RETO 4

1. "El equipo de pruebas utiliza el mismo entorno que los usuarios finales para realizar las pruebas de performance."

- **Evaluación**: Esta práctica es adecuada, ya que asegura que las pruebas de rendimiento simulan el entorno real de los usuarios. Probar en un entorno similar o idéntico ayuda a detectar problemas de rendimiento que podrían no aparecer en entornos de prueba controlados.
- **Alternativa/Mejora**:
  - **Pruebas de rendimiento escaladas**: Además de pruebas en entornos similares a los de los usuarios finales, sería ideal realizar pruebas en escenarios de alto tráfico (concurrentes) que simulen picos en la carga, especialmente en eventos clave como promociones o días de ofertas.
  - Usar herramientas como **JMeter** o **Gatling** para simular un gran número de usuarios y medir el comportamiento del sistema bajo estrés.

2. "Las pruebas de humo se ejecutan para cubrir todas las posibles opciones del usuario y se realizan en cada implementación de una nueva versión, todas las pruebas se realizan manualmente para reflejar la misma experiencia del usuario."

- **Evaluación**: Aunque realizar pruebas de humo para cada versión es una buena práctica, realizarlas **manualmente** para todas las opciones del usuario puede ser **ineficiente y propenso a errores** humanos. Además, el objetivo de las pruebas de humo es verificar que las funciones principales del sistema funcionan correctamente, no cubrir todas las opciones de usuario.
- **Alternativa/Mejora**:
  - **Automatización de pruebas de humo**: Las pruebas de humo deben automatizarse en la medida de lo posible, especialmente para escenarios críticos y repetitivos. Usar herramientas como **Selenium**, **Cypress** o **TestCafe** puede reducir significativamente el tiempo de ejecución y aumentar la confiabilidad.
  - **Pruebas centradas en flujos críticos**: Las pruebas de humo deben enfocarse en los flujos más importantes del sistema (inicio de sesión, búsqueda, compra) en lugar de cubrir todos los posibles escenarios de usuario.

3. "Las pruebas de regresión solo cubren los módulos que probablemente se vean afectados por los cambios realizados en la última versión."

- **Evaluación**: Enfocar las pruebas de regresión solo en los módulos afectados puede ser útil para ahorrar tiempo, pero también es

arriesgado, ya que los cambios pueden tener efectos no previstos en otras áreas del sistema.

- **Alternativa/Mejora**:
  - **Pruebas de regresión automatizadas**: Implementar una suite de pruebas de regresión automatizadas que cubra **todos los módulos críticos** del sistema, no solo los afectados por el cambio. Esto asegura que cualquier cambio no afecte funcionalidad existente.
  - Aplicar **pruebas de regresión basadas en riesgos**: Priorizando los casos de prueba en función de la probabilidad y el impacto de fallos, para optimizar el tiempo de prueba sin dejar de lado la cobertura.

4. "Los casos de prueba se crean en Excel para que sean fáciles de editar. Los casos de prueba se eliminan después de la implementación de esa versión, por lo que se crean nuevos completos en cada iteración."

- **Evaluación**: Usar Excel puede ser flexible a corto plazo, pero a largo plazo puede resultar **ineficiente** y **difícil de mantener**, ya que no permite una gestión centralizada ni la reutilización de casos de prueba. Eliminar los casos de prueba después de cada versión **pierde historial** valioso y puede generar esfuerzo duplicado en cada iteración.
- **Alternativa/Mejora**:
  - Usar una herramienta de gestión de pruebas como **TestRail**, **qTest**, **Zephyr** o **Xray** (integrada con JIRA). Estas herramientas permiten gestionar, versionar y reutilizar casos de prueba fácilmente.
  - **Reutilización de casos de prueba**: Los casos de prueba deberían mantenerse y actualizarse a medida que cambian los requisitos, en lugar de crearse desde cero en cada iteración. Esto ahorra tiempo y asegura consistencia.

5. "Las pruebas comienzan a estar involucradas en el proceso de desarrollo desde la etapa inicial."

- **Evaluación**: Esta es una excelente práctica, totalmente alineada con los principios de **Agile Testing**. La participación temprana de los testers permite identificar posibles defectos desde el principio y prevenir problemas costosos más adelante.
- **Alternativa/Mejora**:
  - **Test-Driven Development (TDD)** o **Behavior-Driven Development (BDD)**: Si no se está implementando ya, sería beneficioso aplicar estas técnicas para garantizar que los criterios de aceptación de las historias de usuario estén bien definidos y validados antes de que comience el desarrollo.

Resumen de las mejoras propuestas:

1. **Pruebas de rendimiento escaladas** para escenarios de alto tráfico, además de usar el entorno de usuario final.
2. **Automatización de pruebas de humo** y enfoque en flujos críticos para optimizar tiempo y evitar errores manuales.
3. **Automatización de pruebas de regresión** que cubra todos los módulos críticos, y no solo los módulos afectados.
4. Uso de **herramientas de gestión de pruebas** en lugar de Excel, y **reutilización** de casos de prueba a lo largo de las iteraciones.
5. Continuar con la práctica de involucrar las pruebas desde la fase inicial, con posibilidad de implementar **TDD/BDD**.

Estas mejoras ayudarían a aumentar la eficiencia del equipo de pruebas y a mejorar la cobertura de calidad en cada iteración de la plataforma, permitiendo detectar problemas más rápido y con menos esfuerzo manual.