**Va a ser pruebas de sistema , probando el software de punta a punta, quiza puede ser de integracion. Tambien de caja negra en general.**

**Semana 1**

Caso de uso es user story (requerimientos), caso de prueba es el documento que indica como se va a testear, diseñado por test manager o equipo de testing y ejecutado por testers generalmente, excepto aquellos que requieran un conocimiento especializado y tecnico como por ejemplo relacionados con el codigo.

**Error defecto falla**

Error es humano // mal interpretacion de un requisito  
defecto es el resultado de un error humano // falta de funcionalidades  
falla el sistema a causa de un/os defecto // fallo global en el sistema, ej que no se puedan conectar, es decir que no se pueda hacer nada.

**7 Principios de testing**

1. La prueba muestra la presencia de defectos, no su ausencia
2. La prueba exahustiva es imposible
3. La prueba temprano ahorra tiempo y dinero
4. Los defectos se agrupan
5. Cuidado con la prueba del pesticida
6. La prueba se realiza de manera diferente según el contexto
7. La ausencia de errores es una falacia.

**Ciclo de vida:**

1. Inicial/nuevo: se recopila la informacion y se registra el defecto
2. Asignado: se analiza y trabaja en la solucion
3. En proceso: se analiza y trabaja en la solucion.
4. Corregido: se realizan los cambios de codigo para soluconar el defecto
5. En espera de verificacion: en espera de que sea asignado a un probador. Es desarrollador esta a la expectativa del resultado de la verificacion.
6. En verificacion:El probador ejecuta una prueba de confirmacion.
7. Verificado: se obtiene el resultado esperado en la prueba de confirmacion
8. Cerrado: el defecto fue corregido y se encuentra disponible para el usuario final.

Participacion de usuario/cliente: Solo al inicio y al final.

**Gestion de defectos:** trabajar con lo que es el tratamiento de los defectos. No existe la ausencia de defectos, con lo cual ¿Qué hacemos con ellos? Trabajarlos, resolverlos, priorizarlos. Estableciendo prioridades.

**Proceso de gestion de defectos:**

1. Detectar (naturalizacion del defecto).
2. Registrar, varia según su contexto, del componente o sistema, nivel de prueba y modelo de desarrollo elegido.
3. Investigacion y seguimiento: me pongo en el lugar del usuario y veo de donde viene el defecto, que fue lo que fallo, tomo acciones y saco concluciones. ¡Analisar!
4. Clasificacion/resolucion: no podemos clasificar lo que antes no analisamos, una vez termina el paso 3, podemos pasar a la fase de resolucion (por parte del equipo de desarrollo). la tarea del tester es probar que los defectos existen, no resolverlos.

Objetivos:  
1-brindar informacion sobre cualquier evento adverso, para identiricar efectos especificos, aislar el problema con una prueba de produccion minima y corregir los defectos potenciales.  
2-Proporcionar a los jefes de prueba un medio para hacer seguimiento a la calidad del producto y del impacto de prueba.  
3-Dar ideas para la mejora de los procesos de desarrollo y prueba.

**Informe de defectos:**

Es un registro de salida. Toda la informacion que se va reportando, va a ser la principal entrada para generar el **documento llamado informe de defectos.  
¿Cómo escribir un buen informe?** Si el defecto se reporta efecientemente las probabilidades de que sea solucionado rapidamente es mayor. Entonces, la solucion de un defecto dependera de la eficioencia con que se reporte.  
**Condiciones a tener en cuenta:** Los bugs deben tener identificadores unicos, Una falla debe ser reproducible para reportarla. Ser especifico, Reportar cada paso realizado para reproducirlo.  
**Problemas mas comunes con los informes:**

* Redactar de manera excesivamente coloquial y ambigua – dar solo captura de pantalla sin nada mas
* no incluir descripcion del resultado esperado para los pasos realizados
* no determinar un patron con el cual el defecto ocurre antes de reportar el mismo
* no leer defecto reportado siguiendo los pasos un mismo para ver que la descripcion era clara.

