Bienvenida!

¿Qué vamos a aprender?

* ¿Cómo? surgió el Testing y conocerás su evolución a lo largo de los años.
* La importancia y los objetivos que tiene el Testing y los desafíos que enfrenta un tester.
* El  "Proceso simplificado" de Testing.
* Las diferentes metodologías de software.
* Sobre la  importancia  del Testing y ¿cómo se relaciona con la actividad de Desarrollo de Software?
* A fondo la etapa de la preparación de tus pruebas
* La diferencia entre Bugs, issues y errores.
* Los tipos de bugs que existen en Testing, su Prioridad y Severidad.
* Las Técnicas para preparar y ejecutar pruebas en ambientes reales de Web y Móvil.
* Sobre los ambientes o entornos de prueba y sobre los diferentes tipos e instancias de prueba que hay.
* Una herramienta para la gestión/Administración de Pruebas y Bug (Bugtracking).
* Las particularidades del UX/UI.
* Sobre testing móvil, identificando apps híbridas, nativas o responsivas.
* Accesibilidad y su forma de testeo.
* El testing de  videojuegos para encontrar bugs, será nuestra introducción como BetaTesters.
* Sobre  las pruebas Api´s y WebServices.
* Bases de datos y como se relacionan con el mundo del Testing.
* Automatización, pruebas  con Selenium y otras más.
* Git y como se utilizan en los proyectos de Testing.
* Los riesgos que se derivan de una mala planificación de las pruebas.

Todo se hará en un ambiente simulado de trabajo.

Historia y evolución del Testing

El 9 de septiembre de 1947, Grace “Amazing” Hopper, encontró el primer defecto de software en la computadora Mark II de la Universidad de Harvard. Tomó nota del fallo y adjuntó una polilla, causante de la falla, al reporte. Por eso se llama “bugs” a los defectos.

¿Cómo es que está relacionado este último video de los inventos con el testing?  
 Porque los inventos necesitaban probarse para poder validar su funcionamiento.

¿Cómo crees que se probaron los inventos de Edison?  
Probando en la práctica a modo de prueba y error.

¿Cómo crees que se determinaba cuando los inventos no funcionaban?  
Cuando no cumplían con su funcionamiento.

“Las pruebas no se limitan al desarrollo de software. Los testers podemos interactuar en cualquier tipo de desarrollo.”

Evolución del Testing

Está subdividida en 5 períodos en la línea de tiempo:

1. *Debuggin (1947 – 1956):*Todas las pruebas realizadas están dirigidas a la corrección directa del código fuente de los programas. Los programadores las realizaban directamente y aun no tenían claro la diferencia que existía entre: checkout, debuggin y testing.
2. *Demostración (1957-*1957):  
   En esta época se encargaban de utilizar de forma masiva el test para asegurar que se cumplía con la especificación. Se realizaban al finalizar el desarrollo del software.
3. *Destrucción (1979 –* 1982):  
   Cambio en la metodología. Se paso de demostrar que un programa era correcto mediante pruebas y demostraciones teóricas a demostrar que el programa no funciona o tiene fallos. De esta manera, se tiene una mayor posibilidad de encontrar fallos. Los test se transforman en casos de prueba que se aplican a los productos de los desarrolladores para encontrar errores y corregirlos.
4. *Evolución (1983 –* 1984):  
   Se comienza a integrar las pruebas de software en el ciclo de vida del desarrollo del software.
5. *Prevención (1985 –* actual):  
   Finalmente, en esta época se diversificaron las pruebas incorporándolas en todas las fases del desarrollo, para con esto poder verificar todos los tipos de componentes, modelos, módulos, subsistemas y sistemas que conforman el software.

¿Qué es el Testing de Software?

Testear un software o una aplicación es el proceso de validar y verificar la funcionalidad de un programa o una aplicación, con el objetivo de garantizar que el producto de software esté libre de defectos (issues). Esto se realiza a partir de los requerimientos solicitados por nuestros usuarios y clientes.

Objetivos:

* Asegurar su funcionamiento y que los errores detectados sean corregidos.
* Asegurar su funcionamiento al momento de que se integren todos los módulos.
* Asegurar que la aplicación cumpla con sus requerimientos.
* Asegurar que cumpla con las expectativas técnicas de rendimiento, capacidad y tiempo de respuesta.
* Aportar calidad sobre el producto que se está probando.

El proceso de testing ayuda para:

1. Detectar bugs.
2. Entregar versiones estables.
3. Garantizar la calidad del producto final.
4. Mantener la confiabilidad en el producto creado.

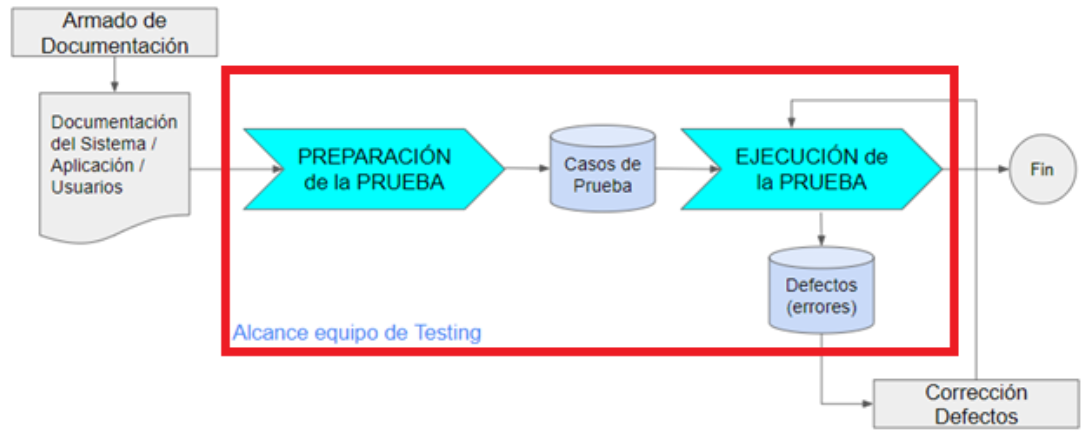
El testing de software es muy importante ya que reduce los cotos de corregir un error fatal del software en una etapa tardía, por ejemplo, cuando lo deployan.

Los desafíos del tester se centran en: encontrar defectos y asegurarse de que sean corregidos antes del deploy. También, no menos importante, el tester tiene el desafío de no ser tomado como “destructivo” o “mala persona” al detectar y reportar los errores “dejados por los desarrolladores en sus programas”.

* ¿Es normal que el Software tenga errores? Si / No y ¿Por qué?  
  Si. Porque errar es humano y los desarrolladores son humanos.
* ¿Por qué pensás que los desarrolladores de software no pueden desarrollar software que simplemente funcione?  
  Desarrollan software que funciona pero con errores. Estos errores aparecen porque somos humanos.

