

«Talento Tech»

Testing QA

Clase 04



Clase N° 4 | QA: Ambientes de trabajo y Definición de ciclos de prueba

Temario:

- Ambientes de desarrollo y builds
- Definición de Release
- Creación de Conjuntos de Pruebas a ejecutar en cada ciclo (Tests Sets)
- Testing dentro del sprint vs. testing al final del sprint
- Estrategias para evitar que QA sea un cuello de botella en Scrum
- Error, Defectos y Fallas.

Objetivos de la clase

En esta clase, exploraremos los ambientes de desarrollo y builds en un proyecto de software, comprendiendo su importancia en el ciclo de vida del desarrollo. También aprenderemos a definir y ejecutar conjuntos de pruebas (Test Sets) en cada ciclo, junto con la identificación y clasificación de fallos, errores y defectos, entendiendo sus diferencias. Finalmente, abordaremos la causa raíz de los defectos, los distintos tipos de defectos, su clasificación y el proceso adecuado para su reporte y documentación, finalizando con ejercicios prácticos para reforzar los conceptos.

Ambientes comunes de desarrollo: ¡El detrás de cámaras del software!

El desarrollo de software comparte un paralelismo interesante con la producción de una película: ambos procesos involucran diversas etapas y escenarios donde se construye, prueba y lanza el producto final. A continuación, exploraremos los diferentes ambientes de desarrollo de software, estableciendo similitudes entre cada ambiente y una etapa correspondiente en la creación de una película.

1. Local: El actor practicando sus líneas en el camarín

Es el entorno de desarrollo personal de cada desarrollador. Aquí escribe y prueba su código antes de integrarlo con el resto del equipo. El camarín de un actor donde practica sus líneas, ensaya sus gestos y se prepara para la filmación.

- **Ejemplo de software:** Un desarrollador crea una nueva funcionalidad para la app de reservas de hoteles que permite a los usuarios buscar hoteles por precio. La prueba localmente para verificar que funciona correctamente antes de compartirla con el equipo.

2. Desarrollo: El ensayo general con todo el elenco

Es el ambiente donde se integra el código de todos los desarrolladores. Se utiliza para realizar pruebas de integración y verificar que las diferentes partes del software funcionan juntas correctamente. El ensayo general de una película donde todos los actores se juntan para ensayar las escenas completas y asegurarse de que la historia fluya.



- **Ejemplo de software:** El equipo de desarrollo integra la nueva funcionalidad de búsqueda por precio con el resto de la app de reservas de hoteles. Se realizan pruebas de integración para verificar que la búsqueda por precio funciona correctamente con las demás funcionalidades de la app.

3. Testing: La revisión del director y los productores

Es el ambiente donde los Testers Analistas de Calidad (como vos) realizan las pruebas funcionales y no funcionales del software. Se simulan las condiciones reales de uso para identificar posibles errores o defectos. La revisión de una película donde el director y los productores evalúan la calidad de la filmación y buscan errores o áreas de mejora.

- **Ejemplo de software:** Vos probás la nueva funcionalidad de búsqueda por precio en el ambiente de testing. Verificás que la búsqueda funciona correctamente, que los resultados se muestran de forma clara y que la app no presenta errores.

4. UAT (User Acceptance Testing): La función de prueba para el público

Es el ambiente donde los usuarios finales prueban el software para verificar que cumple con sus expectativas y necesidades. Una función de prueba de una película donde se invita a un grupo selecto de personas para que vean la película y den su opinión antes del estreno oficial.

- **Ejemplo de software:** Un grupo de usuarios prueba la app de reservas de hoteles en el ambiente de UAT. Utilizan la app para buscar hoteles, reservar habitaciones y realizar pagos. Dan su opinión sobre la usabilidad y la funcionalidad de la app.

5. Staging o Pre-producción: La copia final para el cine

Es un ambiente similar al de producción, donde se realizan pruebas finales antes de lanzar el software al público. Se simulan las condiciones reales de producción para identificar posibles problemas de rendimiento o escalabilidad. La copia final de una película que se envía a los cines para su proyección.

- **Ejemplo de software:** Se realizan pruebas de rendimiento en el ambiente de staging para verificar que la app de reservas de hoteles puede soportar un gran número de usuarios concurrentes sin fallar.

6. Producción: El estreno mundial

Es el ambiente donde el software está disponible para los usuarios finales. Es el escenario donde se proyecta la película.

