## Pruebas funcionales

* Evalúan el funcionamiento del programa basándose en requerimientos funcionales, así como:
  + La funcionalidad
  + Las entradas y salidas
  + El flujo de trabajo del software
  + El manejo de errores y excepciones
  + La interfaz del usuario
  + La integración del software con otros sistemas o componentes
* Negocio da los requerimientos.
* Toda funcionalidad debe ser probada antes de llegar a producción.
* La app de Vueling se llama “Entorno de Producción”, porque es lo que el cliente toca.

#### Ejemplo

##### Pruebas de navegación

Verificar que los enlaces, botones y menús de navegación conduzcan correctamente a las páginas correspondientes y que la estructura del sitio sea coherente.

##### Pruebas de búsqueda

Asegurarse de que la función de búsqueda en el sitio web proporcione resultados relevantes y precisos según los términos de búsqueda ingresados.

##### Pruebas de proceso de compra

Verificar que el proceso de compra funcione correctamente, incluyendo agregar productos al carrito, gestionar el carrito de compras, seleccionar opciones (como tamaño o color), ingresar información de envío y pago, y completar la transacción.

##### Pruebas de autenticación y seguridad

Verificar que el inicio de sesión y el registro de usuarios funcionen correctamente, y que se apliquen adecuadas medidas de seguridad para proteger los datos del usuario.

##### Pruebas de gestión de pedidos

Asegurarse de que los usuarios puedan ver y administrar sus pedidos, realizar seguimiento de envíos, solicitar devoluciones o cambios, etc.

## Pruebas no funcionales

* Evalúan:
  + El rendimiento
  + La seguridad
  + La usabilidad
  + La compatibilidad
  + La escalabilidad
  + La disponibilidad
  + La confiabilidad
* Validan características como seguridad o eficiencia.
* Se puede comprobar incluso antes de entregar a QA.
* Las pruebas empiezan en cómo definimos la arquitectura de software.
* Antes de empezar a desarrollar cualquier user story (funcionalidad a desarrollar que aporta un pequeño valor) hay que definir cómo se va a trabajar. Después se desarrolla y hay que darle cobertura (**unit** i **integration test**).

#### Ejemplo

##### Pruebas de rendimiento

Evaluar el rendimiento del sitio web en términos de tiempo de carga de páginas, capacidad de respuesta durante el proceso de compra y capacidad para manejar cargas de tráfico esperadas.

##### Pruebas de seguridad

Verificar que se apliquen medidas adecuadas para proteger la información personal y financiera de los usuarios, incluyendo el cifrado de datos, protección contra ataques de seguridad, etc.

##### Pruebas de usabilidad

Evaluar la facilidad de uso y la experiencia del usuario en el sitio web, incluyendo la navegación intuitiva, la claridad de la información, la legibilidad de los textos, la visibilidad de los botones de llamado a la acción, entre otros aspectos.

##### Pruebas de compatibilidad

Verificar que el sitio web funcione correctamente en diferentes navegadores web (como Chrome, Firefox, Safari), en diferentes dispositivos (computadoras de escritorio, dispositivos móviles) y en diferentes sistemas operativos.

##### Pruebas de escalabilidad

Evaluar cómo el sitio web maneja un aumento en el número de usuarios y en la cantidad de pedidos, asegurándose de que el rendimiento y la funcionalidad no se vean afectados negativamente.

## Pruebas de caja blanca

* Se examinan los detalles internos de un software.
* Se centran en la funcionalidad externa sin considerar la estructura interna.
* Evalúan el código fuente, la lógica interna y la estructura interna.
* Una herramienta puede ser SonarCube, que mide la **calidad del código** en “**% de cobertura**”.
* **Functional Testing**
  + Se centran en validar que el software cumple con los requisitos y funcionalidades esperadas por los usuarios.
* **Integration Testing**
  + Comprobar cómo funciona el conjunto de todos los elementos.
* **Unit Testing**
  + Hacer tests a nivel unitario de cada elemento.

### Técnicas

#### Prueba de cobertura de código

Se evalúa qué porcentaje del código ha sido ejecutado durante las pruebas. Esto incluye la cobertura de instrucciones, cobertura de ramas de decisión y cobertura de caminos lógicos.

#### Prueba de bucles

Se examinan las estructuras de bucles para asegurarse de que se ejecuten correctamente y se prueben todas las condiciones posibles, incluyendo bucles for, while, do-while, etc.

#### Prueba de condiciones lógicas

Se evalúan las condiciones lógicas dentro del código para verificar si se cumplen todas las combinaciones posibles de condiciones verdaderas y falsas.

#### Prueba de flujos de control

Se comprueba si los diferentes flujos de control, como las estructuras condicionales (if-else, switch) y los saltos (goto), funcionan correctamente y siguen la lógica prevista.

#### Prueba de límites

Se examinan los límites del software, como valores mínimos y máximos permitidos, para verificar que el sistema maneje correctamente los límites y los casos extremos.

## Pruebas de caja negra

* Validan el comportamiento funcional del sistema que nos han pedido desde negocio.
* Se centran en entrada y salida.
* Por ejemplo: si busco un aeropuerto, luego no aparece.