Un tester debe ser: organizado, detallista, intuitivo, comunicativo, persistente

Proceso de testing simplificado

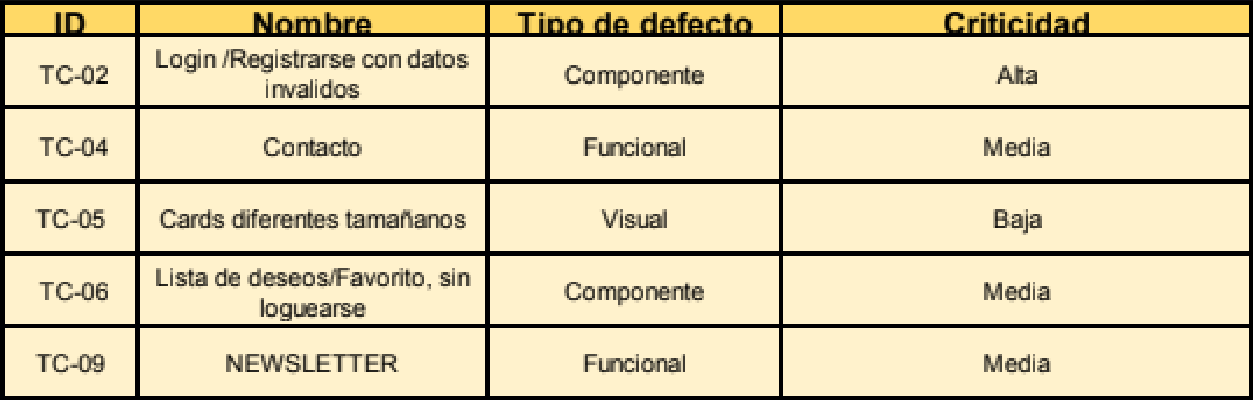


Preparación de prueba: es la descripción de qué y cómo se probará un funcionamiento. Estos serán los *casos de prueba*. Entonces, estos son un conjunto de entrada y condiciones de ejecución que buscan verificar o comprobar que el software se comporta de la manera esperada. Es detallado y estructurado.

*Ejecución de la prueba:* paso siguiente luego de realizar los casos de prueba. Si el resultado obtenido es distinto al resultado esperado, entonces, se registrará un defecto o incidencia para ser corregida por un desarrollador. Para esto se utilizan metodologías como Jira.

*Defectos:* son los errores/incidencias/bugs que se van encontrando al ejecutar los casos de prueba. Se clasifican de la siguiente forma:

* *Categoría:* especificación errónea, faltante, extra
* *Severidad:* bloqueante, mayor, menor, mejora
* *Prioridad:* crítica, alta, media, baja
* *Tipo:* funcionalidad del sistema, integración, seguridad, arquitectura técnica



Introducción al equipo de testing

Resumen de las actividades más comunes:

1. Participar en reuniones donde se planifica la idea, el producto a desarrollar (rol de "prevención").
2. Diseñar y colaborar en el diseño de la Estrategia de Prueba.
3. Analizar los requerimientos de usuario.
4. Diseñar casos de prueba.
5. Participar en reuniones Kick Off
6. Explorar la aplicación y ejecutar los distintos casos de prueba diseñados.
7. Reportar defectos de forma efectiva.
8. Controlar que los defectos hayan sido corregidos, hacer un seguimiento sobre los mismos.
9. Colaborar con los desarrolladores para reproducir los defectos reportados en entorno de prueba.
10. Evaluar los casos de prueba que son candidatos para automatizarse.
11. Sugerir propuestas de mejora al proceso de Testing.

Evolución de las metodologías de desarrollo

*Metodología:* según el diccionario, una metodología hace referencia a un conjunto de procedimientos racionales utilizados para alcanzar un objetivo o a la gama de objetivos que rige un proyecto, trabajo o investigación científica.

* **1835:** Las primeras ideas sobre el desarrollo de manera interactiva e incremental (aún no lo sabían). Fue descrito por Darwin.
* **1891:** Frederick Taylor define la administración científica del trabajo. Expandiendo ideas sobre la división del trabajo a tareas simples y rutinarias, la motivación extrínseca, productividad a través del control y supervisión de los empleados.
* **1924:** Sakichi Toyoda inventa e implementa por primera vez la automatización. Concepto conocido como Jidoka. Inventa una máquina de telar capaz de detectar automáticamente cuándo el hilo se rompía y deteniendo la producción para evitar desperdicios o defectos en el tejido final. Sakichi Toyoda, quien luego en Toyota, aplicaría estos conceptos conocidos hoy en día como Manufactura Lean. Ideas que confrontan directamente el Taylorismo.
* **1939:** Walter Shewhart – conocido como “el padre de la calidad” – publica en su libro Statistical Method from the Viewpoint of Quality Control el proceso de mejora continua y trabajo iterativo e incremental a través de ciclos cortos de “Planificar, Hacer, Analizar y Actuar”.
* **1946:** El ENIAC, la primera computadora digital de la historia.

Siguieron cambiando las metodologías hasta que…

* **2001:** 17 agilistas se juntador a discutir y firmar el “Manifesto Ágil” con los cuatro valores y doce principios.

Antes se utilizaba la metodología *Cascada o Waterfall*. Esta se caracteriza por su secuencialidad, que agrupa tareas dentro de fases o ciclos de desarrollo del proyecto. Además, analiza de forma exhaustiva cada uno de los requerimientos de las fases, cosa que dificulta volver a etapas anteriores del proyecto. Procedimiento lineal. No puede pasar a la siguiente fase hasta que complete en la que está.

Si se entendió lo de mitologías agile, también la actividad de los roles.

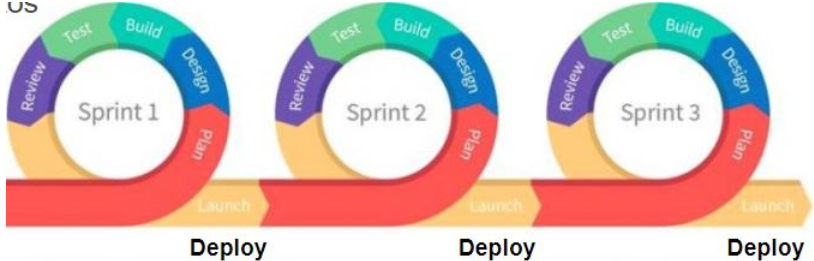
**Ventaja:**

* Pensado para elaborar sistemas pequeños
* Documentación robusta sobre el proyecto
* Se estiman los costos de un proyecto desde el principio
* Se pueden representar y planear de forma más sencilla

**Desventajas:**

* Consume mucho tiempo en proyectos grandes y complejos
* Poca comunicación entre equipos
* Demasiada documentación por evaluar y revisar
* El usuario final no se integra durante la elaboración y el testing del proyecto es hasta el final
* Por lo tanto, los defectos se detectan al final
* Se vuelve compleja la verificación y validación de todo el sistema

*Metodología Agile:* la palabra clave de esta metodología es la “retroalimentación”. Agile es un conjunto de metodologías distintas en las que se trabaja por tareas y períodos de tiempo, llamados sprints. La codificación y las pruebas se realizar de forma iterativa, interactiva e incremental, lo que da como resultado un producto final de calidad que cumple con los requisitos del cliente y a su vez, productos parciales concretos que lo van conformando (sprints).



Cada *sprint* consta de 2 semanas. Se toman decisiones en conjunto con todas las partes interesadas del proyecto. En 2001 lanzaron el “Manifesto Agile” que engloba las mejores prácticas, este documenta esboza 4 valores y 12 principios para aplicarse a las metodologías Agiles.

El objetivo de esto es:

* Desarrollar comunicación eficiente entre los equipos
* Satisfacer al cliente
* Entregar a tiempo (según lo planificado)

*Agile y el “time-to-market” (plazo de vencimiento):* es el tiempo comprendido desde que el producto o servicio es concebido hasta que está disponible para el usuario final. Entonces, las empresas requieren reducir el time-to-market para sobrevivir y asegurar la gratificación instantánea que sus clientes demandan con el uso de nuevas aplicaciones de software.

Los métodos más usados son el SCRUM, el XP y el Kanban entre otros.