- **Ejemplo de software:** La app de reservas de hoteles se lanza al público y los usuarios pueden utilizarla para buscar hoteles y reservar habitaciones.

7. Build: La edición final de la película

Se refiere al proceso de compilar el código fuente del software y crear un ejecutable que se pueda desplegar en los diferentes ambientes. El proceso de edición final de una película donde se juntan todas las escenas, se agregan los efectos especiales y se crea la versión final que se proyectará en los cines.

¿Qué es un Release?



Un "release", en el mundo del software, es como el lanzamiento oficial de una nueva versión de un programa o aplicación. Es como cuando una banda de rock saca un nuevo disco al mercado. Han estado trabajando en él, lo han pulido y ahora lo comparten con el público. En el caso del software, este "disco" puede traer nuevas funcionalidades, mejoras en las que ya existían o incluso correcciones de errores que se habían detectado.

Ejemplo: "Hotel La Posada"

Imagina un hotel llamado "La Posada" que decide modernizar su sistema de reservas y gestión de clientes. Para ello, contratan a una empresa de desarrollo de software que trabaja por releases.

- **Release 1.0:** El primer "release" es como el primer sencillo de la banda. Es la base del sistema. Permite a los empleados del hotel gestionar las reservas de habitaciones, registrar a los huéspedes y asignarles una habitación. Es una versión básica, pero funcional.
- **Release 2.0:** El segundo "release" es como el segundo sencillo, un poco más elaborado. Se añaden funcionalidades como la gestión de tarifas y la generación de informes básicos de ocupación. El hotel ahora puede tener una mejor visión de su negocio.
- **Release 3.0:** El tercer "release" es como el álbum completo. Se integran nuevas funcionalidades como la gestión de reservas online para los clientes, la posibilidad de reservar servicios adicionales (como el spa o el restaurante) y la gestión de programas de fidelización. El hotel ofrece una experiencia más completa a sus clientes.
- **Release 4.0:** Este "release" es como la edición especial del álbum, con canciones extra y versiones acústicas. Se añaden funcionalidades más avanzadas como la integración con sistemas de pago online, la gestión de eventos y la generación de informes personalizados. El hotel se diferencia de la competencia y ofrece un servicio más moderno y eficiente.

¿Por qué es importante cada Release?

Cada "release" de "Hotel La Posada" es importante porque permite al hotel mejorar su gestión y ofrecer nuevas funcionalidades a sus clientes de forma gradual. Es como si la banda fuera lanzando sencillos y luego el álbum completo, manteniendo a sus fans emocionados y con ganas de más. Además, cada "release" es una oportunidad para los desarrolladores de recibir feedback del hotel y corregir errores antes de lanzar la siguiente versión.

Pasos para la creación de un Release (con la intervención del tester)

1. **Crear el Release:** Se definen los Releases como las piezas de código que serán desplegadas progresivamente en los diferentes ambientes de desarrollo, cumpliendo con los criterios de entrada y salida.

Aquí es donde el tester comienza a involucrarse. Desde el inicio, el tester debe estar al tanto de los objetivos del Release y de las nuevas funcionalidades que se incluirán. Esto le permitirá comenzar a diseñar los casos de prueba necesarios.

2. **Completar el contenido del Release:** Se documentan los cambios incluidos en la versión, los casos de prueba asociados y los ambientes donde será desplegado. Se establecen los pasos necesarios para su validación y aprobación antes de su lanzamiento.

El tester juega un papel fundamental en esta etapa. Es el encargado de crear y ejecutar los casos de prueba diseñados previamente. Estos casos de prueba deben cubrir todas las funcionalidades incluidas en el Release, tanto las nuevas como las existentes.

3. **Validación y aprobación:** Una vez que el tester ha ejecutado los casos de prueba, debe documentar los resultados obtenidos. Si se encuentran errores o fallas, se deben registrar y comunicar al equipo de desarrollo para su corrección.

La aprobación del Release no se dará hasta que el tester haya validado que todas las funcionalidades funcionan correctamente y que se cumplen los criterios de calidad establecidos.

¿Qué es un Test Set?

Un **Test Set** es un conjunto organizado de casos de prueba diseñados para validar una funcionalidad, un módulo o un conjunto de requisitos dentro de un software. Su objetivo es estructurar la ejecución de pruebas de manera eficiente, asegurando una cobertura completa del sistema.

