, entrenamiento, soporte, mantenimiento etc.
3. Los manuales de usuario se completan y refinan con la información anterior.
4. Estas tareas se realizan también en iteraciones





# 02

## QA



## QA - Que es?

“Quality Assurance” es un proceso que asegura que todos los procesos de ingeniería de software, métodos, actividades y código funcional sean monitoreados y cumplan con los estándares definidos



## QA - General

Nos asegura la calidad que el cliente espera y revisa que se cumplan los criterios que se acordaron al definir el requerimiento. Se tiene que enfocar en que el producto final no produzca errores, bugs o cualquier falla de software.

Todo equipo de desarrollo debería contar con al menos un responsable encargado de asegurar el correcto funcionamiento del software que se desarrolla.

# QA - Analista de QA - FUNCIONES



Analista de control de calidad son los encargados de asegurarse que el producto, software o app funcione correctamente antes de lanzarla al público o entregarlo al cliente.

# QA - Analista de QA - FUNCIONES



- Se aseguran que no hayan errores, que le sitio sea fácil de utilizar y que todo funcione según lo previsto.
- Encuentran errores pero no los corrigen.
- Revisa el código para verificar que se cumplan estándares, busca optimizaciones y busca de errores o fugas.
- Desarrollar un plan de prueba.
- Documentar los errores.

## QA - QA en equipos de Scrum



Para conseguir que los objetivos que plantean las metodologías ágiles, desarrollo y QA deben ir de la mano e integrados día a día. Cada equipo de metodologías ágiles debería contar con un encargado de QA para que siga el proceso durante el análisis, desarrollo y verificación del requerimiento.

El QA debe contar con parte de negocio y parte técnica para ayudar a traducir lo que el cliente quiere en pruebas y validar los criterios de aceptación.

02

## Actividades Scrum Practico

# Sistema de Control de Viajes (Turismo)

# Scrum - Sistema

Aplicación móvil/**web** enfocada a la planificación de viajes, se basa en la idea de viajar a otros países y hacer más fácil la planificación que esto conlleva. Con la ayuda de la aplicación se da a conocer una lista de opciones para visitar, siendo esta recomendada por las personas que han visitado esos lugares, con esto el usuario solamente tendrá que seleccionar los lugares donde quiere pasar el tiempo y así organizar su tiempo de mejor manera. Los usuarios podrán recomendar nuevos lugares, agregar nuevos lugares y demás opciones. Plan de Suscripción y contacto de guia turistico.



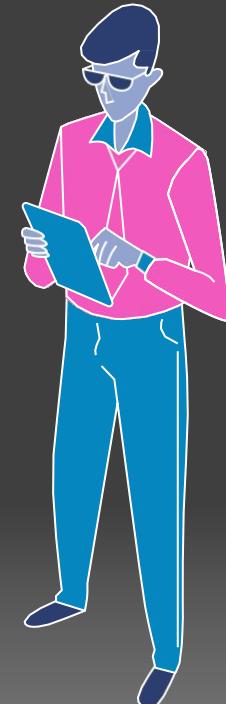
# Scrum - Análisis y Diseño - Historia de Usuario

- Lenguaje: Java / JavaScript
- Framework: Spring
- Arquitectura: MVC
- Backend: PostgresSql
- Control de Versiones: Git
- Host de version de control:  
Github
- Flujo de versionamiento:  
Gitflow
- Herramienta de tablero:  
Gitkraken
- Integracion Continua:  
Jenkins



# Scrum - Análisis y Diseño - Lista de Requerimientos

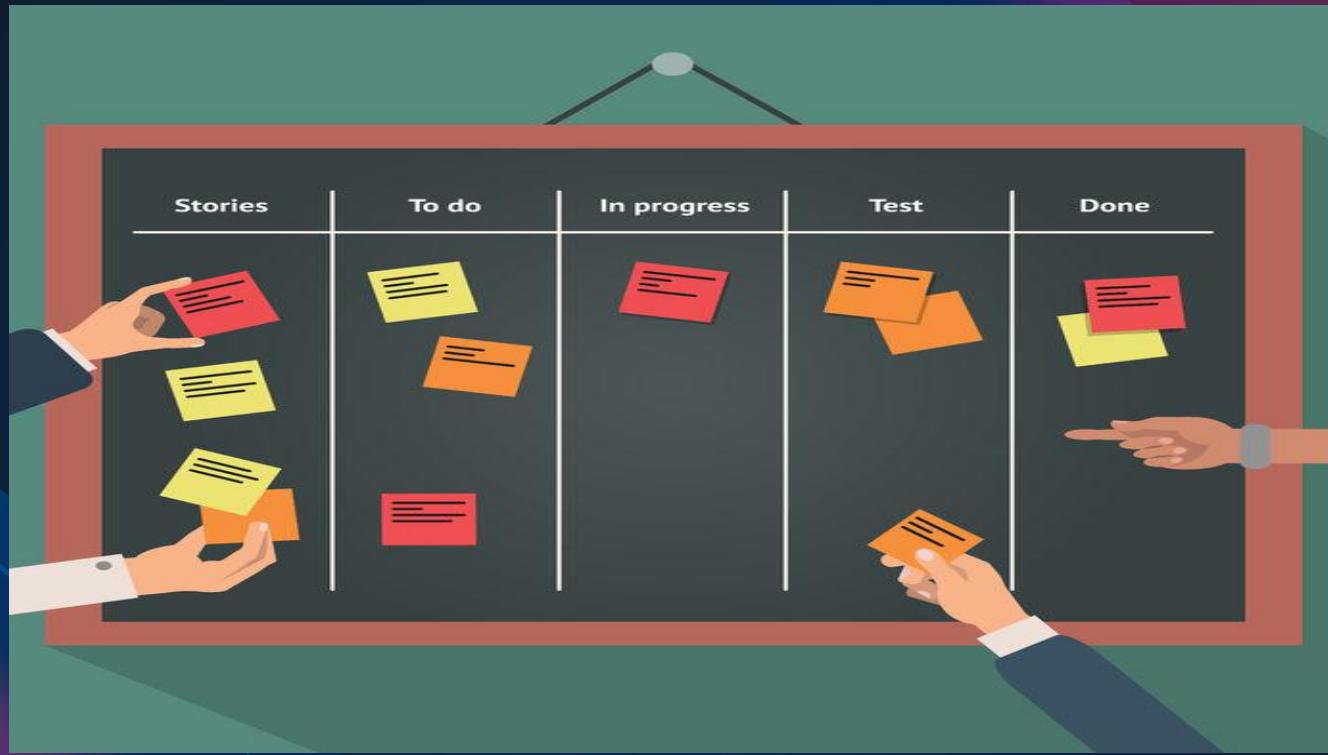
- Buscar por pais 11
- Gestión de precios
- Gestionar calendario personal - Agregar viajes, eliminar y editar. 5
- Clasificación de top lugares.
- Registro de usuario 1
- Login de usuario 1
- Reservaciones de hotel
- Reservaciones de guia 11
- Gestión viajes - agregar presupuestos. 11
- Planear un viaje - Agregar sitios, eliminar y editar. 11
- Recomendación de lugares. 2
- Gestionar plan subscripciones 9
- Gestionar subscripciones por usuario 9
- Mostrar catálogo de sitios 11
- Filtrar catálogo de la página principal 10
- Comentarios de lugares
- Contactar guia turistico
- Valorar guías turísticos y sitios 10
- Diferenciar el tipo de usuario y modular la aplicación.



