**Testing**

**Módulo 1 - Fundamentos de testing y Gestión de Defectos**

**SEMANA 1**

**CLASE 1 – PRIMEROS PASOS**

Testing = Calidad.

Un tester parte siempre de la suposición que un programa contiene errores. Por lo que deberá probarlo para encontrar la mayor cantidad y avisar para que lo soluciones. Brindan calidad a ese desarrollador.

Calidad = Satisfacción del cliente. Testing = Confianza.

Realizar pruebas frecuentes es vital en el proceso de desarrollo.

Existen proyectos en la fase final, que tienen que volver a corregirse por factores humanos:

- Requerimientos poco claros.

- Modificaciones a último momento

- Errores de diseño y testers no calificados.

El testing sirve para:

- Encontrar el mayor número de defectos y remediarlos

- Asegurar el buen funcionamiento del producto

- Lograr un mayor grado de calidad.

**7 principios de testing →**

1) La prueba muestra la presencia de defectos, no su ausencia:

No puede probar que no hay defectos. Reduce la probabilidad de que queden defectos no descubiertos en el software, pero, incluso si no se encuentran, el proceso de prueba no es una demostración de corrección.

2) La prueba exhaustiva es imposible:

No es posible probar todo —todas las combinaciones de entradas y precondiciones—, excepto en casos triviales. En lugar de intentar realizar pruebas exhaustivas se deberían utilizar el análisis de riesgos, las técnicas de prueba y las prioridades para centrar los esfuerzos de prueba.

3) La prueba temprana ahorra tiempo y dinero:

Para detectar defectos de forma temprana, las actividades de testing, tanto estáticas como dinámicas, deben iniciarse lo antes posible en el ciclo de vida de desarrollo de software para ayudar a reducir o eliminar cambios costosos.

4) Los defectos se agrupan:

En general, un pequeño número de módulos contiene la mayoría de los defectos descubiertos durante la prueba previa al lanzamiento o es responsable de la mayoría de los fallos operativos. Tratar de no tener que probar todas las posibilidades, agrupamos comportamientos. Agrupar errores arrastrados.

5) Cuidado con la prueba del pesticida:

Si las mismas pruebas se repiten una y otra vez, eventualmente estas pruebas ya no encontrarán ningún defecto nuevo. Para detectarlo, es posible que sea necesario cambiar las pruebas y los datos de prueba existentes.

6) La prueba se realiza de manera diferente según el contexto:

Por ejemplo, el software de control industrial de seguridad crítica se prueba de forma diferente a una aplicación móvil de comercio electrónico.

7) La ausencia de errores es una falacia:

El éxito de un sistema no solo depende de encontrar errores y corregirlos hasta que desaparezcan ya que puede no haber errores, pero sí otros problemas. Existen otras variables a tener en cuenta al momento de medir el éxito.

**Rol del tester**

Aspecto psicológico:

Humanos tienden a ser orientados a objetivos y establecimientos de metas.

En base a lo que queremos demostrar es el camino que seguiremos y puede traer subconscientemente menos probabilidad de demostrar lo contrario a nuestro pensamiento.

Prueba independiente:

La forma en que se implementa la independencia de la prueba varía dependiendo del modelo de ciclo de vida de desarrollo de software.

Puede ser diferente en desarrollo ágil, pueden los propietarios del producto realizar prueba de aceptación para validar las historias de usuario al final de cada iteración, etc.

Beneficios:

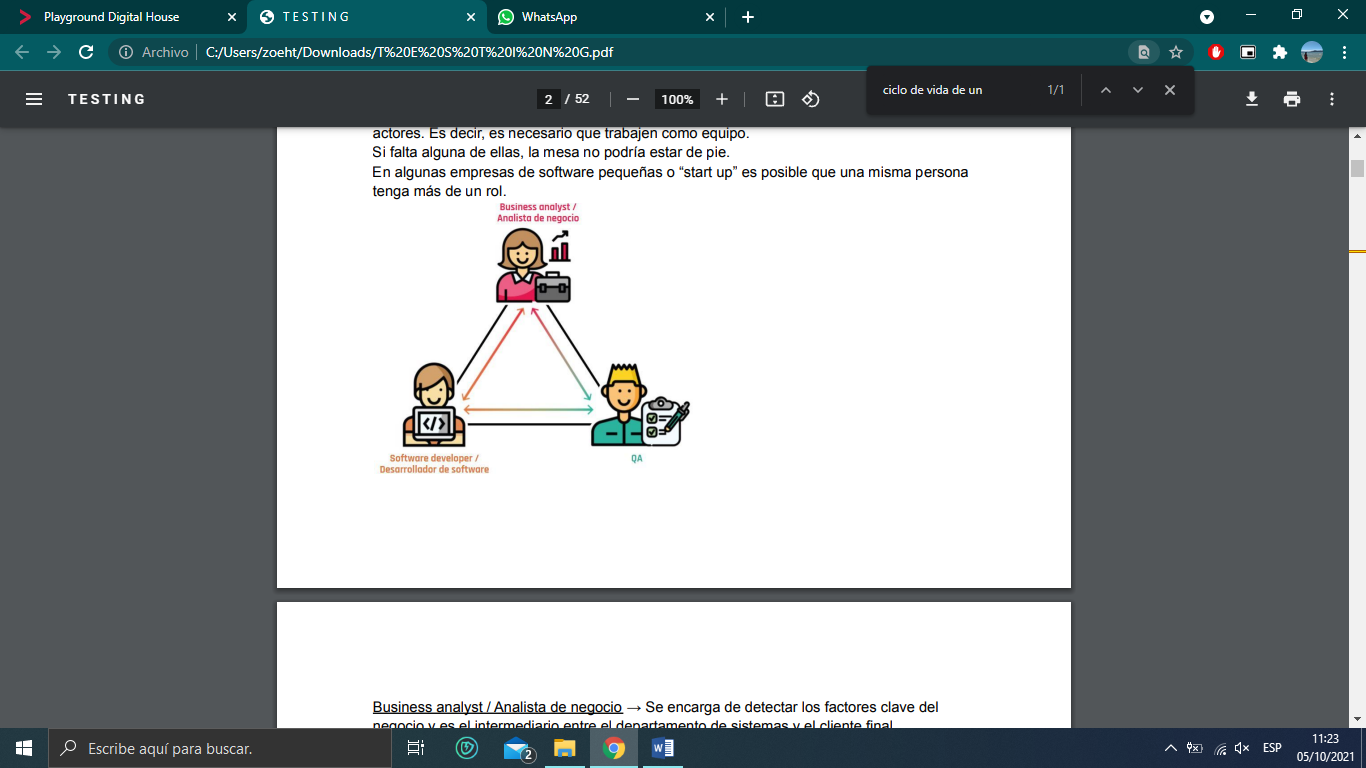
* Reconocimiento de fallos debido a diferente contexto, perspectivas técnicas y sesgos.
* Verificar, cuestionar o refutar las suposiciones hechas por implicados durante la especificación e implementación del sistema.

Desventajas:

* Pueden perder el sentido de la responsabilidad respecto a la calidad.
* Desarrolladores independientes pueden ser vistos como un cuello de botella o ser culpados por retrasos o liberación.
* Probadores independientes pueden carecer de información importante, por ejemplo el objeto de prueba.

**Mesa de 3 patas**

Si bien cada actor tiene un rol definido, es necesario un trabajo en comunión entre los 3 actores. Es decir, es necesario que trabajen como equipo. Si falta alguna de ellas, la mesa no podría estar de pie. En algunas empresas de software pequeñas o “start up” es posible que una misma persona tenga más de un rol.



**CLASE 2 – GESTIÓN DE DEFECTOS**

**Gestión de defectos**

**Ciclo de vida de un defecto**

El proceso que gestiona un defecto desde su descubrimiento hasta su solución 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 más acciones.

|  |  |  |
| --- | --- | --- |
| **Cerrado**  El defecto fue corregido y se encuentra disponible para el usuario final. | 1. **Nuevo/Inicial**   Se recopila la información y se registra el defecto. | **Asignado**  De ser un defecto validado y debe solucionarse asigna al equipo de desarrollo, sino se puede rechazar o diferir.  (en casos como duplicado, diferido, devuelto o rechazado) |
| **Verificado**  Se obtiene el resultado esperado en la prueba de confirmación. |  | **El proceso**  Se analiza y trabaja en la solución. |
| **En verificación**  El probador ejecuta una prueba de confirmación.  Puede reabrirlo, indicando que el defecto no se ha solucionado. | **En espera de verificación**  Espera que sea asignado a un probador. El desarrollador está a la expectativa del resultado de la verificación. | **Corregido**  Se realizan los cambios en el código para solucionar el defecto. |

**Error, defecto y falla.**

Un error es cometido por un programador, esto genera un defecto en el software, lo que culmina en un fallo del sistema.



**Proceso de gestión de defectos**

1. Detectar
2. Registrar (varía según el contexto del componente/sistema/nivel de prueba/modelo de desarrollo elegido.
3. Investigación y seguimiento
4. Clasificación/Resolución

Objetivos

1. Brindar Información sobre eventos adversos, identificar efectos específicos, aislar problema con una prueba de producción mínima 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.

Escribir un buen informe

1. Si el defecto se reporta eficientemente, aumenta probabilidades de solucionarlo rápida y efectivamente.
2. Los bugs deben tener identificadores únicos (ID)
3. Las fallas deben ser reproducibles para reportarla, de no serlo, no es un defecto. Se puede hacer una nota personal para posterior evaluación.
4. Especificidad, no debe haber suposiciones o ideas sobre lo que ocurre u otra información irrelevante para reproducir el defecto.
5. Reportar cada paso para reproducirlo.

Problemas más comunes al escribir informes:

1. Redactar defecto de manera excesivamente coloquial y ambigua.
2. Dar solo una captura del defecto sin indicar contexto
3. No incluir en descripción del defecto cual era el resultado esperado.
4. No determinar un patrón con el cual el defecto ocurre. -es importante para ser directos en cuál es el problema.-
5. No leer el defecto reportado siguiendo los pasos nosotros mismo para ver que la descripción es clara.
6. No incluir información que dada las características del defecto, la misma es de relevancia.

Cuando se detecta un defecto —como parte de las pruebas estáticas—, o se observa un fallo —como parte de las pruebas dinámicas—, la persona implicada debería recopilar los datos e incluirlos en el informe de defectos. Esta información debería ser suficiente para tres fines:

* Gestión del informe durante el ciclo de vida de los defectos.
* Evaluación del estado del proyecto, especialmente en términos de calidad del producto y progreso de las pruebas.
* Evaluación de la capacidad del proceso.

Los datos necesarios para la gestión de los informes de defectos y el estado del proyecto pueden variar en función de cuándo se detecta el defecto en el ciclo de vida, siendo la información requerida menor en etapas anteriores —por ejemplo, revisiones de requisitos y pruebas unitarias—. No obstante, la información básica recopilada debería ser coherente durante todo el ciclo de vida e, idealmente, en todos los proyectos.

Partes de un informe de defectos:

*Id* – Identificador, código único e irrepetible.

*Titulo* – Corto y específico, que se entienda lo que queremos reportar. Cuando el desarrollador o el equipo vean el titulo deben poder interpretar rápidamente qué es, dónde esta y cuán importante es el defecto.

*Descripción* – Describir un poco más el error.

*Fecha del informe del defecto* – Cuándo se detectó el defecto.

*Autor* – Quién detecto el defecto.

*Identificación del elemento de prueba* – Nombre de la aplicación o componente que estamos probando.

*Versión* – Numero que indica en que versión esta la aplicación.

*Entono* – Entorno en que probamos (desarrollo, QA, producción)

*Pasos a reproducir* – Pasos a seguir para llegar al defecto.

*Resultado esperado* – Lo que se espera que suceda, muchas veces según los requerimientos.

