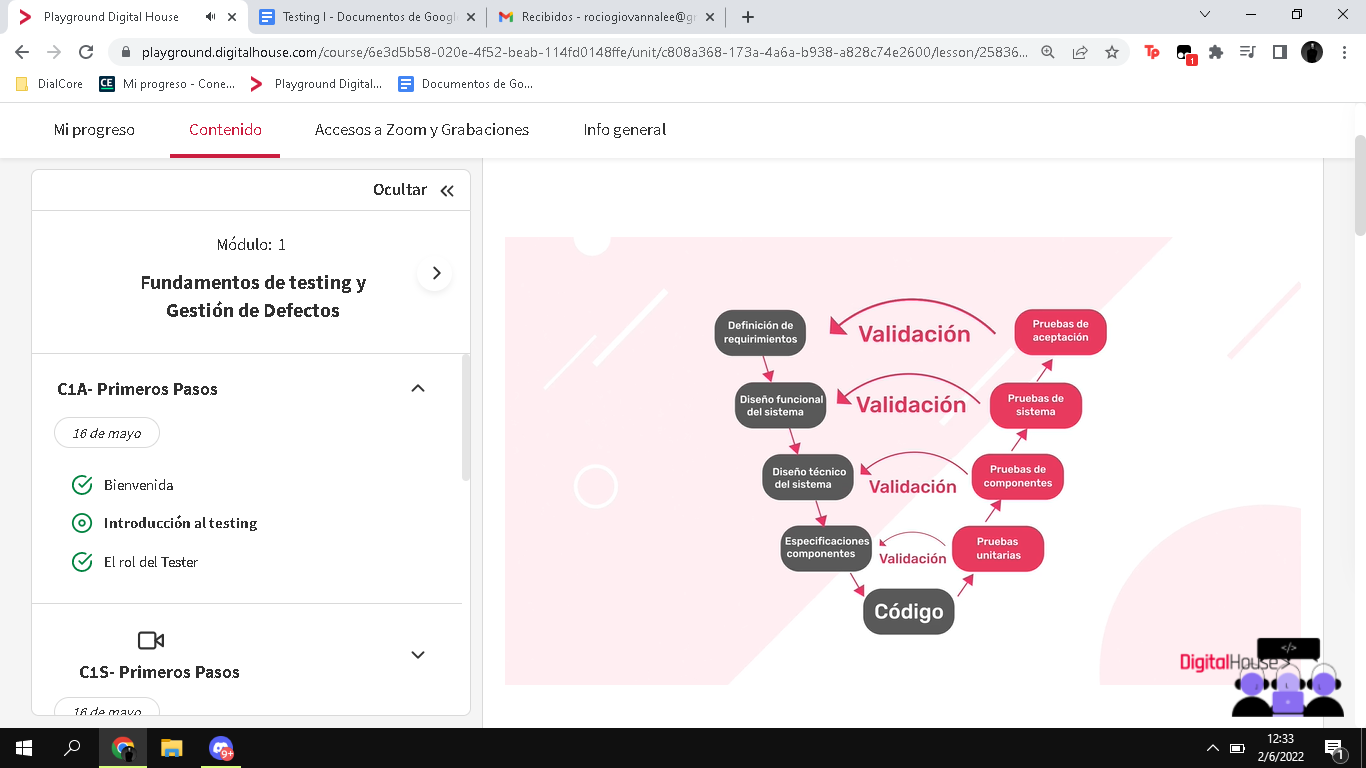
Para obtener un software de calidad debemos testearlo.

Existen distintos niveles de prueba:

* Modelo en V: Validación y Verificación.

****

****

* **Validación:** Son aquellas cuestiones que un analista funcional o de negocios deberá confirmar con el cliente, garantizando un propio entendimiento de las necesidades antes de pasar por la propia ejecución.
* **Verificación**: Hacemos referencia a un control de calidad sobre un producto o servicio.

**7 PRINCIPIOS DEL TESTING**

1 - La prueba muestra la presencia de defectos, no su ausencia: No puede probar que no hay defectos

2 - La prueba exhaustiva es imposible: No es posible probar todo, 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.

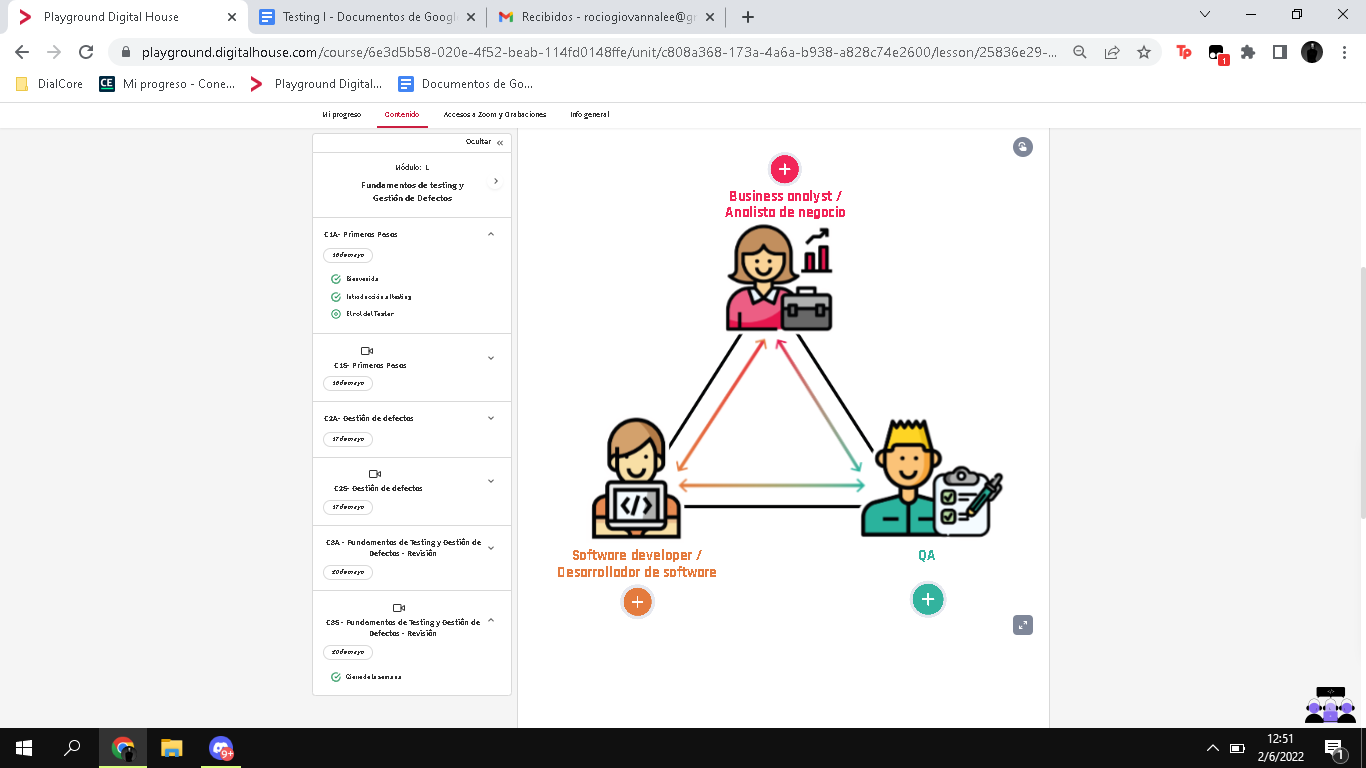
4 - Los defectos se agrupan: En general, un pequeño número de módulos contiene la mayoría de los defectos.

5 - Cuidado con la paradoja 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:

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

Mesa de 3 patas



-El Analista de negocios: Se encarga de detectar los factores clave del negocio y es el intermediario entre el departamento de sistemas y el cliente final.

-Desarrollador de software: Su función es diseñar, producir, programar o mantener componentes o subconjuntos de software conforme a especificaciones funcionales y técnicas para ser integrados en aplicaciones.

-QA: La principal función es probar los sistemas informáticos para que funcionen correctamente de acuerdo a los requerimientos del cliente, documentar los errores encontrados y desarrollar procedimientos de prueba para hacer un seguimiento de los problemas de los productos de forma más eficaz y eficiente.

Error: Es un error del programador en el código.

Defecto: Defecto del software. Consecuencia del error humano en el código.

Falla: Fallo del sistema. Cuando el sistema ya está funcionando pero falla como consecuencia del defecto.

El error lo comete el programador en el código, esto produce un defecto del software, mostrando una falla en el sistema.

El defecto es el reporte de la falla.

**Ciclo de vida de un defecto:**

1. Nuevo/inicial: se recopila la información y se registra el defecto
2. Asignado: se asigna al equipo de desarrollo.
3. En proceso: se analiza y trabaja en la solución.
4. Corregido: se realizan los cambios de código para solucionar el defecto
5. En espera de verificación: en espera de que sea asignado a un probador. El desarrollador está a la espera del resultado de la verificación.
6. En verificación: el verificador ejecuta una prueba de confirmación.
7. Verificado: se obtiene el resultado esperado en la prueba de confirmación.
8. Cerrado: el defecto fue corregido.

Ccccds

PROCESO DE GESTIÓN DE DEFECTOS

1 - Detectar

2 - Registrar

3 - Investigación y seguimiento

4 - Clasificación/ resolución

**CASOS DE PRUEBA**

Un caso de prueba es un conjunto de acciones que se ejecutan para verificar una característica o funcionalidad particular de una aplicación de software.

Es un documento escrito que proporciona información sobre qué y cómo podemos probar.

* Caso de prueba positivo o testing positivo: validan el flujo normal de un sistema bajo prueba. La prueba positiva implica ejecutar un escenario de prueba con solo datos correctos y válidos.
* Caso de prueba negativo o testing negativo: Son aquellos casos de prueba que validan flujos no contemplados dentro de los requisitos de un sistema bajo prueba. Es cuando esperamos que la prueba falle. Por ejemplo, esperar a la validación de mensajes de advertencia. Puede tener un resultado positivo o un defecto.

HAPPY PATH

Es un camino en el que se prueba una aplicación a través de escenarios de prueba cuidadosamente diseñados. Su propósito no es encontrar defectos, sino que un producto funcione como ha sido diseñado.

* solo utiliza escenarios de prueba positivos.

**CASOS DE USO**

Es previo al caso de prueba. Se debe llevar a cabo el análisis de los documentos que van a ser la base para la generación de esos casos de prueba.