## Pruebas dinámicas

* **Requiere** la **ejecución** del código.
* Se valida que **cuando se ejecuta** el software **se cumplen los requisitos** del cliente.
* Se valida que te puedes mover por el dispositivo.

## Pruebas estáticas

* **No requieren ejecución** de software.
* Se realizan en Planning y Refinement para ahorrar tiempo y dinero.
* Evalúan:
  + Especificaciones
  + Requisitos de seguridad
  + Requisitos funcionales
  + Historias de usuario
  + Criterios de aceptación
  + Arquitectura o diseño del código
  + Guías de usuario
  + Páginas web
* Consiste en contemplar requisitos previos:
  + por ejemplo que en seguridad las llamadas tienen que ser https en lugar de http.
* Tienen que ser aspectos visibles antes de empezar a desarrollar.
* Permiten la **detección temprana de defectos** antes de las pruebas dinámicas.

## Pruebas basadas en la experiencia

Las pruebas tienen que ser documentadas y tiene que ser un documento vivo, es decir, un documento que se va cambiando y actualizando, ya que el aplicativo está en constante cambio.

Somos los encargados de dejarlo todo documentado.

## Pruebas de regresión

Las implementaciones nuevas de código no rompen nada de lo anteriormente construido.

### Paradoja del pesticida

Si las pruebas se repiten constantemente y el software está en constante cambio, no encontraremos nada. Para encontrar nuevos defectos hay que cambiar las pruebas o actualizarlas. Pasa sobre todo en pruebas de regresión.

## ¿Ausencia de errores?

**NINGÚN SISTEMA ES INFALIBLE**.

* No existe este concepto.
* El Product Owner o el equipo decide qué errores son más críticos que otros o cuáles son más simples.
* Se debe entrar en la parte de QA lo antes posible por ahorro de tiempo y dinero.
* Se deben hacer pruebas tempranas.
* Habrá que probar en su momento, planificarlo con anterioridad.
* **Hay que avanzarse y planificar antes de desarrollar.**

## CASOS DE PRUEBA

Los casos de prueba tienen una serie de pasos y datos que se deben rellenar para documentar cada caso.

### Casos positivos y negativos

Se valida tanto situaciones positivas como negativas en cuanto a que se comporte de manera esperada. Por ejemplo,

### Caso negativo

NO puedo hacer check in 48h antes del vuelo, lo cual es correcto en cuanto a lógica de negocio.

### Test case

* Escenario que contempla una funcionalidad de entrada y una salida.

### Pack de regresión

* Referencia a pruebas de regresión.
* La funcionalidad nueva que integro no rompe nada de lo anterior.
* Valida el conjunto de funcionalidades

### Smoke test

* Pruebas que se realizan en entorno de producción.
* Valida que los basics funcionan correctamente.
* Pueden ser sacados de la regresión.
* Los test de PRE también pueden servir para PROD.
* Pueden ser manuales o automatizados.

### TestRail

* Herramienta donde se hacen los casos de uso.

### Bug o incidencia

* En general se utiliza bug.

### re-test

* Se vuelve a provar el error para confirmar que se ha fixeado bien.

### Estados

#### Pending

#### Pass

#### Fail

#### Blocked

## Como se trabaja en vueling

* Toda **funcionalidad** se **prueba desde local** del dev **hasta producción**.
* Los **entornos** son **máquinas virtuales** donde se **despliega el código** realizado por el dev **en local**.
* Las máquinas virtuales siempre simulan lo que nos encontraremos en producción.
* **No siempre** se replica el entorno de producción.
* Entornos productivos → producción, No productivos → Dev, INT, QA.
* DEV ENV → entorno que el dev tiene para saber que lo que entrega a QA cumple con req. mínimos necesarios de calidad para ser testeado a nivel MVP. Tiene que cumplir los acceptance test que se plantean en la user story.
* INT ENV → INT es integración. Hay algunos equipos que no tienen DEV y QA, pero sí INT (aunque no es buena práctica).
* QA ENV → Pruebas caja negra, pruebas fucnionales. Product test: parte funcional. A nivel individual el componente (p.ej. buscador) funciona de manera correcta. Validamos que a nivel negocio funciona correctamente.
* Después de QA ENV se hace deploy en PRE. Se integra el componente validado con el resto de aplicativo. PRE siempre tiene lo último de PROD. En PRE ENV se hace test de regresión y de producto. Si todo OK se va a PROD.
* En PROD se hace smoke test (flujos básicos funcionan bien) y se valida que todo funciona bien. PRE y PROD son servidores diferentes. Performance siempre se prueba en PROD. Si encuentro error en PROD pruebo si existe en PRE para descartar problemas. Si encontramos error en PRE se comprueba si pasa en PROD.

## Examen

Pregunta examen: Si encuentro bug en un entorno concreto, saber como se reporta o qué hago. Qué tipo de pruebas se hacen en cada uno

QA PROCESS DIAGRAMA entra a examen

Para el examen: “hazme una regresión del buscador de vueling”.

Consiste en ir más allá de las pruebas que haremos. Qué casos contemplo con cada herramienta.