Material descargable

Introducción a Quality Assurance

Error, defecto y falla

En el terreno profesional de un tester, normalmente siempre se topará con *errores*, ya que son exactamente lo que está buscando al analizar un producto o un proyecto de software. Los **defectos** y las **fallas** se descubren una vez que el producto ya está funcionando del lado del cliente y es este quien las informa.

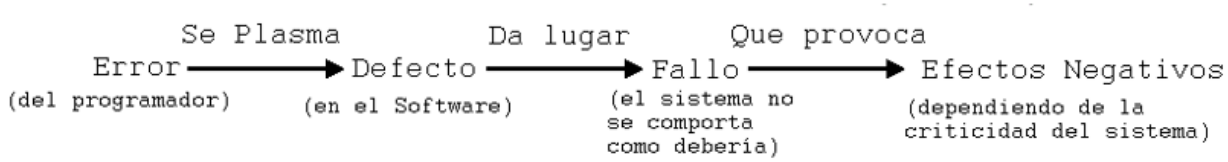
Error	Acción humana que produce un resultado incorrecto
Defecto	Desperfecto en un componente que puede causar que el mismo falle en sus funciones
Fallo	Manifestación visible de un defecto

La identificación de defectos - poder categorizarlos, reportarlos y que eventualmente se solucionen - es parte de las actividades del equipo de Control y Garantía de calidad. El mayor desafío que tiene este equipo es poder realizar las pruebas necesarias en cada momento de las etapas del Ciclo de vida del Desarrollo de Software (SDLC). El objetivo es poder detectar los errores lo antes posible. ¿Por qué? Porque a medida que se avanza en el proceso, los costos para encontrar y corregir errores crecen en forma exponencial.

El momento **más económico** para arreglar un error es durante el periodo de análisis de requerimientos, y luego el costo se va incrementando sin excepción en cada etapa subsiguiente, llegando al escenario más costoso de todos que es en el periodo de mantenimiento luego de la implementación.

¿Quieres saber más sobre la diferencia entre un error, un bug y una falla? Este artículo describe algunos de los desafíos que tenemos al trabajar en español sobre elementos que han sido creados en inglés, y la dificultad que existe para saber cómo nombrar a cada uno de ellos.

Aquí te dejamos un esquema para que recuerdes la secuencia de producción de un error:



Un tester detecta **defectos** y explica cómo **fallan**, acompañando el reporte con un análisis de los **efectos negativos**. El tester no siempre puede saber cuál es el **error** que dio lugar a ese defecto.

Verificación y validación

El proceso de **verificar y validar** es el proceso por el cual se investiga si un sistema de software satisface las especificaciones, requerimientos y estándares indicados y si cumple el propósito deseado.

- **Verificación:** ¿Estamos construyendo el producto **correctamente**?
- **Validación:** ¿Estamos construyendo el producto **correcto**?

Estos dos términos son utilizados indistintamente por muchas personas. Dado que los especialistas en testing se encargan fundamentalmente de la **verificación**, debemos hacer foco en comprender el término y distinguirlo de la **validación**.

Verificación

Es el proceso por el que se determina si el software que está siendo analizado se está diseñando y desarrollando de acuerdo a requerimientos especificados. Los requerimientos o especificaciones actúan como guía en todo el proceso de desarrollo de software. El código que se escribe para un software está basado en esa documentación inicial.

Seth Godin define a la calidad como “estar a la altura de los requerimientos.”

La potente definición de Godin simplifica al máximo nuestra tarea al iniciar un proyecto:

- Asegúrate de haber comprendido lo que se debe hacer
- Haz las preguntas en este momento o luego ya deberás cumplir con lo establecido en ese documento inicial.

La verificación se realiza para corroborar que el software que está siendo desarrollado adhiera a estas especificaciones en todos los momentos de su ciclo de desarrollo. La verificación asegura que la lógica del código esté alineada con estas especificaciones y que no contenga errores.