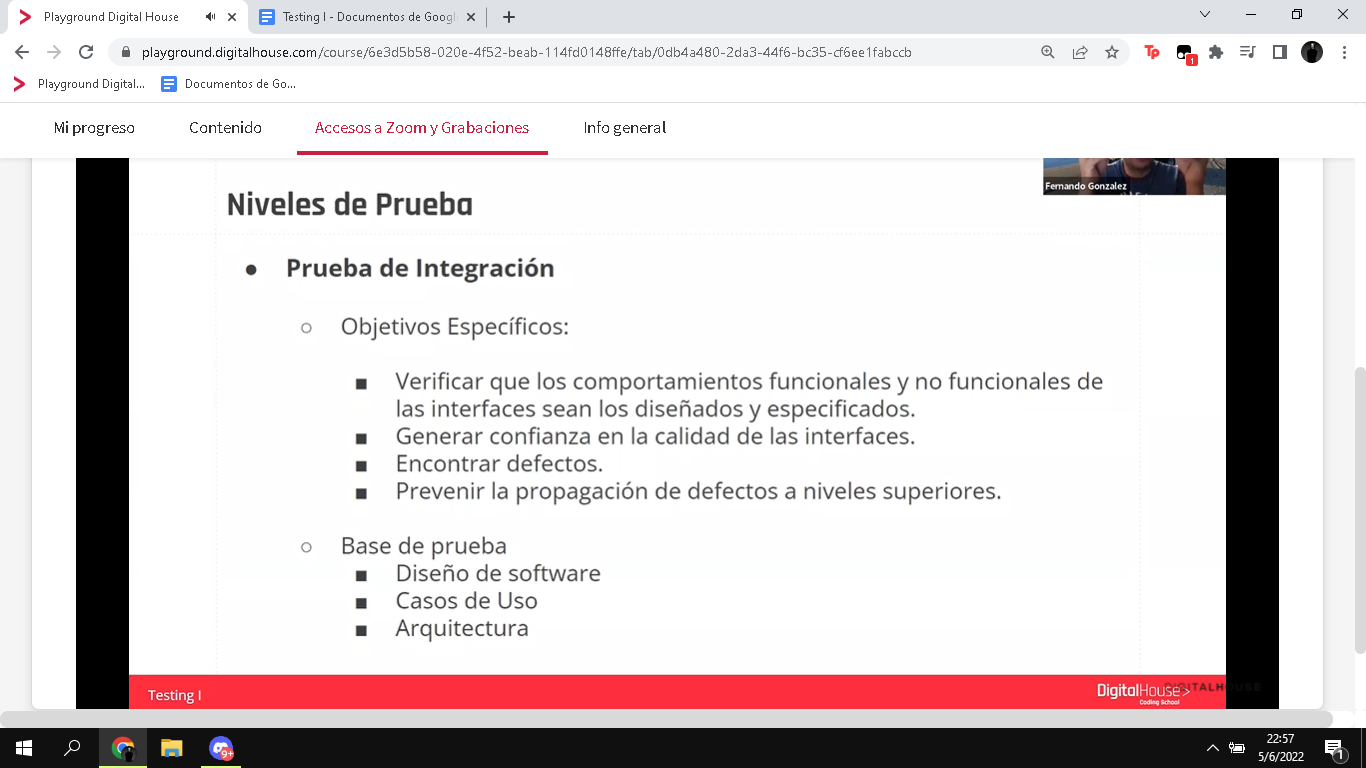
Un caso de uso cuenta la historia de cómo un usuario interactúa con un software para lograr o abandonar un objetivo.

