

TESTING

Software QA

curso básico



Programá
tu futuro

www.tresdefebrero.gov.ar/tecnof3f



(i==2)

```
if (i < inputs.length - 1) {  
    if (i == 2) {  
        atpos = inputs[i].indexOf('E');  
        dotpos = inputs[i].lastIndexOf('.');  
        if (atpos < 1 || dotpos < atpos + 1 ||  
            document.getElementById('errnati')  
        else  
            document.getElementById(div).inn
```

UNIDADES

- ➡ **Unidad 4:** Diseño de casos de prueba

- ➡ **Unidad 5:** Reporte de errores

UNIDAD 4

Diseño de casos de prueba



CASOS DE PRUEBA

Pruebas a partir de especificaciones

Para poder definir pruebas es necesario conocer las especificaciones (o requerimientos) del producto que vamos a probar. Estas especificaciones las podremos conocer a través del ESRE (Especificación de Requerimientos), las historias de usuario, o en algunos casos tendremos que preguntarle al cliente, equipo de proyectos, etc. Ya que es posible que los requerimientos no estén formalmente definidos. Para cada prueba tenemos que definir esto:

- ¿Qué hace el sistema que voy a probar?
- Objetivos de prueba
- Resultados esperados
- Pasos de ejecución
- Condiciones iniciales

EJEMPLO

Para una especificación hay muchos casos de prueba con distintos datos de entrada.

Por ejemplo, una TV:

Especificación: “al apretar la tecla para subir el volumen se escucha más alto”.

Test: “apretar la tecla de subir el volumen y escuchar a ver si está más alto”.

¿Cuál es el valor esperado? Que se escuche más alto.

¿Cuál es el objetivo de la prueba? Verificar que aumenta el volumen.

Otro ejemplo:

Al apretar el botón de prender = prende.

¿Pero si estaba prendida?

¡Hay que aclarar que hay que comenzar con la TV apagada! (condiciones iniciales)

Esto muestra la importancia de especificar las condiciones iniciales. Si aprieto el número 4, debería poner el canal 4. Si desde allí aprieto la tecla de subir,

entonces debería ir al 5. Con esto se ve la importancia de los datos de prueba.

COMPONENTES DE UN CASO DE PRUEBA

- Identificador del caso de prueba, único
- Título / objetivo
- Pasos
- Datos de entrada
- Valores esperados
- Condiciones iniciales
- Tester / equipo / responsable
- Fecha de creación
- Aplicación que se está probando

DISEÑO DE CASOS DE PRUEBA

Al diseñar casos de prueba, hay 2 tipos:

ABSTRACTO

No define los juegos de datos específicos, sino que esto queda a criterio del tester que está ejecutando qué datos usar. Veremos que hay técnicas para elegir estos datos que hacen que el test tenga más o menos sentido.

Ejemplo: “Caso de prueba para validar que el correcto funcionamiento del buscador”.

CONCRETO

Es como un guion paso a paso dónde están indicados los datos de la prueba y se espera que el tester no se salga del mismo. En este caso, el diseñador de las pruebas es quien selecciona los datos y hará tantos casos de prueba como juegos de datos considere necesarios para evaluar la funcionalidad.

Ejemplo: “Caso de prueba para buscar un producto que exista”.

DATOS

Es un elemento aislado, recabado para un cierto fin, pero que no ha pasado por un proceso que lo interrelacione con otros de manera funcional para el fin previsto.

- Ejemplo de datos: 20, Juan, cédula. Información: Se trata del conjunto de datos, añadidos, procesados y relacionados, de manera que pueden dar pauta a la correcta toma de decisiones según el fin previsto.
- Ejemplo de información: La cédula de juan indica que tiene 20 años (Los datos se relacionan para indicar algo, aislados no significan nada; en esta oración indican algo).

TIPOS DE DATOS

Boolean (booleano): Este tipo de dato se emplea para valores lógicos, los podemos definir como datos comparativos, dicha comparación devuelve resultados lógicos (verdadero o falso).

Char (carácter): El tipo de dato carácter es un dígito individual, el cual se puede representar como numéricos (0 al 9), letras (a-z) y símbolos (!"\$&:/).

String (cadena): Es un conjunto de caracteres, o sea un texto. Ejemplos: "Hola mundo", "Vale 100 \$".

Numérico: Este tipo de dato puede ser real o entero, dependiendo del tipo de dato que se vaya a utilizar.



