Challenge Conexa

Wagner Brenda

Parte 1: Análisis y justificación

1. Actividades de planificación de pruebas y su impacto

Mapeo de dependencias entre equipos y componentes:
Antes de planificar nada, necesito saber de quién depende el desarrollo que vamos a testear y qué puede romperse si algo falla

Diseño de una estrategia de cobertura progresiva:
No testeo todo al mismo tiempo. Divido el esfuerzo en capas: core, flujos críticos, periféricos, establezco qué se automatiza y qué no, qué debería ser manual, qué conviene testear en emuladores y qué quizás no. Así se reduce el tiempo de feedback sin sacrificar profundidad.

Relevamiento de requerimientos:
No se trata solo de entender lo que se pide, sino de detectar supuestos no explicitados y traducir lo ambiguo. Esta etapa permite anticipar posibles desvíos y empezar a construir calidad desde antes de iniciar a desarrollar.

Acuerdos sobre datos de prueba, herramientas y ambientes compartidos:
Sin datos limpios o representativos, las pruebas pierden valor, definir las herramientas adecuadas y validar datos permite detectar errores reales y no falsos positivos causados por condiciones mal preparadas.

Identificación de escenarios de negocio prioritarios:
Pensar en el cliente para el que se trabaja es fundamental: qué genera ingresos? Qué no puede fallar nunca? Lo funcional no siempre es lo más importante: lo crítico sí. Entender el negocio es prioritario siempre.

2. Cinco aspectos a evaluar en pruebas no funcionales

1. **Test de estrés:**
   Un sistema puede ser funcionalmente impecable pero colapsar ante 200 usuarios simultáneos.

2. **Seguridad (autenticación, autorización, inyecciones):**
   No hay calidad sin seguridad. Si un usuario puede ver o modificar datos que no le pertenecen, se pierde confianza. Testear ataques comunes, fugas de sesión y

accesos indebidos es clave.

3. **Compatibilidad entre navegadores y dispositivos:**
   Más aún si el producto está orientado a consumidores finales. Verificar comportamiento en distintas plataformas asegura cobertura real de uso. El "funciona solo en Chrome" no es aceptable en producción.

4. **Accesibilidad:**
   No es solo un requerimiento técnico o legal: es una cuestión ética (yo tengo hipoacusia moderada por ej y para mi es importante que a veces cuando el sonido está muy bajo exista la posibilidad de agregar subtítulos a los videos), también validar uso con teclado, contraste adecuado, lectores de pantalla y navegación fluida aporta valor real a la inclusión.

5. **Performance:**
   Cuán rápido responde la UX al usuario, qué tan rápido cargar la web o la app es fundamental para que la experiencia al utilizar la web o la app sea lo más agradable al usuario posible.

3. Tres tipos de pruebas funcionales y su aplicación

1. **Smoke testing:**
   Apenas se despliega algo nuevo, corro un conjunto mínimo para ver si el sistema o la app funciona. Útil en cualquier build rápido o integración continua y permite detectar errores críticos antes de invertir más tiempo.

2. **Pruebas de flujo extremo a extremo (E2E):**
   Las uso cuando quiero simular una experiencia completa de usuario: desde que entra al sitio hasta que paga o se registra. Impacto: asegura que las piezas se integran bien y los flujos reales funcionan.

3. **Pruebas de borde (edge cases):**
   Qué pasa si el usuario carga 909 productos al carrito? Y si pone emojis en su nombre? esto previene errores que no están en la documentación, pero sí en la vida real.

4. Diferencia entre regresión y verificación

- **Regresión:** Evalúa que nada se haya roto después de un cambio.

- **Verificación:** Confirma que lo nuevo se implementó como se definió.

Ejemplo:
Implementás un cambio en la lógica de impuestos de un checkout.

La verificación pasa si el total se calcula bien con los nuevos porcentajes.

Pero si eso rompió la integración con el sistema de facturación (que usaba la lógica anterior), entonces va a fallar la regresión. Una puede salir bien, mientras la otra estalla. Ambas son funcionales, pero una mira adelante (verificación) y la otra para atrás (regresión).

5. Tipo de prueba al revisar documentación o requisitos

Cuando reviso documentación, requerimientos o casos de uso, estoy haciendo una revisión estática.

- Buscar ambigüedades (“login exitoso” no dice mucho).
- Sugerir mejoras en casos de uso (incluir validaciones, errores esperados).
- Identificar omisiones (qué pasa si el usuario cancela el flujo?).

En mi experiencia diría que el 70% de los errores nace antes de que alguien escriba código. Cuestionar lo que está en el ticket de jira o US reduce bugs que después aparecen en testing funcional. A largo plazo ahorra tiempo, recursos y conflictos entre QA, Devs y Producto.