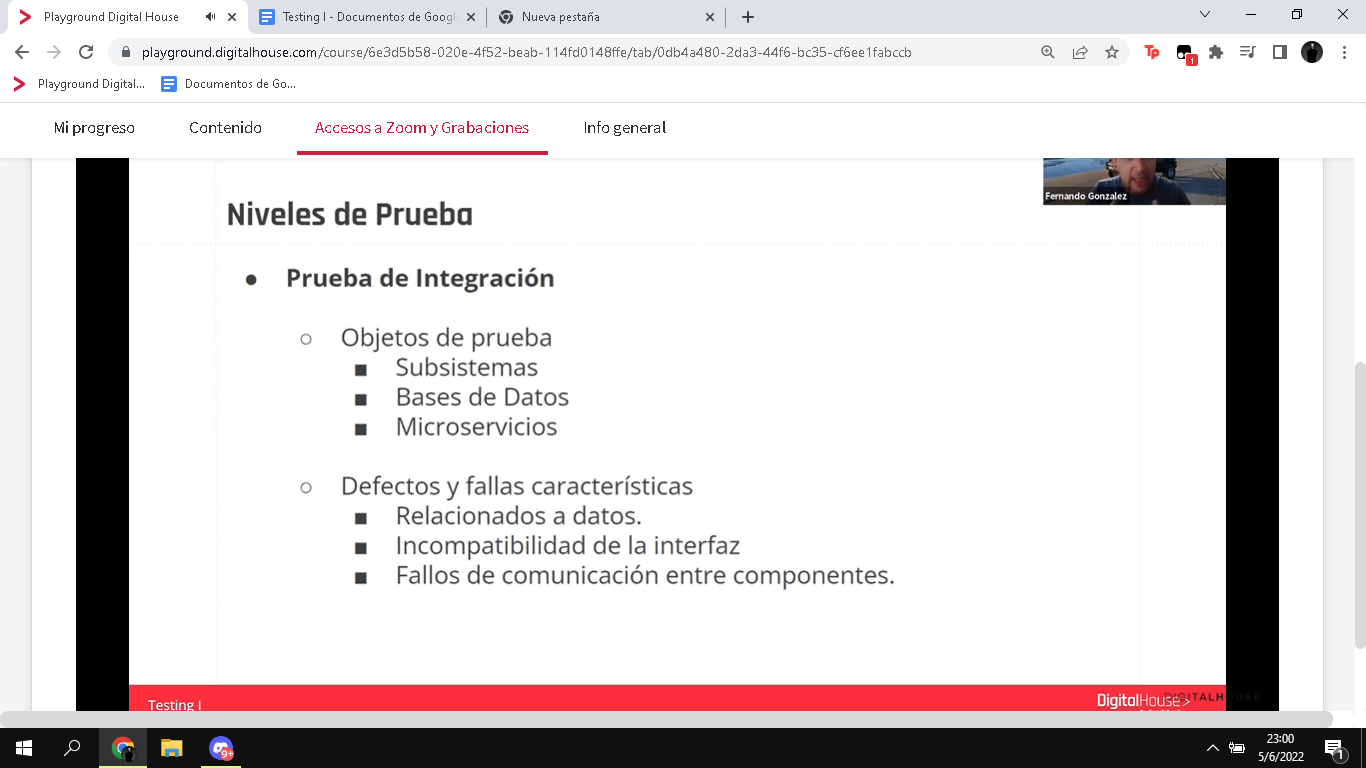
**NIVELES DE PRUEBA**

* Prueba unitaria o de componente

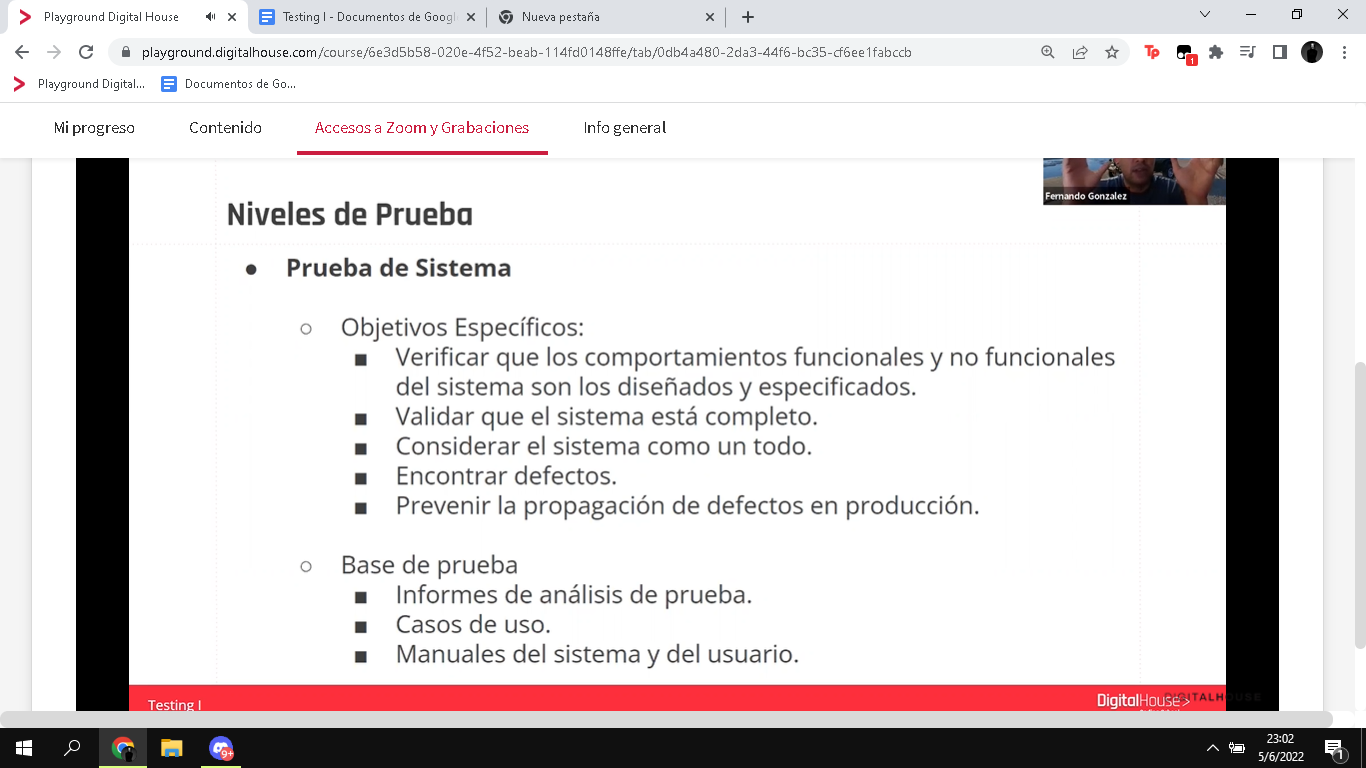


* Prueba de integración

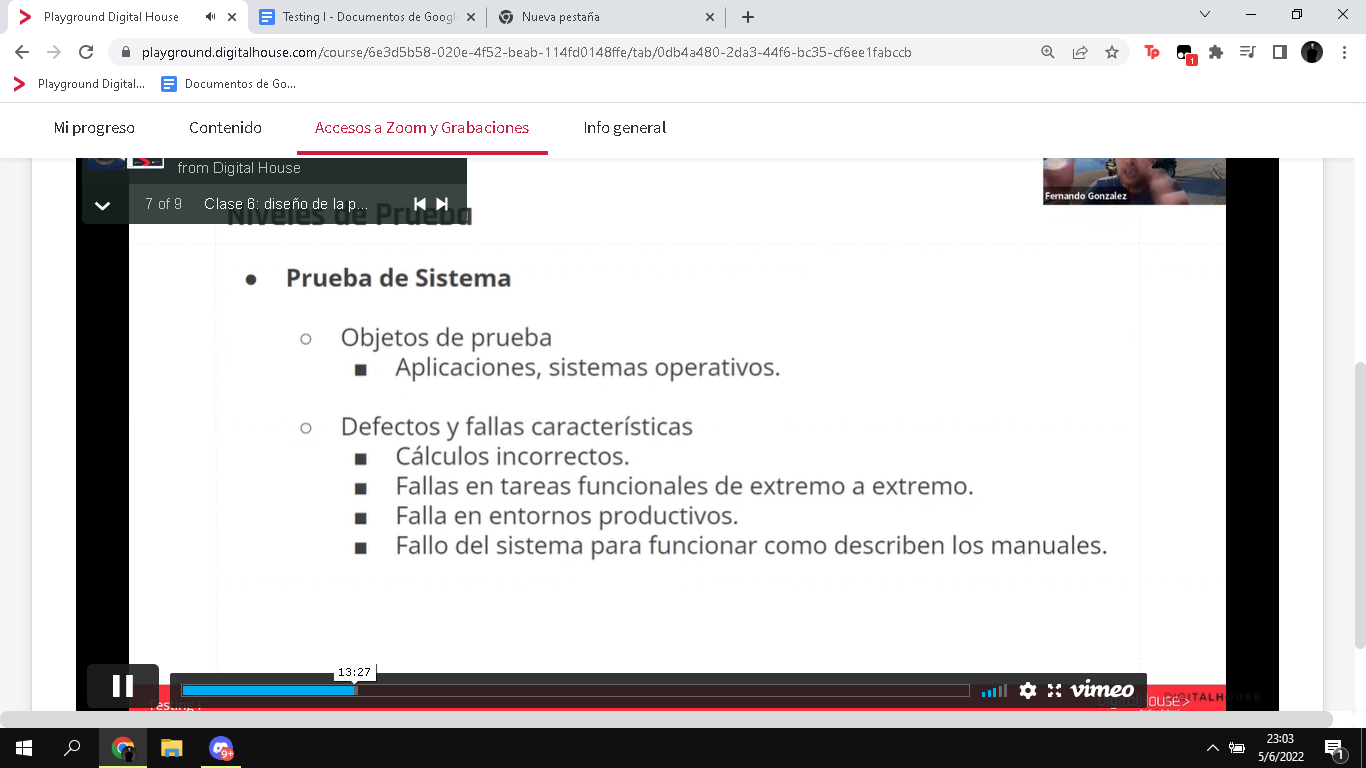




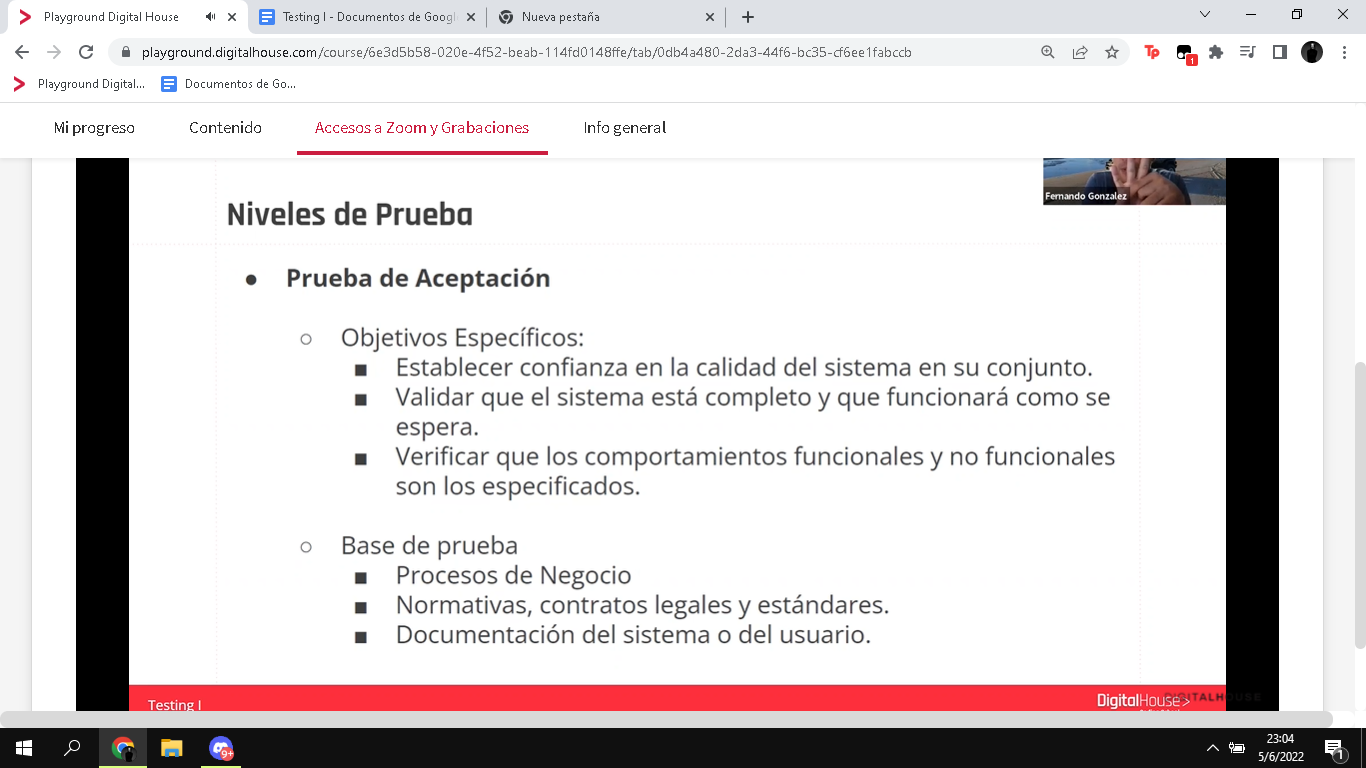
* Prueba de sistema



sigue abajo…

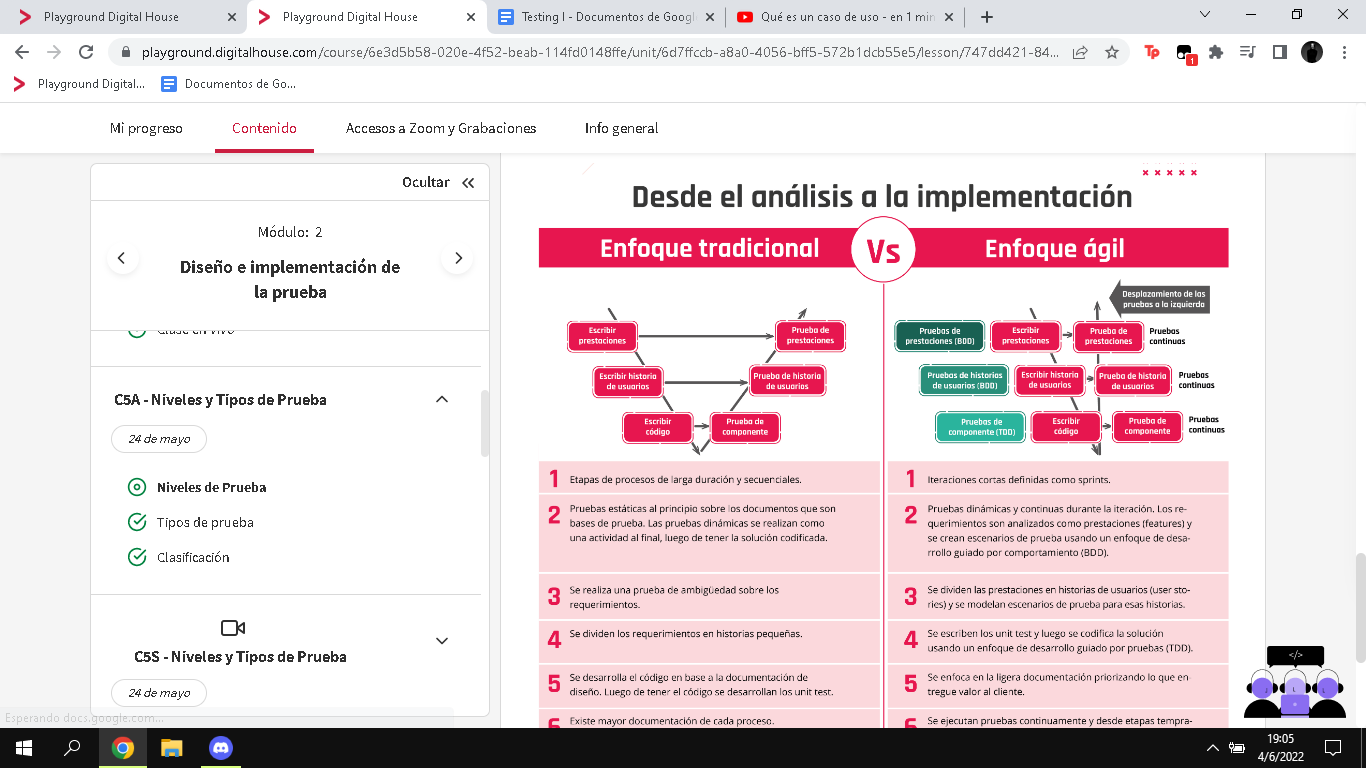


* Prueba de aceptación



IMPLEMENTACIÓN:

Enfoque tradicional y Agil



**TIPOS DE PRUEBA**

Es un grupo de actividades destinadas a probar las características específicas de un sistema.

1. *Prueba funcional*: La prueba funcional de un sistema incluye pruebas que evalúan las funciones que el sistema debe realizar. Las funciones describen **qué hace** el sistema. - Pruebas de sanidad -Pruebas de aceptación - Pruebas de sistema

que hace

1. *Prueba no funcional*: Puede que tengamos un sistema funcionando, pero el usuario está experimentando otro tipo de problemas que no son detectados por las pruebas anteriores. La prueba no funcional prueba “**cómo de bien**” se comporta el sistema.

* Pruebas de rendimiento
* Pruebas de portabilidad
* Pruebas de usabilidad
* Pruebas de carga
* Pruebas de estrés

Se definen comportamiento de sistema en el ambiente operacional, requisitos que dicen como se va a comportar el sistema antes ciertas características.

como lo hace

que pasaria si… por ejemplo se corta la luz como seguiria funcionando el software

1. *Pruebas Estructurales*: Éstas 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.

- Prueba unitaria

- Prueba de integración

1. *Prueba asociada al cambio*: Existen 2 tipos:

* **Prueba de confirmación**: 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. La prueba de regresión implica la realización de pruebas para detectar estos efectos secundarios no deseados.