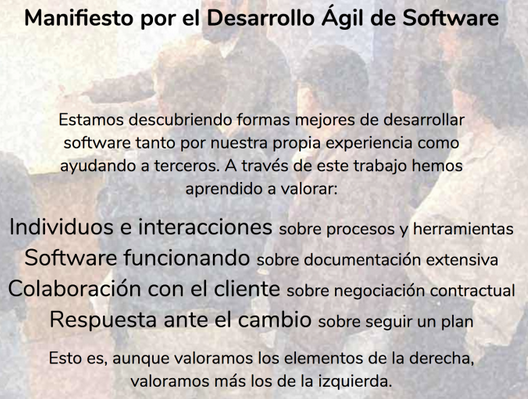
Las entregas en Scrum son parciales y regulares, esto ayuda a los testers a probar solo una parte de la funcionalidad de todo el sistema y asegurarse de que funciona, una vez que se validó se libera, de lo contrario con la existencia de errores es mucho más rápido devolver tal componente al desarrollador para que se corrija y el error no se inyecte nuevamente con el siguiente código.

Los 4 roles del Scrum:

1. Product Owner:  
   Es la figura que representa la voz del cliente. No es un perfil implicado directamente en los proyectos. Se centra en la parte de negocio y ROI (return of inversión) del proyecto.
2. Scrum Master:  
   Es la persona que lidera al equipo, guiándolo para que cumpla las normas y procesos en la metodología scrum. Trabaja de la mano con el Product Owner para maximizar el ROI.
3. Scrum Team (o equipo de desarrollo):  
   Es el equipo encargado del desarrollo y la ejecución del proyecto hasta llegar al producto final. Gran capacidad de autogestión y auto-organización.
4. Skateholders: son todos los perfiles que tienen interés en la evolución del proyecto. Formado por directores, dueños y comerciales.
5. PM: es más de metodologías cascada, relacionada con la parte empresarial. Está en el medio entre los Stakeholders y el PO.

Los 3 pilares del Scrum:

1. Transparencia: toda la información que se genere antes, durante y al final del proyecto, debe poder estar a la vista de todos.
2. Inspección: durante cada Sprint se revisa el avance del proyecto. Se tienen dailies que son juntas de avance al comienzo del día y no pueden pasar los 15 minutos.
3. Adaptación: cada tarea debe ajustarse según las necesidades del momento.



*Metodología Kanban:* es un método de administración de flujo de trabajo para definir, administrar y mejorar los servicios que brindan trabajo de conocimiento. Su objetivo es ayudarlo a visualizar su trabajo, maximizar la eficiencia y mejorar continuamente. La palabra significa “Tablero” y la desarrollo Toyota.



Los 3 apartados del tablero son:

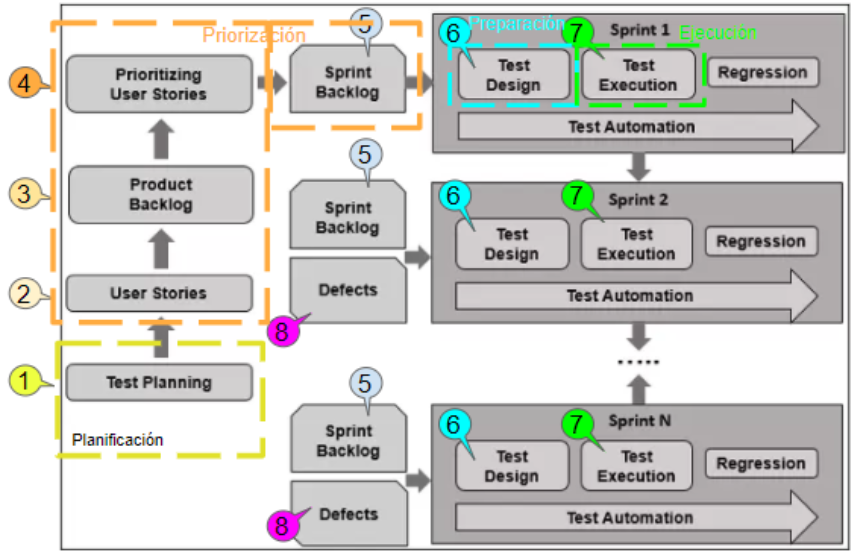
* **TO DO:** trabajo por hacer, ahí es el punto de partida de todos los post-its colocados.
* **DOING:** es el trabajo que está en construcción o bien que está en etapas de pruebas.
* **DONE:** es el artefacto, sistema o funcionalidad ya terminada.

Las *pruebas ágiles* son una práctica de pruebas de software que siguen los principios del desarrollo de software ágil. Se aplica desde el nacimiento del software hasta su deploy y operación. Estas pruebas se enfocan en construir un producto de calidad, utilizando bucles de feedback cortos o sprints para validar nuestras hipótesis. Las prácticas refuerzan la idea de que la calidad es responsabilidad de todo el equipo.

Los principios del agile testing:

* Las pruebas realizadas son continuas
* Proveer retroalimentación constante
* El equipo completo realiza pruebas
* Responder con rapidez a la retroalimentación
* Crear un código simplificado y limpio
* Producir menos documentación
* Probar para descubrir
* Prevenir defectos

*Ciclo de vida agile:* inicia con un pedido del usuario (historias de usuario), el cual se incorpora al product backlog (inventario de pedidos). Estos son priorizados según criticidad de negocio y algún otro aspecto, y con base en eso, se arma la planificación de cada iteración (sprint). Esta se arma en función a la prioridad de los requerimientos y la disponibilidad del equipo.



Ceremonias en Scrum (eventos scrum): es donde tienen lugar los sprints.

*Sprint Planning:* tiene lugar al comienzo del Sprint. Esta ceremonia está diseñada para asegurar que cada miembro del equipo esté preparado y que en cada Sprint se hagan las cosas correctamente. Con frecuencia se relaciona directamente con la duración de cada Sprint. Se define el backlog.

* Duración: 1 semana
* Responsable: PO ya que tiene el control del product backlog

*Daily Scrum:* debe hacerse al menos una vez a la mañana, el equipo se reúne y se comunica el progreso individual, siempre con base en la meta del Sprint. La junta también debe ser informativa. ¿Qué hice ayer para contribuir al sprint goal? ¿Qué voy a contribuir hoy para el sprint goal?

* Duración: No debe superar los 15 minutos
* Responsable: PO o SM

*Sprint Review:* es un momento dedicado a mostrar el trabajo completado durante el sprint a las partes interesadas. De esta forma, pueden ver cómo van las cosas y dedicarse a inspeccionar o adaptar el producto. Generalmente los días viernes. Se le presenta a los Stakeholders las nuevas condiciones que puede afectar al negocio.

* Duración: entre 30 minutos a 1 hora cada sprint
* Responsable: PO

*Sprint Retrospective:* consiste en obtener una retroalimentación rápida con el propósito de mejorar la cultura y desarrollo del producto. Se realiza al final del sprint para que el equipo pueda mirar hacia atrás en su trabajo e identificar elementos que podrían mejorarse.