Imagina que una banda de rock está por lanzar su nuevo disco. Antes de enviarlo a las tiendas, subirlo a Spotify, iTunes, etc., quieren asegurarse de que todas las canciones suenan bien, que no hay errores de sonido, que todas las canciones se escuchen al mismo volumen, que los instrumentos no se escuchen más que la voz del cantante y que el disco en general es de buena calidad.

Para hacer esto, no escuchan el disco entero de una vez. Lo dividen en partes más pequeñas y escuchan cada parte por separado. Por ejemplo, podrían escuchar primero todas las canciones lentas, luego todas las canciones rápidas, luego todas las canciones con solos de guitarra, etc.

Un Test Set es como una de esas partes del disco que escuchan por separado. Es un grupo de "pruebas" que se realizan juntas para verificar que una parte del programa funciona correctamente.

¿Por qué necesitamos Test Sets?

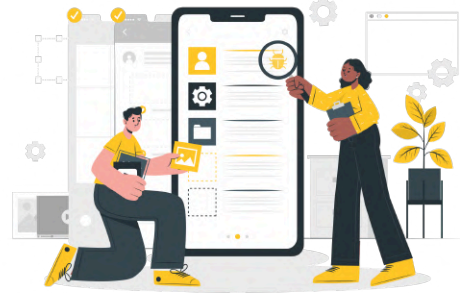


Para asegurarnos de que todo funciona bien, necesitamos probar muchas cosas diferentes. Si probamos cada cosa por separado, puede ser muy confuso y fácil olvidarnos de algo importante.

Los Test Sets nos ayudan a mantener las pruebas organizadas. Es como tener diferentes listas de canciones para escuchar: una lista para las canciones lentas, otra para las canciones rápidas, etc.

¿Cómo se arma un Test Set?

El proceso de armar un Test Set incluye definir lo que se quiere probar, escribir las pruebas correspondientes para cada funcionalidad y luego organizar esas pruebas en un conjunto. Esto permite asegurarse de que todas las partes de la aplicación se validen adecuadamente.



1. Definir objetivos de prueba

("Canciones"):

- Identificar las funcionalidades clave que se quieren probar en la aplicación. Ejemplos:
 - Agregar productos al carrito.
 - Calcular el precio total correctamente.
 - Funcionamiento del pago con tarjeta de crédito.

2. Escribir pruebas específicas ("Pruebas"):

- Para cada funcionalidad, definir instrucciones o acciones que se deben realizar para verificar que la funcionalidad esté funcionando como se espera. Ejemplo para "Agregar productos al carrito":
 - Navegar a la página del producto.
 - Hacer clic en el botón 'Agregar al carrito'.
 - Verificar que el producto aparece en el carrito.

3. Consolidar las pruebas en un Test Set:

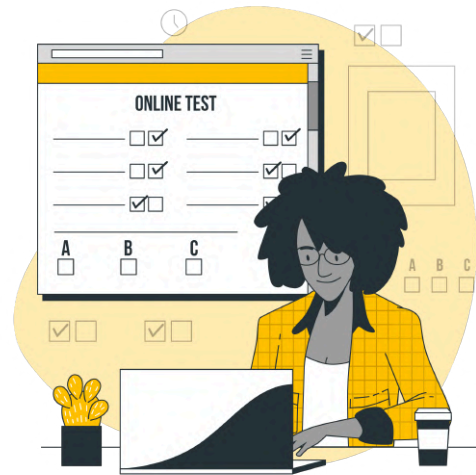
- Reunir todas las pruebas en un único documento o conjunto que servirá como una lista de verificación para evaluar las funcionalidades específicas de la aplicación.

Ejemplo:

Verificar que los builds se desplieguen correctamente en cada ambiente (desarrollo, integración y producción) y que la release cumpla con los criterios de calidad para una app hotelera, identificando y clasificando fallos, errores y defectos.

Casos de Prueba:

En el desarrollo de software, es fundamental garantizar que la aplicación funcione correctamente en diversas etapas de su ciclo de vida. Para el sistema de reservas de una aplicación hotelera, se han diseñado varios casos de prueba que permiten validar funcionalidades críticas en diferentes entornos, desde el desarrollo hasta la producción. Estos casos de prueba están diseñados para asegurar que el software no solo cumple con los requisitos funcionales, sino que también proporciona una experiencia de usuario fluida y sin errores.