**FORMAS DE PRUEBA**

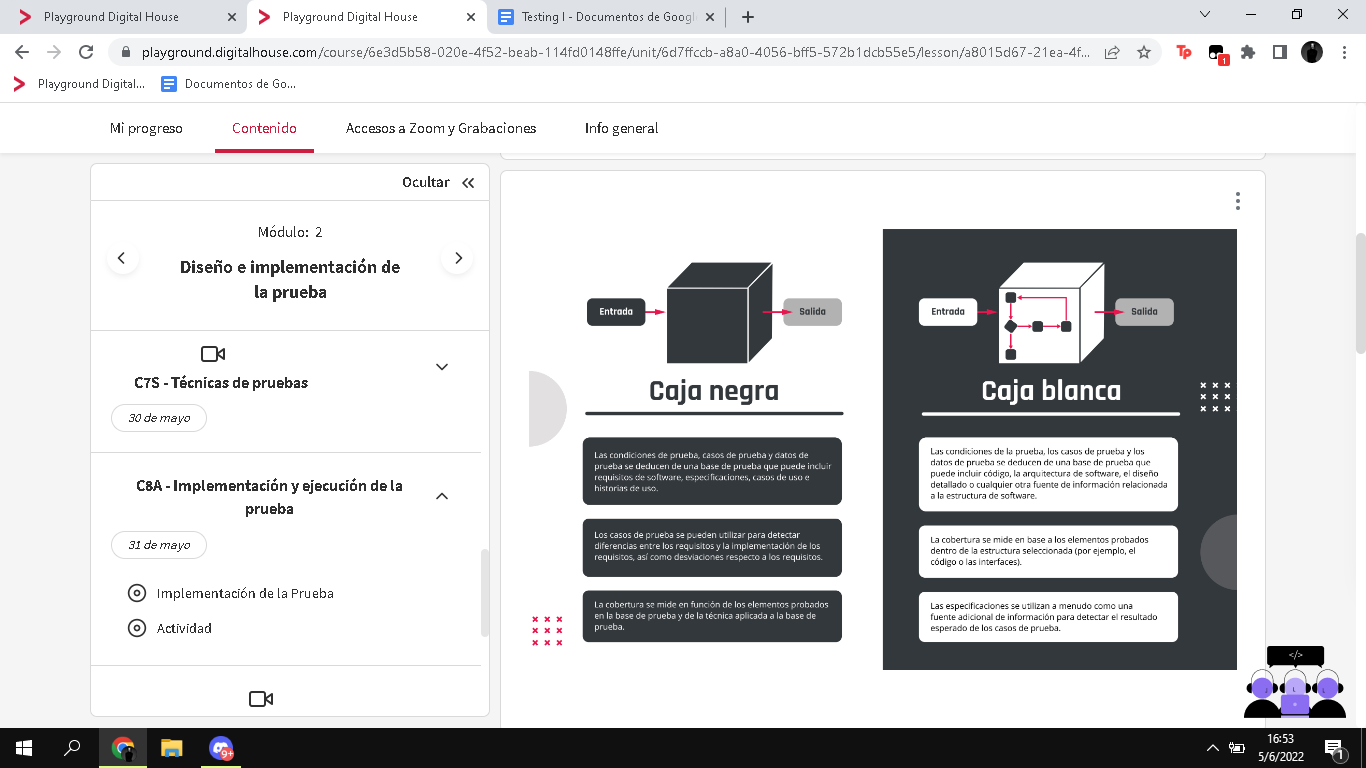
* Manual: Son pruebas manuales realizadas por uno o más testers, simulando las acciones del usuario final.
* Automatizadas: Son pruebas automatizadas ejecutadas por testers con habilidades técnicas

**TÉCNICAS DE PRUEBA**

**Son el punto de partida para el diseño de nuestros casos de pruebas. Estas nos permiten tener una mayor cobertura a la hora de diseñarlos.**

* CAJA NEGRA: En este tipo de técnica de prueba, el elemento es estudiado desde el punto de vista de las entradas que recibe y las respuestas que produce.
* CAJA BLANCA: Es un método por el cual se mira el código y la estructura del producto que se va a probar y se usa ese conocimiento para la realización de las pruebas.

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

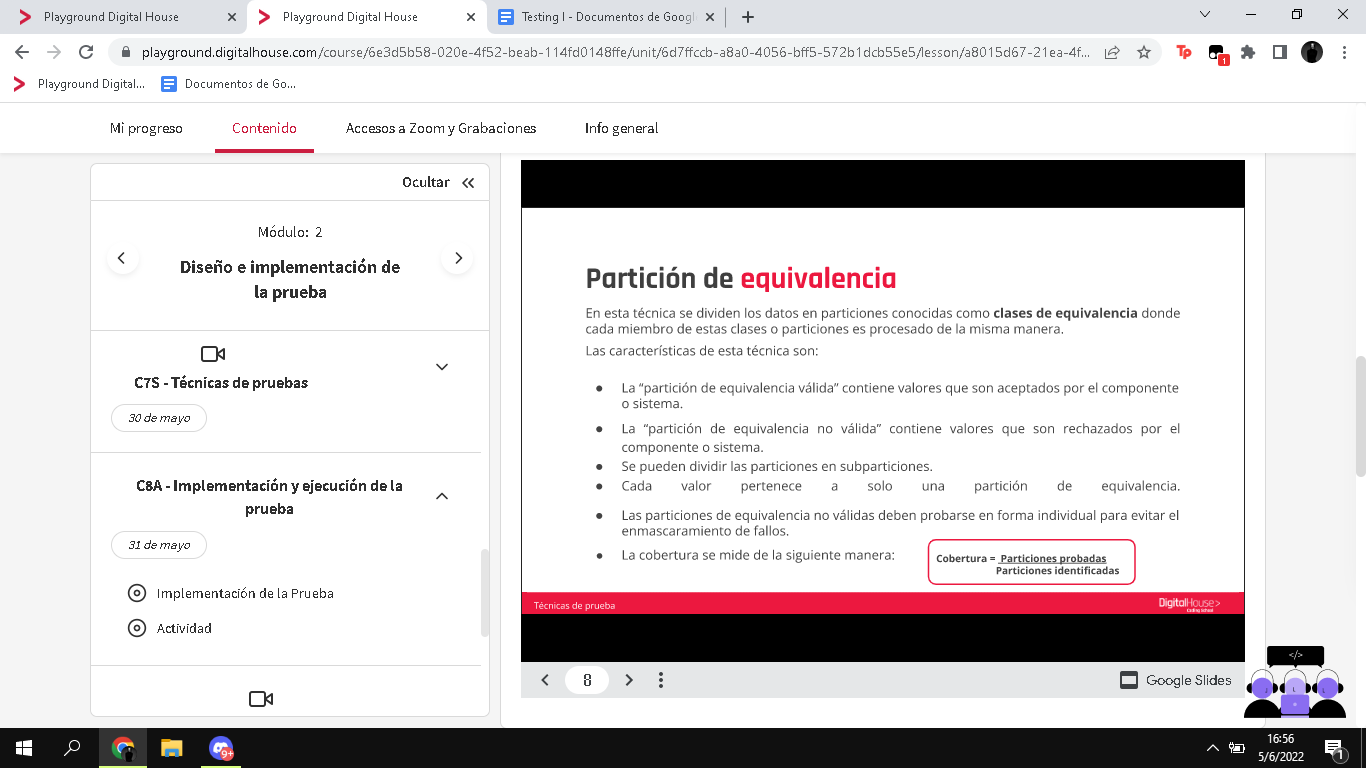


*Clasificación de las técnicas de prueba:*

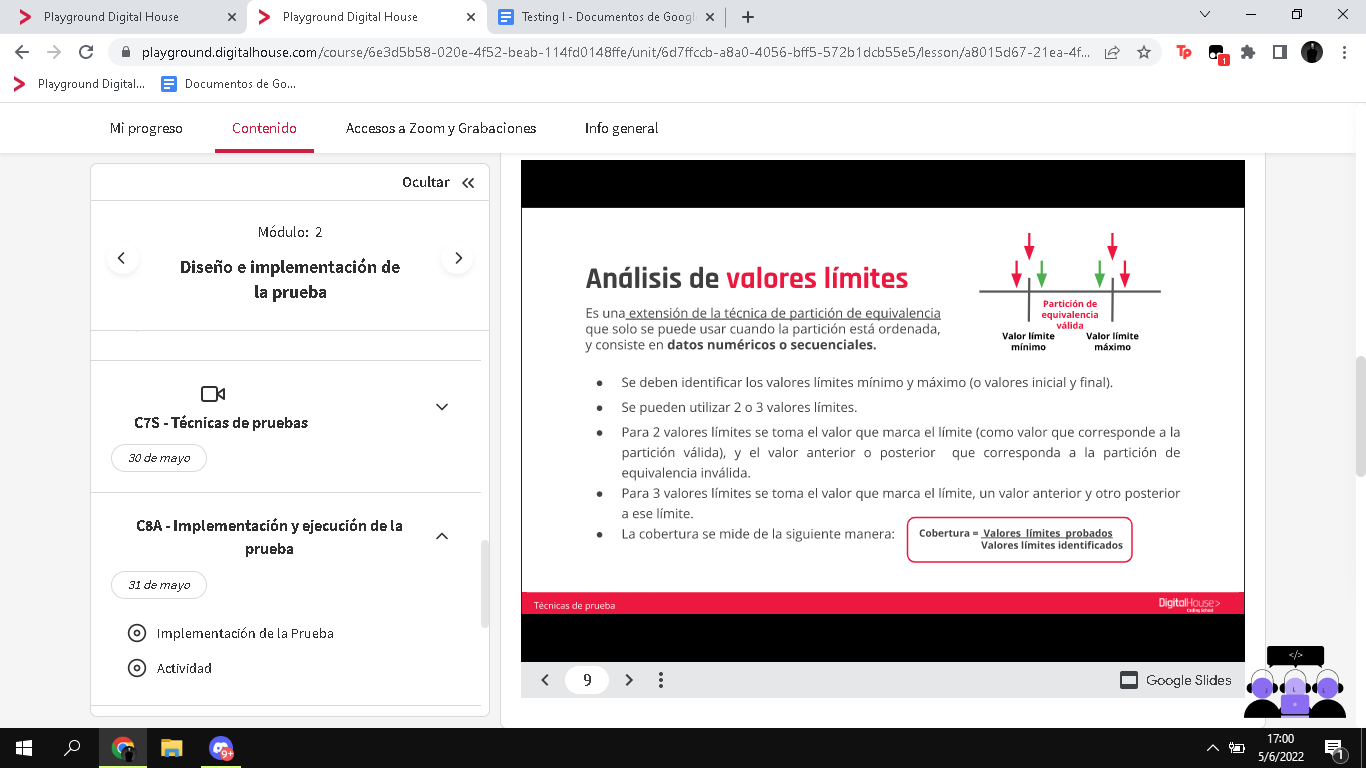
* Técnicas de caja negra: se basan en el comportamiento extraído del análisis de los documentos que son base de prueba(casos de uso, historias de usuario). Son aplicables tanto a pruebas funcionales y no funcionales. Se concentran en las entradas y salidas sin tener en cuenta la estructura interna.
*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

