

1 -. Razones para las pruebas

- Tenemos problemas o el resultado no es el esperado
- Costo alto o fuera de presupuesto
- Implicaciones legales o de estándares tecnológicos

Para el proceso de pruebas es necesario contar con al menos 3 elementos que lograrán cumplir nuestros objetivos:

Metodología: Es donde estableces el criterio o estrategia de cómo se llevarán a cabo las pruebas.

Recursos: Si quieres realizar pruebas y no estás preparado, con el tiempo esas carencias se pueden visualizar como defectos.

Herramientas: Nos ayudarán a optimizar nuestro trabajo.

Definir la calidad del producto: Si estas creando el software correctamente.

Calidad del proceso: Si alguien de tu equipo no te especifica lo que quieres no vas a poder crear un producto que cubra sus necesidades.

Certificaciones, estándares y metodologías para:

- Para individuos
- Para procesos
- Para empresas
- Para servicios/productos = software/hardware
- Para tipo de industrias

2 -. Ciclo de vida del software



Definición de objetivos:

En esta fase se define el alcance general del software y su papel dentro de una estrategia global o dentro del ecosistema que va a funcionar.

Análisis de los requisitos y su viabilidad:

Se trata de recopilar la mayor cantidad de información posible para evaluar la viabilidad del producto, encontrar posibles restricciones y analizar todos los requisitos del cliente.

Diseño:

Alto nivel: Se trata de realizar un diseño básico que valide la arquitectura de la aplicación.

Bajo nivel: Es una definición detallada de la estructura de la aplicación basada en el diseño general.

Programación:

Es la implementación de un lenguaje de programación para crear las funciones definidas durante la etapa de diseño.

Pruebas de verificación:

Aunque en todas las fases anteriores se hacen pruebas en esta fase se cubren: pruebas de componentes, integrales y de sistema.

Prueba beta (o validación):

Se hace para garantizar que el software cumple con las especificaciones originales o también se hacen las pruebas de aceptación.

Implementación:

Se realiza una prueba del sistema implementado para encontrar posibles fallas en la Implementación.

Mantenimiento:

Se hace para todos los procedimientos correctivos (mantenimiento correctivo) y a las actualizaciones secundarias del software (mantenimiento continuo), junto con la actualización de las pruebas.

3 -. Calidad y Defectos

La calidad es una percepción entre lo deseado, analizado y lo que vamos a entregar. La calidad la define el cliente, si esa persona está satisfecha con lo entregado hasta ahí llega la calidad.

Verificación: Es ir en cada etapa revisando que se cumpla lo propuesto por el cliente.

Validación: Antes de entregar al cliente, validamos que efectivamente el conjunto de requerimientos está siendo cumplido con lo entregado.

Anomalía: la manifestación de un error en el software.

Error: una acción humana que produce un resultado incorrecto.

Defecto: imperfección o deficiencia, el cual no cumple sus requerimientos o especificaciones y necesita ser reparado o remplazado.

Fallo: el cese de la habilidad de un producto de cumplir una función requerida o su inhabilidad de funcionar dentro de márgenes previamente especificados.

Problema: dificultad o incertidumbre experimentada por una o más personas, como resultado de un encuentro insatisfactorio con el sistema usado.

Ya entendimos que es necesario definir un proceso que asegure un buen desarrollo del producto, que los estándares o herramientas implementadas deben ayudar a cubrir las métricas definidas para entonces evaluar si el producto tiene el porcentaje o grado de calidad esperado.

4 -. Testing Moderno y las Especialidades de Testing

7 principios del testing moderno:

- **Nuestra prioridad es mejorar el negocio:** El producto que se va a entregar al cliente permitirá hacer funcionar el negocio. Si en algún momento no quieres hacerlo, estás poniendo en riesgo ese negocio porque si el producto no se vende o no es aceptado la empresa puede cerrar o puedes perder el trabajo.
- Nosotros aceleramos el equipo y usamos modelos como **Lean Thinking** y **Teoría de las Restricciones** para ayudar a identificar, priorizar y mitigar cuellos de botella en el sistema:

Cuando queremos hacer algo, lo queremos hacer perfecto y eso puede ser demasiado. Deberías construir en base a procesos cortos para poder encontrar los defectos de una manera más rápida.