* Duración: 2 horas por sprint
* Responsable: SM

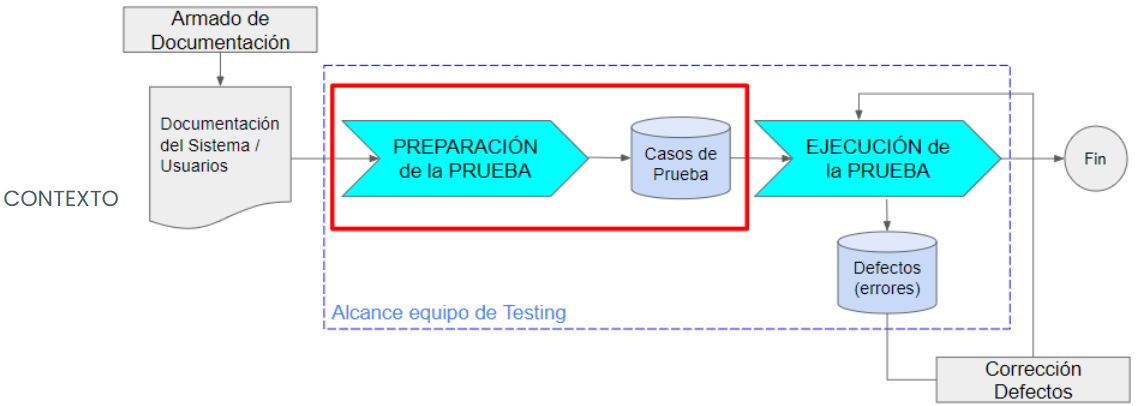
*Reunión de refinamiento:* es como una pre-planning donde el PO debe tomarse el tiempo de ordenar el product backlog antes del sprint. Solo participan el PO, los DEVS y los testers. Sirve para aclarar los requerimientos. Los asistentes deben conocerlas de ante mano.

* Duración: 1 hora por sprint
* Responsable: PO

Otras reuniones:

* *Planning poker:* herramienta del SM para estimar la cantidad de esfuerzo que se necesita para realizar una historia de usuario.
* *Kick-off:* Saque inicial que da inicio a un partido. Sirve para presentar proyectos y presentación del personal en la empresa o en los equipos de trabajo, a modo de punto de arranque en una relación profesional.

Preparación de la prueba



El primer paso es la preparación de la prueba: es el trabajo previo que debemos realizar como testers. Este trabajo involucra desde la lectura de documentación (si es que hay) hasta la investigación del sistema en cuestión. Esto sirve para darnos cuenta de todos los requerimientos funcional, técnicos, cosméticos y lógicos que contendrá la aplicación o sistema.



Sin embargo, nos puede pasar que no contemos con información que nos proporcione una idea de lo que tenemos que testear. En este caso podemos utilizar 2 técnicas:

1. Investigación preliminar:

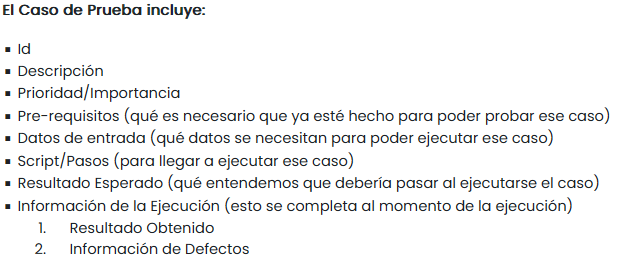
* Leer y analizar la descripción pública en su web y tienda (Google Play o Apple Store)
* Usar la aplicación un rato pensando como usuario (testing exploratorio)
* Tomar notas de lo que se descubre. Hacerse preguntas. ¿Y esto como debería ser?
* Hacer analogía con otra app similar, aplicar sentido común.
* Ir documentando todo en un documento o en herramientas tipo “Confluence” o Google Doc.
* Identificar flujos y funcionalidades críticas e ir documentándolos como “escenarios”
* Identificar funcionalidades secundarias.

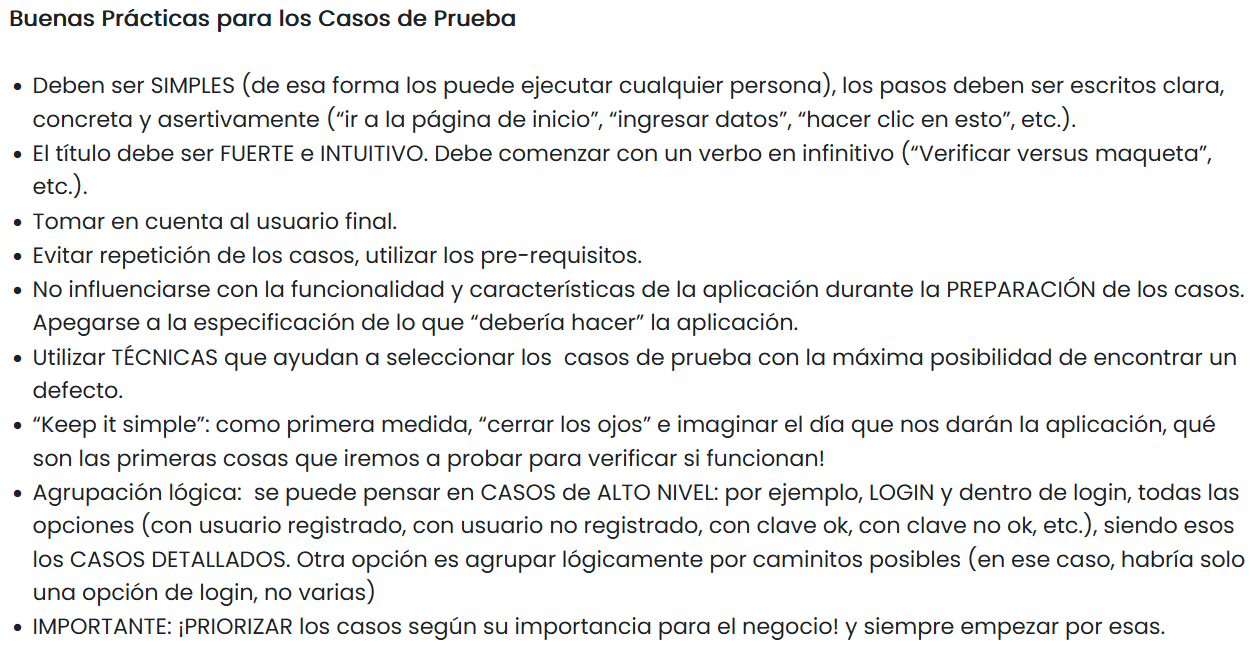
1. Técnica de interacciones:

* Las interacciones específicas de las pantallas (botones de cerrar, info, avanzar, retroceder, guardar, descargar, etc.). Deben ser verificadas, tanto a nivel de posición, tamaño, color, posible superposición, etc., como a nivel de funcionalidad básica.
* Si no se cuenta con la maqueta, al menos se debe hacer una verificación de las cosas “lógicas”.

Diseño de casos de prueba usando técnicas: Variables en juego

Cada *caso de prueba* es un conjunto de entradas y condiciones de ejecución que buscan verificar o comprobar que el software se comporta de la manera esperada ante dichas circunstancias.

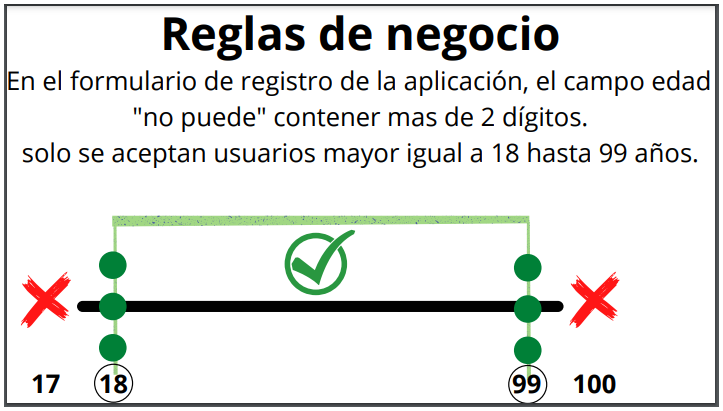




*DDT (Data Driven Testing o pruebas dirigidas por datos):* técnica basada en los datos de entrada y en las variables en juego. Son las particiones de equivalencia, valores límites y combinaciones por pares. Además, hay que sumar una investigación preliminar de la aplicación, en los casos en que no se cuenta con documentación.