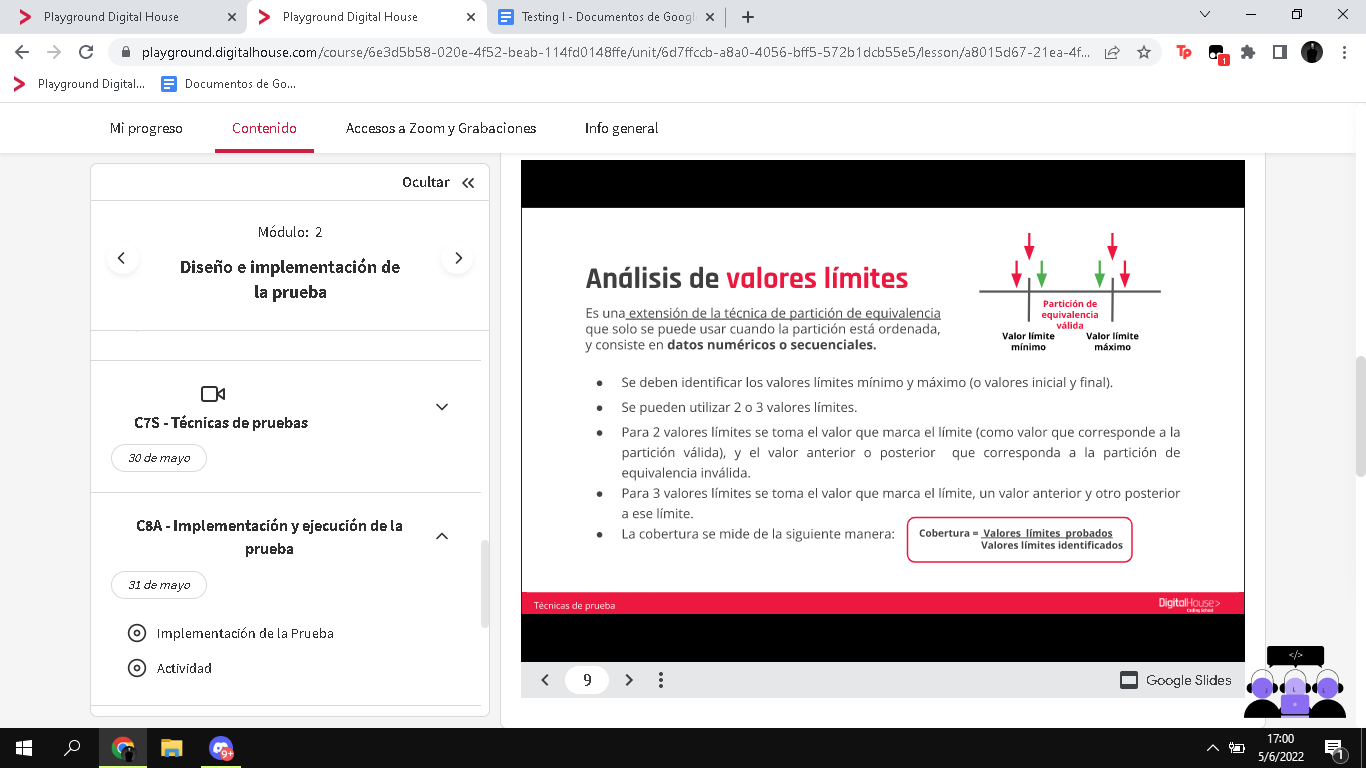
**TÉCNICAS DE CAJA NEGRA:**

* Partición de equivalencias: En esta técnica se dividen los datos en particiones conocidas como clases de equivalencia.



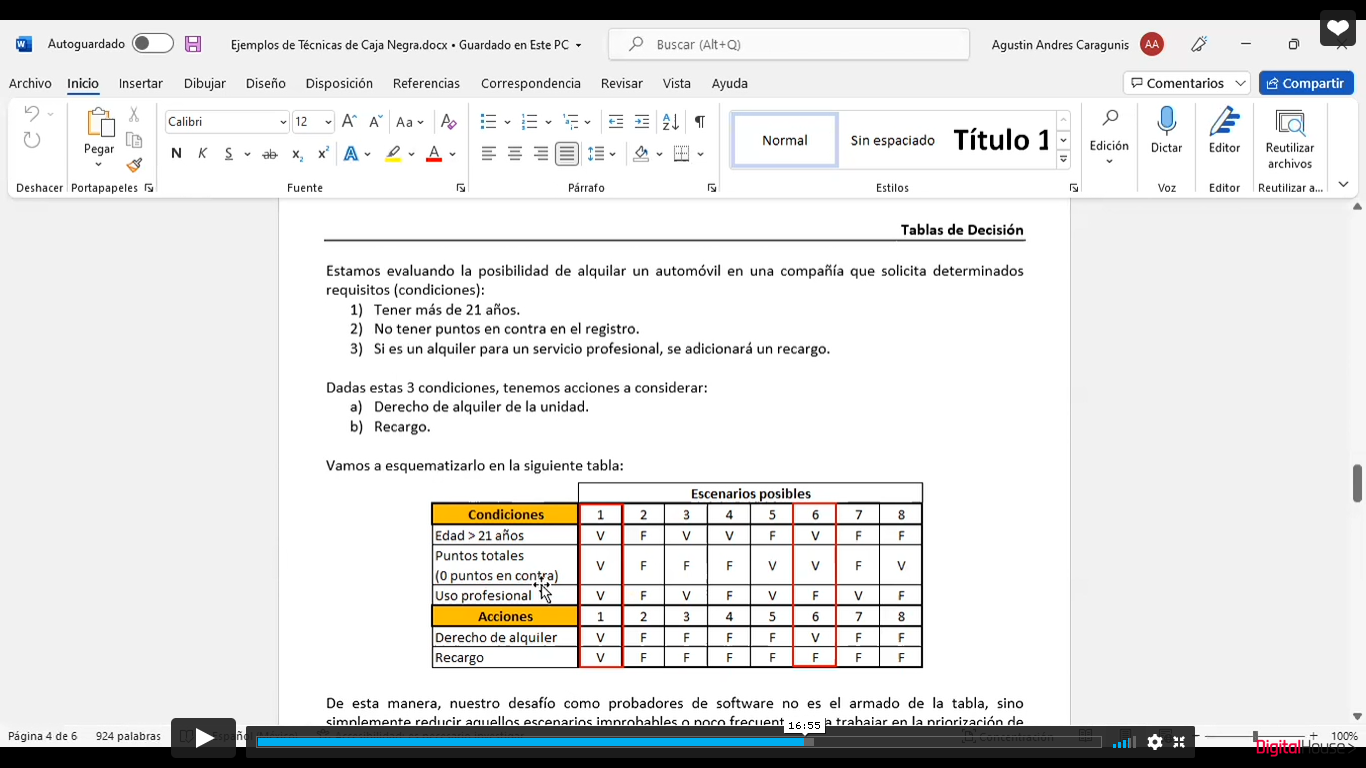
* Análisis de valores límites: Es una extensión de la técnica de partición de equivalencias que sólo puede ser utilizada cuando la partición está ordenada y consiste en datos numéricos y secuenciales.