La aplicación hotelera tiene como objetivo facilitar a los usuarios la búsqueda y reserva de hoteles, así como el manejo de sus reservas. Dado que la aplicación maneja información sensible y transacciones financieras, es crucial realizar pruebas exhaustivas antes de lanzar nuevas versiones. Los casos de prueba abarcan desde la validación del despliegue en el entorno de desarrollo, pasando por la integración funcional en el entorno de pruebas, hasta la validación de la release en producción y la verificación del manejo de incidencias.

A continuación, se detallan los diferentes conjuntos de pruebas, incluyendo los pasos a seguir y los resultados esperados para cada uno.

Test Set-001: Despliegue en Ambiente de Desarrollo

Pasos:

- Desplegar el build de la app hotelera en el ambiente de desarrollo.
- Acceder a la aplicación e iniciar sesión.
- Verificar que la página principal muestre correctamente la lista de hoteles destacados.

Resultado Esperado:

- Build desplegado sin errores.
- Funcionalidad de login y visualización de la homepage operativas.

Test Set-002: Integración y Funcionalidades Clave en el Ambiente de Integración

Pasos:

- Desplegar el build en el ambiente de integración.
- Realizar búsquedas de hoteles, simular un proceso de reserva y cancelar una reserva.
- Verificar la integración con el sistema de pagos.

Resultado Esperado:

- Flujo de búsqueda, reserva y cancelación funciona sin problemas.
- Integración con la pasarela de pagos es correcta y segura.

Test Set-003: Validación de la Release en Ambiente de Producción

Pasos:

- Desplegar el release candidate en el ambiente de producción o staging.
- Ejecutar el test set completo que incluya búsquedas, reservas, visualización de opiniones y procesos de pago.
- Confirmar que se cumplen los criterios de aceptación definidos por Silvia.

Resultado Esperado:

- Todas las funcionalidades operan según lo esperado, sin defectos críticos.
- La release es aprobada para su lanzamiento.

Test Set-004: Validación de Mensajes de Error y Manejo de Incidencias

Pasos:

- Ingresar datos inválidos en el formulario de reserva (por ejemplo, una fecha de check-out anterior al check-in).
- Probar el ingreso de un número de tarjeta de crédito erróneo en el proceso de pago.
- Observar y registrar los mensajes de error generados.

Resultado Esperado:

- Los mensajes de error son claros, específicos y orientan al usuario para corregir la información.
- Los errores se registran y clasifican adecuadamente en el sistema de seguimiento de incidencias.

Test Set-005: Prueba de Carga en Ambiente de Producción

Pasos:

- Simular la concurrencia de 50 usuarios realizando búsquedas y reservas simultáneamente.
- Monitorear el rendimiento del sistema y los tiempos de respuesta.
- Identificar cualquier degradación en el desempeño o cuellos de botella.

Resultado Esperado:

- Tiempos de respuesta por debajo del umbral establecido (por ejemplo, menos de 3 segundos).
- Sistema estable y sin caídas, incluso bajo alta demanda.

En resumen:

Un Test Set es como una lista de canciones para probar un programa. Nos ayuda a mantener las pruebas organizadas y asegurarnos de que no nos olvidamos de nada importante. Es una herramienta muy útil para los testers, que son las personas que se encargan de probar los programas.

Testing dentro del Sprint vs. Testing al Final del Sprint

Los **Test Sets** juegan un papel clave en la planificación y ejecución de pruebas dentro de los Sprints, ayudando a que QA no se convierta en un cuello de botella y asegurando que las pruebas se distribuyan de manera eficiente a lo largo del Sprint.

Testing dentro del Sprint (Pruebas Continuas en Agile)

En un entorno ágil, el testing no es una fase separada, sino que forma parte del ciclo de desarrollo dentro del Sprint. Se adopta un enfoque **"Shift Left"**, lo que significa que las pruebas comienzan lo antes posible para detectar errores a tiempo. Cuando se ejecutan pruebas dentro del Sprint, los Test Sets ayudan a organizar y priorizar los casos de prueba de acuerdo con las funcionalidades desarrolladas.

Características:

- ✓ Se prueban pequeñas funcionalidades en cuanto están desarrolladas.
- ✓ Se realizan pruebas manuales y automatizadas durante todo el Sprint.
- ✓ Se colabora activamente con los desarrolladores para corregir errores rápidamente.
- ✓ Se ejecutan pruebas de regresión continuamente para asegurar estabilidad.