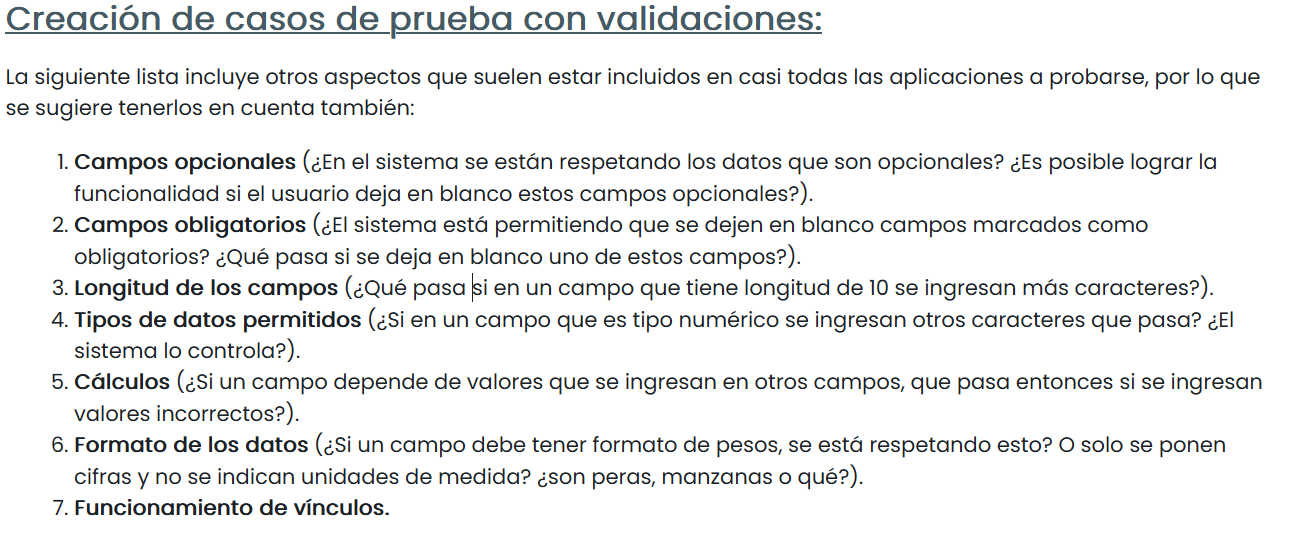
Resumen de los pasos a realizar:

1. Identificar variables
2. Valores interesantes
3. Clases de particiones y equivalencia
4. Valores límites



La técnica de *combinación de variables* permite seleccionar una cantidad acotada de datos de prueba. Indica como combinar los datos para aumentar la probabilidad de encontrar errores sin tener que hacer infinitos casos de prueba.

En resultado esperado habría que hacer el cálculo de los *valores esperados.*



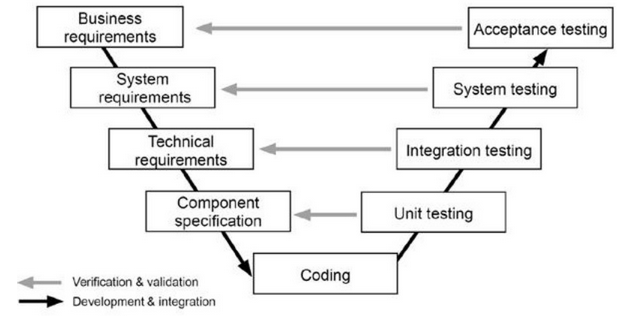
Resumen de como confeccionar casos de prueba:

1. Realizando un análisis preliminar (si no hay documentación). Esto nos ayuda a entender el objetivo de la aplicación, sus funcionalidades principales y críticas, las más masivas, los diferentes escenarios y variables para su ejecución.
2. Tomando como input las historias de usuario y criterios de aceptación.
3. Utilizando técnicas de variables/equivalencias y valores límite.
4. Considerando todas las interacciones en las pantallas.
5. Utilizando el checklist con “otros aspectos” que casi siempre se deben considerar.
6. Lo mas importante, se deben ***identificar las funcionalidades principales.***

Niveles y tipos de prueba

El objetivo de todas las pruebas es detectar defectos. Cada una se puede clasificar según el foco que tenga.

Modelo de desarrollo en V:



Hay 2 tipos de prueba:

* **Funcionales:** sirven para comprobar características críticas para el negocio, la funcionalidad y la usabilidad. Garantizan que las funcionalidades se comportan según lo esperado sin ningún problema. Valida los requerimientos. Incluyen pruebas unitarias.  
  Estas pruebas pueden ser:

1. Unitarias
2. De componentes
3. De humo
4. De integración
5. De regresión
6. De cordura
7. De aceptación

* **No funcionales:** sirven para validar funciones de rendimiento. Se mide la carga y esfuerzo. Se utilizan herramientas y soluciones de automatización. Incluyen pruebas de instalación, pruebas de confiabilidad y pruebas de seguridad.  
  Estas pruebas pueden ser:

1. De rendimiento
2. De carga
3. De estrés
4. De seguridad

***Pruebas funcionales:***

1. *Pruebas unitarias*: es una forma de comprobar que un fragmento de código funciona correctamente. Este tipo de pruebas suelen ejecutarse durante la parte del desarrollo, el objetivo es probar el funcionamiento de el componente y su lógica. Sirve para detectar errores en fases tempranas del desarrollo del software. La suelen hacer los DEVs. Realizar estas pruebas supone un ahorro de costos y tiempo.
2. *Pruebas manuales*: son aquellas pruebas que los testers ejecutan casos de prueba sin ninguna herramienta o script de automatización. Permite asegurarnos de que todas las partes del sistema funcionen correctamente según los requerimientos y condiciones que se definen.
3. *Pruebas de caja negra:* las que hace el tester manual. No conoce el código pero ejecuta pruebas sobre el desarrollo front-end.  
   *Pruebas de caja blanca:* estas se desarrollan del lado del back-end. Se conoce el código que esta detrás de los casos de prueba que ejecutemos.  
   *Pruebas de caja gris:* es la combinación de las 2 anteriores. Identifica errores y defectos debido a la estructura o uso incorrecto de la aplicación.
4. *Pruebas de Humo o Smoke Test:* pruebas para verificar las funcionalidades críticas.
5. *Pruebas de Cordura o Sanity Test:* Se prueba una funcionalidad específica o un error que se corrige para verificar su funcionamiento y ver si no hay otros problemas debido al cambio de otros componentes relacionados.
6. *Pruebas de integración:* pruebas cuando mas de un componente se integran para formar un sistema homogéneo.
7. *Pruebas de regresión:* se llevan a cabo al recibir la compilación del software después de corregir los errores que se encontraron en la ronda inicial de pruebas. Verifica si los errores están corregidos y el resto del software sigue funcionando bien luego de que se aplicaron los cambios de versionado.
8. *Pruebas de localización:* verifica el funcionamiento del software cuando se transforma en una aplicación utilizando un idioma diferente.