*Resultado obtenido o actual* – Lo que realmente sucedió, puede o no coincidir con lo esperado, si no coincide hemos detectado un error o bug.

*Severidad* – Cuan grave s el defecto que encontramos: bloqueado, crítico, alto, medio, bajo o trivial.

*Prioridad* – Qué tan rápido debe solucionarse el defecto: alta, media, baja.

*Estado del defecto* – Nuevo, diferido, duplicado, rechazado, asignado, ne proceso, corregido, en espera de verificación, en verificación, verificado, reabierto y cerrado.

*Referencias* – Link al caso de prueba con el cual encontramos el error.

*Imagen* – Opcional, captura de pantalla del error.

**Módulo 2 - Diseño e implementación de la prueba**

**SEMANA 2**

**CLASE 4 – DISEÑO DE LA PRUEBA**

**Caso de prueba**

Conjunto de acciones que se ejecutan para verificar una característica o funcionalidad particular de una aplicación de software. Documento escrito que proporciona información escrita sobre qué y cómo probar. Es por ello que es de vital importancia en el mundo de la calidad.

Características de un buen caso de pruebas.

1. No asumir – No asumir la funcionalidad y las características de la aplicación mientras se prepara el caso de prueba. Se debe ser fiel a los documentos de especificación y ante cualquier duda, hay que consultar.
2. Asegurar la mayor cobertura posible – Escribir casos de prueba para todos los requisitos especificados.
3. Autonomía – El caso de prueba debe generar los mismos resultados siempre, sin importar quien lo pruebe.
4. Evitar la repetición de caos de prueba – Si se necesita un caso de prueba para ejecutar otro, indicar el caso de prueba por su ID.
5. Deben ser simples – Se deben crear casos de prueba que sean lo más simples posibles ya que otra persona que no sea el autor puede ejecutarlos. Utilizar un lenguaje asertivo para facilitar la comprensión y que ejecución sea más rápida.
6. El titulo debe ser fuerte – Solo leyendo el título, cualquier probador debería comprender el objetivo del caso de prueba.
7. Tener en cuenta al usuario final – El objetivo final es crear casos de prueba que cumplan con los requisitos del cliente y que sean fáciles de usar.

¿Qué debe contener un caso de prueba?

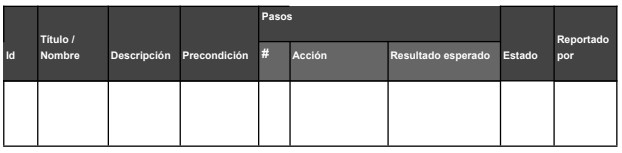
*Identificador* – Numérico o alfanumérico, por lo gral se generan automáticamente.

*Nombre del caso de prueba (conciso)* – Se debe utilizar una nomenclatura que este definida, pero, si no existe, lo recomendable es incluir el nombre de modulo al que corresponde el caso de prueba.

*Descripción* – Debe decir que se va a probar, el ambiente de pruebas y los datos necesarios para ejecutarlo.

*Precondición* – Cosas que deben cumplirse antes de ejecutar el caso de prueba.

*Pasos* – Acciones que deben realizarse para obtener los resultados.

*Resultados esperados* – Lo que le indica al probador cual debería ser la experiencia luego de ejecutar los pasos y determinar si el test fallo o no. 

Testing positivo (+): Son aquellos casos de prueba que validan el flujo normal de un sistema bajo prueba. Es decir, flujos que están relacionados a los requisitos funcionales del sistema bajo prueba.

Testing negativo (-): Son aquellos casos de prueba que validan flujos no contemplados dentro de los requisitos de un sistema bajo prueba.

**Happy Path →**

Es el único camino con el que se prueba una aplicación a través de escenarios de prueba cuidadosamente diseñados, que deberían recorrer el mismo flujo que realiza un usuario final cuando usa la aplicación de manera regular. Generalmente es la primera forma de prueba que se realiza a una aplicación y se incluye en la categoría de prueba positiva. Su propósito no es encontrar defectos, sino ver que un producto o procedimiento funcione cómo ha sido diseñado.

Ventajas:

* Se utiliza para conocer los estándares básicos de la aplicación. Es la primera prueba que se realiza.
* Se utiliza para determinar la estabilidad de la aplicación antes de comenzar con otros niveles de prueba.
* Ayuda a identificar cualquier problema en una etapa temprana y a ahorrar esfuerzos posteriores.

Limitaciones:

* No garantiza la calidad del producto porque el proceso solo utiliza escenarios de prueba positivos.
* Encontrar este camino único requiere un gran conocimiento del uso de la aplicación y necesidades del cliente.

**Casos de uso - Caso de prueba**

Antes de realizar el diseño de los casos de prueba, lo que se debe llevar a cabo es el análisis de los documentos que van a ser la base para la generación de esos casos de prueba. Estos documentos van a asegurar los requisitos del cliente.

Caso de uso: Un caso de uso cuenta la historia de cómo un usuario interactúa con un sistema de software para lograr o abandonar un objetivo. Cada caso de uso puede contener múltiples rutas que el usuario sigue, estos caminos son denominados escenario de caso de uso.

Caso de prueba: Un caso de prueba cubre el software más en profundidad y con más detalle que un caso de uso. Estos incluyen todas las funciones que el programa es capaz de realizar y deben tener en cuenta el uso de todo tipo de datos de entrada/salida, cada comportamiento esperado y todos los elementos de diseño.

¿Cómo combinamos casos de uso con casos de prueba?

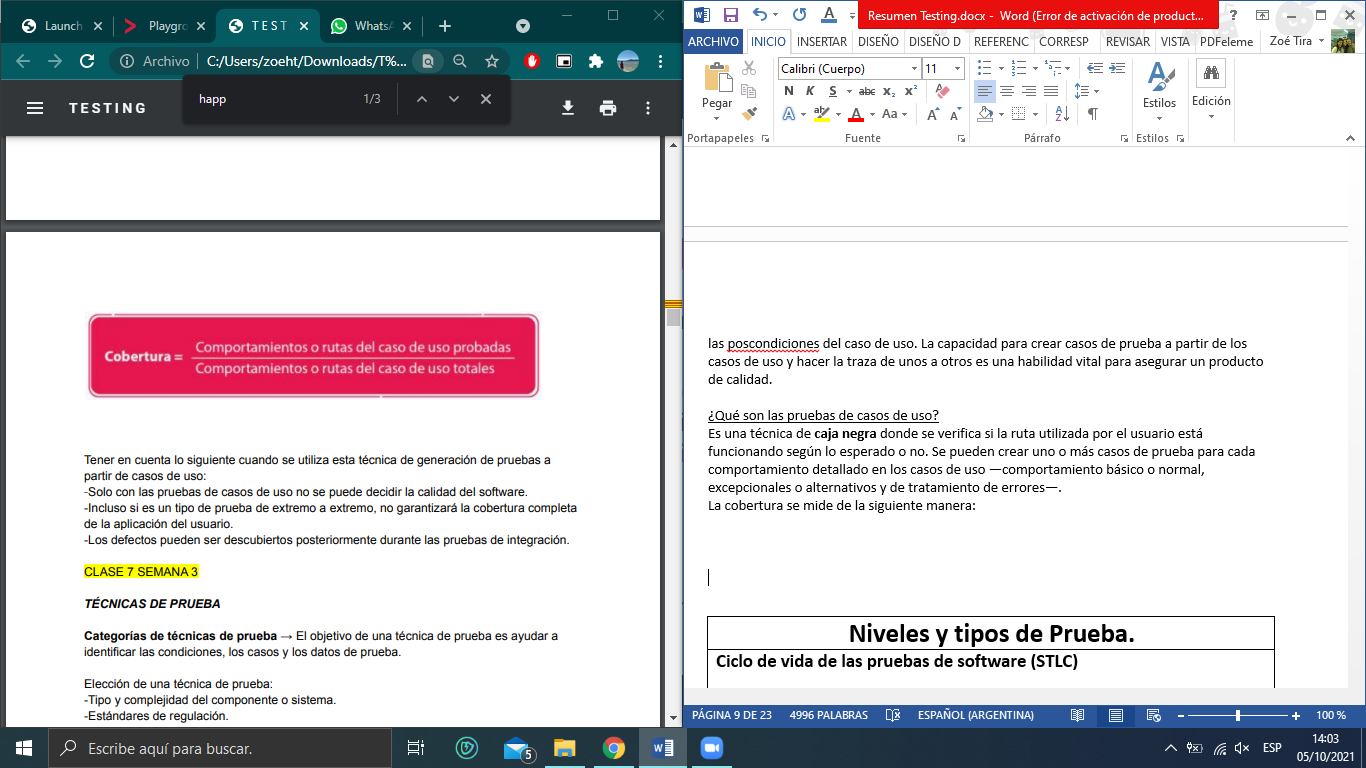
Se puede comenzar escribiendo casos de prueba para el “escenario principal” primero y luego escribirlos para “escenarios alternativos”. Es decir, se escribe uno o más casos de prueba por cada escenario de caso de uso. Los “pasos” de los casos de prueba se obtienen de la secuencia normal o alternativa detallada en los casos de uso.

Tanto el “nombre” como las “precondiciones” del caso de prueba se pueden basar directamente en los mismos campos que existen en el caso de uso. Para el “resultado esperado” de los casos de prueba se debe tener en cuenta la secuencia normal o alternativa y las poscondiciones del caso de uso. La capacidad para crear casos de prueba a partir de los casos de uso y hacer la traza de unos a otros es una habilidad vital para asegurar un producto de calidad.

¿Qué son las pruebas de casos de uso?

Es una técnica de **caja negra** donde se verifica si la ruta utilizada por el usuario está funcionando según lo esperado o no. Se pueden crear uno o más casos de prueba para cada comportamiento detallado en los casos de uso —comportamiento básico o normal, excepcionales o alternativos y de tratamiento de errores—.

La cobertura se mide de la siguiente manera:



Tener en cuenta lo siguiente cuando se utiliza esta técnica de generación de pruebas a partir de casos de uso:

* Solo con las pruebas de casos de uso no se puede decidir la calidad del software.
* Incluso si es un tipo de prueba de extremo a extremo, no garantizará la cobertura completa de la aplicación del usuario.
* Los defectos pueden ser descubiertos posteriormente durante las pruebas de integración.

**CLASE 5 – NIVELES Y TIPOS DE PRUEBA**

**Ciclo de vida de las pruebas de software (STLC)**

No existe un proceso de prueba único 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 desarrollo de software y metodologías de proyecto de uso.
* Niveles y tipos de prueba considerados.
* Riesgos de producto y de proyecto.
* Dominio del negocio.
* Restricciones operativas, incluyendo, pero no limitadas a:

Plazos.

Complejidad.

Tareas principales – aunque no están siempre agrupadas de esta manera en todos los proyectos de software – .

1. Planificación
2. Seguimiento y control
3. Análisis
4. Diseño
5. Implementación
6. Ejecución
7. Conclusión
8. Diseño.

Planificación:

* 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 técnicas, 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 información y proporcionar retroalimentación 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 relación con los criterios de cobertura especificados.
* Determinar si se necesitan más pruebas dependiendo del nivel de cobertura que se debe alcanzar.

*Documento de salida:*

* Informe de avance de la prueba.

Análisis:

Determina “que probar”

*Subactividades:*

* Analizar base de prueba correspondiente al nivel de prueba considerado. -información de diseño e implementación, la implementación del componente o sistema en sí, informes de análisis de riesgo, etc.-
* Identificar defectos de distintos tipos en las bases de prueba – ambigüedades, 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.

Implementación:

Se completan los productos de prueba necesarios para la ejecución de la prueba, incluyendo la secuenciación 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 ejecución.
* 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 ejecución.
* Test suite.