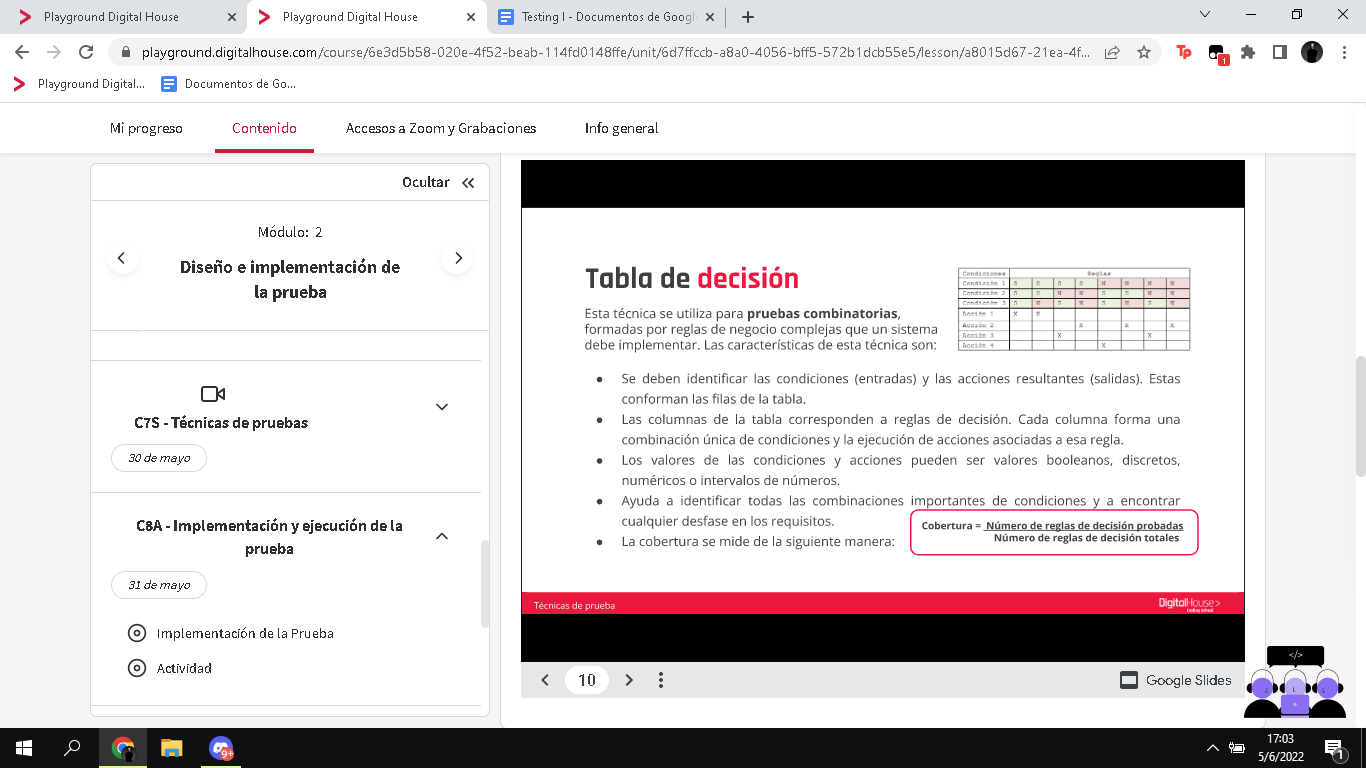




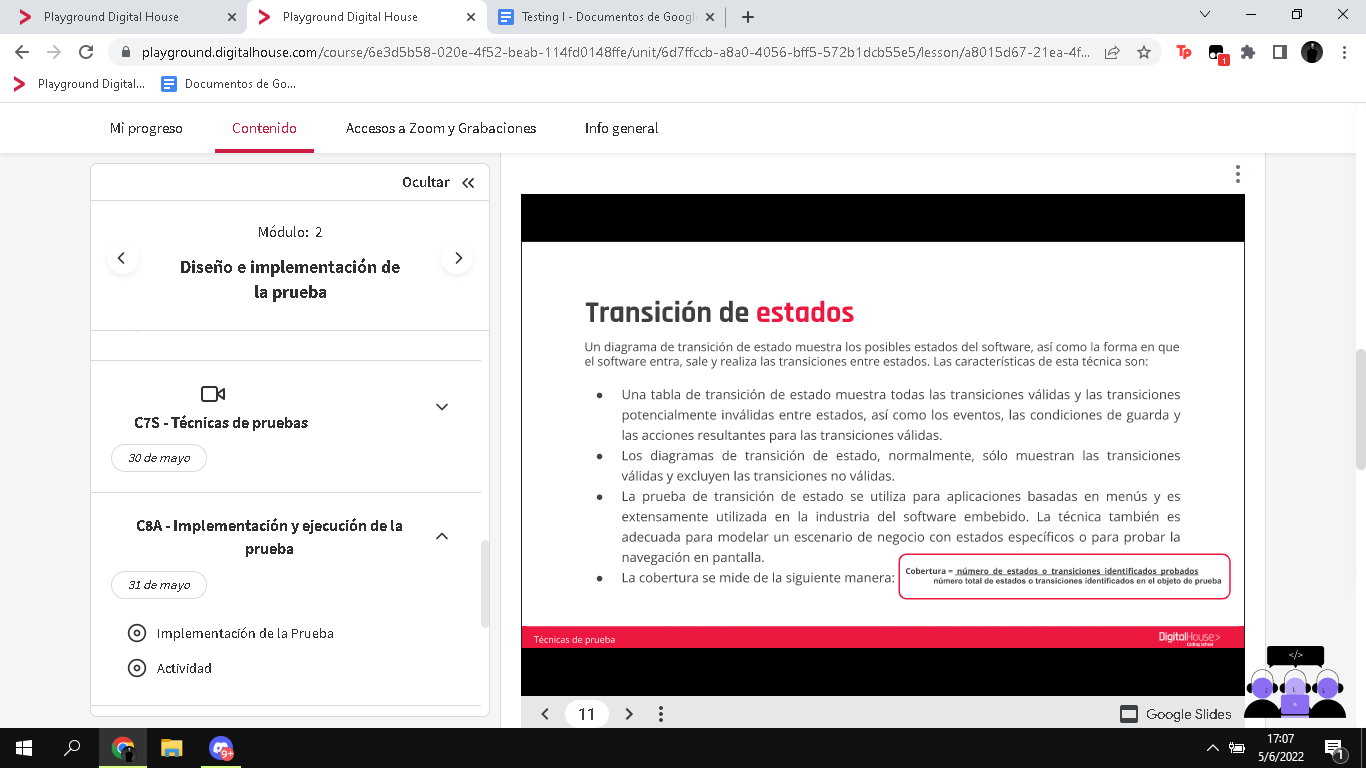
* Tabla de decisión: Esta técnica se utiliza para pruebas combinatorias formadas por reglas de negocio complejas que un sistema debe implementar.

2 a la n → según la cantidad de escenarios de entrada, donde n sería el escenario, entonces si hay 3 escenarios sería una ecuación de 2 al cubo. 3 veces 2, es decir, 8 escenarios posibles. Ejemplo:





* Transición de estados:Muestra los posibles estados de software, así como la forma en la que el software entra y sale y realiza transiciones entre estados.



**TÉCNICAS BASADAS EN LA EXPERIENCIA:**

* Predicción de errores: 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 funcionaba 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: Se diseñan, ejecutan, registran y evalúan 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.
* Pruebas basadas en listas de comprobación: Se diseñan e 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 o conocimiento de lo que el probador cree importante para el usuario y se utilizan debido a la falta de casos de prueba detallados. Se utilizan tanto en pruebas funcionales y no funcionales.
* Las **pruebas de humo** se ejecutan para evaluar la estabilidad de las compilaciones de software iniciales o desarrolladas recientemente.
* Las **pruebas de regresión son la tarea** de verificar y validar las funcionalidades existentes de la aplicación después de cada modificación o en la adición de nuevas funciones.

**Las pruebas de humo son previas a las de regresión.**

PRUEBAS ESTÁTICAS Y DINÁMICAS

* Pruebas estáticas: se basa en la evaluación manual de los productos de trabajo o basada en las herramientas del código, como por ejemplo: requisitos de negocio, historias de usuario y criterios de aceptación, etc.
* Pruebas dinámicas: requieren ejecución del software, componente o sistema. Se complementan con las pruebas estáticas. Para la generación de casos de prueba se utilizan diferentes técnicas caja negra, blanca y basadas en experiencia de usuario.

Estáticas: como documentos. Caso de prueba a validar si se está ejecutando dentro del papel pero si ejecuto ese caso de prueba es dinámica

Previene los defectos por eso son tempranas,

Son documentaciones

Dinámicas: Se está moviendo el código, se está modificando - toda prueba funcional es dinámica (recargar una pantalla, se usa en el momento). Las pruebas no funcionales también son dinámicas

Detecta los defectos por eso son tardías

El aplicativo está funcionando

Sigue abajo…



REQUISITOS

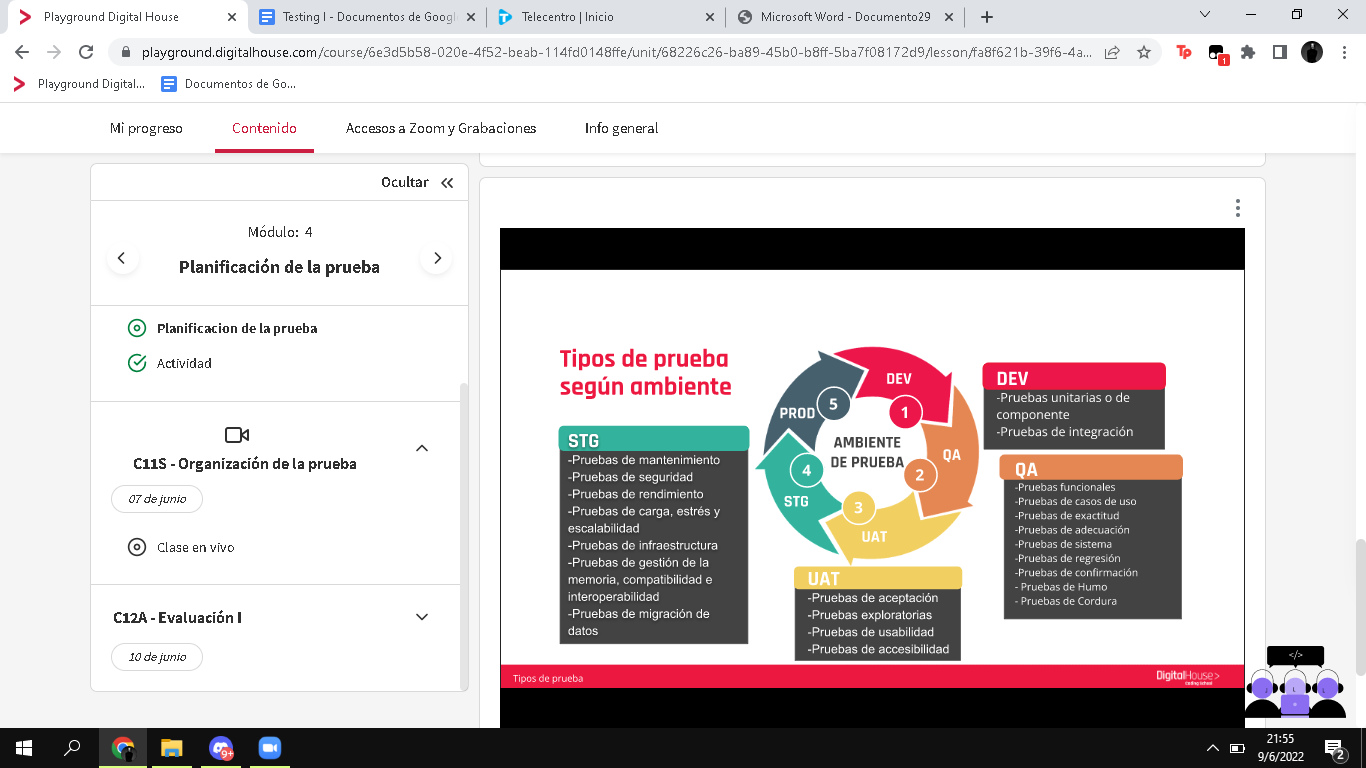
Una de las revisiones que se realizan en las pruebas estáticas es examinar los requisitos del software. Un requisito describe cómo debe comportarse un sistema. Para decir que un sistema tiene calidad deben cumplirse los requisitos **funcionales y no funcionales**.