***Pruebas NO funcionales:***

1. *Pruebas de carga:* se espera que una app maneje una carga de trabajo específica y se prueba de esta manera el tiempo de respuesta en un entorno real. Se prueba su capacidad para funcionar correctamente en un tiempo estipulado y es capaz de manejar la carga.
2. *Pruebas de Estrés:* la app se estresa con una carga de trabajo adicional para verificar si sigue funcionando de manera eficiente y es capaz de manejar el estrés.
3. *Prueba de Volumen:* se prueba para determinar su exactitud y confiabilidad en condiciones adversas agregando grandes cantidades de datos para su procesamiento.
4. *Pruebas de Resistencia:* se prueba la durabilidad del software a partir de un flujo de carga constante y repetido en un patrón escalable.
5. *Pruebas de Seguridad:* se prueba que tan seguro es el soft con respecto a los datos en la red contra ataques maliciosos.
6. *Pruebas Automatizadas:* consiste en la aplicación de herramientas de software para automatizar el proceso manual de revisión y validación de un producto de software que lleva a cabo una persona.

UAT

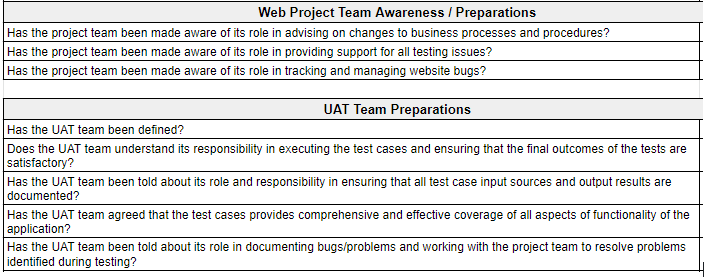
UAT (User Aceptance Testing, Pruebas de Aceptación de Usuario): los usuarios de software reales prueban el software para asegurarse de que pueda manejar las tareas necesarias en escenarios del mundo real. Es uno de los procedimientos finales y críticos del proyecto. Ocurre antes de lanzar un nuevo soft al mercado. Sirve para que el cliente de su visto bueno antes de lanzarlo a producción.

El objetivo principal de estas pruebas es evaluar si el sistema admite situaciones comerciales y de usuario cotidianas, y, garantizar que el sistema sea suficiente y correcto para el uso comercial. También, que el producto sea usable desde la perspectiva de un usuario. Además, de que cumpla todos los requerimientos legales. Por último, certifica que el producto esta listo para moverlo a producción (deploy).

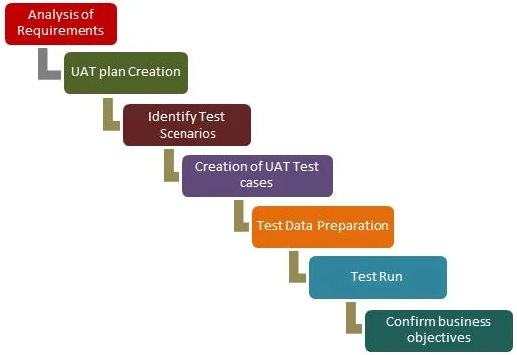
Entonces:

* Debemos evitar que la aplicación se congele
* Muestre un Crash application (Bloqueo)
* Se revisa el rendimiento y performance de la misma
* Será una aplicación de manejo fácil?

El equipo UAT esta conformado por los usuarios finales “reales”. Incluye a todos los roles y grupos de partes interesados. Generalmente cuentan con un formato tipo check list:



Etapas:



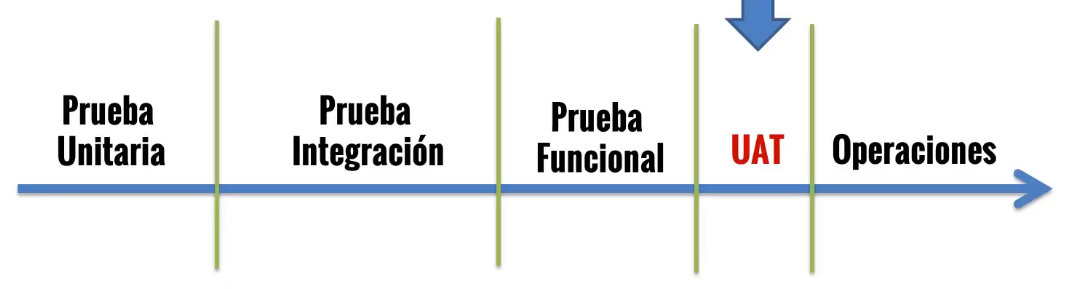
1. *Análisis del requerimiento de negocio:* consta de identificar y desarrollar escenarios de prueba a partir de:

* Requerimiento de proyecto
* Casos de uso
* Diagramas de flujo de proceso
* Documento de requisitos de negocio
* Especificación de requisitos del sistema

1. *Creación del plan de pruebas UAT:* describe la estrategia que se utilizará para verificar y garantizar que una aplicación cumpla con sus requisitos comerciales. Documenta los criterios de entrada y salida para UAT, escenarios de prueba y enfoque de casos de prueba y cronogramas de prueba.
2. *Identificar escenarios de prueba y casos de prueba:* con respecto al proceso empresarial de alto nivel y cree casos de prueba con pasos de prueba claros. Los casos de prueba deberían de cubrir suficientemente la mayoría de los escenarios UAT. Los casos de uso empresarial son una entrada para crear los casos de prueba.
3. *Preparación de datos de prueba:* estos deben ser reales, y deben configurarse por motivos de privacidad y seguridad. El tester de experiencia del cliente, debe estar familiarizado con el flujo de la base de datos.
4. *Ejecute y registre los resultados:* como siempre.
5. *Confirmar que se cumplieron los objetivos de negocio:* Los analistas de negocios o UAT testers deben enviar un correo electrónico de cierre después de la prueba de UAT. Después de la firma, el producto está listo para la liberación a producción. Se entrega el plan de pruebas, los escenarios y casos de prueba de UAT, los resultados de las pruebas y el registro de defectos.

*Información de salida:* se puede observar la evolución de los defectos antes de su entrega de un ciclo al otro.



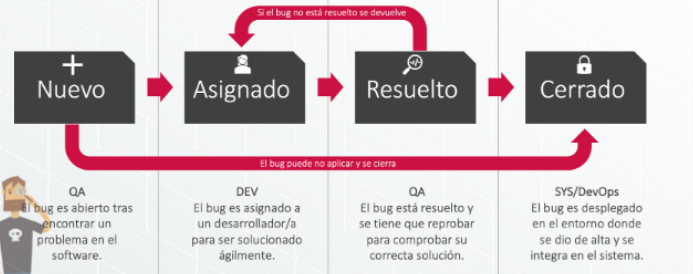


Ejecución de la prueba:

Luego de confeccionar los casos de prueba, vamos a ejecutarlos. Verificaremos cada paso y reportaremos los errores en aquellos casos que sea necesario. En todos los casos, reportaremos el resultado obtenido, y si este coincide con el resultado esperado, entonces el caso habrá pasado si no reportaremos una falla. Hay distintas herramientas para la gestión de proyectos, seguimiento de errores e incidencias. En nuestro caso vamos a usar Jira.

En los casos que se produzca una falla deberemos realizar una evidencia. Esta sirven para demostrar la falla que estamos reportando. 

Ciclo de vida de los defectos:



***Severidad:***

* Bloqueante
* Crítica
* Alta
* Baja

***Prioridad (se la asigna el PO en general):***

* Alta
* Media
* Baja

***Severidad*** es dada por el tester y se clasifica según que tan grave es el bug para la aplicación.

***Prioridad:*** es dada por el líder o el product owner y sirve para darle un orden de resolución a los bugs, comenzando por los que deben de ser resueltos lo antes posible.

Herramientas de testing

Sirven para visualizar, organizar y acomodar los casos de prueba previo a ejecutarse. También, se pueden reutilizar casos de prueba para volver a repetir las pruebas. Por último, se puede obtener estadísticas rápidamente.

UX/UI

***UX (user experience):*** Hace referencia a la dificultad o facilidad con la que los usuarios pueden comprender una web, navegar y completar las tareas que tenga.