Ejecución:

Durante esta actividad se realiza la ejecución 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 utilizando herramientas.
* Comparar los resultados reales con los resultados esperados.
* Informar sobre los defectos en función de los fallos observados.
* Repetir las actividades de prueba, ya sea como resultado de una acción tomada para una anomalía o como parte de la prueba planificada – retest o prueba de confirmación-.
* Verificar y actualizar la trazabilidad entre la base de prueba, las condiciones de prueba, los casos 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 ejecución de prueba.

Conclusión:

*Subactividades:*

* Comprobar que todos los informes de defecto están 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 posterior reutilización.
* Traspaso de los productos de prueba a otros equipos que podrían beneficiarse con su uso
* Analizar las elecciones aprendidas de las actividades de prueba completadas.
* Utilizar la información recopilada para mejorar la madurez del proceso de prueba.

*Documento de salida:*

* Informe resumen de prueba.
* Lecciones aprendidas.

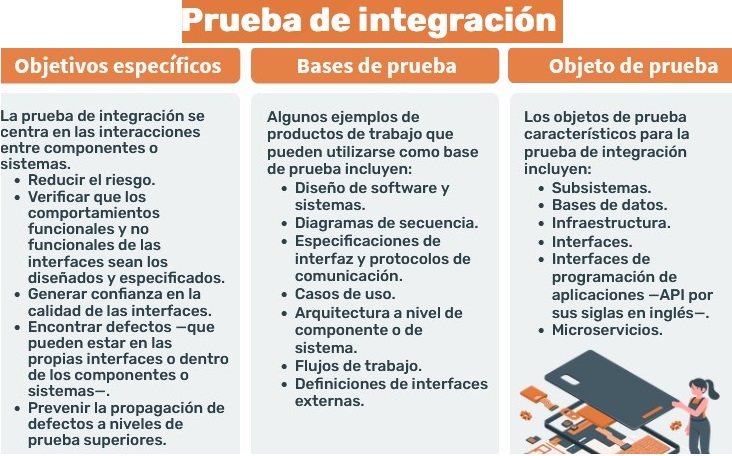
**Niveles de prueba:**

Prueba unitaria o de componente:





Prueba de integración:





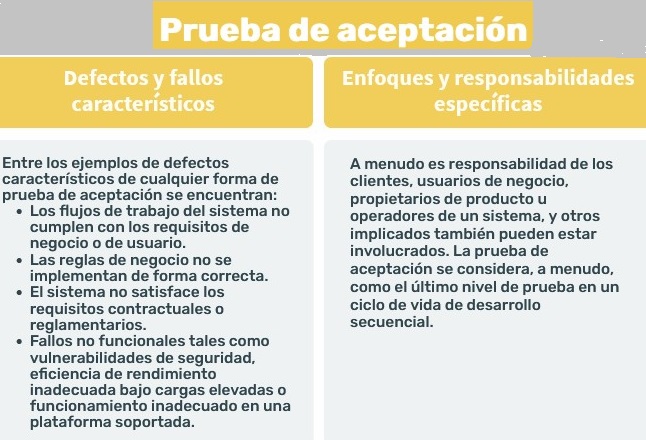
Prueba de sistema:





Prueba de aceptación:





|  |  |
| --- | --- |
| **Desde el análisis a la implementación** | |
| 1. Etapas de procesos de larga duración y secuenciales. | 1. Iteraciones cortas y definidas como sprints. |
| 1. Pruebas estáticas al principio sobre los documentos que son bases de prueba. Las pruebas dinámicas se realizan como una actividad al final, luego de tener la solución codificada. | 1. Pruebas dinámicas y continuas durante la iteración. Los requerimientos son analizados como presentaciones (futuras) y se crean escenarios de prueba usando un enfoque de desarrollo guiado por comportamiento (BDD). |
| 1. Se realiza una prueba de ambigüedad sobre los requerimientos. | 1. Se dividen las prestaciones en historias de usuarios (user stories) y se modelan escenarios de prueba para esas historias. |
| 1. Se dividen los requerimientos en historias pequeñas. | 1. Se escriben los unit test y luego se codifica la solución usando un enfoque de desarrollo guiado por pruebas (TDD). |
| 1. Se desarrolla el código en base a la documentación de diseño. Luego de tener el código se desarrollan los unit test. | 1. Se enfoca en la ligera documentación priorizando lo que entregue valor al cliente. |
| 1. Existe mayor documentación de cada proceso. | 1. Se ejecutan pruebas continuamente y desde etapas tempranas. Toma relevancia la aplicación de integración continua (CI) y distribución o desarrollo continuo (CD). |
| 1. Las pruebas se ejecutan luego de tener el código generado de todo un requerimiento o todo el sistema. | 1. Adquieren gran protagonismo las pruebas automatizadas. |
| 1. las pruebas son generalmente manuales. | 1. Además, la integración continua permite tener una retroalimentación activa. |

**Tipos de pruebas:**

Un tipo de prueba es un grupo de actividades de pruebas destinadas a probar las características específicas de un sistema de software, o de una parte de un sistema, basadas en objetivos de pruebas específicas.

Dichos objetivos pueden incluir:

1. Evaluar las características de calidad funcional tales como la completitud, corrección y pertinencia.
2. Evaluar características no funcionales de calidad, tales como la fiabilidad, eficiencia de desempeño, seguridad, confiabilidad y usabilidad.
3. Evaluar si la estructura o arquitectura del componente o sistema es correcta, completa y según lo especificado.
4. Evaluar los efectos de los cambios, tales como confirmar que los defectos han sido corregidos (prueba de confirmación) y buscar cambios no deseados en el comportamiento que resulten de los cambios en el software o en el entorno (prueba de regresión)

Prueba funcional:

|  |  |
| --- | --- |
| **Definición** | La prueba funcional de un sistema incluye pruebas que evalúan las funciones que el sistema debe realizar. Las funciones describen qué hace el sistema. |
| **Implementación** | La prueba funcional observa el comportamiento del software. |
| **Niveles de Prueba** | Se pueden realizar pruebas funcionales en todos los niveles de prueba. |
| **Alcance** | Los requisitos funcionales pueden estar detallados en los siguientes documentos: especificaciones de requisitos del negocio, épicas, historias de usuarios, casos de uso y/o especificaciones funcionales. |
| **Cobertura** | La cobertura funcional es la medida en que algún tipo de elemento funcional ha sido practicado por pruebas, y se expresa como un porcentaje del tipo o tipos de elementos cubiertos. |

Prueba no funcional:

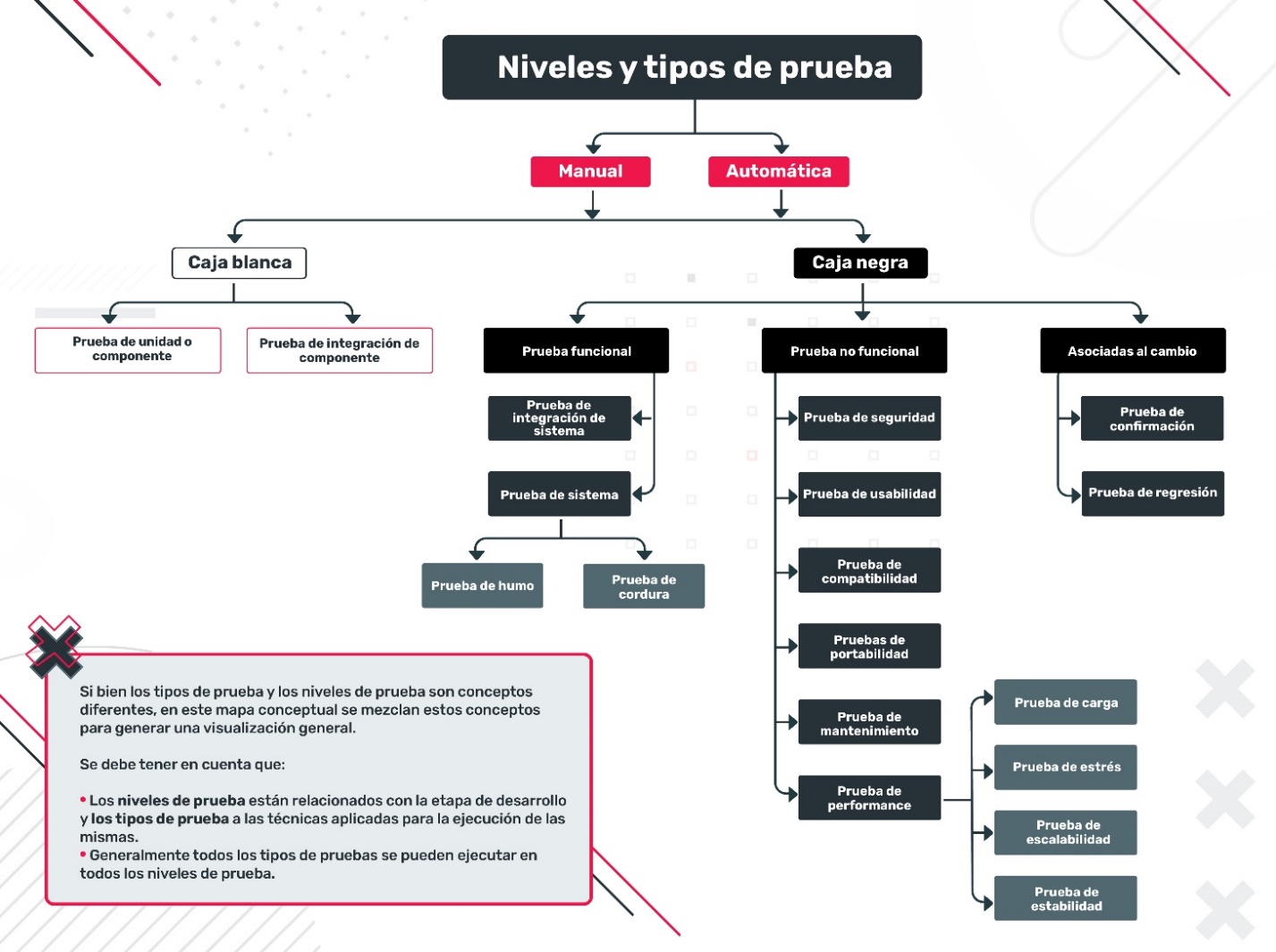
|  |  |
| --- | --- |
| **Definición** | La prueba no funcional prueba “**cómo de bien**” se comporta el sistema. |
| **Implementación** | El diseño y ejecución de la prueba no funcional puede implicar competencias y conocimientos especiales, como el conocimiento de las debilidades inherentes a un diseño o tecnología -por ejemplo: vulnerabilidades de seguridad asociadas con determinados lenguajes de programación-. |
| **Niveles de Prueba** | Se pueden realizar pruebas no funcionales en todos los niveles de prueba |
| **Alcance** | La prueba no funcional del sistema evalúa características como la usabilidad, la eficiencia del desempeño o la seguridad. |
| **Cobertura** | La cobertura no funcional es la medida en que algún tipo de elemento no funcional ha sido practicado por pruebas,  y se expresa como un porcentaje del tipo o tipos de elementos cubiertos. |

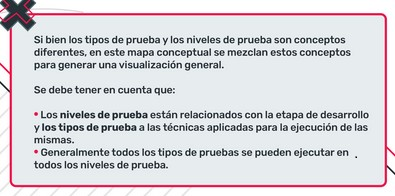
Pruebas estructurales:

|  |  |
| --- | --- |
| **Definición** | Estas pruebas están basadas en la estructura interna del sistema o en su  implementación. La estructura interna puede incluir código, arquitectura, flujos de trabajo y/o flujos de datos  dentro del sistema |
| **Implementación** | El diseño y la ejecución de este tipo de pruebas pueden implicar competencias o conocimientos  especiales, como la forma en que se construye el código, cómo se almacenan los datos, y cómo utilizar las herramientas de cobertura e interpretar correctamente sus resultados. |
| **Niveles de Prueba** | Se puede realizar en el nivel de componente y de integración. |
| **Alcance** | En el nivel de prueba de integración de componentes, la prueba estructural pueden basarse en la arquitectura del sistema, como las interfaces entre componentes |
| **Cobertura** | La cobertura  estructural es la medida en que algún tipo de elemento estructural ha sido practicado mediante pruebas, y  se expresa como un porcentaje del tipo de elemento cubierto. |