¿Qué es testing? La ejecucion de la prueba es solo uno de los pasos en el ciclo de vida de pruebas. Testing es un proceso, conjunto de actividades para encontrar defectos y posteriormente solucionarlos.  
Actividades:

* Ejecucion de prueba yt comprobacion de resultados
* Planificar las pruebas
* Analizar, diseñar e implementar las pruebas
* Informar el avance y resultado de la ejecucion de pruebas
* Evaluar la calidad de un objeto de prueba

Principios de testing:

1. La prueba muestra la presencia de defectos, no su ausencia
2. La prueba exhaustiva se imposible
3. La prueba temprana ahorra tiempo y dinero
4. Los defectos se agrupan
5. Cuidado con la prueba del pesticida
6. La prueba se realiza de manera diferente según el contexto
7. La ausencia de errores es una falacia.

Mesa de 3 patas: business analyst, softwqare developer, quality assurance.

**Semana 2**

**Casos de prueba**

Documento escrito que proporciona informacion escrita sobre que y como probar.

Proceso documentando que define que se esta probando y como.  
El que es relacionado con los requisitos del usuario y el como van a ser los pasos para ejecutar ese caso de prueba.

Que tener en cuenta a la hora de escribir un buen caos de prueba:

1. **Identificador / ID**, la mayoria de herramientas lo generan automaticamente
2. **Nombre del caso de prueba (conciso),** utilizar una nomenclatura que este definida, de no existir se recomiendo incluir el nombre de modulo al que correpsonde le caso de prueba.
3. **Descripcion**, debe decir que se va a probar, ambiente, y datos necesarios para ejecutarlo.
4. **Precondicion,** asuncion que debe cumplirrse antes de ejecutar el caso de pruebas
5. **Pasos**, acciones que se deben realizar para obtener los resultados
6. **Resultados esperados,** lo que se indica al probar cual deberia ser la experiencia luego de ejecutar los pasos y determinar si el test fallo o no.

No es lo mismo diseñar y ejecutar el caso de prueba, aca hablamos de diseño, no de ejecutar casos de prueba.

Estados de diseño: Durante fase de diseño de prueba.

* **En diseño**
* **Diseñado** // ya se creo y esta listo para pasar a un estado de rivison
* **En revision** // responsabilidad de un test manager o persona en detalle de la operativa en el sector de testing
* **Revisado** // listo para pasar a ejecucion

Estados de ejecucion: durante fase de ejecucion de pruebas.

* **No corrido** // no fue ejecuitado
* **Pasado** // pasado correctamente
* **Fallado** // resultado actual discrepante del esperado
* **No aplica** // puede ocurrir por ej: ejecutamos ciertas pruebas pero esas pruebas son de proyectos anteriores o similares, con lo cual para este no aplica y se deja de lado, pero al no correrlo no tiene sentido o seria incorrecto describirlo como fallado
* **Bloqueado** // Tenemos dependencia tal de una condicion que no se puede cumplir al 100% con lo cual no se puede seguir con la ejecucion
* **No completo** // esta en curso, por ejemplo termina la jornada laboral y no terminamos un caso de prueba en curso, la dejamos y queda como no completo hasta retomarla.
* **Diferido** // postergamos su ejecucion, puede que un caso de prueba no sea urgente y se deje para otro sprint.

**Test plan**

Es aquel documento donde documentamos nusetros casos de prueba, documentamos posibles estados. El responsable de documentar esto sera el equipo de testing y tal vez un teste manager sera el responsable de revisar el documento.

**Caracteristicas para un buen caso de prueba.**

* **Deben ser simples:** debe ser lo mas simple posible ya que otra persona diferente al autor debe poder ejecutarlo, usar lenguaje asertivo para facilitar comprension.
* **El titulo debe ser fuerte:** el titulo debe dar una comprension rapida y completa del caso de prueba
* **Tener en cuenta al usuario final:** debemos ser empaticos y crear casos de prueba que cumplan con los requisitos del cliente, brindando facilidad de uso.
* **No asumir:** no asumir funcionalidad y caracteristicas de la aplicación, referirse a documentos de especificacion y ante cualquier duda consultar.
* **Asegurar mayor cobertura posible:** escribir casos de prueba para todos los requisitos especificados.
* **Autonomia:** el caso de prueba debe generar lo mismos resultados siempre, sin importar el sujeto que lo pruebe.
* **Evitar la repeticion de casos de prueba:** si se necesita un caso de prueba par ejectuar otro, indicar el caso de prueba por su id.