# Scrum - Análisis y Diseño - Historias de Usuario

- Como usuario quiero buscar sitios turísticos por país para visualizar los sitios turísticos que me interesen.
- Como administrador del sistema quiero gestionar los precios de suscripciones para tener suscripciones dinámicas conforme el tiempo.
- Como usuario quiero tener la capacidad de gestionar mi calendario personal para agregar viajes que he planeado.
- Como usuario quiero clasificar el catálogo de sitios para tener los sitios más recomendados de un país.
- Como usuario del sistema quiero registrarme al sistema para tener un perfil personalizado dentro de la aplicación.
- Como usuario del sistema quiero logearme para tener los accesos correspondientes a mi información.
- Como usuario quiero crear un plan de viajes para colocar sitios turísticos dentro de él.
- Como usuario quiero agregar un plan de viaje a mi viaje para repetir viaje planeado.
- Como usuario necesito tener la habilidad de agregar sitios a mi plan de viajes para tener una lista dinamica de viajes incremental.
  - Como [Usuario] quiero [objetivo] para [motivo].
  - Como [Usuario] necesito [objetivo] para [motivo].





# Scrum - Sprint Planning Meeting



# Scrum - Daily Meet



03

# Tarea/Hoja de Trabajo

# Hoja de Trabajo

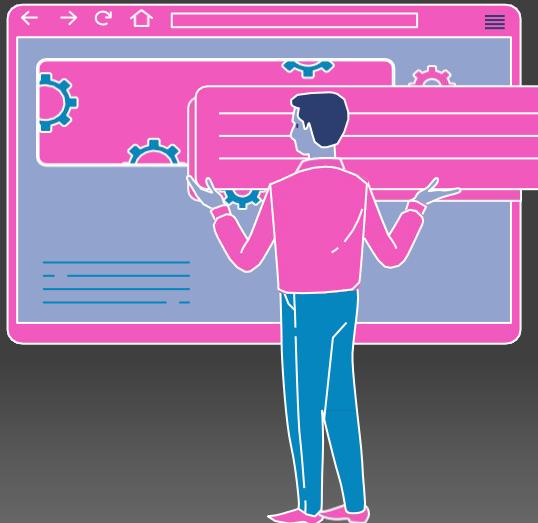
## 3



Errores de Gitflow en  
repositorio



Entrega  
Individual



04

Practica

1

# DUDAS Y COMENTARIOS



# CLASE 5 - Laboratorio

Análisis y Diseño de  
Sistemas 1 Sección A+ 1er  
Semestre 2022

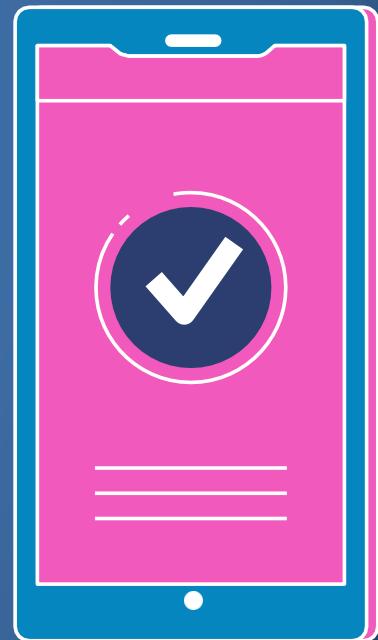
# Temas A Cubrir

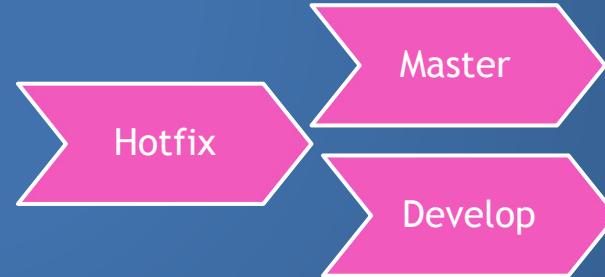
1. Información General
2. Scrum
3. Clase 5 - Pruebas  
Unitarias
4. Tareas/Hoja de trabajo
5. Practica 1/Hoja de  
Calificación
6. Corto 1

0  
1  
Información General

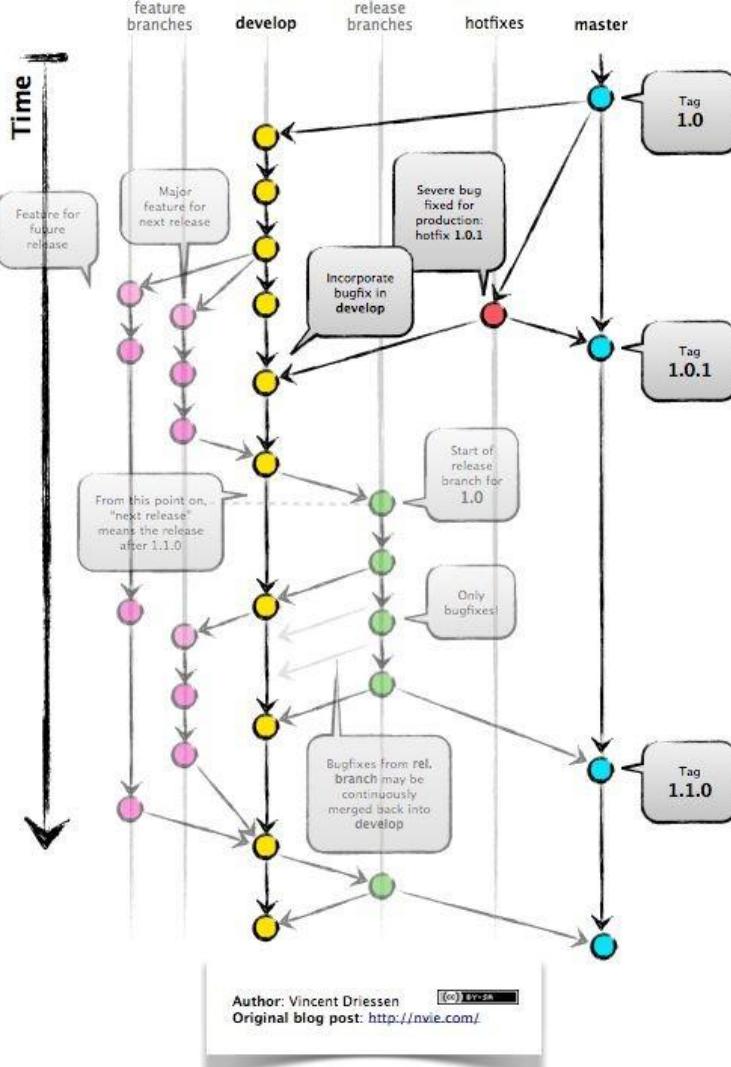
# General

- Nombramiento de ramas
- Nombramiento de versiones
- Definiendo un Tag
- QA
- Scrum y su equipo