- Nosotros somos la fuerza para la mejora continua, ayudando al equipo a **adaptarse y optimizar** para tener éxito, en lugar de proporcionar una red de seguridad para detectar fallas: El cliente puede entender que el producto se va a liberar por fases, es importante que nosotros enfoquemos nuestras pruebas en cada una de esas fases. No tiene que ser todo al inicio y al final, debe haber una distribución que nos permita manejar el riesgo del software
- Nos preocupamos profundamente acerca de la **cultura de calidad** en nuestro equipo, y asesoramos, lideramos y nutrimos el equipo para llevarlos a una cultura de calidad más madura: Al inicio los testers eran personas desarrollando software y un día con tantos defectos y trabajo, separaron los roles para que así hubiese una persona dedicada a realizar las pruebas. El tester puede hacer recomendaciones de herramientas, mejorar el proceso o volverse un coach.
- Nosotros creemos que el cliente es el único capaz de **juzgar y evaluar** la calidad de nuestro producto: Si el cliente está satisfecho con lo entregado y cumple las expectativas entonces has alcanzado la calidad deseada.
- Nosotros usamos **datos de manera extensa y profunda** para entender los casos de uso del cliente y entonces cerrar huecos entre hipótesis del producto e impacto del negocio.
- Expandimos las habilidades de testing y el conocimiento en **todo el equipo**; entendemos que esto reduce o elimina la necesidad de un especialista dedicado al testing.

El tester debe dominar varias áreas necesita entender y tener toda la visión del producto y negocio. Saber sobre herramientas que optimicen el trabajo.

5 -. Especialidades del Testing:



Manual tester: Nos ayuda a definir los casos de pruebas, establecer estrategias. También ejecuta, pero lleva lo necesario para que todos sepan qué hacer.

- Pensamiento lateral, piensa fuera de la caja con una nueva perspectiva, agrega nuevos casos de usos y entiende muy bien al usuario.

Automation tester: Se encarga de agilizar, acelerar el trabajo y actividades que nos quitan la oportunidad de encontrar más casos de usos.

- Conoce de programación, no solo de forma básica, debe conocer cómo crear diseños de frameworks y soluciones. El código crece y las pruebas también, darles ese mantenimiento a las pruebas es un problema común.

Security tester: Encargado para el área de seguridad. Debe ser alguien que aporte valor desde la perspectiva de la seguridad.

- Protocolos, estándares, legalizaciones dependiendo de cada país y marca. Está enfocado en prever ataques, virus, problemas de seguridad, acceso no autorizado. Profundizar en las técnicas y prácticas de seguridad.

Data science tester: Con la manera en que crecen los datos en un proyecto, se necesita a alguien que los analice, agrupe y limpie estos datos.

- Análisis y Limpieza de datos, omite tener un set de pruebas amplio donde la variedad va a

permitir detectar defectos inesperados. Esto puede ser clave para que los resultados no sean falsos positivos

SDET: Es la combinación de un desarrollador que ya sabe hacer pruebas. Con la diferencia de automatiza y hace uso de herramientas que organizan la operación de la entrega de las pruebas. Esta persona se asegura de que las pruebas se ejecuten antes de liberar el código.

- El programador ahora sabe hacer pruebas y conoce de herramientas que le permite entregarlas de una manera automatizada.

DevOps: Conoce todo lo anterior y domina el conocimiento de automatizar el proceso, se asegura de una entrega continua.

- Una automatización de la operación, Entrega Continua. Donde se entregan de forma más rápida las nuevas versiones.

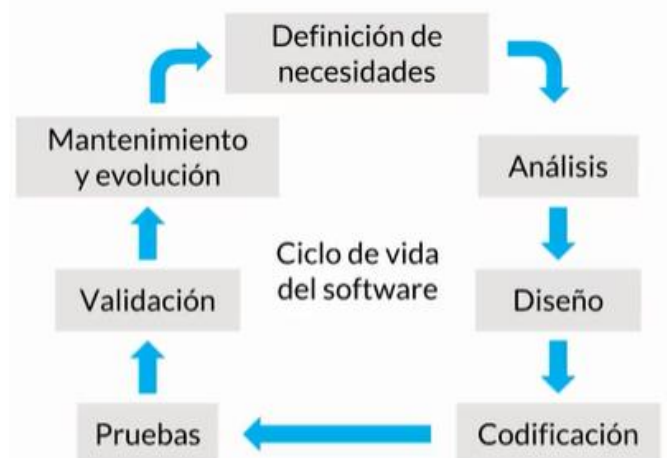
QA Engineer: Quality Assurance. Está enfocado en el producto y en el proceso.