**Testing positivo y Testing negativo:**

**El testing positivo:** trabajamos el flujo normal, lo que esperamos que el software haga a travez de acciones que la persona que ejecuta la prueba pueda identificar.  
Ejemeplo: en un login ingresamos usuario y contraseña correcto, se espera que se pueda pasar a la pagina principal. (con esto ya seria un testing positivo)

**El testing negativo:** buscamos un flujo anormal, por ejemplo buscamos mediante parametros incorrectos, siga un funcionamiento no esperado.  
Ejemeplo: en un login ingresamos usuario y contraseña incorrecto, se espera que NO se pueda pasar a la pagina principal como si el usuario y contraseña fuesen correctos. SI nos dejase pasar estamos en presencia de un defecto.

**Happy path pertenece al testing positivo:** es un camino de prueba cuidadosamente seleccionado para que una persona puede validar cierta funcionalidad.  
Comunmente se utliza en proyectos “incendiados” para poder demostrar que aunque hay problemas, ciertas funcionalidad son efectivas.

|  |  |  |
| --- | --- | --- |
| **Ciclo de vida de un defecto** No existe un proceso de prueba unico y universal, pero existen actividades de prueba comunes que nos ayudan a organizarnos para alcanzar los objetivos establecidos.  El proceso que gestiona un defecto desde su descubrimiento hasta su solucion se denomina ciclo de vida de un defecto.  En cada estado solo existe un responsable del defecto, excepto en estados terminales -cerrado, duplicado-, debido a que no se van a realizar mas acciones. | | |
| **Cerrado**  El defecto fue corregido y se encuentra disponible para el usuario final. | 1. **Nuevo/Inicial**   Se recopila la informacion y se registra el defecto Documento: Test Plan. | **Asignado**  De ser un defecto valiado y debe solucionarse asigna al equipo de desarrollo, sino se puede rechazar o diferer.  (en casos como duplicado, diferido, devuelto o rechazado) |
| **Verificado**  Se obtiene el resultado esperado en la prueba de confirmacion |  | **El proceso**  Se analiza y trabaja en la solucion |
| **En verificacion**  El probador ejecuta una prueba de confirmacion.  Puede reabrirlo, indicando que el defecto no se ha solucionado. | **En espera de verificacion**  Espera que sea asignado a un probador. El desarrollador esta a la expectativa del resultado de la verificacion | **Corregido**  Se realizan los cambios en el codigo para solucionar el defecto |
| **Proceso de gestion de defectos.**   1. Detectar 2. Registrar (varia según el contexto del componente/sistema/nivel de prueba/modelo de desarrollo elegido. 3. Investigacion y seguimiento 4. Clasificacion/Resolucion | | |
| **Objetivos**   1. Brindar Informacion sobre eventos adversos, identificar efectos especificos, aislar problema con una prueba de produccion minima y corregir defectos potenciales. 2. Proporcionar a jefes de prueba un medio para hacer seguimiento de calidad del producto y del impacto en la prueba. 3. Dar ideas para la mejora de procesos de desarrollo y prueba. | | |