# Forma 1



```
graph LR; A[Release] --> B[Master]; A --> C[Develop]; D[Hotfix] --> E[Master]; D --> F[Develop]
```

Release

Master

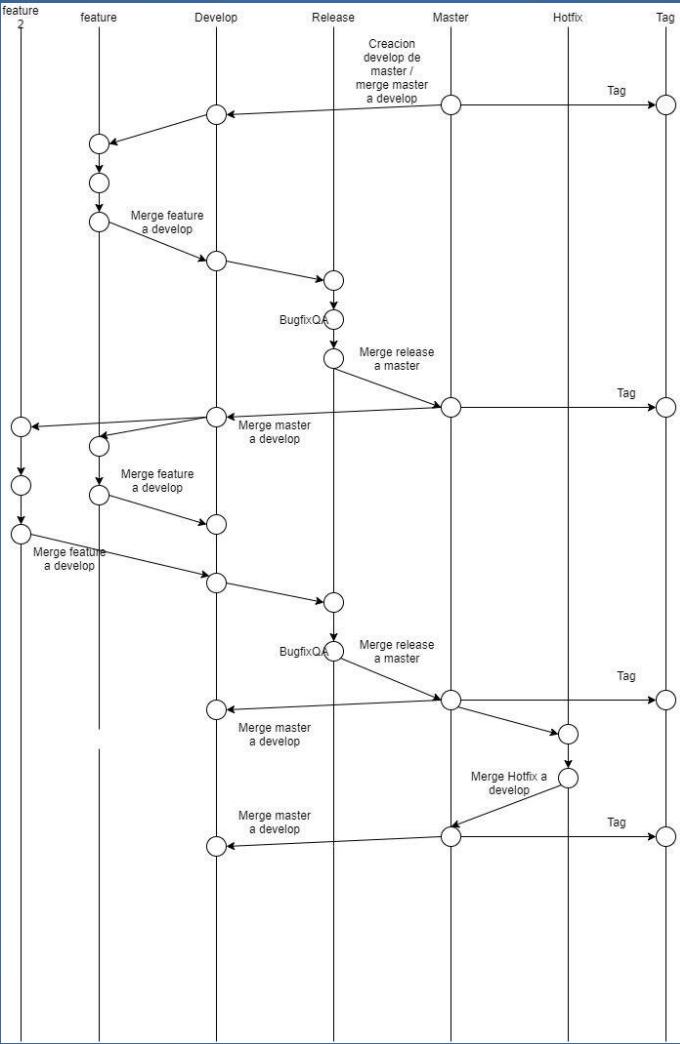
Develop

Hotfix

Master

Develop

## Forma 2



02

## Actividades Scrum Practico

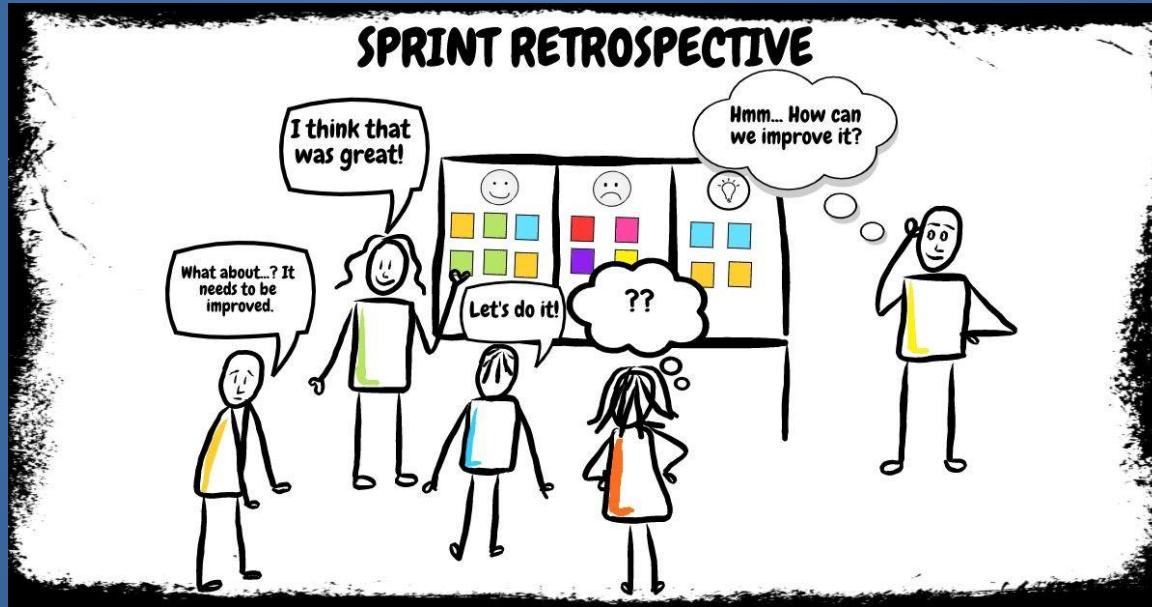
# Scrum - Daily Meet - Practica 1



# Scrum - Sprint Review - Práctica 1



# Scrum - Sprint Retrospective - Practica 1

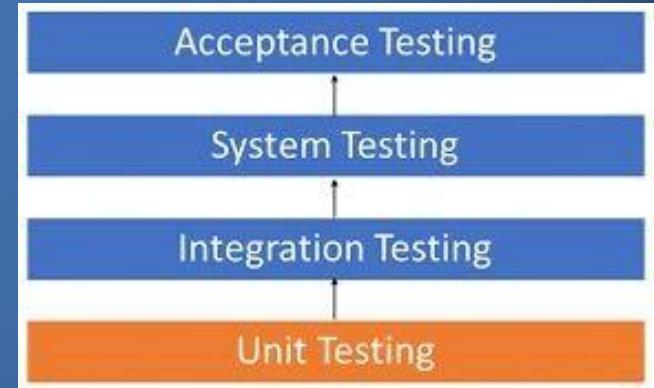


## Unit Testing



03

# Pruebas Unitarias



# Prueba Unitaria

Una prueba unitaria o unit test es un método que prueba una unidad estructural de código.

Comúnmente se piensa que las pruebas unitarias restan tiempo a tareas más importantes.

Son simples y rápidas de codificar, una prueba unitaria no debería tomar más de cinco minutos.

# Características

Debido a la diversidad de definiciones, convendremos que una “buena” prueba unitaria tiene las siguientes características.

- **Unitaria:** Prueba solamente pequeñas cantidades de código.
- **Independiente:** No debe depender ni afectar a otras pruebas unitarias.
- **Prueba métodos públicos:** De otra forma la prueba sería frágil a cambios en la implementación y no se podría utilizar en pruebas de regresión



- Automatizable: La prueba no debería requerir intervención manual.
- Repetible y predecible: La prueba no debe incidir el orden y las veces que se repita la prueba, el resultado debe ser el mismo.
- Profesionales: Las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación.



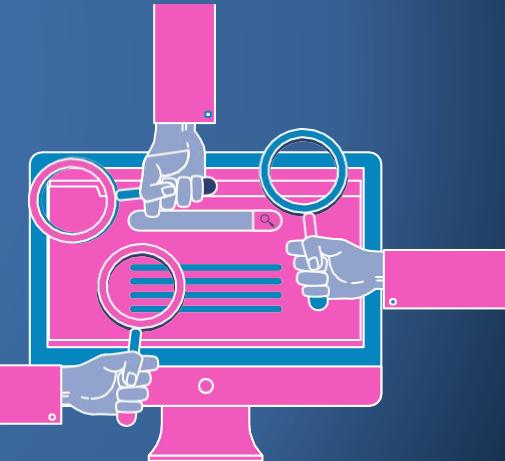
Respecto a la profesionalidad y contrario a lo que piensan muchos desarrolladores - que el desarrollo de pruebas unitarias resta tiempo a tareas más importantes - Las pruebas unitarias por lo general son simples y rápidas de codificar, el desarrollo de una prueba unitaria no debería tomar más de cinco minutos.



## VENTAJAS

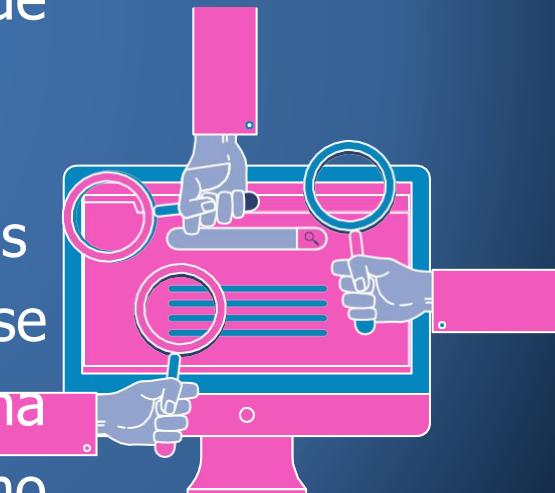
Las pruebas unitarias buscan aislar cada parte del programa y mostrar que las partes individuales son correctas, con las siguientes ventajas

- Fomentan el Cambio: Las pruebas unitarias facilitan la reestructuración del código (refactorización), puesto que permiten hacer pruebas sobre los cambios y verificar que las modificaciones no han introducido errores(regresión).