- Procesos de Calidad

QE: Quality Engineer. Es como un coach, acompaña en las políticas de calidad en la empresa o equipo.

- Soluciones de estrategias de calidad.

Testing Durante el ciclo de vida del desarrollo se software



- **Análisis:** Documentación de las especificaciones de los requerimientos. Se pueden hacer

pruebas de las especificaciones, determinar las validaciones y el flujo de las validaciones.

- **Diseño:** Establecer los diseños visuales. Se puede especificar longitudes, si se aceptan números - etc. Mensajes de salida, que sucede si todo está bien, acepta datos null.
- **Código:** Se puede basarse en módulos - funciones - base de datos. Se puede basarse en backend.
- **Pruebas:** Pruebas de Interfaz - Pruebas del canal - Pruebas de dispositivos. Todo esto haría la revisión de los requerimientos, verificación y validación. Con el cliente se hacen las pruebas de aceptación.

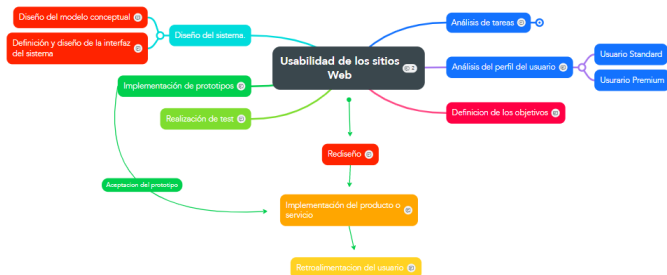
6 -. Estrategias de Pruebas

En este punto se tienen dos interrogantes:

- ¿Qué problema tenemos actualmente?
- ¿O que problemas debemos evitar?

Escenarios y Contextos

- Usabilidad



- Seguridad
- Arquitectura
- Performance
- Escalabilidad

7 -. Testing en desarrollo de software

Testing: Es la exploración de una idea, aprender a conocer como sucede el flujo, se generan datos, se llenan formularios. Esto y la exploración de pruebas nunca terminan, siempre hay nuevas formas de conocer como el usuario está usando el software

Checking: Es cuando sabes qué está pasando y te verificas que siga pasando. Como verificar una maleta antes de viajar para asegurarnos que guardamos todo.

- Solo se ejecutan si sucede algo
- Se ejecutan cada que... libero nuevo código o software.
- Se ejecutan de manera programada.

Desventajas del checking mal empleado

- Pobre cobertura de pruebas.
- Falta de actualización.
- Mal manejo de versiones.

Ventajas del checking bien empleado

- Correr pruebas en paralelo o en múltiples plataformas.
- Reducción del error humano.
- Probar grandes cantidades de datos.

8 -. Testing ágil

Involucra a todos, no solo en al tester, todos en el equipo son tester. La persona con este rol se asegura de la mayor cobertura de pruebas tomando en cuenta todas las necesidades que cada uno de los miembros del equipo también está intentando hacer que funcione. El tester tiene que definir si se está cumpliendo con los requerimientos funciones y los del equipo.

Estrategias Ágiles:

- El testing es de “todo el equipo”
- El testing puede ser independiente
- Integración continua
- Testing guiado por pruebas (Test Driven Development)
- Desarrollo guiado por comportamiento (Behaviour Driven Development)
- Desarrollo guiado por las pruebas de aceptación (Acceptance Test Driven Development)

9 -. Niveles de prueba

- Prueba de Componentes: Componentes son aquellas cosas pequeñas que sueles ver en un video como el botón de pausa, volumen, adelantar, retroceder. Cada una de esas acciones son un componente.
- Pruebas de integración: Una aplicación tiene una serie de componentes que trabajando juntos forman un pequeño sistema, pero

cuando tienes varios de estos sistemas necesitas integrarlos entre ellos.

- Prueba de sistema: Esta parte incluye que estás tomando en cuenta el contexto, no es lo mismo tener las mismas pruebas para iOS, Android y Web.
- Pruebas de aceptación: Si ya probamos que nuestro elemento o acción funcionan, estás pruebas nos aseguran con o sin el cliente que tiene cubierta todas las necesidades requeridas en el software. Es la última verificación.