Prueba asociada al cambio:

|  |  |
| --- | --- |
| **Definición** | Existen 2 tipos de prueba relacionadas al cambio:   * **Prueba de confirmación**: Una vez corregido un defecto, el software se puede probar con todos los  casos de prueba que fallaron debido al defecto, que se deben volver a ejecutar en la nueva versión  de software. El  objetivo de una prueba de confirmación es confirmar que el defecto original se ha solucionado de  forma satisfactoria. * **Prueba de regresión:** Es posible que un cambio hecho en una parte del código, ya sea una  corrección u otro tipo de cambio, pueda afectar accidentalmente el comportamiento de otras partes  del código, ya sea dentro del mismo componente, en otros componentes del mismo sistema, o  incluso en otros sistemas. La prueba de regresión implica la  realización de pruebas para detectar estos efectos secundarios no deseados. |
| **Implementación** | Especialmente en los ciclos de vida de desarrollo iterativos e incrementales (por ejemplo, Agile), las nuevas  características, los cambios en las características existentes y la refactorización del código dan como  resultado cambios frecuentes en el código, lo que también requiere pruebas asociadas al cambio. |
| **Niveles de Prueba** | La prueba de confirmación y la prueba de regresión se realizan en todos los niveles de prueba. |
| **Cobertura** | Los juegos de prueba de regresión se ejecutan muchas veces y generalmente evolucionan lentamente,  por lo que la prueba de regresión es un fuerte candidato para la automatización. |





**Ejemplos de tipos de prueba**

Los siguientes ejemplos están basados en una aplicación bancaria.

**Pruebas funcionales:**

*Prueba de componente:* las pruebas se diseñan con base en la forma en que un componente debe calcular el interés a pagar por un préstamo.

*Prueba de integración de componentes:* las pruebas se diseñan en función de cómo la información de la cuenta capturada en la interfaz de usuario se transfiere a la lógica de negocio.

*Prueba de sistema:* las pruebas se diseñan sobre la base de cómo los titulares de cuentas pueden solicitar una línea de crédito sobre sus cuentas corrientes.

*Prueba de integración de sistemas:* las pruebas se diseñan en función de cómo el sistema utiliza un microservicio externo para comprobar la calificación crediticia del titular de una cuenta.

*Prueba de aceptación:* las pruebas se diseñan con base en la forma en que el empleado del banco tramita la aprobación o rechazo de una solicitud de crédito.

**Pruebas estructurales:**

*Prueba de componente:* pruebas están diseñadas para lograr una cobertura completa de sentencia y decisión para todos los componentes que realizan cálculos financieros.

*Prueba de integración de componentes:* las pruebas están diseñadas para evaluar cómo cada pantalla de la interfaz del navegador pasa datos a la siguiente pantalla y a la lógica de negocio.

*Prueba de sistema:* las pruebas están diseñadas para cubrir las secuencias de páginas web que pueden ocurrir durante una solicitud de línea de crédito (workflow).

*Prueba de integración de sistemas:* las pruebas están diseñadas para evaluar todos los tipos de consulta posibles que se envían al microservicio de calificación crediticia.

*Prueba de aceptación:* las pruebas están diseñadas para cubrir todas las estructuras de archivos de datos financieros soportados y rangos de valores para transferencias de banco a banco.

**Pruebas asociadas al cambio:**

*Prueba de componente:* se construyen pruebas de regresión automatizadas para cada componente y se incluyen dentro del marco de integración continua.

*Prueba de integración de componentes:* las pruebas están diseñadas para confirmar la corrección de defectos relacionados con la interfaz a medida que las correcciones se registran en el repositorio de código.

*Prueba de sistema:* las pruebas de un flujo de trabajo dado se ejecutan de nuevo si cambia alguna pantalla de ese flujo de trabajo.

*Prueba de integración de sistemas:* las pruebas de la aplicación que interactúa con el microservicio de calificación de crédito se vuelven a ejecutar diariamente como parte del despliegue continuo de ese microservicio.

*Prueba de aceptación:* todas las pruebas que han fallado previamente se vuelven a ejecutar después de que se haya corregido un defecto encontrado en la prueba de aceptación.

**SEMANA 3**

**CLASE 7 – TÉCNICAS DE PRUEBA**

**Técnicas de prueba**

El objetivo de una técnica de prueba es ayudar a identificar las condiciones, los casos y los datos de prueba.

Elección de una técnica de prueba:

* Tipo y complejidad del componente o sistema.
* Estándares de regulación.
* Requisitos del cliente o contractuales.
* Clases y niveles de riesgo.
* Objetivo de la prueba.
* Documentación disponible.
* Conocimientos y competencias del probador.
* Modelo del ciclo de vida del software.
* Tiempo y presupuesto.

Clasificación de las técnicas de prueba:

*Técnicas de caja negra:* Se basan en el comportamiento extraído del análisis de los documentos que son base de prueba (documentos de requisitos formales, casos de uso, historias de usuario, etc). Son aplicables tanto para pruebas funcionales como no funcionales. Se concentran en las entradas y salidas sin tener en cuenta la estructura interna.

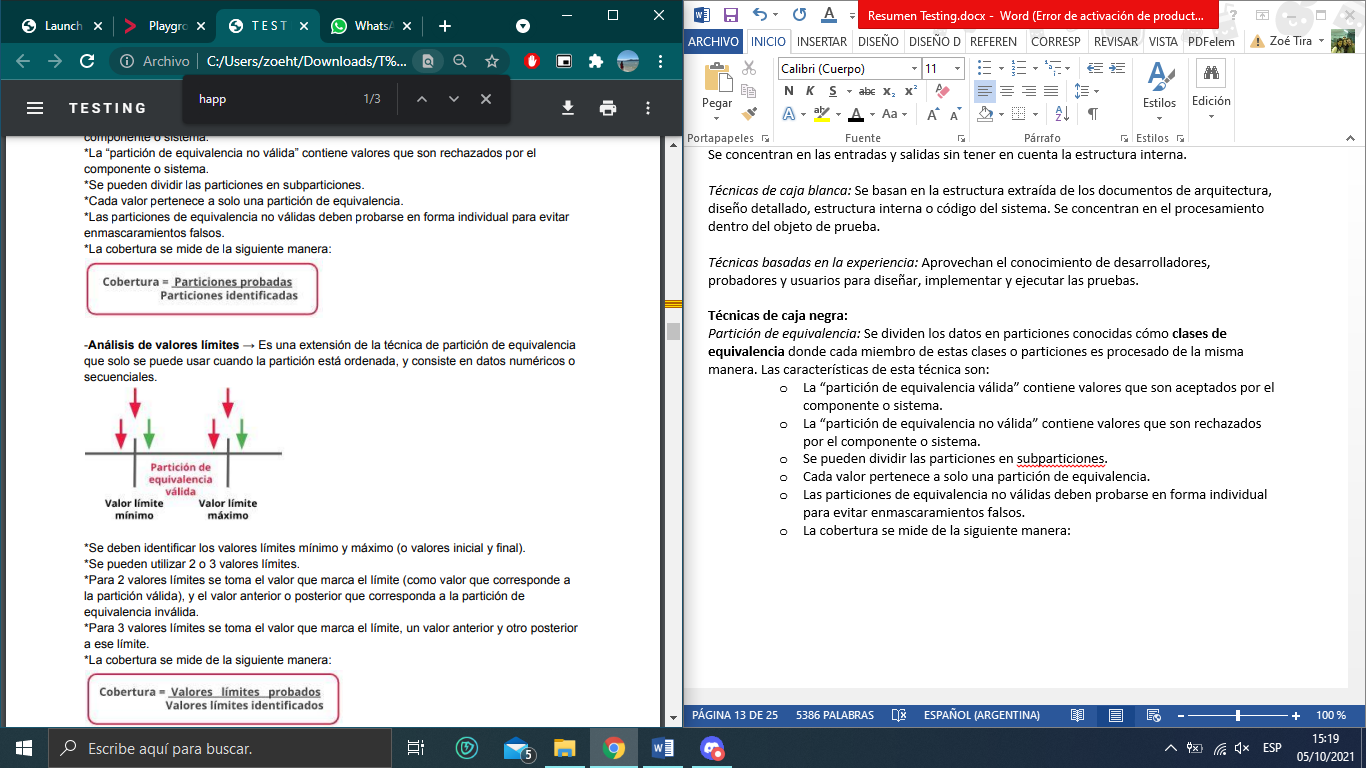
*Técnicas de caja blanca:* Se basan en la estructura extraída de los documentos de arquitectura, diseño detallado, estructura interna o código del sistema. Se concentran en el procesamiento dentro del objeto de prueba.

*Técnicas basadas en la experiencia:* Aprovechan el conocimiento de desarrolladores, probadores y usuarios para diseñar, implementar y ejecutar las pruebas.

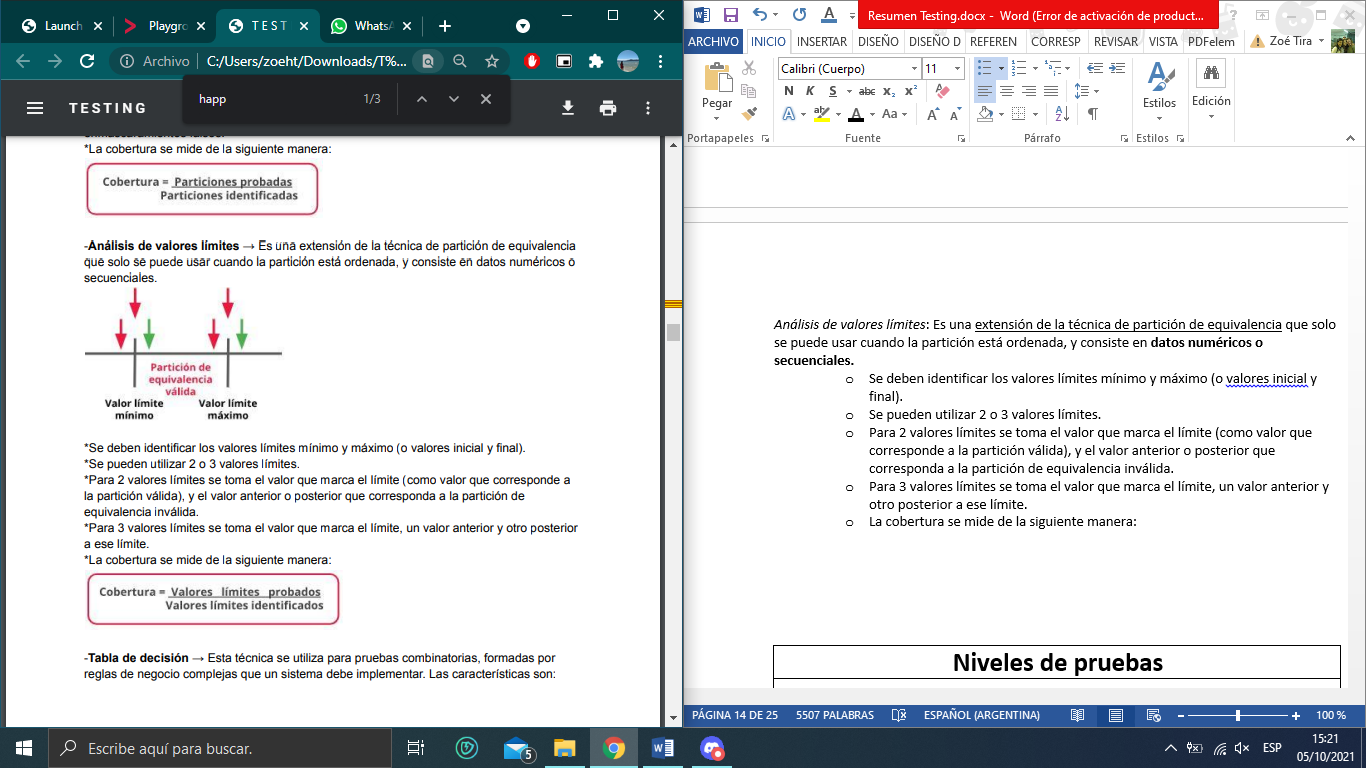
**Técnicas de caja negra:**

*Partición de equivalencia:* Se dividen los datos en particiones conocidas cómo **clases de equivalencia** donde cada miembro de estas clases o particiones es procesado de la misma manera. Las características de esta técnica son:

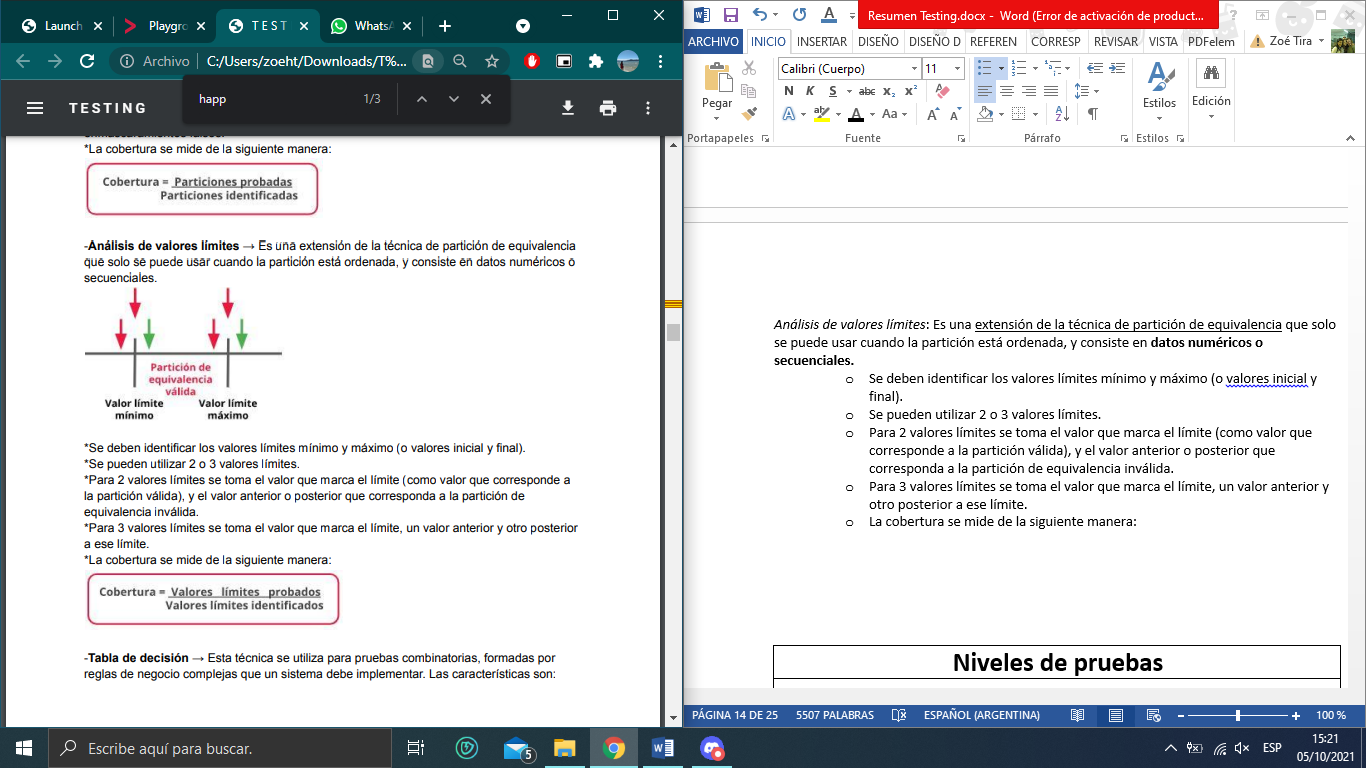
* La “partición de equivalencia válida” contiene valores que son aceptados por el componente o sistema.
* La “partición de equivalencia no válida” contiene valores que son rechazados por el componente o sistema.
* Se pueden dividir las particiones en subparticiones.
* Cada valor pertenece a solo una partición de equivalencia.
* Las particiones de equivalencia no válidas deben probarse en forma individual para evitar enmascaramientos falsos.
* La cobertura se mide de la siguiente manera:



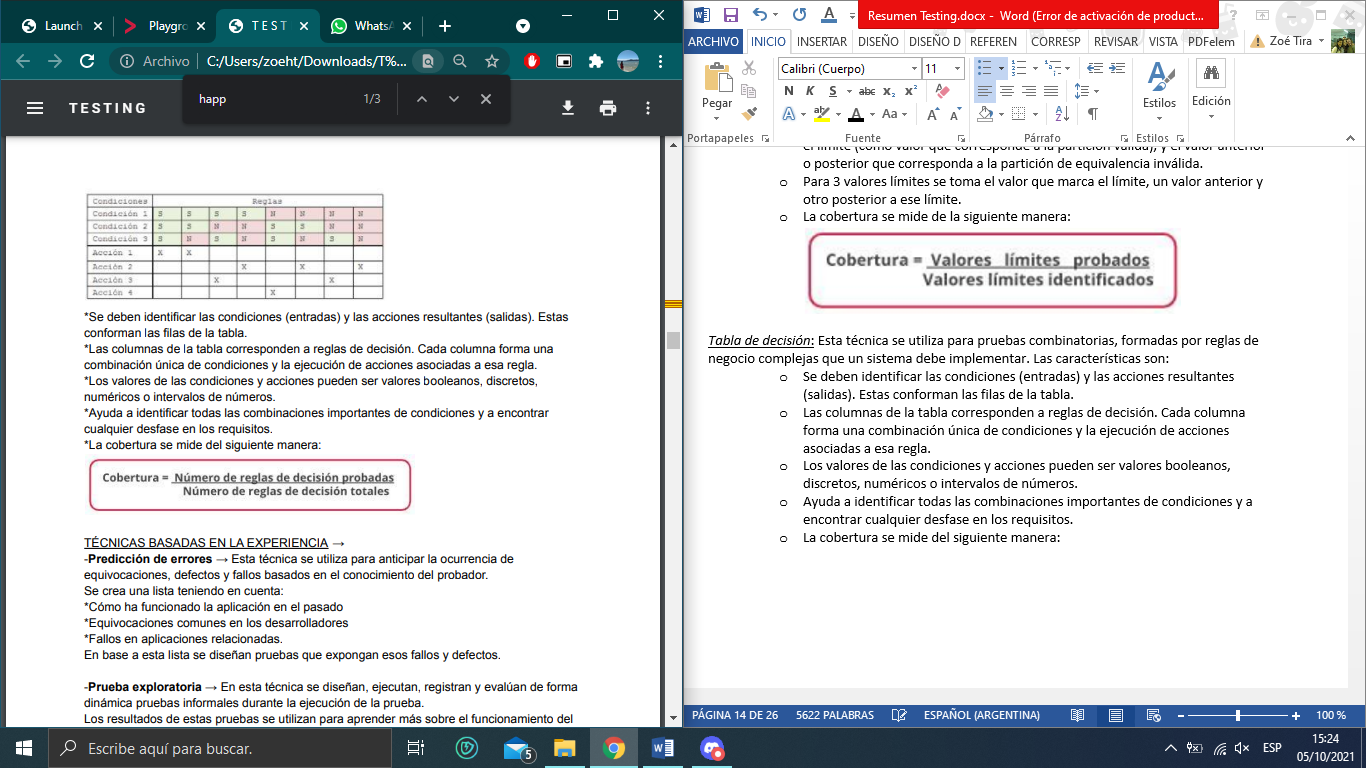
Ejemplo: colores RGB en formato **String** minúscula.

*Análisis de valores límites*: Es una extensión de la técnica de partición de equivalencia que solo se puede usar cuando la partición está ordenada, y consiste en **datos numéricos o secuenciales.**

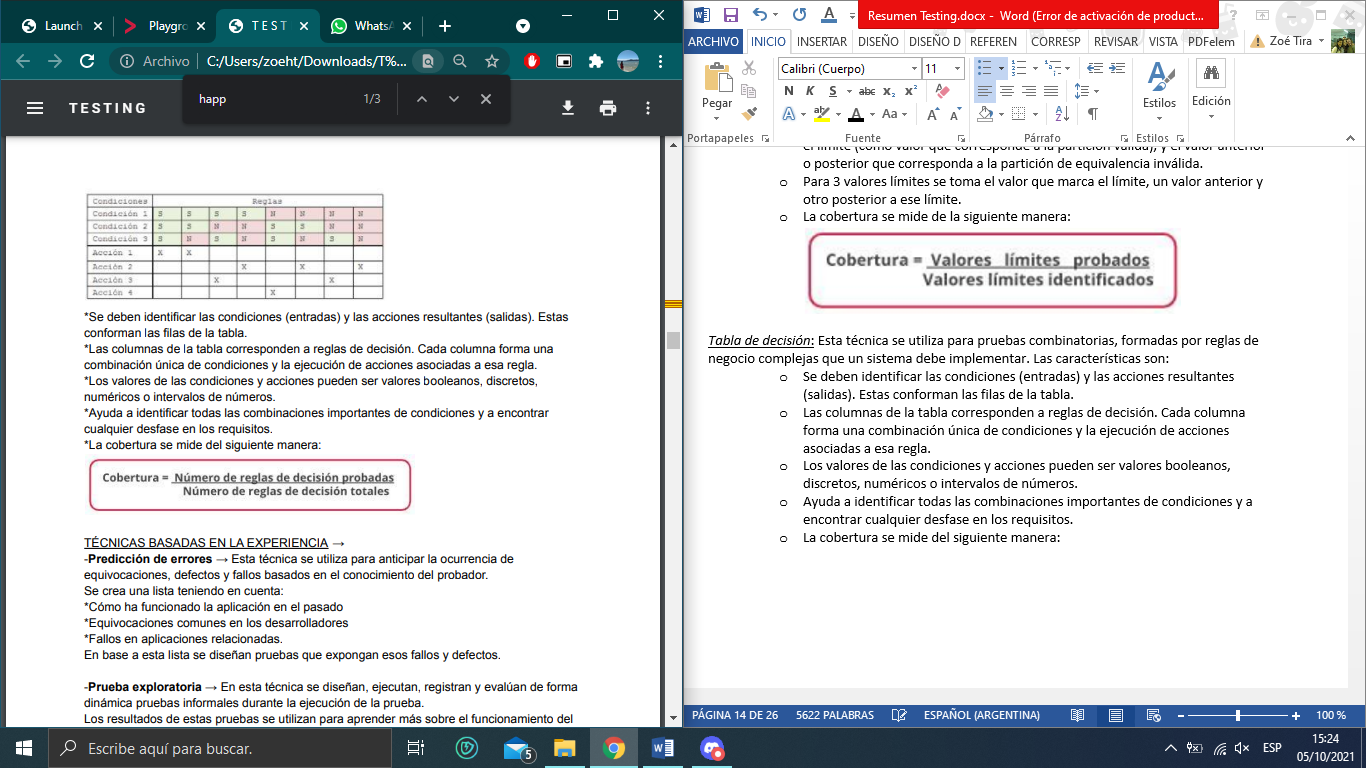
* Se deben identificar los valores límites mínimo y máximo (o valores iniciales y finales).
* Se pueden utilizar 2 o 3 valores límites.
* Para 2 valores límites se toma el valor que marca el límite (como valor que corresponde a la partición válida), y el valor anterior o posterior que corresponda a la partición de equivalencia inválida.
* Para 3 valores límites se toma el valor que marca el límite, un valor anterior y otro posterior a ese límite.
* La cobertura se mide de la siguiente manera:



Ejemplo: dato de entrada corresponde a un número que nos indica el día de la semana (1-7)

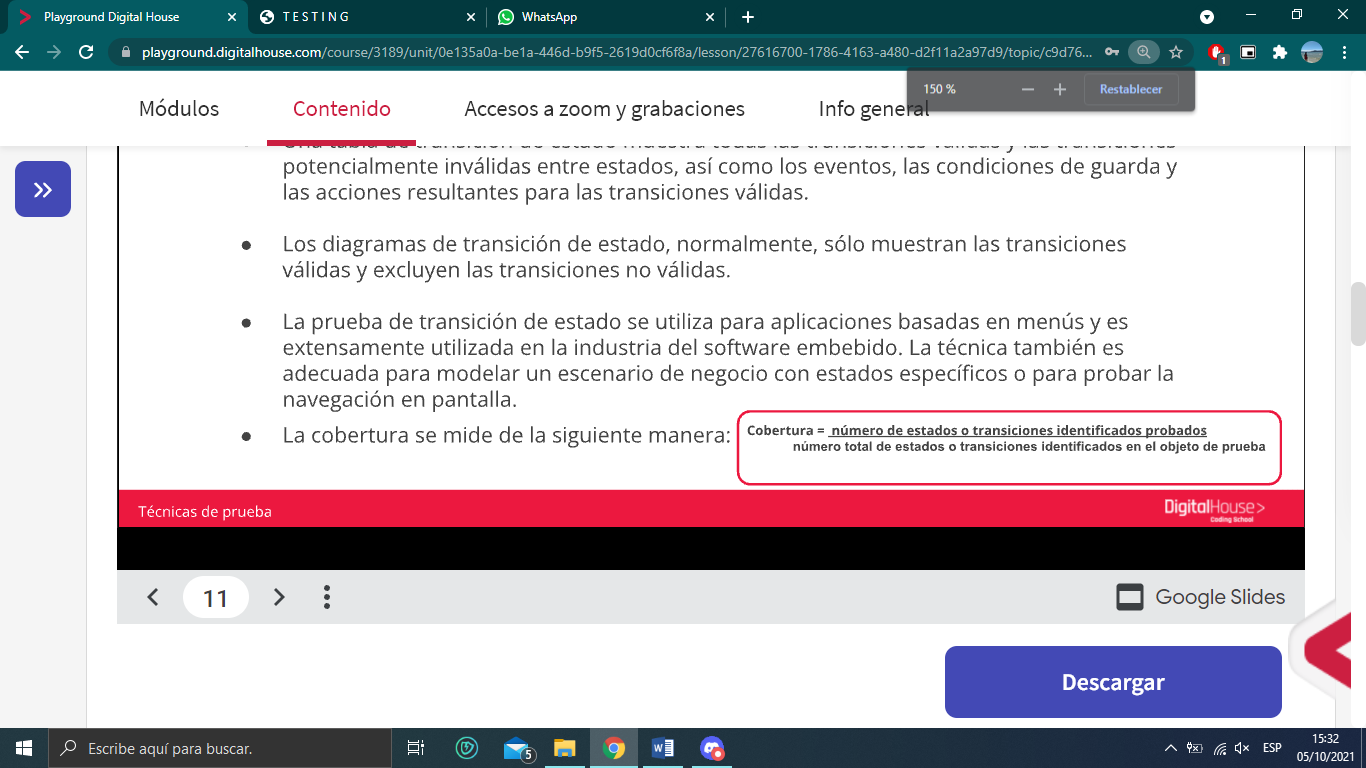
*Tabla de decisión*: Esta técnica se utiliza para pruebas combinatorias, formadas por reglas de negocio complejas que un sistema debe implementar. Las características son:

* Se deben identificar las condiciones (entradas) y las acciones resultantes (salidas). Estas conforman las filas de la tabla.
* Las columnas de la tabla corresponden a reglas de decisión. Cada columna forma una combinación única de condiciones y la ejecución de acciones asociadas a esa regla.
* Los valores de las condiciones y acciones pueden ser valores booleanos, discretos, numéricos o intervalos de números.
* Ayuda a identificar todas las combinaciones importantes de condiciones y a encontrar cualquier desfase en los requisitos.
* La cobertura se mide del siguiente manera:



*Transición de estados:* Muestra los posibles estados del software, así como la forma en que el software entra, sale y realiza las transiciones entre estados. Las características de esta técnica son:

* Una tabla de transición de estado muestra todas las transiciones válidas y las transiciones potencialmente invalidas entre estados, así como los eventos, las condiciones de guarda y las acciones resultantes para las transiciones válidas.
* Los diagramas de transición de estado, normalmente, sólo muestran las transiciones válidas y excluyen las transiciones no válidas.
* La prueba de transición de estado se utiliza para aplicaciones basadas en menús y es extensamente utilizada en la industria del software embebido. La técnica también es adecuada para modelar un escenario de negocio con estados específicos o para probar la navegación en pantalla.
* La cobertura se mide de la siguiente manera:



**Técnicas basadas en la experiencia:**

*Predicción de errores:* Esta técnica se utiliza para anticipar la ocurrencia de equivocaciones, defectos y fallos basados en el conocimiento del probador. Se crea una lista teniendo en cuenta:

* Cómo ha funcionado la aplicación en el pasado.
* Equivocaciones comunes en los desarrolladores.
* Fallos en aplicaciones relacionadas.

En base a esta lista se diseñan pruebas que expongan esos fallos y defectos.

*Prueba exploratoria:* En esta técnica se **diseñan, ejecutan, registran** y evalúan de forma dinámica pruebas informales durante la ejecución de la prueba. Los resultados de estas pruebas se utilizan para aprender más sobre el funcionamiento del componente o sistema. Generalmente se utilizan para complementar otras técnicas formales o cuando las especificaciones son escasas, inadecuadas o con restricciones de tiempo.

*Prueba basada en listas de comprobación:* En esta técnica se **diseñan, implementan y ejecutan** casos de prueba que cubren las condiciones que se encuentran en una lista de comprobación definida. Se crean basadas en la experiencia y conocimiento de lo que el probador cree que es importante para el usuario y se utilizan debido a la falta de casos de prueba detallados. Durante la ejecución puede haber cierta variabilidad, dependiendo de quién ejecuta la prueba y condiciones del contexto. Esto da lugar a una mayor cobertura. Se utiliza tanto en pruebas funcionales cómo no funcionales.

**CLASE 8 – IMPLEMENTACIÓN Y EJECUCIÓN DE LA PRUEBA**

Durante la ejecución de las pruebas, los conjuntos de pruebas se ejecutan luego del despliegue de cambios en los ambientes de prueba como parte del desarrollo planificado dentro de un sprint. La ejecución de pruebas incluye las siguientes actividades principales:

1. Registrar los identificadores y las versiones de los elementos u objetos de prueba, las herramientas de prueba y los productos de prueba.
2. Ejecutar pruebas de forma manual o utilizando herramientas de ejecución de pruebas.
3. Comparar resultados reales con resultados esperados.
4. Analizar las anomalías para establecer sus causas probables.
5. Informar sobre los defectos en función de los fallos observados.
6. Registrar el resultado de la ejecución de la prueba.
7. Repetir las actividades de prueba, ya sea como resultado de una acción tomada para una anomalía o como parte de la prueba planificada.

**Creación de suites**

Durante el desarrollo de software el momento más propenso para la inclusión involuntaria de fallas suele ser cuando se introducen nuevas funcionalidades en la aplicación. Para atacar al inconveniente vemos dos conjuntos o juegos de casos de prueba muy utilizados a la hora del despliegue de nuevas funcionalidades. Son una colección de casos de prueba con un fin específico.

*Pruebas de humo:* Cubren la funcionalidad principal de un componente o sistema. El objetivo es asegurar que las funciones cruciales de un sistema funcionen, pero sin preocuparse por los detalles finos. Con pruebas sencillas y que demandan poco tiempo verificamos que funcionan correctamente ciertos caminos de la aplicación. Generalmente se eligen sólo un conjunto de funcionalidades significativas de la aplicación.

*Pruebas de regresión:* Permiten asegurarnos que los cambios no han dañado las interfaces, los componentes o los sistemas existentes. Además se buscan cambios no deseados en el comportamiento que resulten de cambios en el software o en el entorno. Dentro de un proyecto de automatización lo ideal es comenzar con las pruebas de regresión ya que éstas se ejecutan muchas veces y generalmente evolucionan lentamente. La correcta aplicación de un proceso de prueba durante el despliegue implicaría:

1. Ejecutar las pruebas de humo, y una vez confirmada la ejecución exitosa de estas pruebas estamos en condiciones de regresionar nuestro sistema bajo prueba.
2. Ejecutar pruebas de regresión.

¿Por qué es importante ejecutar estas pruebas?

Las pruebas de humo nos ayudan a confirmar que luego de un despliegue las funcionalidades principales no sufrieron fallas. Por otro lado, las pruebas de regresión terminan de confirmar que todo lo que antes del despliegue funcionaba, sigue funcionando de la misma forma. Es probable que estas dos pruebas se complementen con pruebas específicas relacionadas a las nuevas funcionalidades que se desplegarán.

**Módulo 3 - Análisis y Planificación de la prueba**

**SEMANA 4**

**CLASE 10 – PRUEBAS ESTÁTICAS Y DINÁMICAS**

**Pruebas estáticas y dinámicas**

Las pruebas son una combinación de múltiples actividades del ciclo de vida del software relacionadas con la planificación, el diseño y la evaluación del producto de software, con el objetivo de encontrar los defectos y determinar si el software cumple o no con los requisitos especificados.

Las pruebas estáticas y dinámicas tienen el objetivo de proporcionar una evaluación de calidad de los productos de trabajo e identificar defectos en forma temprana. Estas se complementan entre sí y nos permiten entregar un software con la mejor calidad posible.

**Prueba estática:**

Se basa en la evaluación manual de los productos de trabajo, es decir revisiones, o en la evaluación basada en herramientas del código u otros productos de trabajo, es decir análisis estático. Este tipo de pruebas no requieren la ejecución del software que se está probando.

Se utilizan para examinar cualquier producto de trabajo, por ejemplo:

* Especificaciones, requisitos de negocio, funcionales y de seguridad.
* Épicas, historias de usuarios y criterios de aceptación.
* Especificaciones de arquitectura y diseño.
* Código.
* Producto de prueba: planes, casos, procedimiento y guiones de prueba.
* Manuales de usuario.
* Contratos, planes de proyecto, calendarios y presupuestos.

Ventajas:

Cuando se aplica al principio del ciclo de vida del desarrollo de software, permite la detección temprana de defectos. Esto genera una reducción de costos y tiempo de desarrollo y prueba.

Si el defecto se encuentra en las pruebas dinámicas, solucionarlo requerirá de un cambio de código, realizar una prueba de confirmación y luego incluir los mismos en pruebas de regresión, además de los cambios de toda la documentación asociada.

Defectos encontrados con pruebas estáticas:

* Defectos en los requisitos (inconsistencias, ambigüedades, etc.)
* Defectos de diseño (estructura de base de dato ineficiente, alto acoplamiento, etc.)
* Defectos de codificación (variables con valores no definidos, código inalcanzable o duplicado, etc.)
* Desviaciones con respecto a estándares (falta de uso de estándares de codificación)
* Especificaciones de interfaz incorrectas (unidades de medida diferente, etc.)
* Vulnerabilidades de seguridad (susceptibilidad a desbordamiento de la memoria intermedia)
* Diferencias o inexactitudes en la trazabilidad o cobertura de la base de prueba (falta de pruebas para un criterio de aceptación)
* Defectos de mantenibilidad (mala reutilización de componentes, modularización inadecuada, etc.)