Dado que el proceso de verificación suele ser complejo debido al tamaño del proyecto o la complejidad del mismo, se utilizan varios métodos de verificación. Algunos de ellos son: inspección del código, revisiones de código, revisiones técnicas, entre otros. Los equipos de testing (QA) también pueden usar modelos matemáticos y cálculos para predecir el comportamiento del código y verificar su lógica (etapa de análisis de requerimientos).

¿NECESITAS UN EJEMPLO?

Supongamos que debemos testear una aplicación para teléfonos móviles. O dicho como lo diría un profesional: una app mobile.

Esta verificación tiene tres fases:

1. La verificación de los requerimientos.
2. La verificación del diseño.

3. La verificación del código.

Paso 1: es verificar y confirmar que los requerimientos están completos, son claros y correctos. Antes de que una aplicación vaya a ser desarrollada, el equipo de QA testea que los requerimientos del negocio o del cliente estén completos y correctos.

Paso 2: La verificación del diseño es el proceso por el cual el equipo se asegura que el software cumple con los requerimientos de diseño y lo hace a través de mockups o evidencias gráficas. Aquí el equipo de QA testea si los layouts, prototipos (mockups), los flujos de navegación, el diseño de la arquitectura y los modelos lógicos de las bases de datos están a la altura de los requerimientos funcionales y no-funcionales del cliente.

Paso 3: La verificación del código tiene por objetivo evaluar el código para ver si está (a) completo, (b) correcto y (c) si es consistente (al ser testeado, responde en forma previsible). En esta etapa el equipo de QA chequea si el código fuente, las interfaces de usuario y el modelo de la base de datos de la aplicación cumplen con las especificaciones de diseño.

Lo más importante a recordar de este ejemplo es que la verificación se realiza en forma interna, dentro de los equipos que van a estar trabajando con el desarrollo de la aplicación. Es importante hacer este proceso antes de iniciar el trabajo ya que asegura que no se trabaje en direcciones equivocadas, gastando recursos escasos. Una vez más, el equipo de testing está para asegurarse de que se trabaje en forma eficiente, anticipando los problemas y ahorrando tiempo y dinero.

La realidad: muchas organizaciones y startups tardan mucho en implementar procesos de verificación. Y suelen hacerlo cuando ya han constatado más de una vez que se gasta mucho más en “lo hacemos rápido, sin un plan” que “lo hacemos bien, verificando paso por paso y luego lo construimos.”

Conclusiones: La verificación es un proceso continuo, interno que se debe realizar en todas las etapas del desarrollo de software. La verificación es el testing estático.

Sus principales ventajas son:

1. Actúa como control de calidad en cada etapa del proceso de desarrollo de software.
2. Permite que el equipo de desarrollo produzca productos que se ajusten a los requerimientos y a las necesidades del cliente.
3. Ahorra tiempo ya que se detectan los defectos en las etapas tempranas del desarrollo de software.
4. Reduce o elimina los defectos que pueden surgir en las etapas más tardías del desarrollo de software.

Validación

La validación es el proceso de chequear que se esté desarrollando el producto correcto. A diferencia de la verificación que va desde los requerimientos hacia el código, la validación se hace en dirección opuesta. Va escalando por las distintas capas de desarrollo y va chequeando que el software desarrollado sea el producto correcto. Es por ello que este proceso lo lleva adelante, en su mayor parte, el equipo de desarrollo, y se realiza cuando el producto ya está listo para ser entregado. La validación es el testing dinámico.