Enteros: son los valores que no tienen punto decimal, pueden ser positivos o negativos y el cero.



Reales: estos caracteres almacenan números muy grandes que poseen parte entera y parte decimal. Fecha: Se manejan distintos tipos de datos según el lenguaje de programación o base de datos para almacenar fechas, lo importante es que tienen distintos formatos aceptados. Ejemplo: "02/10/2015" formato clásico dd/mm/aaaa (día/mes/año).

Multimedia: El término multimedia se utiliza para referirse a cualquier objeto o sistema que utiliza múltiples medios de expresión físicos o digitales para presentar o comunicar información. De allí la expresión multimedios. Los medios pueden ser variados, desde texto e imágenes, hasta animación, sonido, video, etc.

TESTING POSITIVO Y NEGATIVO

Test Positivo (o limpio): Intenta mostrar que el producto satisface sus requerimientos.

- ¿Cómo hago un test positivo?

Armo mi test con el requerimiento que aparece en la especificación.

- Ejemplo: Para el carrito pide que se puede ingresar el nombre y el apellido del cliente, hago mi caso para ingresar el nombre Juan y apellido Pérez (un nombre y apellido cualquiera).

Test Negativo (o sucio): El objetivo es romper el sistema.

- ¿Cómo hago un test negativo?

Trato de hacer algo contrario a lo que el sistema espere que se haga.

- Ejemplo: Para el mismo requerimiento del carrito de nombre y apellido, tratar de poner un número en lugar de un nombre y un apellido. El resultado debería ser que se muestre un aviso de error (nunca debería caerse el sistema e impedir que siga registrando el usuario).

COBERTURA DE PRUEBAS

Es un elemento aislado, recabado para un cierto fin, pero que no ha pasado por un proceso que lo interrelacione con otros de manera funcional para el fin previsto.

¿Cuántos datos tengo que poner en cada pantalla?

¿Cuántos valores distintos puedo poner en cada campo de la pantalla?

¿Cómo combinar esos datos?

Además, puedo ejecutar en distintos navegadores, en distintos celulares, sistemas operativos, etc. Entonces la respuesta es: ¡infinitos casos!

Claramente, no tenemos tiempo para probar todo.

La forma en la que los testers estaremos aportando calidad al software es principalmente buscando fallos, por supuesto. Digamos que si no encuentro fallo todo el costo del testing me lo pude haber ahorrado, **¿no? ¡Nooo!**

Porque si no encontramos fallos, entonces tenemos más confianza en que los usuarios encontrarán menos problemas.

¿Se trata de encontrar la mayor cantidad de fallos posible? ¿Un tester que encuentra cien fallos en un día hizo mejor su trabajo que uno que apenas encontró 15? De ser así, la estrategia más efectiva sería centrarse en un módulo que esté más verde, que tenga errores más fáciles de encontrar, me centro en diseñar y ejecutar más pruebas para ese módulo, pues ahí encontraré más fallos. **¡NO!** La idea es encontrar la mayor cantidad de fallos que más calidad le aporten al producto. O sea, los que al cliente más le van a molestar.

¡Ojo! *El objetivo no tiene por qué ser el mismo a lo largo del tiempo, pero si es importante que se tenga claro en cada momento cuál es.*

Puede ser válido en cierto momento tener como objetivo del testing encontrar la mayor cantidad de errores posibles, entonces seguramente el testing se enfoque en las áreas más complejas del software o más verdes.

Si el **objetivo** es dar seguridad a los usuarios, entonces seguramente se enfoque el testing en los escenarios más usados por los clientes. Existen distintas técnicas de diseño de casos de prueba, que permiten seleccionar la menor cantidad de casos con mayor probabilidad de encontrar fallas en el sistema.

Por este motivo, estos casos se consideran los más interesantes para ejecutar, ya que el testing exhaustivo no solo es imposible de ejecutar en un tiempo acotado, sino que también es muy caro e ineficiente.

¿QUÉ ES LA COBERTURA DE PRUEBAS?

Básicamente, es una medida de calidad de las pruebas. Se definen cierto tipo de entidades sobre el sistema, y luego se intenta cubrirlas con las pruebas. Es una forma de indicar cuándo probamos suficiente, o para tomar ideas de que otra cosa probar (pensando en aumentar la cobertura elegida).

Para verlo aún más simple, podríamos decir que la cobertura es como cuando barres la casa.