|  |
| --- |
| **Ciclo de vida de las pruebas de software (STLC)** |
| No existe un proceso de prueba unico y universal, pero si actividades de prueba comunes que nos ayudan a organizarnos para alcanzar los objetivos establecidos.  **Proceso de prueba en contexto:**   * Modelo de ciclo de vida de desartrollo de software y mtodologias de proyecto de uso. * Niveles y tipos de prueba considerados. * Riesgos de producto y de proyecto. * Dominio del negocio. * Restricciones operativas, icluyendo, pero no limitadas a: 1) Plazos. 2)Complejidad |
| **Tareas principales.**   1. Planificacion 2. Seguimiento y control 3. Analisis 4. Diseño 5. Implementacion 6. Ejecucion 7. Conclusion 8. Diseño. |
| **Planificacion**  Definen objetivos y enfoque de la prueba dentro de restricciones impuestas por contexto.   * Determinar el alcance, los objetivos y los riesgos * Definir el enfoque y estrategia general. * Integrar y coordinar las actividades a realizar durante el ciclo de vida del software. * Definir las especificaciones de tecnicas, tareas de prueba adecuadas, las personas y otros recursos necesarios. * Establecer un calendario de pruebas para cumplir con un plazo limite * Generar el plan de pruebas.   Documentos de salida:   * Plan de prueba – general y/o por nivel de prueba-. |
| **Seguimiento y control.**  Reunir informacion y proporcionar retroalimentacion y visibilidad sobre las actividades de la prueba. Como parte del control, se pueden tomar acciones correctivas, como cambiar la prioridad de las pruebas, el calendario y reevaluar los criterios de entrada y salida.  **Subactividades:**   * Comprobar resultados y registros de la prueba en relacion con los criterios de cobertura especificados. * Determinar si se necesitan mas pruebas dependiendo del nivel de cobertura que se debe alcanzar.   **Documento de salida**   * Informe de avance de la prueba. |
| **Analisis**  Determina “que probar”  **Subactividades:**   * Analizar base de prueba correspondiente al nivel de prueba considerado. -informacion de diseño e implementacion, la implementacion del componente o sistema en si, informes de analisis de riesgo, etc.- * Identificar defectos de distintos tipos en las bases de prueba – ambiguedades, omisiones, inconsistencias, inexactitudes, etc.- * Identificar los requisitos que se van a probar y definir las condiciones de prueba para cada requisito * Capturar la trazabilidad entre la base de prueba y las condiciones de prueba.   **Documento de salida**   * Contrato de prueba que contiene las condiciones de la prueba. |
| **Diseño**  Se determina “como probar”  **Subactividades**   * Diseñar y priorizar casos de prueba y conjuntos de casos de prueba de alto nivel. * Identificar los datos de prueba necesarios * Diseñar el entorno de prueba e identificar la infraestructura y las herramientas necesarias * Capturar la trazabilidad entre la base de prueba, las condiciones de prueba, los casos de prueba y los procedimientos de prueba.   **Documento de salida:**   * Casos de prueba de alto nivel diseñados y priorizados. |
| **Implementacion**  Se completan los productos de prueba necesarios para la ejecucion de la prueba, incluyendo la secuenciacion de los casos de prueba en procedimientos de prueba.  **Subactividades:**   * Desarrollar y priorizar procedimientos de prueba * Crear juegos de prueba (test suite) a partir de los procedimientos de prueba * Organizar los juegos de prueba dentro de un calendario de ejecucion * Construir el entorno de prueba y verificar que se haya configurado correctamente todo lo necesario. * Preparar los datos de prueba, los procedimientos de prueba y los juegos de prueba.   **Documento de salida:**   * Procedimientos y datos de prueba * Calendario de ejecucion * Test suite. |
| **Ejecucion**  Durante esta actividad se realiza la ejecucion de los casos de prueba.  **Subactividades:**   * Registrar los identificadores y las versiones de los elementos u objetos de prueba. * Ejecutar y registrar el resultado de las pruebas de forma manual o tuilizando herramientas. * Comparar los resultados reales con los resultados esperados * Informar sobre los defectos en funcion de los fallos observados. * Repetir las actividades de prueba, ya sea como resultado de una accion tomada para una anomalia o como parte de la prueba planificcada – retest o prueba de confirmacion-. * Verificar y actualizar la trazabilidad entre la base de prueba, las condiciones de prueba, los cvasos de prueba, los procedimientos de prueba y los resultados de prueba y los resultados de la prueba.   **Documento de salida:**   * Reporte de defectos. * Informe de ejecucion de prueba |
| **Conclusion**  **Subactividades**   * Comprobar que todos los informes de defecto estan cerrados. * Finalizar, archivar y almacenar el entorno de prueba, los datos de prueba, la infraestructura de prueba y otros productos de la prueba para su posterios reutilizacion. * Transpaso de los productos de prueba a otros equipos que podrian beneficiarse con su uso * Analizar las elecciones aprendidas de las actividades de prueba completadas. * Utilizar la informacion recopílada para mejorar la madruez del proceso de prueba.   **Documento de salida:**   * Informe resumen de prueba. * Lecciones aprendidas. |