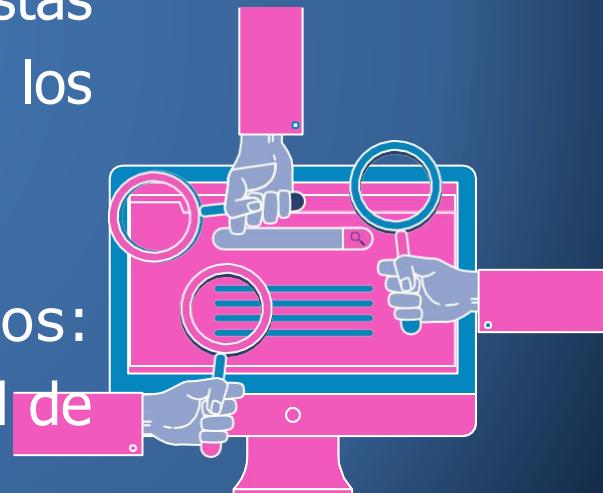
## VENTAJAS

- Simplifican la Integración: Permiten llegar a la fase de integración asegurando que las partes individuales funcionan correctamente. De esta manera se facilitan las pruebas de integración.
- Documentan el Código: Las propias pruebas pueden considerarse documentación, ya que las mismas son una implementación de referencia de cómo utilizar la API.



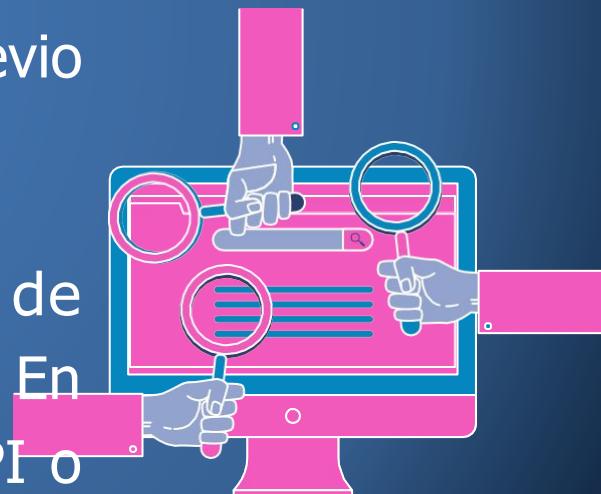
## VENTAJAS

- Separación de la Interfaz y la Implementación: La única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.
- Menos Errores y Facilita Localizarlos: Las pruebas unitarias reducen la cantidad de errores y el tiempo en localizarlos.



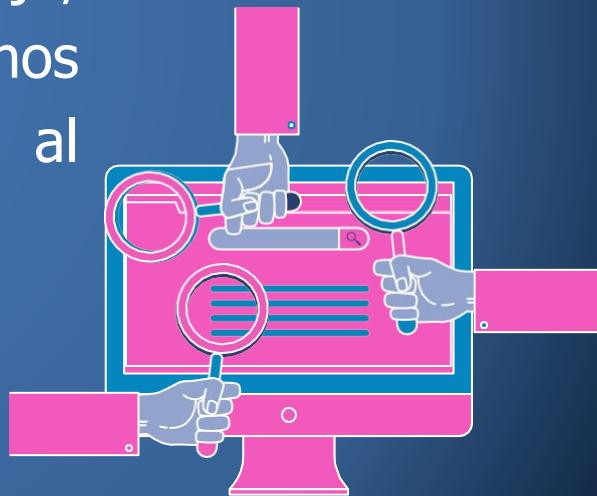
## VENTAJAS

- Pueden Mejorar el Diseño: La utilización de prácticas de diseño y desarrollo dirigida por las pruebas (Test Driven Development/TDD) permite definir el comportamiento esperado en un paso previo a la codificación.
- Puede Ser la Forma Más Simple de Verificar el Funcionamiento: En situaciones como el desarrollo de una API o componente que brinda servicios del cual no



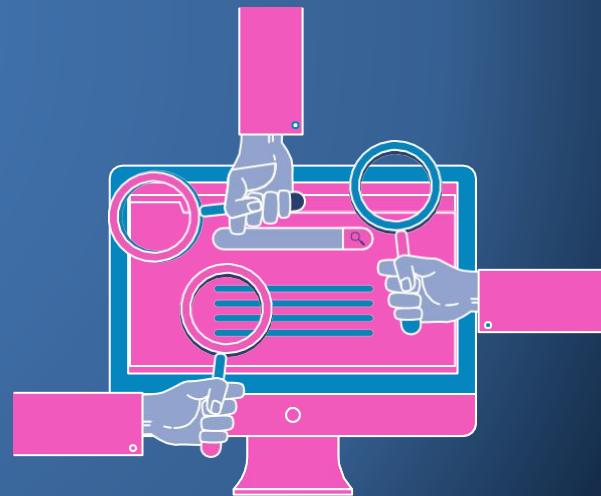
## VENTAJAS

- Ahorra tiempo en el futuro: Adoptar el uso de pruebas unitarias como una disciplina generalizada puede ser un trabajo costoso, pero no debe ser considerado un desventaja, debe entenderse que esta actividad nos permitirá ahorrar tiempo en el futuro al disminuir la ocurrencia de errores



## VENTAJAS

- Mejora el código: La utilización de pruebas unitarias permitirá mejorar progresivamente la calidad del código en la medida que los desarrolladores aumentan la calidad de las pruebas y la cobertura.



# Consideraci

## ones

Como en la adopción de cualquier otra disciplina, la incorporación de pruebas unitarias es posible que ocurran problemas o limitaciones.

- Por estar orientada a la prueba de fragmentos de código aislados, las pruebas unitarias no descubren errores de integración, problemas de rendimiento y otros problemas que afectan a todo el sistema de conjunto.
- En algunos casos será complicado anticipar inicialmente cuales son los valores de entradas adecuados para las pruebas, en esos casos las pruebas deberán evolucionar e ir incorporando

# Consideraci

## ones

Si la utilización de pruebas unitarias no se incorpora como parte de la metodología de trabajo, probablemente, el código quedaría fuera de sincronismo con los casos de prueba.

- Otro desafío es el desarrollo de casos de prueba realistas y útiles. Es necesario crear condiciones iniciales para que la porción de aplicación que está siendo probada funcione como parte completa del sistema al que pertenece.

# Pruebas unitarias utilizando objetos simulados

- Existen pruebas unitarias que utilizan objetos simulados que sustituyen a los objetos reales utilizados por la clase o fragmento de código a utilizar.

Ejemplo:

Se tiene la clase calculadora que necesita un objeto de acceso a datos para obtener información de la base de datos, este objeto es el objeto real. Si quisiéramos realizar una prueba de la clase Calculadora, deberíamos pasarle al objeto una conexión válida a la base de datos y asegurarnos que los datos que necesita existan.



# Pruebas unitarias utilizando objetos simulados

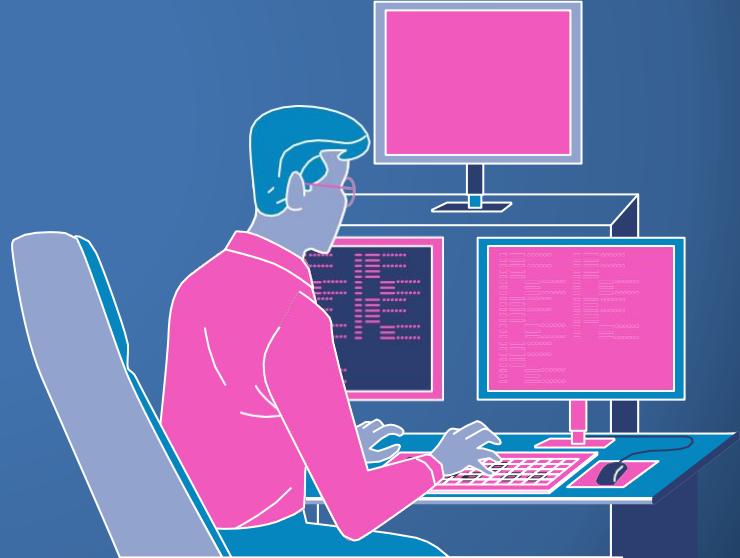
- Este proceso de determinar los datos que el objeto necesita e insertarlos a la base de datos es mucho trabajo para una prueba unitaria.
- En lugar de esto, se puede proveer una instancia falsa del objeto de acceso de datos que regrese lo que necesitamos para la prueba.