Para que ya vayas teniendo una idea, en el proceso de validación, estas son las actividades más relevantes:

1. Testeo de caja negra (Black box testing - BBT) - Testeo de sistema completo. Responsables: equipo de QA/testing
2. Testeo de caja blanca (White box testing - WBT). Responsables: desarrolladores
3. Testeo unitario (Unit testing - UT). Responsables: desarrolladores
4. Testeo de integración o integraciones (Integration testing). Responsables: desarrolladores

El extraño caso del UAT (User Acceptance Test). El UAT es el último test del proceso de verificación (y está a cargo de usuarios reales, beta testers o usuarios designados por el cliente), o puede ser considerado como parte de la Validación ya que es parte del proceso de validación del producto. Hoy en día la línea que divide Verificación de Validación es muy difusa ya que los procesos se han vuelto más

dinámicos, respondiendo a cambios mucho más ágiles propios del desarrollo de software ágil.

Si en el campo profesional estos dos procesos se superponen, ¿por qué es importante que aprendamos estos conceptos?

Volviendo al punto de partida inicial: la **verificación** sirve para saber si estamos construyendo el **producto correctamente** (de acuerdo a lo pedido) y la **validación** nos sirve para saber si hemos construido el **producto correcto** (el que necesita el mercado o el cliente y que va a generar valor).

Verificación	Validación
Incluye chequear documentos, diseños, código y programas.	Incluye el testeo y validación del producto real.
En su mayoría, son pruebas estáticas (*static testing*).	En su mayoría, son pruebas dinámicas (*dynamic testing*).
No incluye la ejecución del código.	Incluye la ejecución del código.
Los métodos utilizados son reviews, repasos, inspecciones de código, demostración y análisis.	Los métodos utilizados son BBT (black box testing), WBT (White box testing) y testeos no funcionales. También incluye: performance testing, regression testing, security testing y testing de sistema.
Revisa que el software cumpla con los requerimientos. Documenta lo que no cumple o lo que falta cumplir.	Chequea que el software cumpla con los requerimientos y las expectativas del cliente. Documenta el feedback del cliente.

Permite encontrar defectos en etapas tempranas del desarrollo.	Permite encontrar defectos que no se pudieron encontrar durante la verificación.
El objetivo de la verificación es la aplicación y la arquitectura y requerimientos de software.	El objetivo de la validación es el producto en sí.
El equipo de QA realiza la verificación.	El equipo de desarrollo ejecuta la validación con ayuda del equipo de Testing.
Ocurre antes de la validación.	Ocurre luego de la verificación.
En su mayoría, consiste en el análisis de documentación y la realizan personas.	En general, consiste en la ejecución de un programa y la realiza una computadora.

Ejemplo de Verificación

Vamos a dar un paseo por una verificación. Estaremos verificando un automóvil y una aplicación de software al mismo tiempo.

1. **Inspección:** se refiere a examinar un producto o sistema sin intervenir en él. Puede ser la simple manipulación física o tomar medidas, por ejemplo.

Automóvil: inspeccionar visualmente el automóvil para asegurarnos de que cumpla con los requerimientos especificados: levanta cristales eléctrico, asientos ajustables, aire acondicionado, sistema de navegación a bordo, kit de remolque, etc.

Software: examinar visualmente el software para constatar que existan las pantallas solicitadas, chequear que estén los campos necesarios para el ingreso de datos (nombre de usuario, por ejemplo), verificar que existan todos los botones para las funcionalidades solicitadas, etc.

2. **Demostración:** es la manipulación del producto como se espera que sea usado, para verificar que se comporte como se planificó o de acuerdo a las expectativas.

Automóvil: poner en uso los comandos de las ventanas y los asientos para verificar que funcionen correctamente, encender el vehículo para corroborar que el aire acondicionado produzca aire frío, dar una vuelta con el automóvil para tener una idea de aceleración y rango de maniobras como fue descrito en los requerimientos.

Software: ingresar todos los campos en las pantallas y seleccionar los botones que cumplan con lo solicitado, esperando la respuesta específica. Asegurar que los datos devueltos son del tipo requerido.

3. **Prueba:** es la verificación del producto o sistema utilizando una serie de estímulos, datos o ingresos predeterminados para corroborar que el producto produzca un resultado específico y predefinido en los requerimientos.