Una forma de saber si barriste toda la casa es contando si barriste cada habitación. Esto te ayuda a saber cuál es la calidad de la barrida y al mismo tiempo es un indicador de cuando parar de barrer.

Ahora, lograr el 100% de cobertura con ese criterio, ¿indica que la casa está limpia? NO, porque la cocina no la barrí.

Entonces, ¡jojo! Manejar el concepto con cuidado, tener cierto nivel de cobertura es un indicador de la calidad de las pruebas, pero nunca es un indicador de la calidad del sistema, ni me garantizará que está todo probado.

¿PARA QUE ME SIRVE?

- Medida de calidad de cómo barro
- Me indica cuándo parar de barrer
- Me sugiere qué más barrer

Unos criterios pueden ser más fuertes que otros, entonces el conocerlos me puede dar un indicador de qué tan profundas son las pruebas, cuándo aplicar uno y cuándo otro.

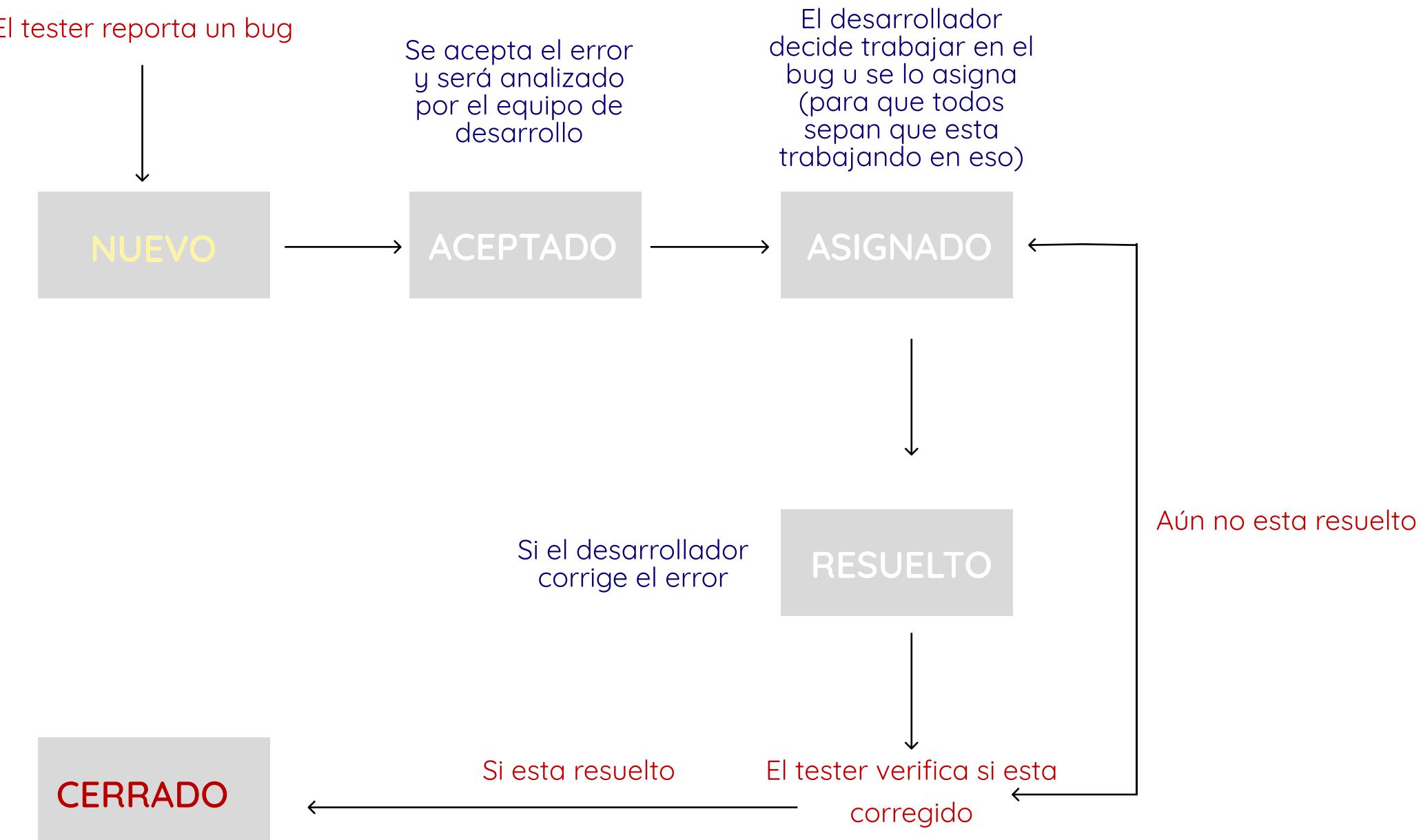
- Criterio 1: barrer cada habitación.
- Criterio 2: barrer cada pieza (habitaciones, comedor, cocina, baño, etc.).
- Criterio 3: barrer cada pieza, incluso en las esquinas, porque ahí hay más posibilidades de que se acumule suciedad.