# Objetos Simulados

 Stub  
Mock

 Proxy

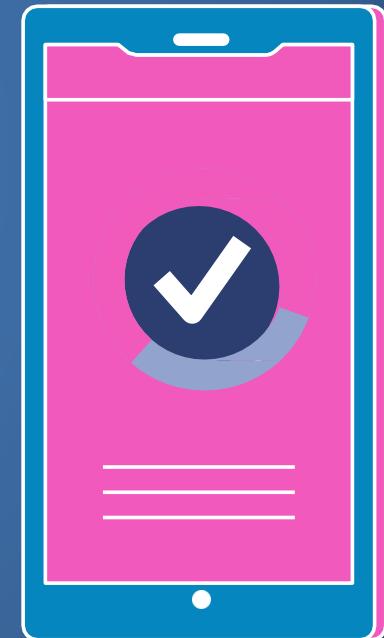


# QUE SON?

Son objetos que se parecen a un objeto de verdad pero no tienen dependencia de otros objetos.

# Ventajas

- Se pueden programar de manera sencilla con las especificaciones que deseamos para realizar la prueba
- Permite registrar cualquier tipo de interacción que haya ocurrido con el objeto.

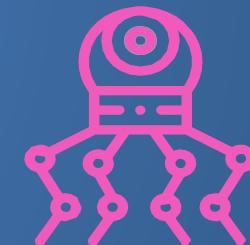


## STUB

- Es un objeto falso que se le programan objetos de retorno.
- Está programado solo para dar respuestas.
- Es el objeto simulado más liviano

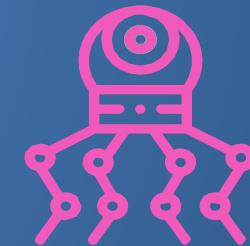
## FUNCIONES

- Siempre regresa los mismos valores sin importar el input.
- Se pueden utilizar para simular objetos de base de datos.



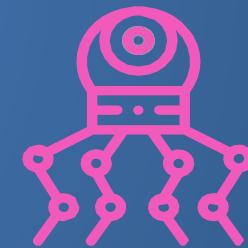
# MOCK

- Es un objeto falso que se le programan objetos de retorno.
- Guarda llamadas a métodos.
- Es usado para guardar y verificar interacciones.
- Considerado el más poderoso y flexible de los objetos.
- Otorga control total del comportamiento de los objetos.



# Spy / Proxy

- Es un objeto falso que se le programan objetos de retorno.
- Cualquier comportamiento que no ha sido mockeado son reales y funcionan como un objeto real.
- Técnicamente es sustituir el objeto real y reemplazar solo algunos métodos.
- Útil para testear objetos con muchos



# CORTO 1

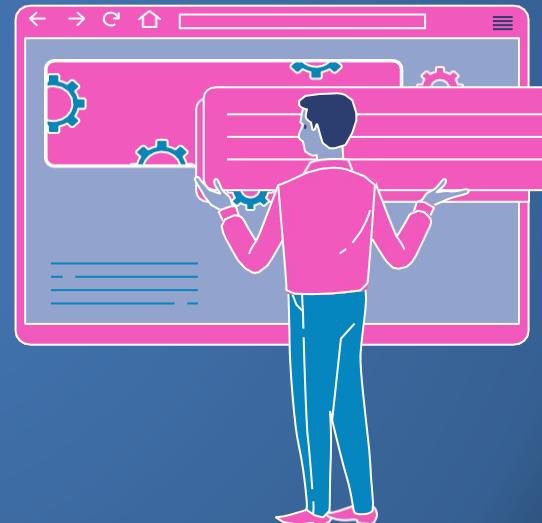


## Curso Pruebas Unitarias

[https://www.udemy.com/course/junit-quick-start-for-beginners  
-java-unit-testing](https://www.udemy.com/course/junit-quick-start-for-beginners-java-unit-testing)



Entrega  
Individual



05

Practica 1 /  
HC

# DUDAS Y COMENTARIOS



# CLASE 6 - Laboratorio

Análisis y Diseño de  
Sistemas 1 Sección A+  
1er Semestre 2022

# Agenda

1. Extreme Programming
2. Test Driven Development
3. Behavior Driven Development

0

1

EXTREME PROGRAMMING

Es una metodología ágil enfocada en no nonsense que se enfoca en cuatro actividades

Crear código porque si no se crea código, no se hace nada.  
Testear porque si no se realizan pruebas no se sabe cuando se termina de codear

Escuchar porque si no se escucha no se sabe que codear o que probar  
Diseñar para lograr Crear, Testear y Escuchar indefinidamente.

# Valores

- Comunicacion
- Sencillez
- Retroalimentaci  
ón
- Valentia
- Respeto

# Principios

- Feedback inmediato.
- Tendencia a lo sencillo
  - You aint gonna need it: Mientras una función no se solicite, no se debe implementar.
  - Dont repeat yourself: Debe evitarse repetir tareas y diseñar código de manera que se deben aplicar los cambios en varios puntos.

# Principios

- Modificaciones incrementales: Modificaciones en pasos pequeños.
- Aceptación de modificaciones.
- Alta calidad.

# Técnicas

Las prácticas de XP son instrucciones muy concretas.

Los valores y principios también se presentan en otras metodologías pero las técnicas concretas de extreme programming son totalmente diferentes.

# Feedback fino

Se trabaja en ciclos extremadamente cortos para que se pueda comprobar el código una y otra vez.

- Test-driven development: Pruebas primero, código después.
- Planning game:
  - On-site customer: Representante del cliente forme parte del equipo para resolver dudas, aportar ideas o transmitir prioridades.

# Feedback fino

- Pair Programming:
  - Un desarrollador escribe código
  - El otro desarrollador comprueba, transmite consejos y señala errores.

# Proceso Continuo

Los equipos XP están continuamente rehaciendo su código, con la idea que refactoring mejore el código fuente y elimine repeticiones.

La integración continua permite a los desarrolladores escribir su trabajo varias veces al día en el proyecto (Integracion en cuestión de horas) y así comprobar continuamente aportaciones y los implicados.

# Proceso Continuo

Los programas y actualizaciones que funcionan se publican lo antes posible en small releases.

# Comprensión Conjunta

Diseño simple: Todo el mundo puede entender el código, todo lo que complique el texto debe eliminarse.

Coding-standards: Para que todo el equipo pueda trabajar en estrecha colaboración.

Propiedad conjunta del código: Evita enfocarse en que parte de responsabilidad y errores tiene cada miembro, se comparte la responsabilidad tanto de errores como éxito.

# Comprensión Conjunta

System metaphor: Consiste en describir el proyecto de la manera más sencilla posible, realizando nombres autodescriptivos en la medida de lo posible, para que la integración de miembros sea más sencilla.

# Bienestar de los desarrolladores

El bienestar del equipo es importante para el éxito del proyecto.

Un empleado descansado y motivado puede ofrecer un rendimiento y resultados de calidad.

En extreme programming se impone semana de 40 horas y se evitan las horas extra a toda costa.