***UI (user interface):*** refiere a todo aquello con lo que los usuarios interactúan de forma directa, desde el packaging hasta un sitio web.

¿Por qué hacer testeos de experiencia de usuario? Porque una web con una mejor UX ofrecerá mayor rendimiento. Se utilizan para comprobar si el comportamiento de los usuarios en la web se ajusta a nuestras expectativas. Para probar diferentes prototipos, diseños y estrategias UX para determinar cuáles son los más efectivos. Para detectar posibles errores de diseño o en la UX. Para obtener mas información general sobre el comportamiento de los usuarios para tomar decisiones de desarrollo.

¿Cómo se hace el test? Primero hay que definir qué es exactamente lo que se quiere medir. Segundo, decidir qué tipo de pruebas haremos. Tercero, definir nuestro porcentaje de éxito y otras métricas clave. Por último, analizar los resultados de las pruebas UX.

**Test A/B / Split Testing:** Consiste en comparar dos variantes de un mismo elemento para saber cuál funciona mejor. Se puede probar sobre casi cualquier elemento del diseño.

**Test UX Libre:** Consiste en un evaluador que ve a un usuario usar la aplicación.

**Text UX guiado:** Similar al anterior salvo que el usuario recibe instrucciones y cuestionarios específicos.

**Test de UX móvil / escritorio:** Se trata de comparar las versiones mobile y de escritorio del sitio web para ver si ofrecen el mismo rendimiento.

Existen 12 leyes de diseño que deben ser aplicadas al diseñar una aplicación.

Aplicaciones Móviles

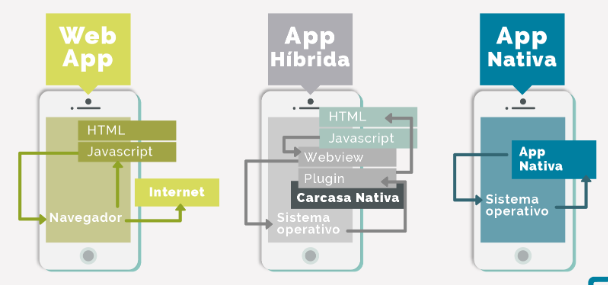
*¿Por qué testearlas?* Porque un buen producto significa un incremento en los ingresos. Reduce el riesgo del proyecto. Garantiza un buen producto final.

Clasificación:

1. *Nativas:* Es la que se ejecuta en el dispositivo de manera más rápida y avanzada en cuanto a las funciones. Desarrollos que se construyen para determinados S.O. No dependen de una conexión a internet. Objetive-C (iOS) y Java (Android). Pros: UX, alto rendimiento, protección de datos, funcionalidad complete, personalización. Cons: tiempo de desarrollo y costo. Ejemplos: Google Maps, LinkedIn, Twitter, Telegram, PokemonGo.
2. *Híbridas:* Combina las características de las aplicaciones web y nativas. Pros: multiplataforma, rápida de construir, menor inversión, fácil de cambiar y actualizar, personalización. Cons: conexión a internet, capacidades limitadas, pobre experiencia de usuario, desarrolladores de app nativas. Ejemplos: Discord, Netflix, Instagram, TikTok.
3. *Web App:* Es la que se puede acceder desde cualquier navegador móvil. Desarrolladas con las mismas herramientas que la página HTML, CSS, JS. Necesitan internet para brindar todas las funcionalidades. Pros: Reducción de costo desarrollo, rápida de construir y probar, fácil de mantener. Cons: no funciona sin internet, pocas funcionalidades de componentes nativos. Ejemplos:wordpress.com, médium.com, emol.com.

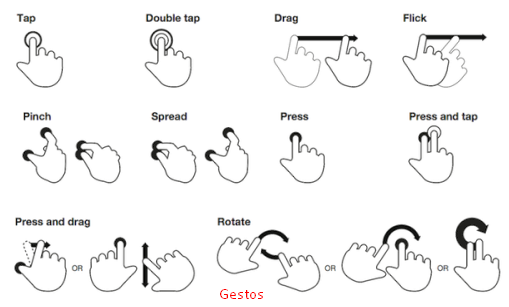
Se debe saber en que versión del S.O. en que se probaba la aplicación móvil. Lo informa el cliente o los devs. Se *recomienda* utilizar dispositivos físicos o reales de pruebas. Los simuladores y emuladores son aplicaciones limitadas, por lo tanto no se realiza un buen testeo de UX. XCode sirve para dispositivos iOS y Android Studio para los Android.





Aspectos claves de las pruebas móviles:

1. ***Funcionalidad:*** Registro/Login. Modo vertical y horizontal. Acciones básicas, opciones del menú, títulos, textos y botones. Notificaciones, mensajes error/éxito. Gestos básicos.



1. ***Usabilidad:*** aseguran que se construya una aplicación móvil intuitiva y fácil de usar. Es recomendable incorporar a los usuarios desde que entregan el mockup para obtener su feedback. De esta misma forma, se podrían incorporar scripts para pruebas UAT. Lo *incomodo* es un bug.
2. ***Conectividad:*** Probar si funcionan bien para diferentes velocidades de conexión. Para las aplicaciones nativas e hibridas es fundamental verificar si funcionan correctamente en modo fuera de línea, avión, con 3/4/5G o Wi-Fi. A veces es necesario tener chips de diferentes compañías.
3. ***Rendimiento:*** ayudan a garantizar que la aplicación móvil funcione como se espera bajo cargas de trabajo diferentes y específicas. Para esto se recomienda *contabilizar* el tiempo desde que se abre la app hasta que se muestra la primera pantalla, no debe exceder los 2 segundos. *Verificar* el consumo de la batería. *Chequear* si se sobrecalienta el dispositivo móvil. *Revisar* el uso de memoria.
4. ***Interrupciones:*** aseguran que funcione como se espera cuando se encuentre ante eventos abruptos e inesperados, como una llamada, notificación, etc. ¿Qué ocurre al recibir una llamada o un SMS? ¿Qué sucede después de que el usuario apaga el dispositivo se apaga inesperadamente? ¿Cómo se comporta la app al recibir una notificación?
5. ***Compatibilidad:*** se debe definir que versiones del sistema operativo soporta la aplicación móvil y probar el diseño responsivo, en varios navegadores (Chrome, Safari, Firefox, Edge, Opera y Samsung), el comportamiento en diferentes versiones del S.O., el funcionamiento en distintos móviles con diferentes especificaciones de hardware.
6. ***Localización:*** aseguran que la aplicación funcione como se espera en diferentes mercados, países y regiones. Podemos probar que las traducciones sean correctas y que no sean traducciones generadas automáticamente, comprobar que la aplicación muestre la hora correctamente en diferentes zonas horarias, confirmar que los textos y elementos de la interfaz se vean y funcionen en diferentes idiomas sin problemas.

Testing de Accesibilidad

Es un subconjunto de las pruebas de usabilidad. Sirve para asegurarse que los usuarios que tienen alguna discapacidad puedan usar las apps en cualquier dispositivo. Un 15% de la población padece alguna discapacidad. Las pautas de accesibilidad al contenido web (WACG 2.0 – estándar técnico para un alto nivel de accesibilidad) están basadas en 4 principios:

1. **Información perceptible:** alternativas de texto para contenido que no es tal, por ejemplo, una imagen que la reemplace un texto referido a ella si no se carga.
2. **Interfaz de usuario operable:** el sitio web debe ser navegable e interactivo a través del teclado.
3. **Información comprensible:** el idioma debe ser muy claro.

Este testing debe formar parte del ciclo de vida normal de las pruebas de software.

