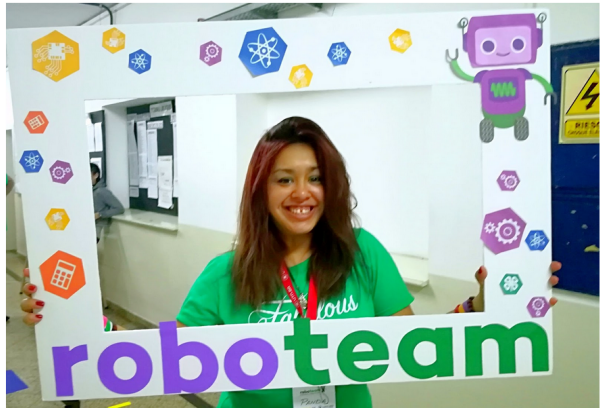
Certificación avanzada en

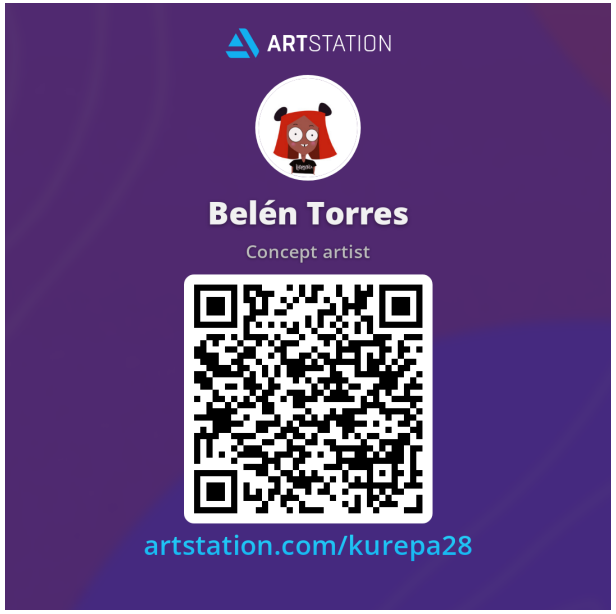
DevOps

Calidad

UNC FCEFyN mundosE PEOPLE & BUSINESS SCHOOL

Quién soy?


