

Bienvenido a QA/Pruebas de software

- QA es gestión de riesgos para la calidad del producto.
- Las pruebas son un deporte de equipo en SDLC
- Aprenderás manual + automatización + ingeniería de calidad.

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level> | <https://agilemanifesto.org/>

Cómo utilizar este curso

- Primero mire las diapositivas y luego practique las prácticas de laboratorio.
- Utilice guiones como guía de narración
- Construya un proyecto de cartera según el capítulo 12

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Evergreen + Temas de actualidad cubiertos

- Evergreen: diseño de pruebas, riesgo, defectos, estrategia de automatización.
- Tendencias: desplazamiento a la izquierda, puertas de calidad CI/CD, observabilidad
- Herramientas modernas: Playwright, API/pruebas de contrato, k6

Referencias: <https://dora.dev/> | <https://playwright.dev/docs/intro>

Resultados de aprendizaje

- Diseñe pruebas efectivas con una cobertura clara
- Cree comprobaciones automatizadas de UI/API mantenibles
- Integre puertas de calidad en los procesos de CI/CD

Referencias: <https://docs.github.com/en/actions> |
<https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>

¿Qué son las pruebas?

- Las pruebas evalúan la calidad y el riesgo del software.
- Las pruebas proporcionan información, no una prueba absoluta
- La calidad incluye atributos funcionales y no funcionales.

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level> | <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>

Atributos de Calidad (ISO 25010)

- Los atributos clave incluyen rendimiento, seguridad y mantenibilidad.
- La corrección funcional es sólo una parte de la calidad
- La estrategia de prueba debe asignarse a los atributos objetivo.

Referencias: <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>

Principios de prueba (orientados a ISTQB)

- Las pruebas tempranas ahorran costos y retrabajos
- Los defectos se agrupan en una pequeña cantidad de módulos.
- Las pruebas se desgastan y necesitan revisión periódica

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Pruebas estáticas versus dinámicas

- Estático: revisiones, tutoriales, linters, SAST
- Dinámico: ejecutar software con casos de prueba.
- Ambos son complementarios y deben coexistir.

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level> | <https://owasp.org/www-project-top-ten/>

Niveles y tipos de pruebas

- Niveles: componente, integración, sistema, aceptación.
- Tipos: funcional, no funcional, estructural.
- Seleccione el nivel/tipo según el riesgo y la velocidad de retroalimentación

Referencias: <https://martinfowler.com/articles/practical-test-pyramid.html> | <https://www.selenium.dev/documentation/>

Diseño de casos de prueba útiles

- Partir de los requisitos y criterios de aceptación.
- Incluir escenarios positivos, negativos y marginales.
- Mantenga las pruebas independientes y reproducibles

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Partición de equivalencia

- Agrupar las aportaciones en clases de comportamiento esperado.
- Pruebe un representante por clase válida/inválida
- Reduce el recuento de casos preservando la cobertura

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Análisis de valor límite

- Los defectos suelen ocurrir cerca de los límites de entrada
- Pruebe los valores mínimo, máximo y justo dentro/fuera
- Aplicar a rangos, duraciones, fechas y límites.

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Prueba de tabla de decisiones

- Usar reglas para combinaciones de condición/acción
- Excelente para precios, elegibilidad y permisos.
- Garantiza la integridad de las reglas de negocio

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Pruebas de transición estatal

- Estados del modelo y transiciones permitidas.
- Probar cambios de estado válidos e inválidos
- Útil para flujos de trabajo y sistemas de ciclo de vida

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Conceptos básicos de las pruebas web

- Verificar rutas críticas de UX y flujos centrales
- Verifique el comportamiento responsivo y en varios navegadores
- Incluir comprobaciones de accesibilidad y estado de error.