9 -. Tipos de Pruebas

- **Pruebas funcionales:** Cómo funciona un sistema, qué debe estar haciendo, cómo está interactuando el usuario con él. (caja negra)
- **Pruebas no-funcionales:** El usuario puede estar experimentando otro tipo de cosas que aún funcionando puede tener la sensación de lentitud, falta de legibilidad o claridad. Esas características de usabilidad están asociadas a estas pruebas. (caja negra)
- **Pruebas estructurales:** Tienen que ver con la tecnología y el stack usado para construir nuestro producto. Nos tenemos que asegurar que nuestra base de datos o servidor funcionen de la manera correcta. Son conocidas como pruebas de caja blanca.
- **Prueba de manejo de cambios:** Es probar nuevamente un componente ya probado para verificar que no ha sido impactado por actualizaciones
- **Pruebas estáticas:** Muchas veces no son consideradas en los proyectos porque significa revisar código, documentación, verificar información documentada de la forma correcta.
- **Pruebas dinámicas:** Se enfocan en comportamientos externos visibles durante la ejecución del software.

10 -. ¿Qué son los elementos?

- Contratos, planes y calendarios del proyecto, así como su presupuesto.
- El análisis de requerimientos.
- Especificaciones o reglas de negocio:
 - Técnicos
 - Seguridad

- Las definiciones de:
 - Historias del usuario
 - Criterios de aceptación
 - Mockups
- El diseño de arquitectura
- Las pruebas (Testware), puntos de verificación CI
- Guías de usuario
- Evaluación/revisión del código

Beneficios

- Detectar y corregir defectos de manera más eficiente
- Identificar y priorizar la ejecución de pruebas en etapas posteriores.
- Prevenir defectos
 - Que no son fácilmente detectables durante las pruebas dinámicas
 - Durante la etapa de análisis y diseño
- Cubrir aspectos como:
 - Inconsistencias, ambigüedades, contradicciones, definiciones inexactas, requerimientos redundantes.
- Reducir el retrabajo e incrementar la productividad.
- Reducir el costo el tiempo.
- Mejorando la comunicación entre todos los miembros del equipo.

11 -. Definición y diseño de pruebas

¿Qué hace un tester?

- Encontrar problemas
 - Si no encuentra problemas antes de que el producto sea entregado al cliente, entonces su testing es ineficiente.
- Documentar problemas
 - Si cuando encuentra problemas no sabe documentar y reproducir los pasos correctos su testing genera retrabajo y sube el costo.
- Comunicar problemas
 - Si como representante de la calidad del producto no sabe argumentar y proteger los intereses del negocio o los clientes, entonces su testing no agrega valor.

12 -. Pruebas de Caja

Negra (frontend): No podemos observar como fue construida, no vemos el código, no sabemos su arquitectura, no tendemos nociones mas que la interfaz que estamos interactuando.

- Participa de equivalencia
- Valores limite
- Tabla de decisiones
- Transición de decisiones
- Casos de usos

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

- Cobertura de declaración
- Cobertura de decisiones o código
 - Sentencias
 - Decisiones
 - Condiciones

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

- Casos de negocios
- Pruebas end-to-end
- Pruebas de integración

13 -. Gestión monitoreo y control

Etapas:

- **Planeación:** Definir los objetivos de las pruebas es muy importante, al no tener una estrategia clara termina causando una pobre cobertura de pruebas. Los elementos a considerar para una buena planeación son la estimación, recursos, el alcance y objetivo.
- **Monitoreo y Control:** Durante el monitoreo lo que estamos buscando son esas métricas que nos digan si estamos llevando avances o tenemos retrasos. Son nuestras alertas cuando nuestro plan no se está ejecutando.
- **Análisis:** Incluye decidir cuáles son esas prioridades que nos ayudará a definir qué debemos probar.

- **Diseño:** Normalmente cuándo estas iniciando las pruebas es necesario crear un mapa de ideas. Después de esto, puedes realizar el diseño a detalle de qué va a incluir cada caso de uso.
 - Casos de alto nivel.
 - Diseñar y priorizar pruebas.
 - Identificar el entorno de pruebas.
 - Hacer una trazabilidad entre pruebas y sus condiciones.
- **Implementación:** También nos aseguramos de contar con la estructura necesaria para realizar las pruebas, con un ambiente, datos y dónde documentar o realizar las pruebas.
- **Ejecución:** En esta etapa las suites de pruebas se ejecutan de acuerdo con el programa o el plan diseñado con anterioridad. Se suelen agrupar los casos de pruebas para que no estén desorganizado y podemos hacerles un buen seguimiento.
- **Finalización:** Cuando queremos cerrar el ciclo de las pruebas, necesitamos saber qué porcentaje se cubrió, ejecutó, cuántos defectos se derivaron, aprender lecciones sobre el proceso.