**Prueba dinámica:**

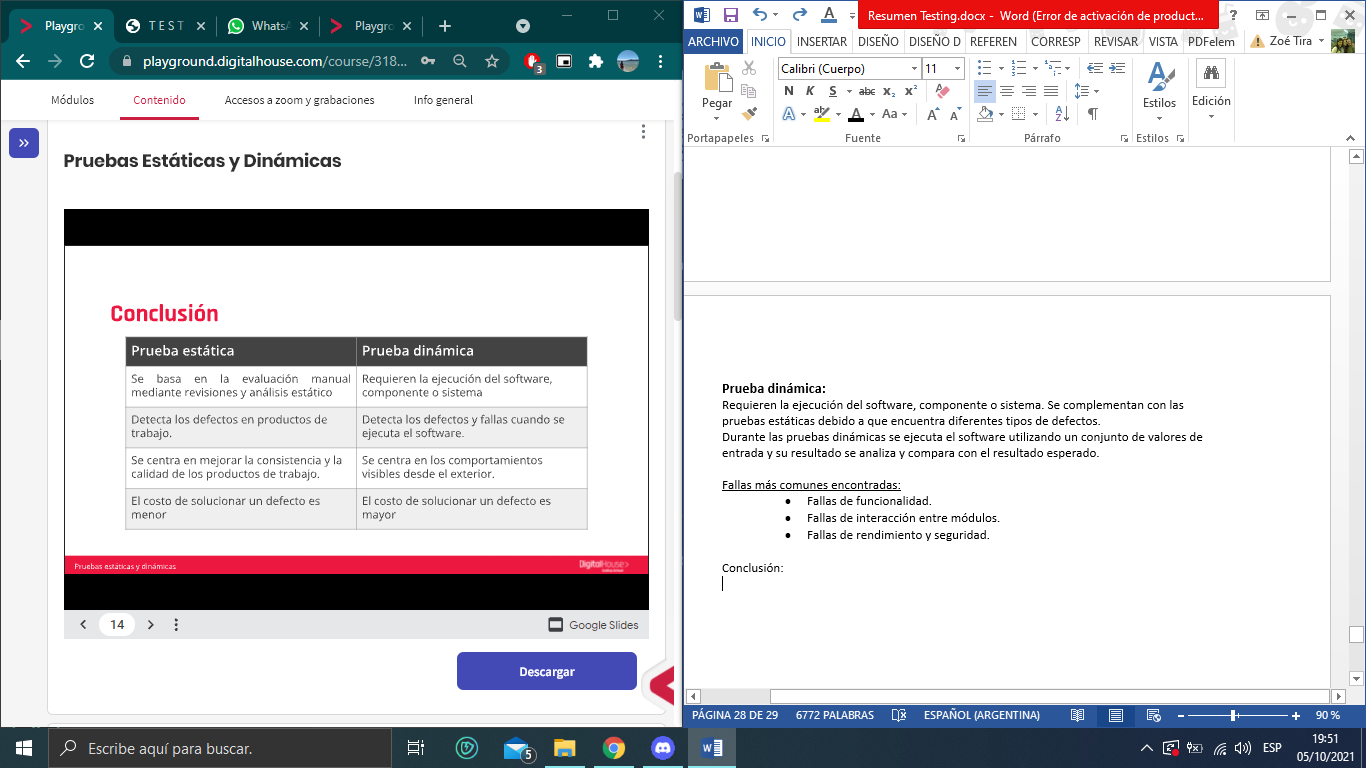
Requieren la ejecución del software, componente o sistema. Se complementan con las pruebas estáticas debido a que encuentra diferentes tipos de defectos.

Durante las pruebas dinámicas se ejecuta el software utilizando un conjunto de valores de entrada y su resultado se analiza y compara con el resultado esperado.

Fallas más comunes encontradas:

* Fallas de funcionalidad.
* Fallas de interacción entre módulos.
* Fallas de rendimiento y seguridad.

**Conclusión:**



**Proceso de revisión**

Dentro de las pruebas estáticas, una forma de detectar errores es mediante un proceso de revisión.

Las revisiones consisten en examinar cuidadosamente un producto de trabajo con el principal objetivo de encontrar y remover errores. Pueden ser realizadas por una o más personas.

Las revisiones pueden ser:

* Revisiones formales: Tienen roles definidos, siguen un proceso establecido y deben ser documentadas.
* Revisiones Informales: No siguen un proceso definido y no son documentadas formalmente.

El grado de formalidad del proceso de revisión está relacionado con factores, como el modelo del ciclo de vida del desarrollo del software, la madurez del proceso de desarrollo, la complejidad del producto del trabajo que se debe revisar, cualquier requisito legal y/o la necesidad de un rastro de auditoría.

**Componentes en el proceso de revisiones formales:**

Roles:

* *Autor:* Creador del producto de trabajo bajo revisión y quien corrige los defectos, en caso de ser necesario.
* *Dirección:* Planifica y controla las revisiones.
* Facilitador: Asegura el funcionamiento efectivo de las reuniones de revisiones y las modera.
* *Líder de revisión:* Asume la responsabilidad general de la revisión; decide quiénes estarán involucrados y organiza cuándo y dónde se llevará a cabo.
* *Revisores:* Identifican posibles defectos en el producto de trabajo bajo revisión; pueden representar diferentes perspectivas —por ejemplo, probador, programador, usuarios, operador, analista de negocio, experto en usabilidad—.
* *Escriba:* Recopila los posibles defectos encontrados, puntos abiertos y decisiones en las revisiones.

Tipos:

* *Guiada | Walkthrough:* Dirigida por el autor del producto de trabajo, el escriba es obligatorio. El uso de listas de comprobación y la preparación individual previa son opcionales. Puede variar de informal a formal. Puede o no haber documentación de defectos potenciales.
* *Técnica:* Se realiza entre pares técnicos del autor. Si hay una reunión, debe haber una preparación individual previa. Tienen que existir un facilitador y un escriba obligatoriamente, quien idealmente no será el autor. Se elaboran registros de defectos potenciales e informes de revisión.
* *Inspecciones:* Los resultados se documentan formalmente. Requieren preparación individual, tienen roles definidos: autor, director, facilitador, escriba, revisores y líder de revisión, pueden incluir también un lector dedicado. El autor no puede actuar como facilitador, líder de revisión, lector o escriba. Se hace uso de listas de comprobación. Se elaboran registros de defectos potenciales e informes de revisión.

Técnicas:

* *Ad hoc*: Los revisores leen el producto de trabajo de forma secuencial y a medida que van identificando los defectos, los van documentando, recibiendo poca o ninguna orientación en el proceso.
* *Basada en escenarios y ensayos:* Fundada en el uso esperado del producto de trabajo descrito en un documento, por ejemplo, en un caso de uso.
* *Basada en listas de comprobación*: Los revisores detectan defectos a partir de un conjunto de preguntas basadas en defectos potenciales, producto de la experiencia del autor de la lista.
* *Basada en roles:* Los revisores evalúan el producto de trabajo desde la perspectiva de usuarios experimentados, inexpertos, adultos, niños o roles específicos en la organización, como un administrador de usuarios, de sistemas o un probador del rendimiento.
* *Basada en perspectiva:* Esta técnica es similar a una revisión basada en roles, los revisores adoptan los diferentes puntos de vista del usuario final, del personal de marketing, del diseñador, del probador o del personal de operaciones.

Actividades:

* *Planificar:* Definir el alcance, establecer objetivos, roles, tiempo y plazos. Definir y comprobar el cumplimiento de criterios de entrada y salida para revisiones más formales.
* *Iniciar revisión:* Distribuir el material e instruir a los participantes.
* *Revisión individual (preparación individual):* Revisión del material y tomar notas de los defectos encontrados.
* *Comunicar y analizar:* Comunicar defectos a los responsables.
* *Corregir e informar:* Comunicar los defectos potenciales encontrados en una reunión de revisión. Elaborar informes de hallazgos, documentar las características de calidad y comprobar criterios de salida.

**Requisitos**

Una de las revisiones que se realizan en las pruebas estáticas es examinar los requisitos del software. Un requisito define las funciones, capacidades o atributos intrínsecos de un sistema de software, es decir, describe cómo debe comportarse un sistema. Para decir que un sistema tiene calidad deben cumplirse los requisitos **funcionales y no funcionales**.

Requisitos funcionales:

Definen lo que un sistema permite hacer desde el punto de vista del usuario. Estos requisitos deben estar especificados de manera explícita. Por ejemplo: “El campo de monto acepta únicamente valores numéricos con dos decimales” (pruebas funcionales y de sistema).

Requisitos no funcionales:

Definen condiciones de funcionamiento del sistema en el ambiente operacional. Ejemplos:

* *Requisito de usabilidad*: la usabilidad se define como el esfuerzo que necesita hacer un usuario para aprender, usar, ingresar datos e interpretar los resultados obtenidos de un software de aplicación (pruebas de usabilidad).
* *Requisito de eficiencia:* relacionado con el desempeño en cuanto al tiempo de respuesta, número de operaciones por segundo, entre otras mediciones; así como consumo de recursos de memoria, procesador y espacio en disco o red (pruebas de rendimiento, pruebas de carga, estrés y escalabilidad, pruebas de gestión de la memoria, compatibilidad e interoperabilidad).
* *Requisito de disponibilidad:* disposición del sistema para prestar un servicio correctamente (pruebas de disponibilidad).
* *Requisito de confiabilidad:* continuidad del servicio prestado por el sistema (pruebas de seguridad).
* *Requisito de integridad:* ausencia de alteraciones inadecuadas al sistema (pruebas de seguridad, pruebas de integridad).
* *Requisito de mantenibilidad:* posibilidad de realizar modificaciones o reparaciones a un proceso sin afectar la continuidad del servicio (pruebas de mantenimiento y de regresión).

**Módulo 4 - Análisis y Planificación de la prueba**

**SEMANA 4**

**CLASE 11 – ORGANIZACIÓN DE LA PRUEBA**

**Organización de la prueba**

¿Qué pasa si luego de la publicación del proyecto se debe realizar un cambio imprevisto o si se visualiza un error que necesita ser corregido? No podemos hacer los cambios directamente sobre el ambiente en el que el cliente está utilizando el software, porque podríamos romperlo y dejarlo inoperativo. Es por eso que deben existir diferentes ambientes de trabajo, donde se pueda desarrollar y probar los cambios antes de que llegue al ambiente del cliente.

¿Qué es un ambiente?

*Ambiente de trabajo:* entorno con todos los recursos necesarios para que se pueda ejecutar un sistema.