**Requisitos funcionales**

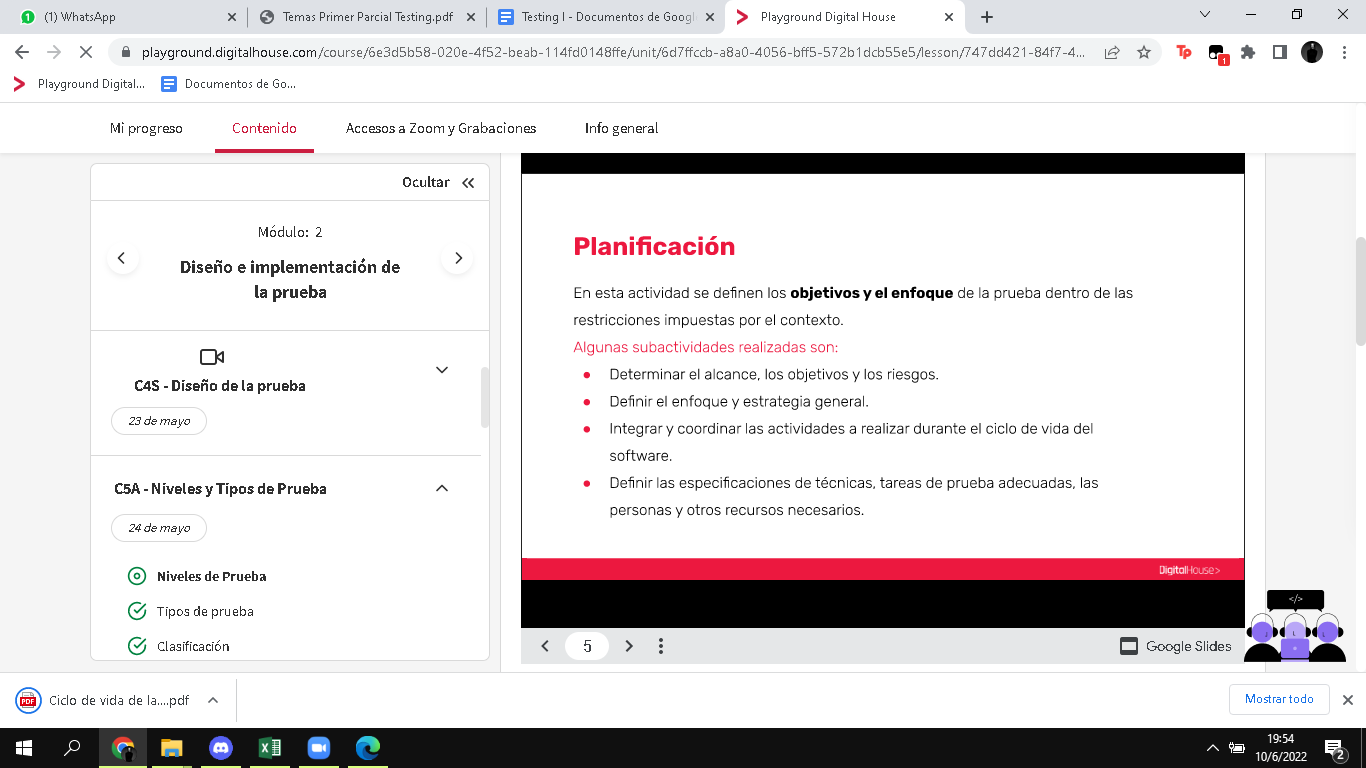
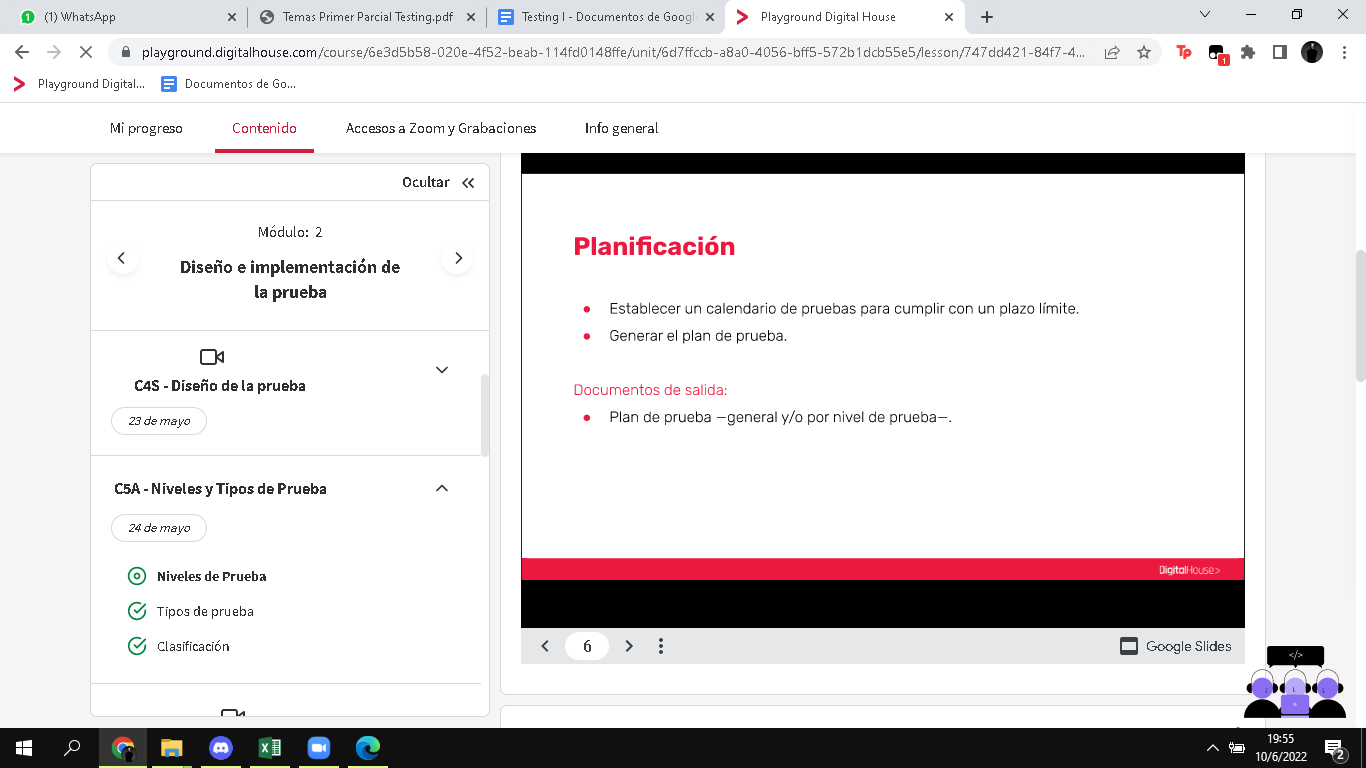
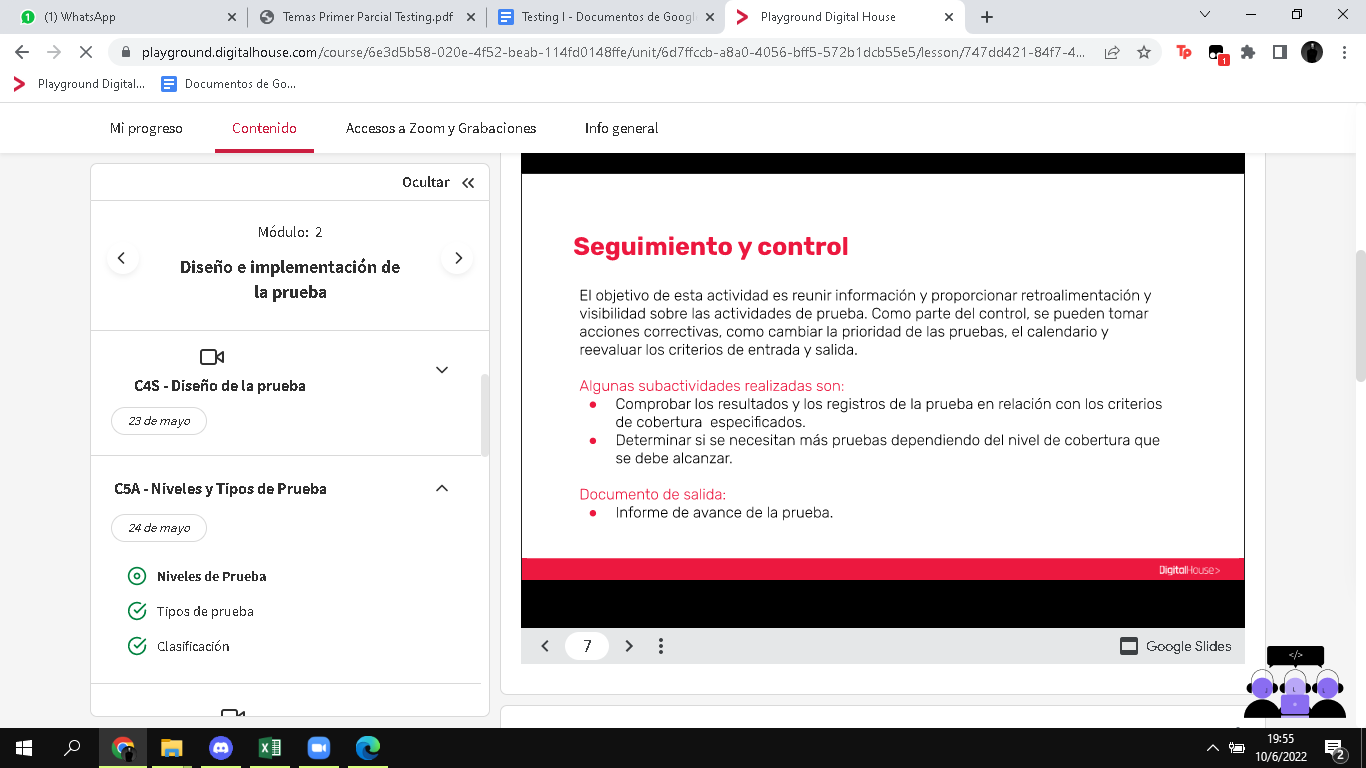
Definen lo que un sistema permite hacer desde el punto de vista del usuario. Estos requisitos deben estar especificados de manera explícita.

**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
* **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
* **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).
* 

NIVELES Y CICLOS DE UN SOFTWARE

* 
* 
* 
* 

**DEBUGGING/ DEPURACIÓN**

Los desarrolladores llevan a cabo un proceso de debugging, conocido también como depuración. Esto implica buscar los errores cometidos en el código fuente para poder corregirlos.

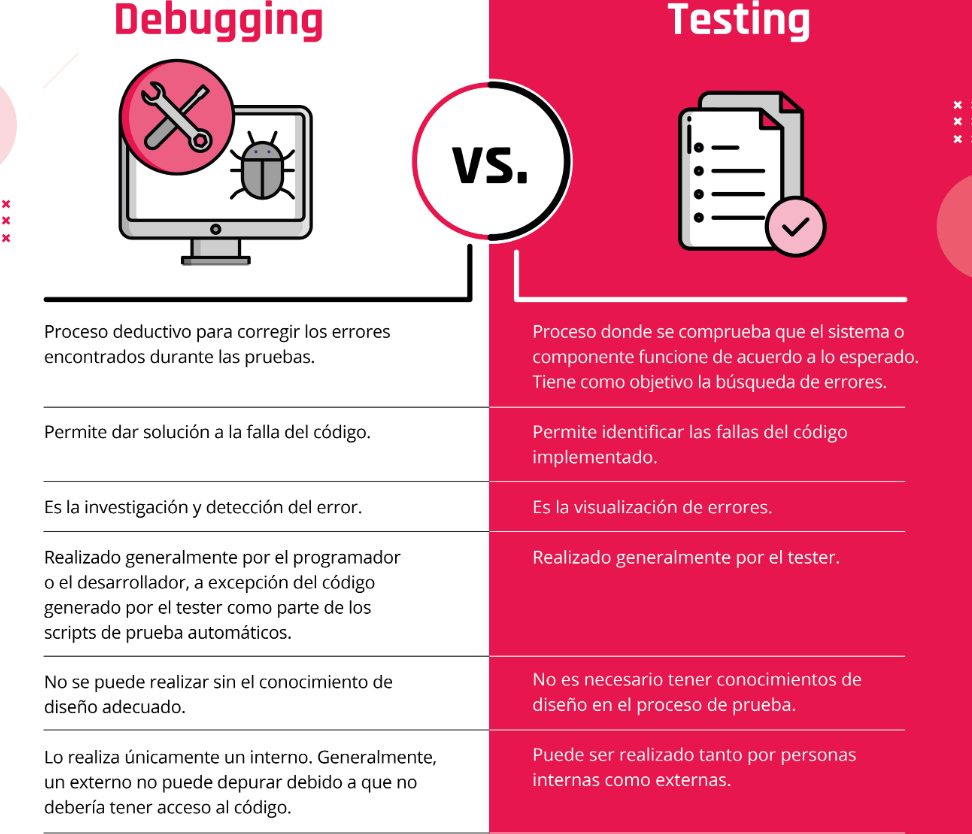
- Debug: Se puede realizar utilizando herramientas como las del navegador: Chrome, Dev Tools, etc. O la opción de Debug dentro de un framework o IDE como VScode.