VENTAJAS	INCONVENIENTES
Relación estrecha con el cliente	Mayor esfuerzo de trabajo
Ausencia de trabajos de programación innecesarios	El cliente se implica en el proceso
Software estable debido a continuas pruebas	Requiere mucho tiempo
Menos errores gracias a la programación en pareja	Relativamente caro
Ausencia de horas extra, gestión propia de tiempo	Requiere control de versiones obligatoriamente
Aplicación rápida de cambios	Requiere autodisciplina en la aplicación
Código de comprensión sencilla en todo momento	

02

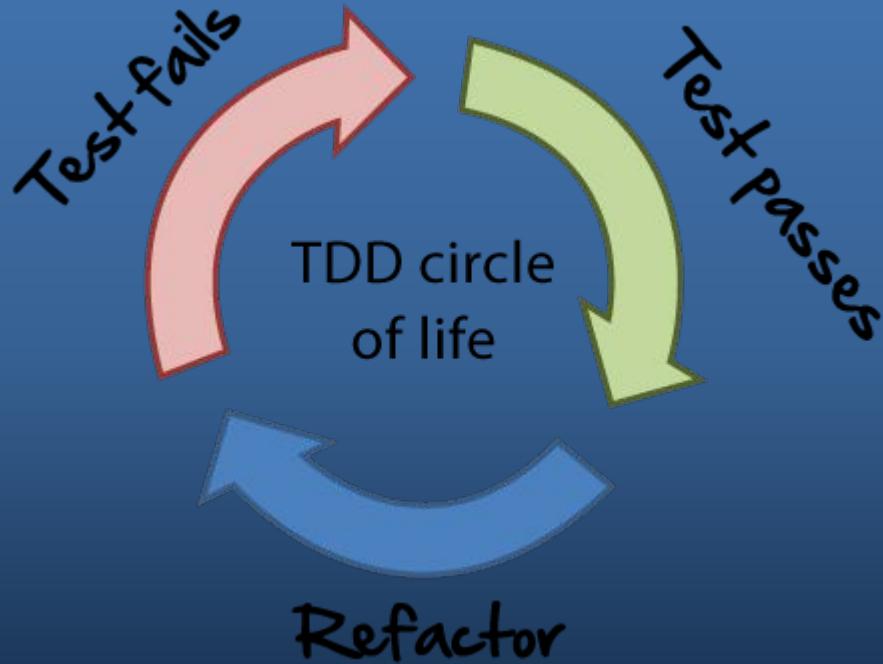
# Test Driven Development

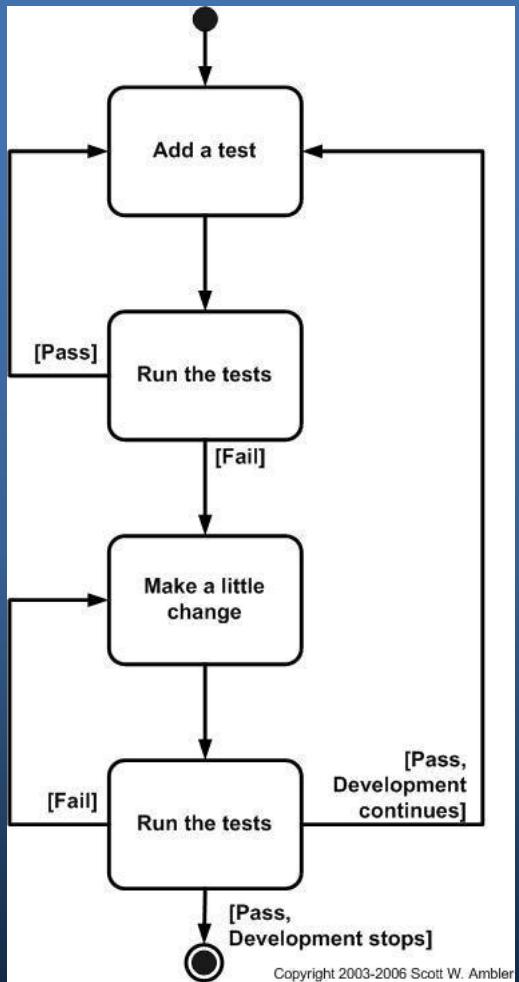
# De que trata?

Se realizan las pruebas antes de agregar funcionalidad al sistema logrando:

- Definir éxito desde el inicio
- Ayuda a descomponer el diseño en pequeñas partes
- Una gran cantidad de test que prueban que lo que realizamos funciona.

# Ciclo





# Reglas

- No se puede escribir código de producción al menos que sea para que una prueba fallida pase.
- No escribir más de la prueba unitaria si ya es suficiente para que falle.
- No realizar más código para que la única prueba fallida pase

# Pasos

1. Escribir test para caso
2. Correr Test
3. Test falla
4. Se escribe código
5. Se evalúa si se puede refactorizar el código y proceder.
6. Si el refactor afecta las demás pruebas se realiza mal.
7. Correr Test
8. Test Pasa
9. Repetir

03

# Behavior Driven Development

# Que es?

Es una rama de TDD que utiliza descripciones de software fácil de entender, sin necesidad de alto conocimiento en informática, para los requerimientos del usuario.

# Requerimientos antes de utilizar BDD

- Cada requerimiento debe convertirse en historias de usuario.
- Cada caso debe ser un escenario de un usuario en el sistema.
- Se debe tomar como especificación de comportamiento de un usuario envés de prueba unitaria de una clase.
- Tener clara el formato GIVEN-WHEN-THEN y

## GIVEN-WHEN-THEN

- Given: En español “dado”, especifica el escenario y las precondiciones.
- When: En español “cuando”, las condiciones de las acciones que se van a ejecutar.
- Then: En español “entonces”, El resultado esperado, las validaciones a realizar

## EJEMPLO GIVEN-WHEN-THEN

- Given/Dado que el usuario no ingresó ningún dato al formulario.
- When/Cuando realiza click en submit
- Then/Entonces se deben indicar al usuario las validaciones para enviar el formulario.

## ROLE-FEATURE-REASON

- As a: En español “como”, Se especifica el tipo de usuario.
- I want: En español “deseo”, Las necesidades que tiene.
- So that: En español “para”, Las características para cumplir el objetivo.

## EJEMPLO ROLE-FEATURE-REASON

- As a/Como cliente interesado.
- I want/deseo registrarme por medio del formulario
- So that/Para que se cree un usuario para mi persona.

# Ventajas

- No se definen pruebas sino comportamientos.
- Mejora comunicación entre desarrolladores, testes, usuarios y gerencia.
- Curva de aprendizaje es más corta que TDD.
- Cubre un público más amplio.
- El enfoque de definición ayuda a una aceptación común de las funcionalidades antes del desarrollo.
- Encaja bien con metodologías ágiles que utilizan historias de usuario.

# Ejemplo

Caracteristica: Suma de dos números

Como matematico novato

Yo quiero obtener la suma de dos cifras Para aprender a sumar

Escenario: Sumar dos numeros

positivos Dado que estoy en la

aplicación

Cuando ingreso los números  
1 y 3 Y solicito el resultado

Escenario: Sumar dos numeros negativos

Dado que estoy en la aplicación Cuando ingreso los números -1 y -3 Y solicito el resultado del cálculo

Entonces el resultado debe ser -4

Escenario: Sumar numero negativo y uno positivo Dado que estoy en la aplicación

# DUDAS Y COMENTARIOS



# CLASE 7 - Laboratorio

Análisis y Diseño de  
Sistemas 1 Sección A+ 1er  
Semestre 2022

# Agenda

1. Pruebas Funcionales
2. Pruebas No Funcionales
3. Behavior Driven Development
4. Hoja de Trabajo 4
5. Practica 2

01

# Pruebas Funcionales

# Que

# son?

Se centran en comprobar que los sistemas desarrollados funcionen acorde a las especificaciones funcionales y requerimientos del cliente.

- Validan que los sistemas sean eficientes y sin errores.

# Ventajas

- Mitigación del riesgo de aparición de fallos en producción.
- Cumplimiento de objetivos del proyecto en calidad y resultado.
- Permite evitar problemas con proveedores, elevación de costos y permite generar mayor confianza en el sistema.

# Fases

01

## Análisis de requerimientos

Revisión de la documentación entregada por el jefe de proyecto de desarrollo.

02

## Plan de Pruebas

Generación de documento que será entregado al Jefe de Proyecto de desarrollo donde se identifican las consideraciones para la ejecución de pruebas, responsabilidades, métricas de aceptación, etc.)

03

## Casos de Pruebas

Diseño de los casos de prueba que se utilizarán para realizar la certificación del software validando que cumpla con lo solicitado por el usuario.

# Fases

04

Ejecución

Ejecución de los casos de pruebas diseñados. Se realiza detección de errores en la aplicación o incidencias.