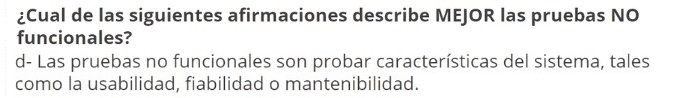
|  |
| --- |
| **Niveles de pruebas**  Grupo de actividades que estan integradas, pero no se mesclan con los tipos de prueba. |
| Pruebas llevadas a cabo por los desarrolladores, necesita personas con concimiento tal que excede a los testers. Un ejemplo de componente es un boton. prueba de componente también llamada prueba unitaria es por definición la prueba que se lleva a cabo luego de haber construido el componente.  Para toda prueba de componente hay que tener en cuenta: (a) las bases de la prueba, es decir: los requisitos del componente, su diseño detallado y su código; (b) los objetos de prueba más típicos, es decir: los componentes, ó clases, ó unidades, ó módulos, más programas y/o conversión de datos y/o migración de programas; y finalmente (c) los módulos de la/s base/s de datos.  UNA UNIDAD DE SOFTWARE ESPECIFICA, UN EVENTO. El login, un boton. |
|  |
| Las pruebas de integración dentro del software testing chequean la integración o interfaces entre componentes, interacciones con diferentes partes del sistema, como un sistema operativo, sistema de archivos y hardware o interfaces entre sistemas. Las pruebas de integración son un aspecto clave del software testing. Integracion entre 2 componentes, por ejemplo primero alta de usuario y despues login. |
|  |
| Ejemplo de prueba de sistema: alta de usuario, verificar las pruebas de zoom, ver todo el material de cada clase, ver le progreso que tengo en una clase. Es trabajo de un tester, es diferente de la prueba de aceptacion en que la realiza un tester y no el usuario final. |
|  |
| Es similar a la prueba de sistema, pero es llevada a cabo por el usuario o cliente. |
|  |

**Tipos de pruebas de software:**

**Pruebas funcionales**

Las pruebas funcionales se llevan a cabo para comprobar las características críticas para el negocio, la funcionalidad y la usabilidad. Las pruebas funcionales garantizan que las características y funcionalidades del software se comportan según lo esperado sin ningún problema. Valida principalmente toda la aplicación con respecto a las especificaciones mencionadas en el documento Software Requirement Specification (SRS). Los tipos de pruebas funcionales incluyen pruebas unitarias, pruebas de interfaz, pruebas de regresión, además de muchas.

**Pruebas no funcionales**

Las pruebas no funcionales son como pruebas funcionales; sin embargo, la principal diferencia es que esas funciones se prueban bajo carga para el rendimiento de los observadores, fiabilidad, usabilidad, escalabilidad, etc. Las pruebas no funcionales, como las pruebas de carga y esfuerzo, normalmente se llevan a cabo mediante herramientas y soluciones de automatización, como LoadView. Además de las pruebas de rendimiento, los tipos de pruebas no funcionales incluyen pruebas de instalación, pruebas de confiabilidad y pruebas de seguridad. Estos requisitos no se exteriorizan, son tasitos. Portalibilidad, escalabilidad, performance, etc. El cliente no lo suele pedir pero se esperan que se haga.

**Pruebas unitarias**

Las pruebas unitarias se centran en probar piezas/unidades individuales de una aplicación de software al principio del SDLC. Cualquier función, procedimiento, método o módulo puede ser una unidad que se someta a pruebas unitarias para determinar su corrección y comportamiento esperado. Las pruebas unitarias son las primeras pruebas que los desarrolladores realizan durante la fase de desarrollo.

**Pruebas de integración**

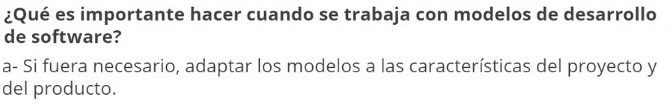
Las pruebas de integración implican probar diferentes módulos de una aplicación de software como grupo. Una aplicación de software se compone de diferentes submódulos que trabajan juntos para diferentes funcionalidades. El propósito de las pruebas de integración es validar la integración de diferentes módulos juntos e identificar los errores y problemas relacionados con ellos.