Ventajas:

- Se detectan errores antes, reduciendo costos de corrección.
- Se mejora la comunicación entre testers y desarrolladores.
- Se garantiza que cada incremento del producto sea funcional al final del Sprint.

Ejemplo práctico: Un tester en un equipo Scrum revisa los criterios de aceptación de una User Story antes de que los desarrolladores comiencen a programar. Luego, a medida que se desarrolla la funcionalidad, ejecuta pruebas manuales y automatizadas para validar su correcto funcionamiento.

Testing al Final del Sprint (Enfoque Tradicional dentro de Scrum)

Si las pruebas se dejan para el final del Sprint, QA debe validar todas las funcionalidades juntas, lo que puede generar retrasos. En este caso, los Test Sets ayudan a organizar la ejecución masiva de pruebas.

Características:

- ✓ Se prueba todo el incremento de software una vez finalizado el desarrollo.
- ✓ No se realizan pruebas parciales antes de completar el Sprint.
- ✓ Se pueden acumular defectos que no se identificaron a tiempo.

Desventajas:

- Aumenta el riesgo de no cumplir con la Definition of Done.
- Si se encuentran errores al final, no hay suficiente tiempo para corregirlos.
- Puede generar estrés en el equipo de QA y retrasos en la entrega del producto.

Ejemplo práctico: Un equipo de desarrollo trabaja en varias funcionalidades sin que se realicen pruebas intermedias. Al final del Sprint, los testers encuentran fallos críticos que requieren cambios de código, retrasando la entrega del producto.

Estrategias para evitar que QA sea un cuello de botella en Scrum

Cuando QA se convierte en un cuello de botella, significa que las pruebas se acumulan y retrasan la entrega del producto. Para evitarlo, es fundamental adoptar estrategias que permitan la integración temprana del testing en el proceso de desarrollo.

Estrategias para optimizar el trabajo de QA en Scrum:

1) Automatización de pruebas

- ✓ Implementar pruebas automatizadas en los flujos más críticos.
- ✓ Ejecutar pruebas de regresión automáticas en cada nueva versión.
- ✓ Usar herramientas como Selenium, Cypress o JUnit para pruebas automatizadas.

Ejemplo: Un equipo de QA automatiza las pruebas de login y checkout para evitar probarlas manualmente en cada Sprint.

2) Definir criterios de aceptación claros y testables

- ✓ QA debe participar en la definición de User Stories para garantizar que los criterios de aceptación sean medibles y probables.
 - ✓ Se pueden utilizar técnicas como BDD (Behavior Driven Development) para escribir criterios en lenguaje claro (Given, When, Then).

Ejemplo:

- **User Story:** Como usuario, quiero recibir un email de confirmación después de completar una compra.
- **Criterio de aceptación:** El email debe enviarse automáticamente dentro de los 5 minutos posteriores a la compra.

3) Implementar pruebas en paralelo al desarrollo

- ✓ Se pueden realizar pruebas en un entorno de desarrollo antes de que la funcionalidad esté terminada.
- ✓ Los testers pueden diseñar casos de prueba mientras los desarrolladores escriben código.

Ejemplo: Mientras el equipo de desarrollo trabaja en la funcionalidad de "Filtrar productos por categoría", QA redacta los casos de prueba y ejecuta pruebas en el backend.

4) Uso de entornos de integración continua (CI/CD)

- ✓ Permite ejecutar pruebas automatizadas en cada nueva integración de código.
- ✓ Se pueden usar herramientas como Jenkins, GitHub Actions o CircleCI para ejecutar pruebas automáticamente.

Ejemplo: Un equipo configura un pipeline de CI/CD donde cada nueva funcionalidad pasa automáticamente por pruebas unitarias y de regresión antes de integrarse en la versión final.

5) Aplicar Testing Exploratorio

- ✓ Mientras se automatizan casos de prueba, se pueden realizar pruebas exploratorias para detectar defectos inesperados.
- ✓ Permite a los testers descubrir fallos en la experiencia de usuario que las pruebas automatizadas no detectan.

Ejemplo: Un tester explora la interfaz de una app móvil buscando inconsistencias en el diseño y la navegación.