14 -. Roles y responsabilidades



- **Especialista en pruebas manuales:** se enfoca en la estrategia, definición, ejecución y cobertura de pruebas para cumplir los requerimientos, echando mano de cualquier técnica para obtener información suficiente y así cumplir con las asignaciones correspondientes.

- **Especialista en pruebas técnicas.** trabaja muy de cerca con el tester manual, mientras que el tester manual define las pruebas, el tester técnico acelera la capacidad de ejecución de las pruebas. Esto lo hace implementando herramientas que permitan la automatización de pruebas, o la correcta selección de datos de pruebas, o el monitoreo de la ejecución de las pruebas.
- **Líder del equipo de pruebas.** generalmente dentro de sus responsabilidades es volverse un facilitador de servicios, información y herramientas para el equipo de pruebas, para poder estimar presupuestos, recursos y tiempos respecto al plan de desarrollo de software.
- **Ingeniero de calidad.** ya no solamente está al pendiente del producto y los procesos, comienza a involucrarse más con el negocio, ayudando tanto a testers como cualquier otro miembro del equipo a llevar cabo pruebas que reduzcan, en todas las etapas del ciclo de vida del software, el error humano.

15 -. Retrabajo

Es la principal causa del retraso, de que la estimación de tiempo falle, de que costos suban. Cuando estos suceden aumentamos exponencialmente el trabajo de todos

- Falta o mala documentación
- Falta de capacitación o dominio en las herramientas utilizadas
- Falta de capacitación o dominio en el software a desarrollar
- Falta de comunicación

15.1 -. Gestión de Bugs

La mala administración, malas prácticas o falta de seguimiento entorpece las tareas de todo el equipo sino que además sumamos el retrabajo en la mala documentación puede que nuestro proyecto se salga de presupuesto o tiempo.

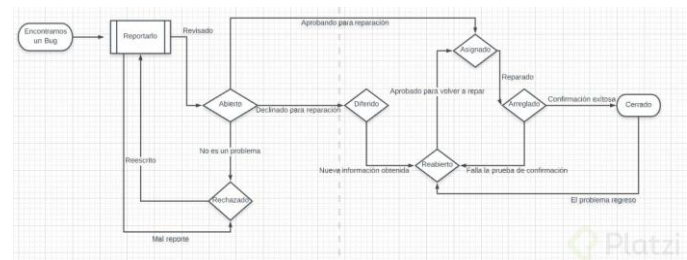
Razones por las que aparecen defectos:

- Hay presión de tiempo en la entrega del software

- Descuidos en el diseño
- Inexperiencia o falta de conocimiento
- Falta de comunicación en los requerimientos
- Diseño complejo de código
- Desconocimiento de las tecnologías usadas

Preguntas a realizar para construir un proceso de gestión de bugs:

- ¿Qué debe de hacer la persona que encuentre el defecto?
- ¿En qué herramienta debe documentar el defecto?
- ¿Cómo vamos a almacenar la información?
- ¿Qué información requiere el equipo de desarrollo para poder resolver un defecto?
- ¿Cuáles son los estatus que se manejan para que fluya la resolución del defecto?
- ¿Cuáles son los criterios de aceptación de cierre del defecto?



Repositorio y monitoreo de defectos

Una vez instaurado el proceso de gestión de bugs, también se debe precisar quien tiene acceso a los bugs y cuales son los permisos que tiene, por cuanto tiempo se almacenan, etc.

15.2 -. Defectos y sugerencias

Defectos: Es aquello que no cumple los requerimientos funciones, de diseño, de arquitectura y es la consecuencia de un error humano en el código o la interpretación de la información.


Sugerencias: Es cómo la experiencia del usuario se ve afectado. La lentitud del proyecto, la legibilidad, combinación de colores, la forma de navegar no es adecuada.