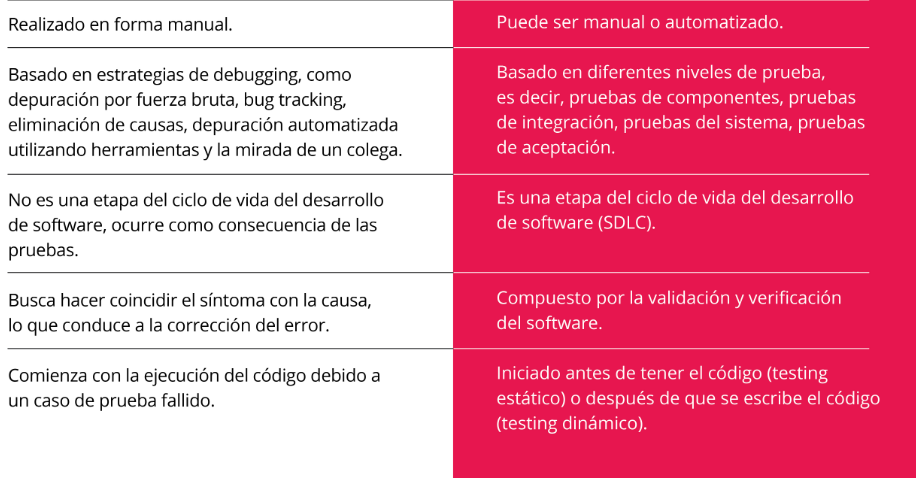
- Breakpoint: Es un punto de interrupción en nuestro código para obtener la ejecución del programa en líneas específicas y analizar la situación del mismo. Revisando por ejemplo el estado de las variables.

- Debugging: un proceso de debugging, conocido también como depuración. Este implica buscar los errores cometidos en el código fuente para poder corregirlos.

Es un proceso de encontrar, analizar y remover las causas de fallos en el software. Se realiza la ejecución paso a paso del programa para analizar las variables y sus valores.

**Diferencia entre debugging y testing**





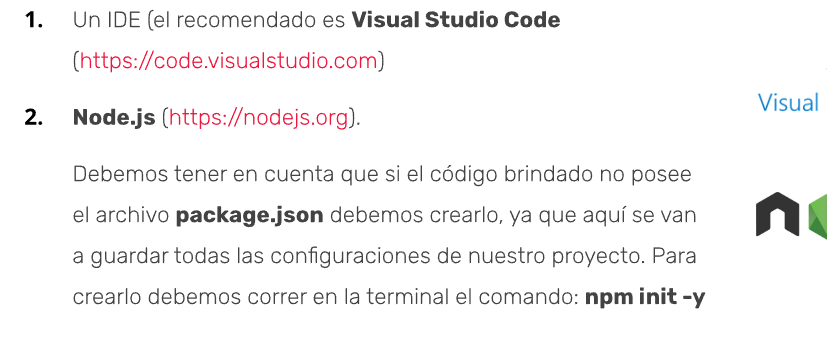
**UNIT TESTING**

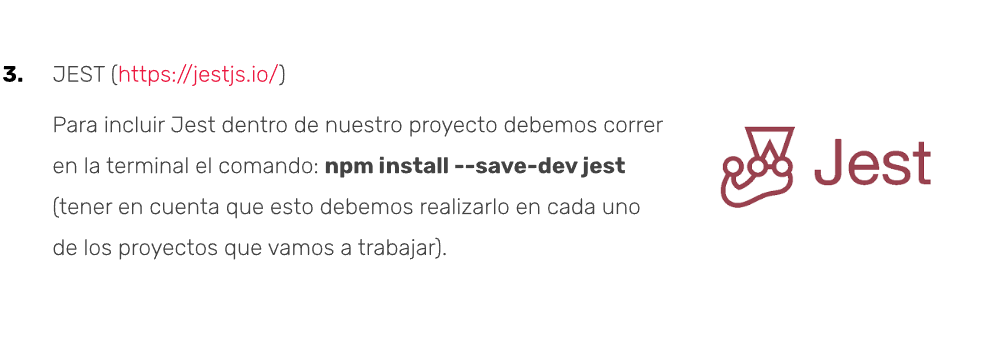
## Prueba de componente o prueba unitaria : Son pequeños test creados específicamente para cubrir todos los requisitos del código y verificar sus resultados. Para generar estos test se utilizan técnicas de caja blanca. Son generalmente pruebas automatizadas escritas y ejecutadas.

## El proceso general para la creación de estos unit test consta de tres partes: 1. **Acuerdo o criterio de aceptación, 2. Escritura del test, 3.Confirmación .**

El objetivo principal es aislar cada unidad del sistema para identificar y corregir los defectos.

Sus pruebas unitarias pueden ser (frameworks/librerías) pueden ser manuales o automatizadas.



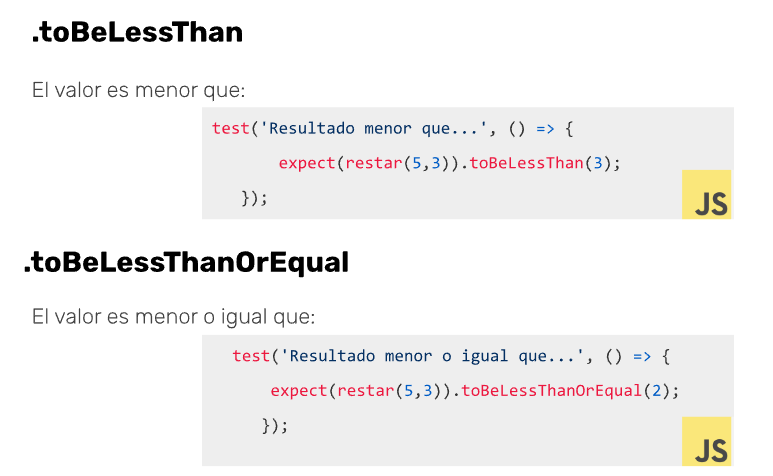


**MATCHERS CON JEST**

Jest utiliza matchers para probar los diferentes valores que puede tener un código:













**ARRAYS Y STRINGS**:



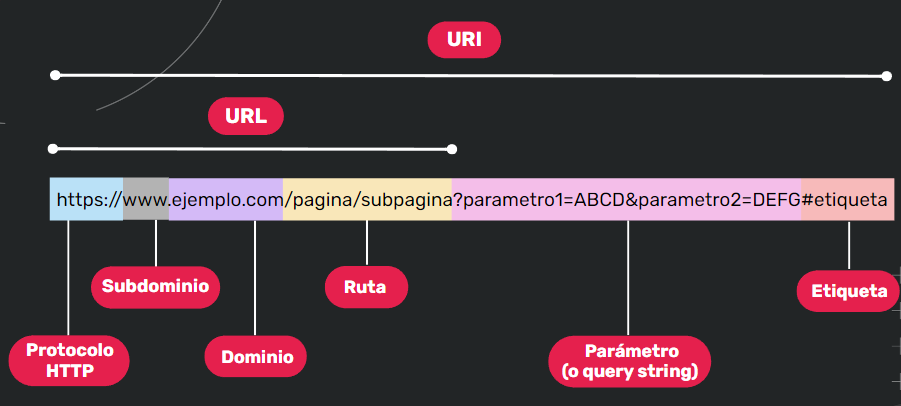


**TESTING DE APIS**

Estas pruebas consisten en hacer peticiones HTTP (get, post, put y delete) y luego verificar la respuesta.

**HTTP Y SUS MÉTODOS**

El protocolo de transferencia de hipertexto (*hypertext transfer protocol* o HTTP) es un sencillo protocolo cliente-servidor que articula los intercambios de información entre un servidor y una aplicación que consume estos servicios. Esta comunicación se logra gracias a los métodos HTTP, los cuales nos permiten enviar y recibir información.



* **URL** (Localizador de Recursos Uniforme) se utiliza para encontrar la ubicación de un recurso.
* **URI** (Identificador de Recursos Uniforme) es el nombre único con el que cada archivo se encuentra identificado a fin de ser encontrado rápidamente
* **Subdominio** de una URL es todo lo que aparece entre el protocolo y el primer punto que lo separa del dominio.
* **Dominio** es un nombre único que sirve para identificar una página web, por un lado, y encontrarla con más facilidad, por el otro
* **Ruta** normalmente indica páginas y subpáginas que podemos encontrar en un sitio web.
* **PARÁMETRO(O QUERY STRING)** Parámetro es lo que viene **después del signo de interrogación** ?. En una URL puede haber **varios parámetros**. Y, cuando es el caso, estos parámetros se separan con el símbolo de ampersand **&**. Los parámetros pueden indicar diferentes cosas: algunas veces tienen que ver con una búsqueda en el sitio, otras son parámetros de campañas publicitarias, etc.

**MÉTODOS GET Y POST**

Podemos verlo por la consola de desarrollador de Chrome o ingresando a [Postman API Platform](https://www.postman.com/) y pegando la url que nos pasan.

*Postman Tests* permite asegurarnos de que un API funciona como se esperaba. Nos permite establecer que las integraciones entre los servicios funcionen de manera confiable y verificar que los nuevos desarrollos no hayan roto ninguna funcionalidad existente. Nos ayuda a verificar resultados, como el estado exitoso o fallido, la comparación de los resultados esperados, etc.

**GET**: Se utiliza para recuperar información de una URL específica y analizar la información obtenida a partir de los test.

**POST**: A través de esta solicitud enviamos los datos y el api nos devuelve una respuesta que valida que la creación sea exitosa.

Status:

* 200 ok. Respuestas satisfactorias. Significa que la solicitud se realizó correctamente.
* 300 : Redirecciones.
* 400 Bad Request: Errores de cliente. Esta respuesta significa que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.

En Postman:

Cada uno de los test es ejecutado con el objeto pm y en concreto con el método .test().

Para poder acceder al contenido de la respuesta de las invocaciones tenemos el objeto pm.response y su método .json() que nos permitirán acceder a los elementos de la respuesta en JSON.

Otro método importante es el que nos permite realizar una comprobación de contenido, este es pm.expect.