Conclusión:

Los Test Sets son herramientas clave para organizar y optimizar las pruebas en Scrum, evitando cuellos de botella y asegurando que las validaciones se realicen de manera continua dentro del Sprint.

Si se utilizan correctamente, permiten a QA trabajar de forma más eficiente y garantizar la calidad del software sin retrasar el desarrollo.

¿Qué diferencias hay entre Error, Defecto y Fallo?



Término	Definición	Características / Ejemplo
Error	Equivocación humana en el desarrollo o en el diseño.	Ejemplo: Un programador confunde la variable que almacena el número máximo de habitaciones disponibles, configurando el límite incorrecto.
Defecto	Imperfección en el código resultante del error, que puede llevar a comportamientos inesperados.	Ejemplo: Debido al error en la lógica, la app muestra la disponibilidad de habitaciones incorrecta, dejando ver más habitaciones de las que realmente hay.
Fallo	Manifestación observable del defecto durante la ejecución de la aplicación.	Ejemplo: Un huésped intenta reservar una habitación y la app se cierra inesperadamente o muestra un mensaje de error, impidiendo la reserva.

- **Error:** Se trata de la equivocación cometida durante la creación del software, por ejemplo, un fallo en la lógica del algoritmo que calcula la disponibilidad de habitaciones.
- **Defecto:** Es la consecuencia directa del error, donde la app de hotel tiene una imperfección en su funcionalidad, como mostrar datos erróneos sobre la disponibilidad.
- **Fallo:** Es lo que el usuario experimenta en tiempo real; por ejemplo, si al realizar una reserva la app se bloquea o no procesa correctamente la información, se manifiesta el fallo del sistema.

¡Trabajando en Talento Lab!



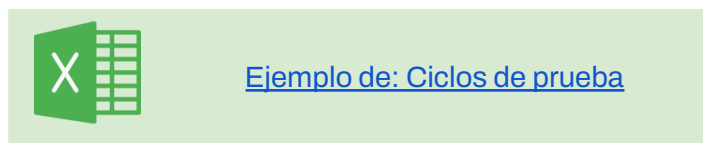
Hoy, Silvia y Matías te guían en una nueva aventura: descubrir los ambientes de desarrollo y builds. En esta jornada, Silvia nos muestra cómo cada entorno –desde desarrollo hasta producción– es vital para garantizar que nuestros builds se conviertan en versiones sólidas del software.



Matías nos acompaña para definir qué es una Release, ese momento crucial en el que todo se junta para lanzar una versión estable y funcional del producto. Además, aprenderemos a crear y ejecutar Test Sets en cada ciclo, lo que nos permitirá identificar y clasificar fallos, errores y defectos, y comprender la causa raíz de cada incidencia.

¡Veamos un ejemplo de cómo organizar un ciclo de pruebas!

Para que te hagas una idea, acá tenemos una planilla de ejemplo de cómo se organiza un ciclo de pruebas manualmente:



Ejercicios Prácticos

- **Creación de conjuntos y ciclos de prueba:**
 - Seleccioná las user stories generadas del proyecto creadas en los ejercicios prácticos anteriores para poder avanzar.
 - Crear y relacionar los casos de pruebas creados con las user stories y criterios de aceptación correspondientes.
 - Crear los conjuntos y ciclos de pruebas a ejecutar.

Preguntas para Reflexionar

- ¿Por qué es importante utilizar diferentes ambientes de desarrollo en un proyecto de software?
 - ¿Qué ambiente es clave para detectar errores a tiempo?
 - ¿Cómo impacta el tester en un "release"?
 - ¿Por qué son útiles los "test sets"?
 - ¿Teniendo en cuenta lo desarrollado y trabajado con el sitio Tech Lab, pueden identificar cuales son aquellas grietas, etiquetadas como Error, Defecto y Fallo?
-

Próximos Pasos

En la próxima clase, exploraremos en profundidad los problemas y defectos que pueden surgir durante el desarrollo de software, comprendiendo su origen, impacto y gestión.

Analizaremos los conceptos de error, defecto y fallo, los tipos de defectos que se pueden encontrar, la importancia de identificar la causa raíz y los efectos de un defecto, cómo determinar la severidad y prioridad de un defecto, y cómo realizar un reporte de defectos efectivo.



Buenos Aires
aprende
Agencia de Habilidades para el Futuro

BA Buenos
Aires
Ciudad