UNIDAD 5

Ciclo de vida de un BUG



CICLO DE VIDA DE UN BUG



CICLO DE VIDA DE UN BUG

Bug, bicho, incidente, defecto, error, son distintas formas de llamarle a lo mismo. Es lo que los testers estamos buscando todo el tiempo. Como una Mantis que busca bichos para comerlos, y de ahí el nombre de una de las herramientas gratuitas más famosas, Mantis Bug Tracker (gestor de bugs)

Ciclo de vida de un incidente:

El flujo más típico de los incidentes (o bugs) es el que se representa en el siguiente esquema. Considera que en rojo aparecen las acciones que hace el tester y en azul aparecen las acciones que hace el desarrollador. Luego verás en las cajas las distintas situaciones en las que se encuentra un bug, las cuales son:

CICLO DE VIDA DE UN BUG

Puede pasar que el desarrollador no entienda el incidente y necesite más información. Por eso también veremos que hay bugs que aparecerán como “Se necesita más información”. Si llegase a suceder, el tester debe proveer más información y así el desarrollador podrá corregirlo. También hay situaciones en

las que un tester reporta un bug que quizá no era un bug, sino que el tester había entendido mal cómo debería funcionar el software, esperaba otra cosa, y por eso reportó el bug. En ese caso, el desarrollador pasará el bug ha “Resuelto” con una nota indicando que eso no era un error. El tester ahí deberá aceptar el comentario y cerrar el incidente, a menos que considere que sí es un error, con lo cual podría reabrirlo y continuar “discutiendo” con el desarrollador.

Realmente todos estos tipos de interacción serían muy complicados de seguir si no fuese porque contamos con herramientas que nos ayudan a ordenar nuestro trabajo.

REPORTE DE ERRORES

¿Qué NO puede faltar en un reporte de Error?

Los datos que todo reporte debería tener son:

Título: que describa el error.

El nombre del tester: En caso que haya preguntas de como reproducir el problema, es importante que los desarrolladores puedan identificar a la persona que reportó el incidente.

Fecha: Es importante capturar la fecha en que el incidente se reportó para poder saber cuánto tiempo se demoró en resolver.

Aplicación y versión: Es importante detallar que aplicación estamos probando, y que versión de la misma se está probando.

Ambiente del Tester: En caso de reportar un error web diríamos que navegador se estaba usando, en caso de ser un reporte de un incidente en un dispositivo móvil diríamos que teléfono/modeloversión del sistema operativo.

Los pasos: necesitamos comunicar cómo reproducir ese error, algo así como la receta para reproducir el problema. Es necesario detallar bien los pasos para poder reproducir el error y que no ocurra el caso del teléfono descompuesto.

Captura de Pantalla: Al mostrar los pasos de como reproducir, o el resultado obtenido, ayuda mucho poder tener capturas de pantalla que le permitan al desarrollador ver el problema más claramente.

Resultado Obtenido: El error de lo que obtuvimos en la prueba

Resultado Esperado: Lo que deberíamos obtener según la especificación.

Importancia: se suele tomar una escala de valores como bajo, medio, alto, crítico, y con esto se registra la importancia que se le debería dar al error, en base a qué tanto impacto puede tener en el usuario si encuentra el error, y en base a la probabilidad que el error aparezca (es un error en algo muy usado, entonces habrá mayor posibilidad de que suceda).

ESCRIBIENDO REPORTES DE ERROR EFECTIVOS

Todo reporte de incidente es una **comunicación escrita** al equipo del proyecto sobre la **calidad del software bajo prueba**. Muchas veces es tu habilidad para comunicar los incidentes de una aplicación, y no la severidad inherente a los bugs, lo que determina que los incidentes sean corregidos o no. Ustedes pueden estar pensando: “*Pero... yo odio escribir y no soy bueno para eso! ¿Cómo puede depender el destino del incidente del texto del reporte?*”

Es tentador pensar que los bugs hablan por sí mismos, que cualquier persona en su sano juicio puede automáticamente ver que cierto error es horrible y debería ser corregido.