**Performance Testing**

Las pruebas de rendimiento son un tipo de pruebas no funcionales, realizadas para determinar la velocidad, estabilidad y escalabilidad de una aplicación de software. Como su nombre indica, el objetivo general de esta prueba es comprobar el rendimiento de una aplicación con respecto a los diferentes puntos de referencia del sistema y de la red, como la utilización de la CPU, la velocidad de carga de la página, el control de tráfico máximo, la utilización de recursos del servidor, etc. Dentro de las pruebas de rendimiento, hay varios otros tipos de pruebas, como pruebas de carga y pruebas de esfuerzo

**Pruebas de aceptación**

En ingeniería de software y pruebas de software, las pruebas de aceptación pertenecen a las últimas etapas previas a la liberación en firme de versiones nuevas a fin de determinar si cumplen con las necesidades y/o requerimientos de las empresas y sus usuarios.



**Caja blanca, gris y negra.**

Cuándo estamos refiriéndonos a “una caja” es la manera de observar el contenido del Software, ya sea que no tenemos noción más que la interfaz con la que estamos trabajando, caja negra. Cuándo podemos ver todo el contenido como una caja de cristal, caja blanca. Las integraciones, los datos cómo fluyen de un lugar a otro, dónde no conozco el código ni veo interfaz pero puedo ver cómo fluye la información a través de las redes, caja gris.

**Negra**: No podemos observar cómo fue construida, no vemos el código, no sabemos su arquitectura, no tenemos nociones más que la interfaz que estamos interactuando.

* Partición de equivalencia: en esta tecnica se dividen los datos en particiones conocidas como clases de equivalencia donde cada miembro de estas clases o particiones es procesado de la misma manera.  
  Las validas nos mostrara las particiones que el software acepte, las invalidas las que no acepte.  
  Ej: RGB son los colores que acepta, cualquier otro es invalido.
* Valores límite: Es una extencion de la tecnica de particion de equivalencia que solo se puede usar cuando la particion esta ordenada y consiste en datos numericos o secuenciales. Estableciendo un valor limite inferior y uno superior.
* Tabla de decisiones: Tiene que ver ocn valores de entrada que toman como salida una resultante, por ejemplo con condicion A y B se toma decisión 1. Con condicion A y C se toma decisión 2, con decisión B y C se toma decisión 3.
* Transición de estados: un diagrama de transicion de estado muestra los posibles estados del software, asi como la forma en que el software entra, sale y realiza las transiciones entre estados.  
  Buscamos mostrar o excibir los diferentes estados que el software presenta. Lo importante es el ultimo estado que el software presenta.   
  Técnica de diseño de pruebas de caja negra en la cual los casos de prueba son diseñados para ejecutar transiciones de estado válidas e inválidas.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Estado 1 | Estado 2 | Estado 3 | Estado 4 | Estado 5 | Estado final |
| Nacido | Soltero | Con hijo/s |  |  | Con hijo/s |
| Nacido | Soltero | Casado | Viudo |  | Viudo |
| nacido | soltero | casado | Divorciado | Casado | Casado |

* Casos de usos

• **Participación de Equivalencia.** Esos grupos de datos que pueden entrar para casos exitosos o para casos no exitosos.

• **Valores Límite.** Se puede tener usado un rango de valores.  
• **Tabla de Decisiones.** Va enfocada si tuviéramos valores seleccionables.  
• **Transición de Estados.** Cómo el componente se comporta.  
• **Casos de Uso.** Realizar escenarios que pueda realizar el usuario.

**Blanca**: Es como una caja de cristal, puedo ver todo lo que hay adentro e incluso puedo ser parte del equipo que desarrolla el software.

* Cobertura de declaración
* Cobertura de decisiones

• **Cobertura de declaraciones.** Las declaraciones son todo aquello que tienes dentro del código y estás asumiendo que es lo que se pide que haga, al decir cobertura es, dependiendo el tipo de software, los requerimientos, el objetivo, se establece un porcentaje de cobertura, esto significa que, cada línea de código debería ser ejecutada al menos una vez, cada sentencia debería de ejecutarse alguna vez.  
• **Cobertura de código.** Que se evite tener código obsoleto.