Referencias: <https://www.selenium.dev/documentation/> |
<https://web.dev/learn/accessibility>

Fundamentos de las pruebas API

- Validar estado, esquema y reglas comerciales.
- Verifique los contratos de idempotencia, autenticación y error
- Automatice las comprobaciones de API para obtener comentarios rápidos

Referencias: <https://learning.postman.com/docs/tests-and-scripts/write-scripts/test-scripts/>

Pruebas por contrato (CDC)

- Verificar los contratos API productor-consumidor
- Capte la tendencia de la integración antes
- Reducir la dependencia del E2E completo y frágil

Referencias: <https://pact.io/> | <https://docs.pact.io>

Pruebas exploratorias para Web/API

- Utilice cartas para centrarse en la exploración con límites de tiempo
- Documente observaciones y nuevos riesgos rápidamente
- Introducir los hallazgos en suites de regresión

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Casos de seguridad y abuso

- Incluir pruebas de autenticación, autorización, inyección y sesión.
- Asigne comprobaciones críticas al Top 10 de OWASP
- Priorizar por impacto y exposición empresarial

Referencias: <https://owasp.org/www-project-top-ten/>

La estrategia de automatización primero

- Automatice comprobaciones estables, repetibles y de alto valor
- Mantenga pruebas manuales para descubrimiento y juicio de UX
- Trate la automatización como un código de producto de ingeniería

Referencias: <https://martinfowler.com/articles/practical-test-pyramid.html>

Pirámide de pruebas en la práctica

- Comprobaciones de unidad/servicio más rápidas, menos UI E2E
- Las pruebas de UI cubren únicamente viajes críticos para el usuario
- Equilibre la confianza con el costo de ejecución

Referencias: <https://martinfowler.com/articles/practical-test-pyramid.html>

Mitigación de pruebas inestables

- Ponga en cuarentena pruebas no deterministas rápidamente
- Eliminar el estado compartido y las dependencias de tiempo
- Realice un seguimiento de las tendencias de confiabilidad a lo largo del tiempo

Referencias: <https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html>

Arquitectura de prueba mantenible

- Utilice objetos de página/guión cuando corresponda
- Centralice accesorios y pruebe los creadores de datos
- Mantenga las afirmaciones explícitas y legibles

Referencias: <https://playwright.dev/docs/intro> |
<https://docs.pytest.org/en/stable/>

BDD: cuándo y cuándo no

- Utilice Gherkin para lograr un entendimiento compartido
- Evite convertir archivos de funciones en scripts de bajo nivel
- Prefiere escenarios legibles para el negocio

Referencias: <https://cucumber.io/docs/gherkin/reference/>

Por qué Playwright + Python

- Soporte para varios navegadores con primitivas modernas
- Modelo de localizador sólido y afirmaciones basadas en la web
- El ecosistema Python se integra bien con los flujos de trabajo de control de calidad

Referencias: <https://playwright.dev/docs/intro> |
<https://docs.pytest.org/en/stable/>

Flujo de trabajo principal del dramaturgo

- Inicie el navegador/contexto/página de forma determinista
- Utilice localizadores resilientes y afirmaciones explícitas
- Recopilar rastros/capturas de pantalla de fallas

Referencias: <https://playwright.dev/docs/intro> |
<https://playwright.dev/docs/test-assertions>

Patrones de integración de Pytest

- Utilice accesorios para la reutilización del montaje/desmontaje
- Parametrizar pruebas para escalar la cobertura
- Utilice marcadores para segmentar conjuntos por riesgo

Referencias: <https://docs.pytest.org/en/stable/>

Pruebas basadas en datos y conscientes del medio ambiente

- Leer configuración de variables de entorno
- Separar secretos del código fuente
- Utilice un ciclo de vida de datos de prueba limpio por ejecución

Referencias: <https://12factor.net/config>

Depuración de pruebas de IU fallidas

- Reproducir con modo encabezado y rastros.
- Inspeccionar la red/consola antes de cambiar las afirmaciones
- Corregir errores del producto antes de las pruebas de parcheo

Referencias: <https://playwright.dev/docs/trace-viewer-intro>

Conceptos básicos de las pruebas de rendimiento

- Definir objetivos de rendimiento alineados con SLO
- Ejecute pruebas de referencia, carga, estrés y remojo
- Analice el tiempo de respuesta y el comportamiento de la tasa de error