roboteam

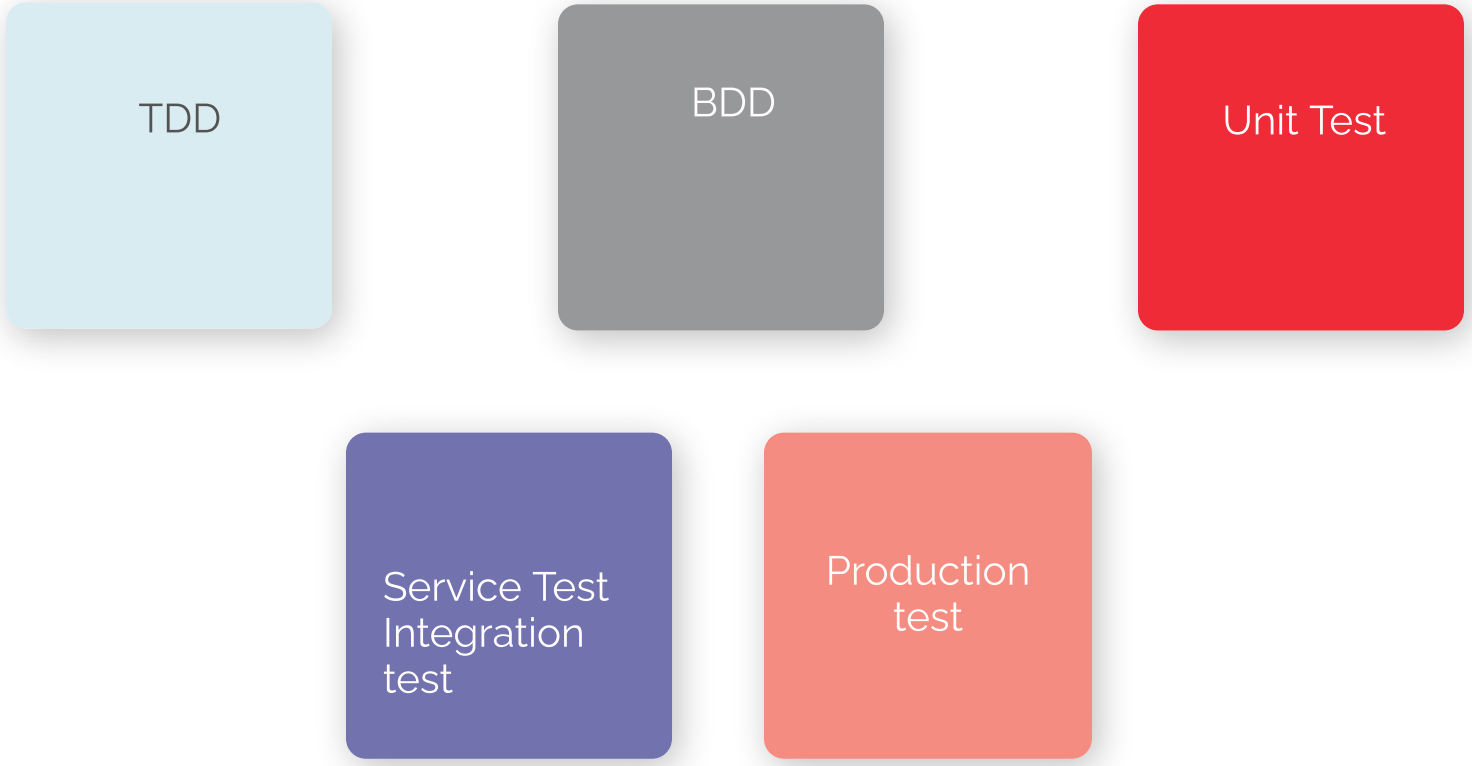

ARTSTATION
Belén Torres
Concept artist
artstation.com/kurepa28


Torres, Anahi Belén
Nombre
Ekumen
Institución/Empresa
XI JAR Bariloche, 2022
EKUMEN
XI Jornadas Argentinas de Robótica
San Carlos de Bariloche, 9-11 de Marzo de 2022

Para recordar lo visto en las últimas clases

Quizziz

¿Qué vamos a ver hoy?




Antes de Testing
Después de Testing

Calidad / ¿Qué es?

La calidad debe de ser un **pilar fundamental** dentro de esta filosofía de trabajo y al aplicar entornos de entrega continua, debemos de tener en cuenta que el ciclo de producción es mucho más corto y tenemos claros unos objetivos inamovibles: desarrollar, probar y poner en producción.
Si queremos poner algo en producción, es completamente imprescindible la realización de pruebas, y no solo unitarias, sino funcionales y no funcionales.

Calidad / ¿Qué es?

El ciclo completo de pruebas debe de ser completo, teniendo pruebas unitarias, de integración, manuales funcionales y automáticas (basadas en las manuales), sin contar con las no funcionales que nos darán buena cuenta de la seguridad y del rendimiento (entre otras muchas cosas) de lo que estemos probando.

Calidad

Dentro de un ciclo de DevOps, debemos de descartar el uso de pruebas manuales al completo, ya que es imposible cubrir la velocidad requerida a lo largo de cada entrega. Es importante, tener claro, que la idea de DevOps no se convierta en un entregar rápido y corriendo sin tener la garantía y la calidad que nuestros clientes nos demandan.

Deben asegurarse de que todos sus casos de prueba estén automatizados y alcancen una cobertura de código cercana al **100%**.

TDD (Test driven development)

El proceso de diseño de software, combinando TDD con metodologías ágiles, sería el siguiente:

- El cliente escribe su historia de usuario. Se escriben junto con el cliente los criterios de aceptación de esta historia, desglosándolos mucho para simplificarlos todo lo posible.
- Se escoge el criterio de aceptación más simple y se traduce en una prueba unitaria. Se comprueba que esta prueba falla.
- Se escribe el código que hace pasar la prueba.
- Se ejecutan todas las pruebas automatizadas.
- Se refactoriza y se limpia el código.
- Se vuelven a pasar todas las pruebas automatizadas para comprobar que todo sigue funcionando. Volvemos al punto anterior con los criterios de aceptación que falten y repetimos el ciclo una y otra vez hasta completar nuestra aplicación.

TDD (Test driven development) Ejemplo

Supongamos que el cliente nos pide que desarrollemos una calculadora que sume números (es lo primero que se me ha ocurrido).

Acordamos con el cliente que el criterio de aceptación sería que si introduces en la calculadora dos números y le das a la operación de suma, la calculadora te muestra el resultado de la suma en la pantalla.
Partiendo de este criterio, comenzamos a definir el funcionamiento del algoritmo de suma y convertimos el criterio de aceptación en una prueba concreta, por ejemplo, un algoritmo que si introduces un 3 y un 5 te devuelve un 8:

```
public void testSuma() { assertEquals(8, Calculadora.suma(3,5)); }
```

Primero escribo cómo debe funcionar mi programa y después, una vez lo tengo claro, paso a codificarlo.Al escribir el test estoy diseñando cómo va a funcionar el software.

TDD (Test driven development) Ejemplo

Al escribir el test estoy diseñando cómo va a funcionar el software, pienso que para cubrir la prueba voy a necesitar una **clase Calculadora** con una función que se llame **Suma** y que tenga dos parámetros.
Esta clase todavía no existe, pero cuando la cree, ya sé cómo va a funcionar. Este caso es muy trivial, pero muchas veces no sabemos exactamente qué clases hacer o qué métodos ponerle exactamente.
Es más, a menudo perdemos el tiempo haciendo métodos y clases que pensamos que luego serán útiles, cuando la cruda realidad es que muchas veces no se van a usar nunca. **Con TDD sólo hacemos lo que realmente necesitamos en ese momento**.