Sugerencias convertidas en defectos / actualizaciones de software

- Hace lenta la operación

- Detiene parcial o totalmente el proceso
- El contenido o el flujo confunde al usuario
- Deja cometer muchos errores al usuario
- La traducción o el lenguaje empleado no es correcto
- No funciona sin internet

¿Como reportar una sugerencia o defecto?

Sistema Operativo Windows 10 Plataforma: Laptop Hora: 00:00 Precondición: debemos tener registrado. Dato de prueba correo registrado: xxx@gmail.com						
Nombre	Descripción	Pasos	Resultados esperados	Resultados Actuales	Elemento Adjunto	Defecto
Mensaje erroneo al registrar el correo duplicado	Mensaje de error con cuenta de correo	1.- Ir al sitio http**** 2.- Dar click en la opcion de registro 3.- Ingresar cuenta duplicada				109
		4.- Dar click en el boton registrarse	La cuenta ha sido dada de alta. Recibo un mensaje de confirmacion	Recibi un mensaje de error: "Un error ha ocurrido"		100

11 -. Depuración (Debugging)

Debugger: Es una herramienta que nos ayuda a encontrar todos estos errores ya sea de sintaxis, advertencias de seguridad, etc. Nos permite ejecutar línea por línea, detener la ejecución temporalmente, visualizar el contenido de las variables, cambiar el valor del entorno de ejecución para poder ver el efecto de una corrección en el programa.

Beneficiados

- Programador: Requiere cada vez que programa ir depurando lo que ejecuta y escriba para que cumpla con su objetivo
- Tester: Le ayuda a reducir el tiempo de análisis que después puede ser asignado para el desarrollador
- Analista: Puede ser para analisis de encontrar información de un historial sobre cómo se comporta un sistema.

Objetivo: Vamos a analizar cómo se comporta el sistema, cómo se transfieren los datos, cómo se procesa la información. Tenemos la capacidad de tener nuestro código en cualquier momento para conocer cómo funciona.

Errores: Generan oportunidades de mejora

Síntomas de Errores

- Obtención de salida incorrecta.
- Realización de operaciones fuera de los normal.

- No finalización del programa (ciclos infinitos, etc.).
- Caídas del programa.

El depurador (Debugger) permite:

- Ejecutar línea por línea.
- Detener la ejecución temporalmente en una línea de código concreta o bajo determinadas condiciones.
- Visualizar el contenido de las variables.
- Cambiar el valor del entorno de ejecución para poder ver el defecto de una corrección en el programa.

Tipos de herramientas:

- Debugger
- Manual
- Local / Remota

Herramientas:

- Mensaje de advertencia
- Estándares de compilación
- Verificación sintáctica y lógica

11.1 -. Técnicas de Depuración

Las técnicas de depuración deben ir cambiando de ser reactivas a ser preventivas. Debemos recordar que parte de los principios del testing moderno es tratar de ir corrigiendo nuestras técnicas, implementar mejores prácticas y hacer uso de mejores herramientas. El debugging debería ser la última técnica que utilizas.

Técnicas de depuración:

- **Debugging:** Observar valores de variables, detener temporalmente la aplicación.
- **Logs:** Hacer un vaciado de cómo las variables van cambiando y es más fácil rastrear la información.
 - Desventajas de no usar Logs:
 - Visibilidad nula de errores.
 - Metodología de trabajo no estandarizada.
 - Accesos e información descentralizada.
 - Incremento del tiempo de respuesta.

- **Historial:** Agiliza la forma de monitorear y observar los comportamientos de nuestro software. Comparando valores, agrupando información.
 - Ventajas de generar un historial/reporte:
 - Aplicar técnicas de Machine Learning.
 - Mejorar la gestión y el control de la información.
 - Detectar amenazas de red o virus.
 - Prevenir fugas de información, así como comportamientos inadecuados.
- **Reportes:** Observar anomalías, acelerar el tiempo de respuesta, prevenir ataques o fallas.

Pasos para depurar:

- Ir al módulo que falla
- Establecer breakpoints
- Diseñar una matrix de pruebas
- Establecer los datos de prueba
- Comenzar a depurar

Fase 1 Encontrar el error

Pasos para depurar:

1. Ir al módulo que falla
2. Establecer breakpoints:
 - a. En asignación de valores
 - b. Procesamientos de valores
 - c. Cambio de estados
3. Diseñar una matriz de pruebas
4. Establecer los datos de prueba
5. Comenzar a depurar

Fase 2 Corregir el error:

12 -. Pruebas de verificación

- Tratan de reproducir el escenario fallido con los datos usados. Pero sería un error usar los mismos datos para después asumir que el error fue corregido.
- Se buscan nuevos escenarios donde se utilicen valores relativos. Como Otras plataformas,

otros sistemas operativos, otros exploradores, otros dispositivos.