05

Reporte de Cierre

Corresponde a la generación de reporte de cierre con los resultados finales de la certificación de software.

# Tipos de Pruebas Funcionales

Pruebas  
unitarias

Pruebas de  
integración

Pruebas de  
componentes

Pruebas de  
regresión

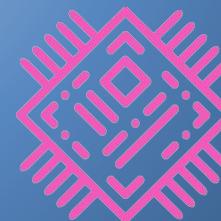
Pruebas de  
aceptación

Pruebas de  
humo

Pruebas de  
cordura

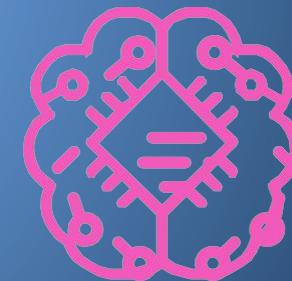
# Pruebas Unitarias

- Se ejecutan de forma independiente para comprobar que el resultado sea el requerido.
- Verifica las funcionalidades y usabilidades de los componentes.
- Aseguran que cada célula del código desarrollado de un componente brinde los resultados adecuados.
- Componentes que se pueden probar:
  - Prueba de UI
  - Prueba de carga
  - Inyección de SQL
  - Prueba de login



# Prueba de Humo

- Se realizan para verificar si las funcionalidades más significativas de la aplicación funcionan o no.
- Se trata de verificar que la funcionalidad crítica del sistema realmente funcione.
- De ser exitosa significa que es una compilación estable.



# Prueba de Integración

- Generalmente son automatizadas.
- Prueba componentes individuales con el objetivo de verificar como los módulos que trabajan de forma individual funcionen bien cuando se integren.
- Permite que todo actúe como partes de un solo sistema en lugar de aplicaciones aisladas.



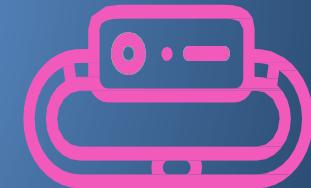
# Prueba de Regresión

- Los desarrolladores pueden modificar y mejorar funcionalidades del desarrollo. Lo que causa posibilidad de que puedan causar efectos inesperados en su comportamiento.
- Aseguran que los cambios o adiciones no hayan alterado ni eliminado las funcionalidades existentes.
- Detecta errores que pueden haber sido



## Prueba de Corrección para modificaciones menores.

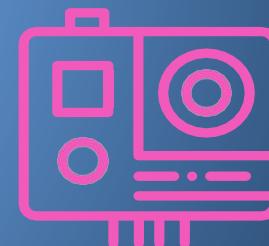
- Determina que las modificaciones hayan solucionado el problema y que la corrección no haya generado más problemas.
- Detecta errores que pueden haber sido introducidos accidentalmente en la compilación existente.
- Analizan profundamente la compilación



# Pruebas De Aceptación del UsUARIO

• Última fase del proceso de testing.

- Los usuarios reales del software lo usan para verificar que cumpla con las tareas requeridas en un ambiente real.
- Esto se puede realizar en un servidor de preproducción dedicado a estas pruebas con una copia del proyecto en producción.



02

# Pruebas no Funcionales

# Que son?

Pruebas enfocadas en validar un sistema o aplicación por medio de sus requerimientos no funcionales



Permite conocer los riesgos que corre el producto e indica si tiene un mal desempeño o bajo rendimiento.



Ayuda a establecer cuánta carga puede manejar el servidor o el sistema.



Permite explicar lo que soporta el producto y si cumple con las expectativas de los clientes



# Beneficios

- Mejora la experiencia de usuario.
- Utiliza métricas importantes para poner a punto el sistema.
- Determina cuellos de botella como la configuración de base de datos
- Ayuda a establecer si una nueva versión está preparada para producción.
- Permite a los interesados conocer el rendimiento real comparado a las expectativas



# Tipos de Pruebas No Funcionales

- Pruebas de Carga
- Pruebas de Estrés
- Pruebas de Volumen
- Pruebas de Configuración
- Pruebas de Usabilidad
- Pruebas de Seguridad
- Pruebas de Resistencia
- Pruebas de escalabilidad
- Pruebas de recuperación
- Pruebas de mantenibilidad

04

# Hoja de Trabajo 4

## Curso Pruebas Unitarias

<https://www.udemy.com/course/junit-quick-start-for-beginners-java-unit-testing>

05

## Practica 2

DUDAS  
Y  
COMENTARIOS



# CLASE 8 - Laboratorio

Análisis y Diseño de Sistemas 1

Sección A+ 1er Semestre 2022

Cesar Sazo

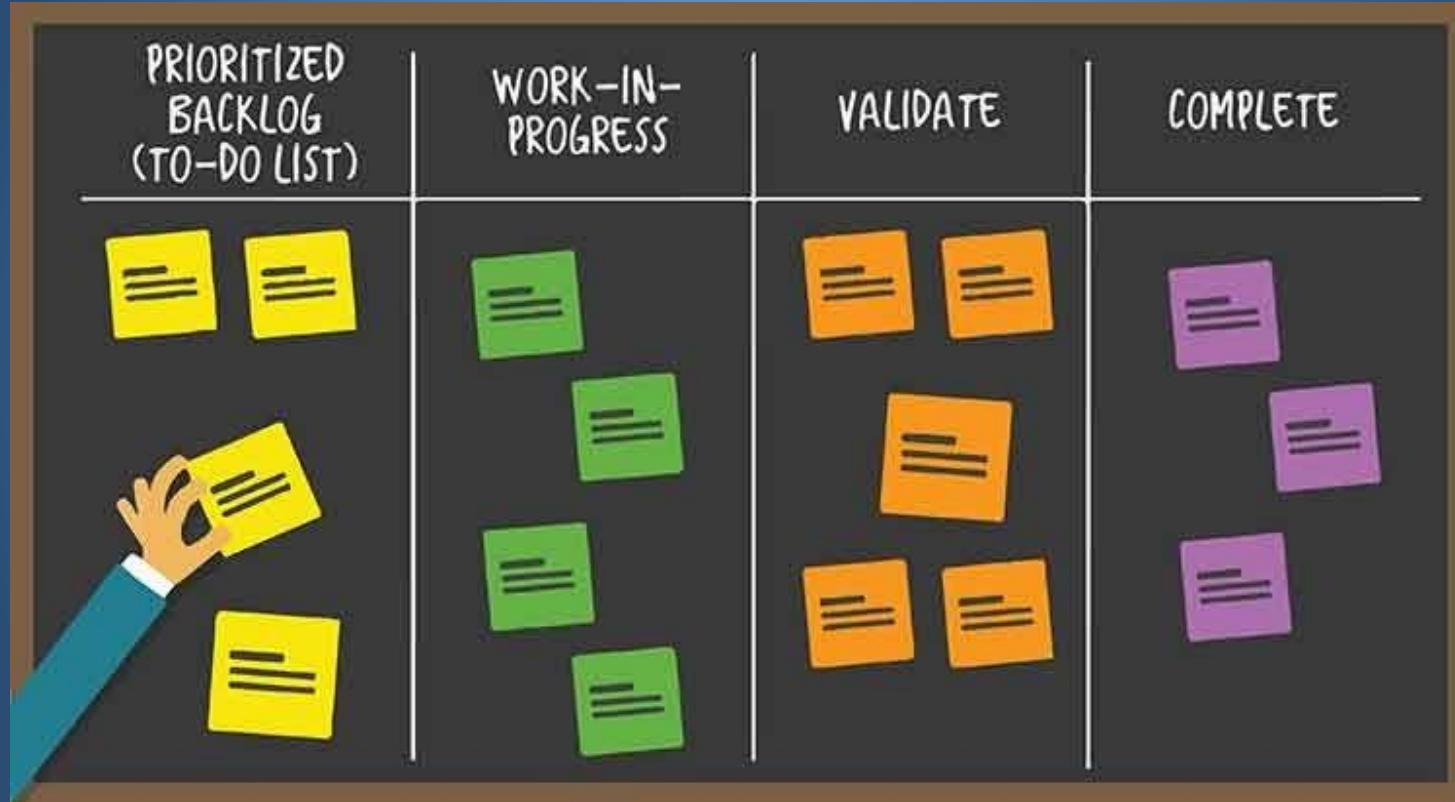
# Agenda

1. Tema: Kanban
2. Devops
3. Práctica 2
4. Corto 2
5. Actividad Sprint

01

Kanban

# Kanban



# Kanban

Es una metodología ágil que surgió en Toyota con la implementación del sistema de producción just in time el cual hace enfoque en producir a base en la demanda de los clientes.