En Argentina la ley n°26.653 reglamenta el tema. El monitoreo y evaluaciones esta a cargo del ONTI. Entiende por accesibilidad a “la posibilidad de que la información de la página web pueda ser comprendida y consultada por personas con discapacidad y por usuarios que posean diversas configuraciones en su equipamiento en sus programas”.

Herramientas para verificar la accesibilidad: ADA, WCAG, WAVE.

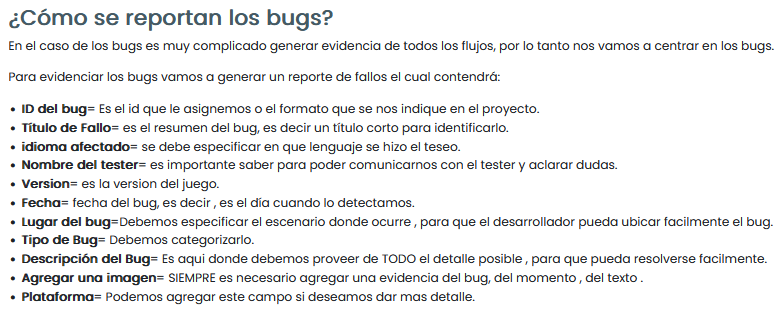
Testing de Videojuegos

Se pueden realizar pruebas relacionadas con la experiencia, con la funcionalidad o la localización.

*Técnicas:* pruebas de *funcionalidad* (errores de gráficos corruptos, interfaz, estabilidad, problemas mecánicos, congelación, bloqueos de progresión, problemas con la animación), *combinatoria* (analiza y estructura todas las salidas y entradas del juego por lo que se puede obtener una imagen clara sobre distintas combinaciones y resultados concebibles), *ad-hoc* (pruebas aleatorias no estructuradas wild-child), *compatibilidad* (ayuda a validar si la interfaz de usuario es compatible con todos los tamaños de pantalla, requisitos de hardware) y *regresión*. También hay pruebas de rendimiento y seguridad.

Los 3 tipos de errores más comunes:

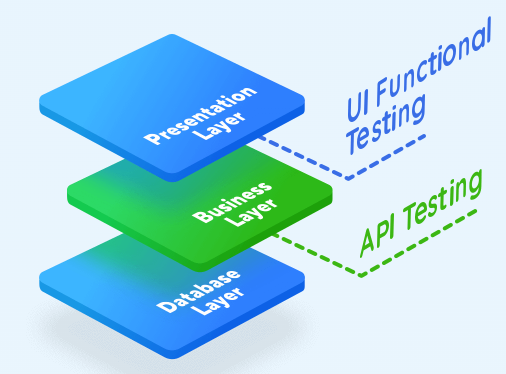
* **Glitch**: cuando un gráfico no carga de manera completa, personas quedan atrapados dentro de otros, superficies u objetos.
* **Bug:** habla de algo más serio, se refiriere a un error de programación.
* **Exploit:** deficiencias que existen en el juego para obtener una ventaja.





**APIs**

Las API son un conjunto de comandos, funciones y protocolos que permiten la comunicación entre softwares. Es un tipo de prueba de integración. Se considera como una prueba de bajo nivel ya que no interactúa directamente con la interfaz de usuario.



*Los Web Services* son un conjunto de estándares y protocolos para intercambiar datos entre aplicaciones que se conectan a la aplicación *a través de internet*. Esa es su principal diferencia con API. Su lenguaje intermedio sería a través de XML con el protocolo SOAP. Otra forma de comunicación, es a través de la arquitectura REST aplicando XML o JSON.

*SOAP (simple object Access protocol):*  es un protocolo diseñado para poder facilitar la comunicación entre plataformas basado en XML y en otro lenguaje denominado WSDL.

*REST:* arquitectura mas moderna y liviana, la mas común. Utiliza operaciones como GET (leer), POST (crear), PUT (actualizar), PATCH (modificar) y DELETE (borrar).

Tipos de prueba:

* **Pruebas Funcionales:**En estas pruebas lo que debemos primero es probar los diferentes tipos de [códigos de estado](https://mindhublab.com/mod/url/view.php?id=26445).
* **Pruebas de Seguridad**: Se prueba la Autentificación.
* **Pruebas de Rendimiento:** pruebas de carga, pruebas de estrés, pruebas de escalabilidad, pruebas de volumen, etc. Estas pruebas sirven para validar la carga de usuarios y que la API pueda responder correctamente a dicha carga.
* **Pruebas de Integración:** Podemos hacerlo verificando la integración con otras API integradas en un mismo proyecto.
* **Documentación**: La documentación es muy importante al momento de probar; no podemos comenzar a probar APIs si no tenemos documentación asociada a ella.

Nuestro trabajo como tester es el de validar cada una de las funcionalidades que ofrece la Api al usuario y verificar que cada código de respuesta sea el correcto de acuerdo con la petición realizada en el front.

Herramientas:

* SoapUI: permite probar, simular y generar código de servicios web de forma ágil, partiendo del contrato de los mismos en el formato WSDL y con vínculo SOAP sobre http.
* Postman: Es un cliente HTTP que nos da la posibilidad de testear ‘HTTP Request’ a través de una interfaz gráfica de usuario. Su ventaja es que la interfaz de usuario es mucho mas amigable e intuitiva, dispone de colecciones y agrupaciones de peticiones.

Tipos de tokens:

1. **Token Bearer:**Tipo de header de una petición HTTP que carga con información relevante al tipo de usuario con el objetivo de darle autoridad para ejecutar la acción requerida en la petición.
2. **Apy Key:** identificador que sirve como el medio de autenticación de un usuario para el uso de los servicios proporcionados por Sovos en los Web Services o en el API Rest.
3. **Basic Auth:** Este método utiliza el encabezado HTTP Authorization compuesto de la palabra Basic seguido de una cadena codificada en base64 que contiene un nombre de usuario y una contraseña separadas por dos-puntos (:).  
   Por ejemplo, este tipo de auth necesita de un password y una contraseña, finalmente una vez provisto de estos dos datos se construirá una cadena como esta:  69876357:testpassword\_DEMOPRIVATEKEY23G4475zXZQ2UA5x7M
4. **OAuth 1.0:** Este protocolo  permite a los usuarios de aplicaciones autorizar a aplicaciones que consuman sus APIs sin necesidad de que las contraseñas viajen en las peticiones.
5. **OAuth 2.0:** Es un Framework que facilita la delegación de la autorización del cliente, verificando su acceso a las APIs, A diferencia con el anterior el Oauth 2.0 No es un protocolo.

Bases de datos

Sirve para almacenar información y poder manipularla de alguna forma conveniente. Se generan mapeos de datos que hay que validar su funcionamiento. Cada transacción que se realiza con la base de datos debe cumplir con las propiedades *acid:*

* **Atomicidad**significa que una transacción falla o pasa. Esto significa que incluso si una sola parte de la transacción falla, significa que toda la transacción ha fallado. Por lo general, esto se llama la regla de 'todo o nada'.
* **Consistencia:** Una transacción siempre dará como resultado un estado válido de la base de datos.
* **Aislamiento:** Si hay varias transacciones y se ejecutan todas a la vez, el resultado / estado de la base de datos debería ser el mismo que si se ejecutaran una tras otra.
* **Durabilidad:** Una vez que se realiza y se confirma una transacción, ningún factor externo como la pérdida de energía o el bloqueo deberían poder cambiarla.

Se debe testear el correcto funcionamiento de las operaciones CRUD.

Se deben realizar consultas SQL apropiadas para validar estos objetos complejos.