- Otras plataformas
- Otros síntomas operativos
- Otros exploradores
- Otros dispositivos

Pruebas de regresión

- Las matrices de pruebas cuando se implementan otros dispositivos u otros exploradores nos ayuda a tenerlos en cuenta nuestros puntos de verificación para que no sufran un impacto.
- La matriz de prueba nos funciona para casos donde no solo vemos los defectos, sino que todo lo que ya funciona siga funcionando.
- Nos ayuda a tener una claridad con los casos de prueba claves que pueden ser automatizados.

Documentación

- Comentarios en el código
- Documentación técnica
- Pruebas unitarias
- Pruebas específicas
- Matrices de pruebas
- Plan de pruebas

13 -. Automatización de pruebas

¿Cuándo estamos listos para automatizar?

- ✓ Tenemos pruebas repetitivas
 - ✗ Pero no están identificadas
- ✓ Buscamos optimizar la ejecución de pruebas
 - ✗ Solo escribimos script sin agrupar
- ✓ Hemos definido un Framework
 - ✗ No se estandarizan las pruebas

Pruebas unitarias:

Tienen que ver con un pedazo de código que el desarrollador está codificando, pero no tienen que ver con todo el flujo de negocio y proceso del software.

Pruebas de integración:

Cómo hacemos que el conjunto del equipo que libera pedacitos de software funcionen juntos y no hagan defectos adicionales.

Pruebas funcionales o de aceptación:

Estas pruebas no necesariamente forman parte de los requerimientos especificados por el cliente, una recomendación para automatizar estas pruebas es que deban cumplir con los requerimientos dados por el cliente.

TDD (Test Driven Development):

El desarrollo va a estar enfocado haciendo primera las pruebas y después el código. Haciendo que el desarrollo sea muy específico con la mayor cobertura y no pongamos líneas de código que no van a funcionar o no se usan.

Proceso TDD

- Escribimos una prueba
- Ejecutamos la prueba: Falla
- Se escribe el código
- Ejecutamos la prueba: Pasa

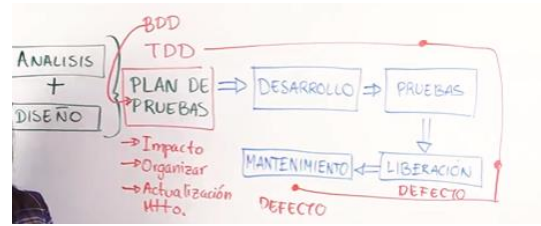
BDD (Behavior Driven Development):

Si primeros vamos a escribir las pruebas, debemos hacerlo bien y usando un lenguaje sencillo, simple para que la sirva al equipo para entender qué es lo que queremos hacer.

BDD es el desarrollo guiado por el comportamiento. Es un proceso que proviene de la evolución del TDD.

En BDD también se escriben pruebas antes del código. Pero en vez de ser pruebas unitarias son pruebas que van a verificar que el comportamiento del código es correcto desde el punto de vista de negocio.

“Entre mas claros los casos de prueba, mas eficiente la cobertura de pruebas. Entre menos errores o ambigüedad tengan los casos de pruebas, son más fácil de ejecutar o automatizar”.



14 -. Gherkin

es un lenguaje de texto plano con estructura, usamos palabras que no son comandos, pero permiten entender en un modo de pseudocódigo qué es lo que se tiene que hacer. Está diseñado para ser fácil de aprender y ser entendido por todos.

Ventajas

- Simple
- Palabras claves o keywords
- Estandariza los casos de uso
- Reduce el tiempo de diseño

Principales keywords usados en Gherkin

- Feature: El usuario abre 1 puerta de perilla para salir # comentarios
- Scenario: El usuario tiene una puerta cerrada.
- Given: Girando la perilla, And: empujando la puerta, When: La puerta abre hacia afuera, then: La puerta queda abierta, And: el usuario puede salir, But (steps)
- Background
- Sceneario outline
- Examples