**Gris**: Pueden ser la integraciones, cómo fluye el código y puedo ver como se transmiten los datos a través de las redes.

* Casos de negocios
* Pruebas End-to-End
* Pruebas de integración

• **Casos de Negocio.** Es necesario conocer cómo el usuario interactúa, qué datos ingresa y qué datos van a ser retornados.  
• **Pruebas End to End.** Cómo se están agregando datos y aún no queremos ver datos de salida.  
• **Pruebas de integración.** Ver cómo viajan esos datos, la respuesta y la comunicación de cómo fluyen los datos entre diferentes servicios.

**Pruebas basadas en la experiencia:** Aprovechan el conocimiento de desarrolladores, probadores y usuarios para diseñar, implementar y ejecutar la prueba.  
Son pruebas diseñadas en base al conocimiento de los anteriormente mencionados y usan ese conocimiento para validar el flujo valido o invalido.

* Prueba de prediccion de errores
* Basadas en listas de comprobacion
* Pruebas exploracion

**Pruebas de humo y regresion.**

**Estan relacionadas al cambio, es decir que son dinamicas.**

**Prueba de humo** es verificar las funcionalidad core de una implementacion, lo basico y que no deberia estar ausente. Se realiza cuando se va a agregar un nuevo mvp, puede ser nueva funcionalidad o correccion de defectos.

**Una suit de regresion** es la verificacionda la continuidad en el buen funcionamiento de otras funcionalidades, dada la implementacion de otra nueva, manteniendo la integridad del sistema aunque se incluyan cambios. Se realiza cuando se va a agregar un nuevo mvp, puede ser nueva funcionalidad o correccion de defectos. Cuando automatizamos debemos empezar por las pruebas de regresion

Dentro de las suits de regresion, van a contener lo mismo que las pruebas de humo, asi que puede y va a haber duplicidad entre ambas pruebas. Todos los test cases deberian pertenecer a un suite de humo o regresion por que ambas son importantes y se jeecutan a lo largo del desarrollo de un sistema, siendo relevantes para el funcionamiento saludable del producto en construccion y relevantes para la calidad final del producto.

**Pruebas estaticas y dinamicas:**

**Estaticas:** aquellas que evaluamos de forma manual, en un tiempo especifico y determinado, por eso son estaticas.  
revisamos documentacion, una linea de codigo, cosas puntuales que no necesiten la ejecucion del software, pueden ser llevadas a cabo por gente que entienda del codigo.  
Estructura, ver si un IF esta cerrado, etc.  
Suelen ser de caja gris, pero siempre se tiende hacia un lado o al otro en mayor medida.

**Conceptos basicos de la prueba estatica**

* Especificaciones, requisitos de negocio, funcionales y de seguridad.
* Epicas, historias de usuarios y criterio de aceptacion. (epicas son cuestiones que hacen a la funcionalidad pero no estan demasiado desglosadas, por lo cual hay que desmenuzar para atacar esos puntos uno a uno.) Ej: testear un carrito de compra y todos sus componentes, que se cargue el carro, que se ejecute la compra, que se realice el checkout, etc.
* Especificaciones de arquitectura
* Codigo
* Prudctos de prueba, planes, casos procedimientos y guiones de prueba.
* Manual de usuario
* Contratos, planes de proyectos, calendarios y presupuestos.

**Dinamica:** Esta relacionado con la ejecucion del software, sistema o componente.  
Puede ser basda en experiencia, se complementan con las estaticas debido a que encuentra diferentes tipos de defectos, para la generacion de casos de prueba se utilizan diferentes tecnicas de caja negra, blanca o basdadas en la experiencia de usuario.