Automóvil: acelerar el automóvil de cero a 100 km/h. Verificar que pueda ser realizado en 5.2 segundos. Acelerar en una curva bajo condiciones de control, produciendo 0.8 fuerza G, sin que el vehículo pierda tracción.

Software: ingresar el tipo y modelo de automóvil, con levanta cristales eléctrico, dirección asistida, y el resto de las opciones definidas en el plan de pruebas, seleccionar el botón de “obtener precio ya” y que la aplicación devuelva el valor preciso de “$43.690”.

4. **Análisis:** es la verificación del producto o sistema utilizando modelos, cálculos y equipos de pruebas especializados. Esta etapa permite que se puedan hacer predicciones sobre el desempeño o *performance* típicos del producto o software basados en resultados confirmados de las pruebas. También se pueden combinar estos resultados para ofrecer mayor información sobre el producto para poder estimar los rangos límites de performance.

Automóvil: completar una serie de aceleraciones a unas revoluciones/m por un tiempo determinado, mientras se monitorea la vibración del motor y su temperatura, verificando que se logran los resultados deseados. Utilizar esta

información para predecir el punto de falla del motor. Ej, las rev/m máximas toleradas por un tiempo estimado.

Software: completar una serie de pruebas en las que un número predeterminado de usuarios ingresan las características del automóvil que están intentando cotizar e inician la función “obtener precio” al mismo tiempo. Se mide el tiempo de respuesta para corroborar que la función devuelve un precio dentro de los límites de tiempo preestablecidos. Se analiza la relación entre el incremento de usuarios en el sistema y el tiempo que le toma a la función devolver el precio. Se documentan los resultados y el tiempo de cada prueba para ver si se degrada la performance a medida que el sistema recibe mayor carga para detectar cuándo es el momento en el que el sistema deja de cumplir con las expectativas definidas en los requerimientos.

Evaluación de criticidad y prioridad

Entender cómo clasificar a un error o defecto en el momento en el que lo encontramos es clave en el proceso de testing.

Si el ciclo de producción de software está funcionando de manera eficiente, el equipo de pruebas (QA) debería ser capaz de identificar y alertar sobre errores antes de que se manifiesten ante clientes o usuarios finales. No obstante, reconocemos que esto representa un escenario ideal. En la práctica, nos enfrentamos a múltiples líneas de código y colaboración simultánea, ya sea en grandes empresas con procesos bien establecidos o en contextos de pequeñas empresas o startups, donde pocas personas realizan numerosas tareas de manera superpuesta.

En esta superposición de actividades y responsabilidades, es común observar cómo los errores (bugs o defectos) salen a la luz. En ocasiones, los descubrimos con escaso margen de tiempo antes de la entrega del producto, cuando ya no es posible realizar cambios.

Para simplificar esta tarea, existe la **matriz de análisis** simple en la Evaluación de Criticidad y Prioridad en la Gestión de Errores, una herramienta visual que ayuda a

determinar la importancia y la urgencia de abordar un defecto o error en el software. Esta matriz generalmente utiliza dos dimensiones: criticidad y prioridad.

- **Criticidad**: Indica el impacto del error en el funcionamiento general del sistema o en las actividades del usuario. Puede clasificarse en categorías como "Crítico", "Importante" o "Menor".
- **Prioridad**: Refleja la urgencia de corregir el error. Puede dividirse en niveles como "Alta", "Media" o "Baja".

La combinación de la criticidad y la prioridad de un error en la matriz proporciona una guía para decidir el orden en que se deben abordar y corregir los problemas. Los errores con mayor criticidad y prioridad alta se consideran los más urgentes y deben tratarse de manera prioritaria.

La matriz de análisis ayuda a los equipos de desarrollo y QA a asignar recursos de manera eficiente, centrándose en los errores que tienen un impacto significativo en el funcionamiento del software y requieren una corrección inmediata.

La siguiente representación visual muestra una matriz de análisis simple que puede resultar útil para determinar la acción a seguir al descubrir un defecto.