Referencias: <https://grafana.com/docs/k6/latest/> | <https://sre.google/sre-book/service-level-objectives/>

k6 para comprobaciones prácticas de rendimiento

- Escribir escenarios de carga basados en JavaScript
- Integre ejecuciones en CI para regresiones
- Utilice umbrales para fallar en la degradación crítica

Referencias: <https://grafana.com/docs/k6/latest/>

Fundamentos de las pruebas de accesibilidad

- Utilice WCAG como estándar de referencia
- Automatice comprobaciones comunes con herramientas como ax
- Agregar comprobaciones manuales del teclado y del lector de pantalla

Referencias: <https://www.w3.org/TR/WCAG22/> | <https://www.deque.com/axe/>

Pruebas de seguridad en control de calidad

- Priorice los principales riesgos web de OWASP
- Incluir escenarios de uso indebido/abuso en el diseño de pruebas.
- Colabore con AppSec para evaluaciones más profundas

Referencias: <https://owasp.org/www-project-top-ten/>

Fiabilidad y observabilidad

- Utilice registros, métricas y seguimientos para investigar fallas
- Diseñar pruebas para exponer el riesgo operativo.
- Colaborar con SRE/DevOps en aprendizaje de incidentes

Referencias: <https://opentelemetry.io/docs/> | <https://sre.google/sre-book/table-of-contents/>

CI/CD y puertas de calidad

- Ejecute pruebas automatizadas en cada cambio significativo
- Fusiones de bloques en fallos críticos de calidad
- Mantenga las canalizaciones rápidas para preservar el flujo de desarrolladores

Referencias: <https://docs.github.com/en/actions> | <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>

Acciones de GitHub para control de calidad

- Definir flujos de trabajo como código en YAML
- Dividir trabajos por tipo de prueba y paralelizar
- Publicar informes/artefactos para mayor visibilidad

Referencias: <https://docs.github.com/en/actions>

Estrategia de ramificación e integración

- Prefiere sucursales de corta duración y fusiones frecuentes.
- Utilice prácticas basadas en troncales para reducir el riesgo de fusión
- Mantener la rama principal liberable

Referencias: <https://trunkbaseddevelopment.com/>

Libere la confianza y el control de versiones

- Utilice versiones semánticas para señales de cambio claras
- Vincula las decisiones de liberación a la evidencia, no a la intuición
- Seguimiento de la preparación para la reversión y los riesgos conocidos

Referencias: <https://semver.org/>

Mayús-Izquierda y Mayús-Derecha juntas

- Mayús-izquierda: previene defectos antes
- Shift-right: monitorear y validar en producción
- Utilice ambos para obtener ciclos completos de retroalimentación de calidad

Referencias: <https://dora.dev/> | <https://opentelemetry.io/docs/>

Informes de defectos que ayudan a los equipos

- Escriba pasos de reproducción claros y esperados frente a reales
- Adjunte evidencia: registros, capturas de pantalla, entorno
- Priorizar por impacto en el usuario/negocio

Referencias: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Métricas de control de calidad que importan

- Seguimiento de los principales indicadores (tasa de escamas, defectos escapados)
- Realice un seguimiento de los resultados de entrega junto con las métricas de calidad
- Utilice métricas para mejorar los sistemas, no castigar a las personas

Referencias: <https://dora.dev/> |

<https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html>

Hoja de ruta profesional: desde control de calidad hasta ingeniero de calidad

- Fortalecer los fundamentos de las pruebas y el pensamiento del producto.
- Desarrollar habilidades de codificación para automatización y herramientas.
- Desarrollar CI/CD, observabilidad y comunicación de riesgos.

Referencias: <https://docs.pytest.org/en/stable/> | <https://playwright.dev/docs/intro>

Plano del proyecto final

- Seleccione una aplicación web real con dependencias API
- Crear estrategia de prueba + núcleo de regresión automatizado
- Ejecute en CI y publique artefactos de panel de calidad

Referencias: <https://docs.github.com/en/actions> | <https://playwright.dev/docs/intro> | <https://learning.postman.com/docs/tests-and-scripts/write-scripts/test-scripts/>