Este sistema de producción tenía como propósito minimizar los desperdicios sin afectar la producción y generar más valor.

La metodología está enfocada en llevar a cabo tareas pendientes.

# 4 Principios de Kanban



## Calidad Garantizada

Todo lo que se produce debe salir a la primera sin margen de error.  
Por lo que se enfoca en la calidad y no en la rapidez



# 4 Principios de Kanban



## Reducción del desperdicio

Se basa en hacer solamente lo necesario. Ni más ni menos y siempre de forma correcta.



# 4 Principios de Kanban



## Mejora Continua

Está diseñado para implementarse con una mínima resistencia, por lo que trata de pequeños y continuos cambios incrementales y evolutivos



# 4 Principios de Kanban

## Flexibilidad

Permite priorizar y escoger tareas según sean las necesidades del negocio y del momento.



# Kanban

La metodología Kanban utiliza los siguientes conceptos:

- Tarjeta Visual
- Tablero
- WIP
- Flujo

# Tablero

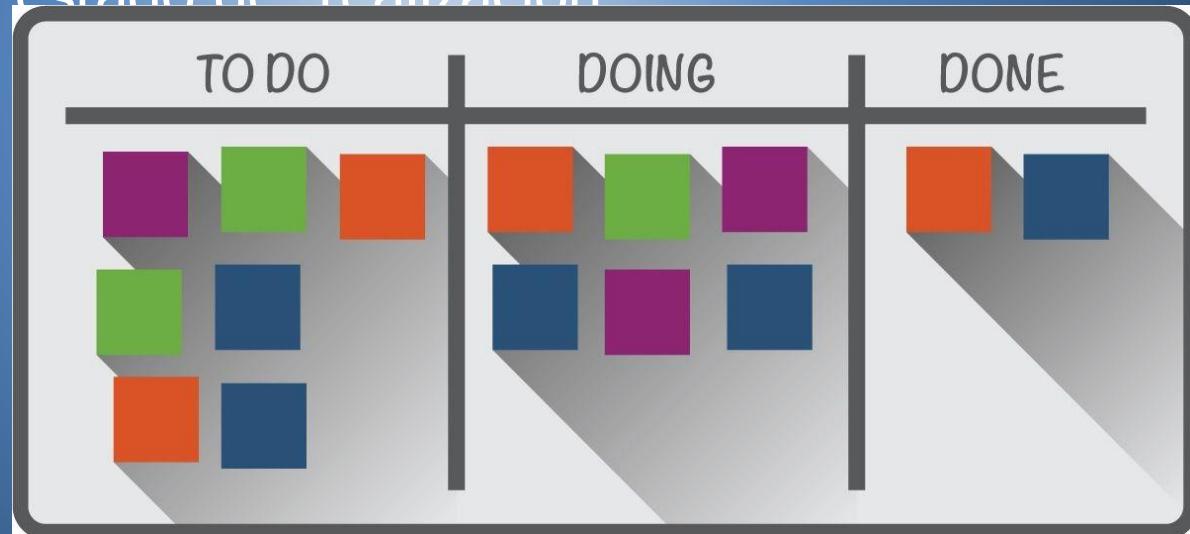
Todo el trabajo realizado por el equipo gira entorno al tablero. Se debe asegurar que el trabajo dentro del tablero se pueda visualizar por todos los miembros del equipo así como el flujo y bloqueos que se encuentren.

El tablero debe mostrar:

- Tarjetas
- Flujo
- WIP

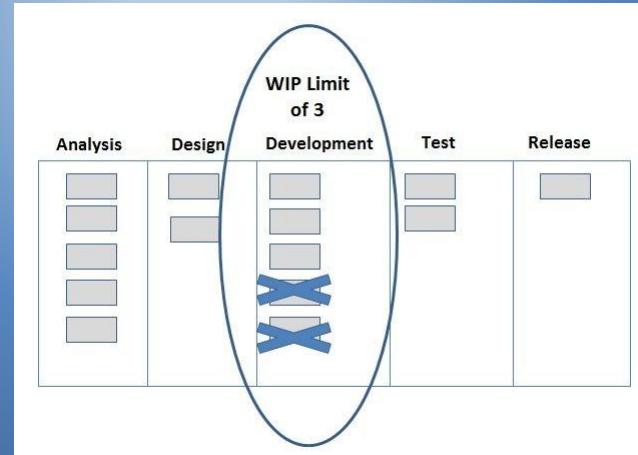
# Tarjetas

Kanban o Señal Visual son las representación de las tareas en el tablero y debe moverse por el tablero según sea su estado de realización



# WIP

Work In Progress o trabajo en progreso, Kanban utiliza el WIP en cada paso del flujo para limitar las tareas que se pueden estar trabajando en la sección del flujo.



# Flujo

El flujo son los pasos y procesos dentro del tablero de Kanban en donde todas las tareas deben de seguir todos los pasos del flujo, La tarjeta Kanban pasa a estar terminada después de seguir el flujo.



# Practicas Kanban

01

Visualizar el  
flujo de trabajo

03

Políticas administrativas  
explícitas

05

Mejorar  
Colaborativamente

02

Limitar WIP

04

Administrar el  
flujo

# Practicas

- Visualizar el flujo: Visualización de todas sus tareas y elementos en una tabla contribuye a que todos los miembros del equipo se mantengan al corriente con su trabajo.
- Limitar WIP: Se deben establecer metas realistas, se debe mantener el equilibrio del flujo de trabajo, limitando la cantidad de trabajo que se realiza y así prevenir el exceso de compromiso en la cantidad de tareas que no se podrán terminar.

# Practicas

- Políticas administrativas: Criterio con el cual se considera que una tarea termino en alguna fase del flujo.
- Administrar el flujo: Al visualizar el avance de las tareas por el tablero se puede identificar como el valor está fluyendo y permite observar mejoras, definirlas e implementarlas.

# Prácticas

- Mejorar colaborativamente: El equipo Kanban debe mejorar constantemente, por lo que se puede utilizar muchos métodos de autoexaminación para ver las fallas del equipo o proceso y enfocarse en esas áreas para mejorar.
  - Dónde están los cuellos de botella?
  - Puede dar problema alguna política planteada?
  - Los procesos están definidos para enfrentar fallas?

# Beneficios Claves

- Estimula el rendimiento: El análisis profundo y estimaciones que permiten medir el rendimiento ayuda a detectar cualquier problema existente y ajustes al flujo de trabajo para obtener mayor eficiencia.

- Trabajo en equipo: La metodología permite beneficiarse del enfoque visual y facilita la forma en la que la información se transmite entre los miembros del equipo de trabajo, facilitando labor e interacciones.



# Beneficios Claves

- Distribución del trabajo: Cómoda visión general de las tareas en curso. El flujo constante de tareas reduce el tiempo de espera y el tiempo dedicado, permite al colaborar seleccionar la tarea que estime sea mejor darle enfoque.
- Rapidez en identificación y solución de problemas: El tablero visual nos permite identificar problemas rápidamente y poder actuar lo más pronto posible para solucionarlo.



01

Kanban

02

# Información Practica 2

03

Corto 2

04

# Practica 3 y 4 GOOGLE CLOUD

# Temas a cubrir

- Scrum
- Gitflow
- Testing funcional (Unitarias y End2End)
- SonarQube
- Integración continua
- Documentación
- Azure DevOps

# DUDAS Y COMENTARIOS