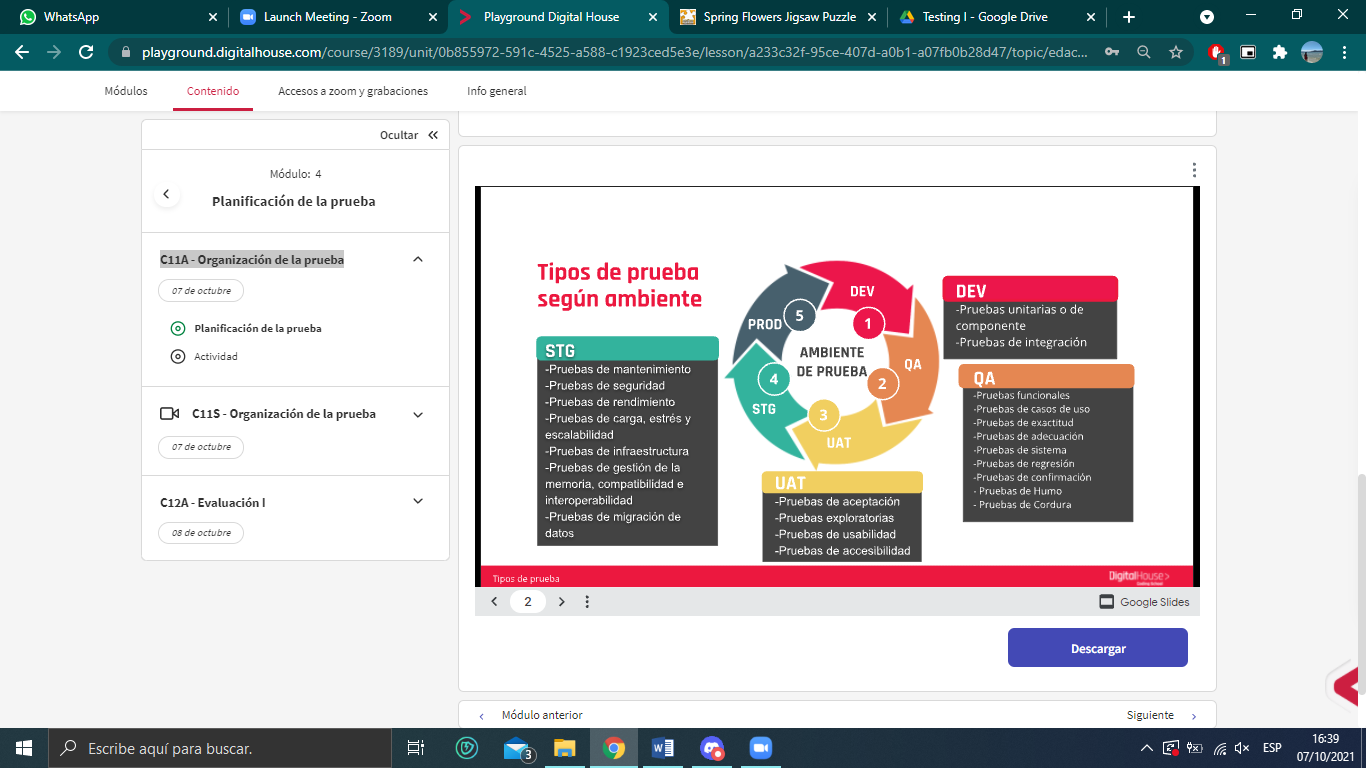
La ejecución de las pruebas se realiza en diferentes espacios de trabajo de acuerdo a la etapa del ciclo de desarrollo en el que se encuentre el sistema. Esto nos permitiría desarrollar aplicaciones de forma segura y con entornos diferenciados para realizar la programación, realizar pruebas, compartir resultados con los clientes y permitirles realizar pruebas y prácticas; y finalmente publicar una aplicación robusta y estable.

Niveles de ambientes:

Entonces, podemos decir que es conveniente distinguir los siguientes entornos:

* 1. *Ambiente de desarrollo o DEV:* este entorno es el espacio de trabajo donde el programador desarrolla el código de la aplicación, realiza pruebas iniciales y comprueba si la aplicación se ejecuta correctamente con ese código. Este ambiente puede ser local o en la nube, de acuerdo a la necesidad del proyecto.
  2. *Ambiente de pruebas o QA :* el entorno de pruebas suele estar ubicado en un servidor en la nube o en una granja de servidores locales (laboratorio). Permite minimizar incidencias en etapas posteriores, ya que el tester ejecutaría las primeras pruebas de funcionalidad en este ambiente.
  3. *Ambiente de UAT:* el entorno de UAT (o de pruebas de aceptación de usuario) permite a los usuarios del cliente poder verificar que los cambios realizados son los que realmente se solicitaron, evaluando a su vez accesibilidad y usabilidad.
  4. *Ambiente de preproducción o STAGE:* este entorno debería poseer una configuración técnica idéntica a la que nos encontraremos en el entorno de producción. El propósito principal de este entorno es emular al entorno de producción con el fin de probar las actualizaciones y asegurar que estas no corromperán la aplicación en los servidores en producción cuando sean desplegadas. De esta forma se minimizan las caídas del sistema y corte de los servicios en producción.
  5. *Ambiente de producción o PROD:* este es el entorno donde finalmente se ejecuta la aplicación, donde acceden los usuarios finales y donde se trabaja con los datos reales de negocio. Es un servidor que posee las mismas características y configuración que tendrá el servidor de preproducción. Aunque, en este caso, puede estar configurado por más de un servidor, para efectos de balanceo de carga en aplicaciones que requieren una infraestructura con capacidad de manejar un tráfico de usuarios pesado y miles de conexiones concurrentes.

¿Qué tipos de puebas se pueden hacer en cada ambiente?



En el ambiente de PROD (producción) no se desarrollara ningún tipo de prueba ya que los probadores no tienen acceso a este ambiente y, en caso de que tengan acceso, no se deben realizar acciones que generen datos, se corre el riesgo de ingresar dato basura, se interfiere en los datos de seguimiento (creo que el profe dijo que podía ser catastrófico)

DEV:

* Pruebas unitarias o de componente: Se centra en los componentes que se pueden probar por separado. Objetivo: encontrar defectos en el componente y verificar que los comportamientos funcionales y no funcionales del componente son los diseñados y especificados.
* Pruebas de integración: Se centra en las interacciones entre componentes o sistemas. Objetivo: encontrar defectos en las interfaces o dentro de los componentes o sistema y verificar que los comportamientos funcionales y no funcionales de las interfaces sean los diseñados y especificados.

QA:

* *Pruebas funcionales:* Incluyen pruebas que evalúan las funciones que el sistema debe realizar. Los requisitos funcionales pueden estar descritos en productos de trabajo tales como especificaciones de requisitos de negocio, épicas, historias de usuario, caso de uso y especificaciones funcionales.
* *Pruebas de casos de uso:* Proporcionan pruebas transaccionales, basada en escenarios, que deberían emula el caso de uso del sistema.
* *Pruebas de exactitud:* Comprenden el cumplimiento por parte de la aplicación de los requisitos especificados o implícitos y también puede abarcar la exactitud del cálculo. (Puede ser que haya dicho algo de la calculadora? Ejemplo del 2+2?)
* *Pruebas de adecuación*: Implican evaluar y validar la eficiencia de un conjunto de funciones para la consecución de las tareas especificadas previstas. Estas pruebas pueden basarse en casos de uso.
* *Pruebas de sistema:* Se centra en el comportamiento y las capacidades de todo un sistema o producto, a menudo teniendo en cuenta las tareas extremo a extremo que el sistema puede realizar y los comportamientos no funcionales que exhibe mientras realiza esas tareas.
* *Pruebas de regresión:* Implican la realización de pruebas para detectar efectos secundarios no deseados, luego de cambios hechos en una parte del código que puedan afectar accidentalmente el comportamiento de otras partes del código.
* *Pruebas de confirmación:* Consiste en volver a ejecutar los pasos para reproducir el fallo o los fallos causados por un defecto en la nueva versión de software, una vez corregido el defecto, para así confirmar que el defecto original se ha solucionado satisfactoriamente o detectar efectos secundarios no deseados.
* *Pruebas de cordura:* Es una prueba de regresión acotada que se centra en una o unas pocas áreas de funcionalidad. Se utiliza para determinar si una pequeña sección de la aplicación sigue funcionando después de un cambio menor.
* *Pruebas de humo*: Se lleva a cabo para asegurar si las funciones más importantes de un programa están trabajando correctamente, pero sin molestarse con los detalles más finos.

UAT:

* *Pruebas de aceptación*: Se centra normalmente en el comportamiento y las capacidades de todo un sistema o producto. Además, pueden producir información para evaluar el grado de preparación del sistema para su despliegue y uso por parte del cliente (usuario final)
* *Pruebas exploratorias:* Se diseñan, ejecutan, registran y evalúan de forma dinámica pruebas informales (no predefinidas) durante la ejecución de la prueba. Los resultados de la prueba se utilizan con el objetivo de aprender más sobre el componente o sistema y crear pruebas para las áreas que pueden necesitar ser probadas con mayor intensidad.
* Pruebas de usabilidad: Evaluan la facilidad con la que los usuarios pueden utilizar o aprender a utilizar el sistema para lograr un objetivo especifico en un contexto.
* Pruebas de accesibilidad: Incluyen y evalúan la accesibilidad que presenta un software para aquellos con necesidades particulares o restricciones para su uso. Esto incluye a aquellos usuarios con discapacidades.

STG:

* *Pruebas de mantenimiento:* Se cetra en probar los cambios en el sistema, asi como en probar las piezas no modificadas que podrían haberse visto afectadas por los cambios. El mantenimiento puede incluir lanzamientos planificados y no planificados.
* *Pruebas de seguridad:* Las pruebas de seguridad se podrían definir como el conjunto de actividades que se llevan a cabo para encontrar fallas y vulnerabilidades en el sistema, buscando disminuir el impacto de ataques y pérdida de información importante.
* *Pruebas de rendimiento:* Se implementan y se ejecutan para evaluar las características relacionadas con el rendimiento del destino de la prueba, como los perfiles de tiempo, el flujo de ejecución, los tiempos de respuesta y la fiabilidad y los limites operativos.
* *Pruebas de carga, estrés y escalabilidad:* Una prueba de carga garantiza que un sistema pueda controlar un volumen de tráfico esperado. Una prueba de estrés es en la que se somete al sistema a condiciones de uso extremas para garantizar su robustez y confiabilidad. Las pruebas de escalabilidad garantizan la escalabilidad de un sistema, es decir, que pueda soportar el incremento de demanda en la operación. También se pueden realizar en QA encontrando el correspondiente escalar con respecto a un ambiente de PROD.
* *Pruebas de infraestructura:* Incluyen todos los sistemas informáticos internos, los dispositivos externos asociados, las redes de internet, la nube y las pruebas de virtualización.
* *Pruebas de gestión de la memoria:* Evalúan el estado y la integridad de la memoria del sistema para identificar problemas potenciales.
* *Pruebas de compatibilidad:* Incluyen las pruebas para comprobar que el sistema es compatible con todos los navegadores de internet y todos los sistemas operativos del mercado.
* *Pruebas de interoperabilidad:* Se refieren a aquellas donde se realiza la evaluación de la correcta integración entre distintos aplicativos, sistemas, servicios o procesos que conforman una plataforma o solución tecnológica.
* *Pruebas de migración de datos*: Incluyen las pruebas realizadas al transferir datos entre tipos de dispositivos de almacenamiento, formatos o sistemas de cómputo.

1. El happy path, es un tipo de prueba positiva y que busca validar el flujo normal de un sistema. Por ejemplo, si tenemos que testear un Login, ¿cuál sería el camino ideal? Que el usuario ingrese un nombre de usuario y contraseña correctos, haga clic en el botón loguearse y que la aplicación se abra. Este camino ideal, es lo que conocemos como happy path. Son los primeros test que deberíamos hacerle a nuestra aplicación.

2. Si, al testing positivo y negativo lo podemos categorizar como una modalidad de casos de prueba.

3. En el día a día, los tester generalmente hacemos pruebas en el ambiente de QA o de producción. Por ahora, esta correcto que utilicen el ambiente de QA para el diseño de los casos de prueba. No le den muchas vueltas, vamos a profundizar el tema de los ambientes más adelante en el cursado

4. Con respecto a la severidad y prioridad, es más que nada una evaluación subjetiva y lo vas mejorando con los años. Pero por ejemplo si no me puedo loguear en la aplicación y esto lo tienen que solucionar si o si, sino el usuario está bloqueado. Entonces su severidad y prioridad es alta. En cambio, si por ejemplo al ingresar mal la contraseña me sale un mensaje de error y no lo puedo cerrar pero si me puedo loguear, esto tiene una severidad baja, porque puedo seguir trabajando con la app.