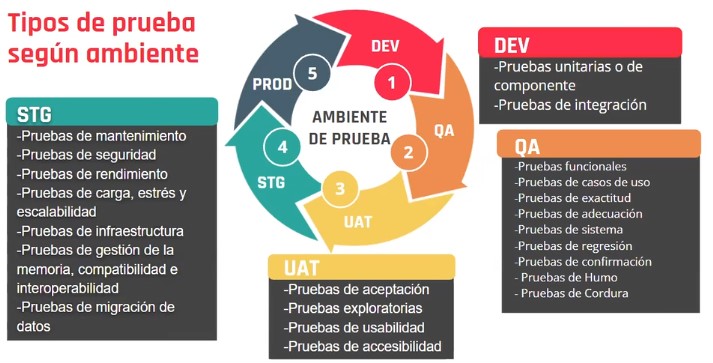
**Proceso de revision:** seleccionamos un producto de trabajo cuidadosamente, su trabajo es encontrar y remover errores, puede ser realizadas por uno o mas personas.

Revisiones formales: roles definidos, siguen un proceso establecido y deben ser documentadas. Ej: auditorias programadas.  
Una revision es de las mas formales. CREO

Revisiones informales: no siguen un proceso definido y no son documentadas formalmente. Ej: empresas pequeñas donde todos hacen todo y dependiendo de demandas/disponibilidad/requisitos se tapan baches con estas revisiones informales. No hay plazos claros y documentacion escasa o nula.

Requisitos /requerimientos: definen condiciones de funcionamiento del sistema en el ambiente operacional.

* Requisito de usabilidad
* Eficiencia
* Disponibilidad
* Confiabilidad
* Integridad
* Mantenibilidad

Lo evalua un test manager

**No desarrollaremos en ambiente de produccion debido a que**:

EN general los probadores no tienen acceso a este ambiente  
en el caso de tener acceso y realizar pruebas  
no se deben realizar acciones que generen datos  
Se corre el riesgo de ingresar datos basura  
Se interfiere en los datos de seguimiento.

**Primer kahoot**

1. Cuales son los niveles de prueba?  
   Componente-integracion-sistema-aceptacion.
2. Uno de los objetivos de la prueba de aceptacion es establecer confianza en la calidad del sistema en su conjunto.  
   Verdadero.
3. La prueba no funcional observa el comportamiento del software.  
   Falso
4. Un caso de uso es la base de prueba para las pruebas de sistema.  
   Verdadero.
5. Una de las bases de prueba para las pruebas de componente es el codigo.  
   Verdadero
6. La prueba funcional prueba “que tan bien” se comporta el sistema.

**Segundo kahoot**

1. ¿Que es una tecnica de prueba?  
   Una guia, Como vamos a trabajar las pruebas, Un modelo para entregar software de calidad.
2. Particion de equivalencia es una tecnica de…  
   Caja Negra.
3. Analisis de valores limites es una tecnica de caja blanca  
   Falso.
4. Pruebas basadas en comprobacion es una tecnica de Caja Blanca  
   Falso.
5. ¿A que clasificacion responden las pruebas basadas en listas de comprobacion.  
   Basadas en la experiencia.

**Ejercicio PG**

¿Por qué es necesario aplicar tecnicas de pruebas para el diseño de software?

Nos enfocamos en como vamos a trabajar las pruebas de software. Posteriormente revisarlas, analizarlas, diseñarlas, implementarlas y ejecutarloas para nuestro software bajo alcance.

¿Cuál es la diferencia entre pruebas de caja negra y de caja blanca?

Las tecnicas de caja negra no tarbajan con el codigo, pero si desde la interfaz de usuario, mientras que las tecnicas de caja blanca si trabajan con la revision del codigo y su ejecucion bajo diferentes instancias.

¿Qué relacion existe entre la particion de equivalencia y el analisis de valores limites?  
Ambas definen las clases validas e invalidas, ambas son de caja negra.  
Mientras particiond e equivalencia acepta cualquier valor de datos, mientras uqe valores limites acepta valores numericos detallados.

Explique brevemente el significado de una clase de equivalencia nov alida.  
Son aquellas clases que el software estara rechazando, es decir, no aceptando como tales.

¿Por qué es necesario aplicar tecnicas de prueba par el diseño de las mismas?

Nos enfocamos en como vamos a trabajar las pruebas de software, posteriormente revisarlas, analizarlas., diseñarlas, implementarlas y ejecutarlas para nuestro software bajo alcance.