TDD (Test driven development) Ejemplo

Por supuesto, si intentamos pasar este test nos dará **un error**, porque la clase Calculadora aún no existe. Ahora pasamos a escribir el código de la clase, es fácil porque ya sabemos exactamente cómo se va a comportar:

```
public class Calculadora { public static int suma (int a, int b) { int c = a + b; return c; } }
```

Ahora ejecutamos la prueba y ya tenemos el código funcionado con la prueba pasada. Una vez todo esté funcionando, pasamos a **refactorizar** y a eliminar código duplicado, este ejemplo es extremadamente sencillo, y en un caso real no haríamos tantos pasos para algo tan evidente, pero el código mejorado podría ser por ejemplo:

BDD (Behavior Driven Development)

Es una estrategia de desarrollo dirigido por comportamiento, que ha evolucionado desde TDD (Test Driven Development), aunque no se trata de una técnica de testing.

A diferencia de TDD, BDD se define en un idioma común entre todos los stakeholders, lo que mejora la comunicación entre equipos tecnológicos y no técnicos. Tanto en TDD como en BDD, las pruebas se deben definir antes del desarrollo, aunque en BDD, las pruebas **se centran en el usuario** y el comportamiento del sistema, a diferencia del TDD que se **centra en funcionalidades**

BDD (Behavior Driven Development)

- Ya no estamos definiendo **'pruebas'**, sino que está definiendo **'comportamientos'**.
- Mejora la comunicación entre desarrolladores, testers, usuarios y la dirección.
- Debido a que **BDD se específica utilizando un lenguaje simplificado** y común, la curva de aprendizaje es mucho más corta que TDD.
- Como su naturaleza no es técnica, puede llegar a un público más amplio.
- El enfoque de definición ayuda a una aceptación común de las funcionalidades previamente al desarrollo.
- Esta estrategia encaja bien **en las metodologías ágiles**, ya que en ellas se especifican los requisitos como historias de usuario y de aceptación.

BDD (Behavior Driven Development) Ejemplo

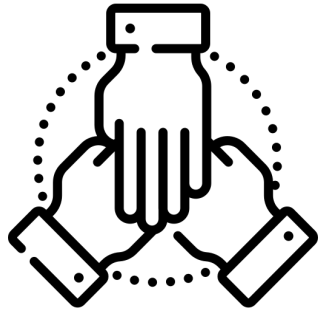
```
Feature: Búsqueda en Google
  Como usuario web, quiero buscar en Google para poder responder mis dudas.

  Scenario: Búsqueda simple en Google
    Given un navegador web en la página de Google
    When se introduce la palabra de búsqueda "pingüino"
    Then se muestra el resultado de "pingüino"
```

BDD (Behavior Driven Development) Ejemplo

FEATURE y una descripción. La descripción es opcional y puedes añadir tantas líneas como necesites. Como práctica recomendada Agile, debes incluir la historia del usuario, defiendo ejemplos concretos. **FEATURE** podría tener más **SCENARIOs**, pero por simplicidad, este ejemplo tiene solo uno. Cada SCENARIO se ejecutará independientemente de los otros **SCENARIOs**: ¡el resultado de un SCENARIO no tiene relación con el siguiente!

HERRAMIENTAS

TRABAJO EN GRUPO



TAKE A BREAK!

y en 15´volvemos.

UNC FCEFyN

mundosE PEOPLE & BUSINESS SCHOOL

Unit Testing

Las **pruebas unitarias o unit testing** son una forma de comprobar que un fragmento de código funciona correctamente. Es un procedimiento más de los que se llevan a cabo dentro de una metodología ágil de trabajo. Las pruebas unitarias consisten en **aislar una parte del código** y comprobar que funciona a la perfección. Son pequeños tests que validan el comportamiento de un objeto y la lógica.

Integration test

Las pruebas de integración **verifican que los diferentes módulos** y/o **servicios** usados por nuestra aplicación funcionen en armonía cuando trabajan en conjunto. Por ejemplo, pueden probar **la interacción con una o múltples bases de datos**, o asegurar que los microservicios operen como se espera. Las pruebas de integración son típicamente el paso siguiente a las pruebas unitarias. Y son generalmente más costosas de ejecutar, ya que requieren que más partes de nuestra aplicación se configuren y se encuentren en funcionamiento.

Production test

Las pruebas de producción difieren de otras pruebas en que no se trata simplemente de identificar errores (aunque eso sigue siendo importante).
Las pruebas de producción se dividen en **dos categorías**

El **monitoreo del rendimiento** responde preguntas como "¿es el nuevo código más eficiente que el anterior?" o "¿el sistema escala correctamente?"

Las pruebas **comparativas** (que cubren las pruebas Canary y las pruebas A/B) responden preguntas como "¿el nuevo código mejora la experiencia de usuario?" o "¿el nuevo código es más eficaz?". Veamos cada uno de estos con más detalle.

MUCHAS GRACIAS!

CONTACTO

https://www.linkedin.com/in/anahi-belen-torres/

UNC FCEFyN mundosE PEOPLE & BUSINESS SCHOOL