Pero lamentablemente, no es lo que sucede. Pero la buena noticia es que su habilidad para comunicarse efectivamente con los desarrolladores y el equipo del proyecto, no está predeterminada por cómo les ha ido en la clase de español (o inglés) del colegio. No se trata de escribir una prosa fluida con palabras interesantes. Ni siquiera se trata de gramática y ortografía correcta. **Se trata de expresarse claramente, utilizando las palabras estrictamente necesarias para explicarse.**

Si se usa demasiadas palabras la idea se pierde, en cambio muy pocas palabras y los demás llenan los espacios con sus propias suposiciones. Se trata nada más que de saber **exactamente qué es lo que hay que comunicar**. Si no están seguros de la naturaleza del bug, entonces no importa cuán bien puedan escribir el reporte de error, igual nadie más sabrá tampoco de qué incidente se trata. Este reporte de incidentes presenta cuatro cosas que ustedes pueden comenzar a hacer desde ya para aumentar las chances de que las personas presten atención a los errores que reportan.

CONOZCAN A SU AUDIENCIA

En cualquier curso de escritura les dirán que deben conocer para quién están escribiendo. Los reportes de error no son una excepción.

Hay al menos dos audiencias para cualquier reporte de error: la persona que tiene que corregir el error y la persona o grupo que decide el destino del bug (a veces ambas cosas pueden ser hechas por la misma persona, pero aun así en cada rol tiene intereses/necesidades diferentes).

Vamos a llamar “**el desarrollador**” a la primera audiencia – es decir *la persona que debe corregir el error*. *El desarrollador* – precisa pasos claros e inequívocos para reproducir el problema, necesita detalles precisos sobre lo que hicieron y vieron.

Llamaremos a la segunda audiencia el “**Comité de Revisión de Errores**” – es decir *la persona o grupo que decide sobre el destino del bug*- precisan entender las consecuencias de elegir no corregir un error.

Para lograr que los errores se corrijan, su rol es ayudar al comité a ver que los riesgos de no corregir el error son mayores que los riesgos de corregirlo. Cuanto más comprendan cómo trabajan sus desarrolladores y el comité, mejor pueden adecuar los reportes de error a sus necesidades. Hagan el esfuerzo de conocer personalmente a sus interlocutores. Si pueden participar de las reuniones del Comité de Revisión de Bugs, háganlo. Aprenderán mucho sobre cómo piensan los destinatarios de los reportes de error.

Elian un buen título, la breve frase que describe el bug es generalmente denominada, el título o la descripción del error. Es la parte más importante del reporte de error. Los miembros del Comité de Revisión de Errores, a menudo deciden que un incidente puede aplazarse basándose solamente en el título; si éste es débil, los miembros podrían determinar que no vale la pena invertir más tiempo en el bug (al fin y al cabo, hay otros 145 más para revisar en las 2 horas siguientes)

He aquí algunos ejemplos:

EJEMPLOS

Demasiado largo: El programa se cae cuando eliges salir del menú de archivo, justo antes de que la base de datos se tornara accesible y estuvieras salvando los cambios en un registro.

Detalles insuficientes: El programa se cae.

Demasiado vago: Problema cuando la base de datos está fuera de línea.

El equipo del proyecto también usa el título para referirse al bug y para buscarlo. La gente tiende a recordar mejor las palabras que los números. Por lo tanto, en lugar de recordar el bug 23423 la gente retendrá el bug que “No se puede instalar en Windows 2000” y va a usar esas palabras claves para buscar el bug en el futuro.

Es difícil escribir una descripción de error buena y concisa. Es probable que pasen más tiempo elaborando el título perfecto para el bug que escribiendo el resto del reporte de error. Asegúrense que el título es suficientemente corto para ser desplegado enteramente en la pantalla del error (sin desplazamientos) y en los reportes generados por la herramienta de registro de incidentes. El título **NO** tiene que ser gramaticalmente perfecto, **tiene que ser corto e ir al grano**.

Describe clara e inequívocamente los pasos, estos pasos le suministran al Desarrollador la información sobre “dónde vive” el bug, para que pueda ser encontrado y corregido. También provee información al Comité de Revisión de Errores sobre las circunstancias en las que el error ocurre.

FORMA RECOMENDADA

1. Ejecutar la aplicación / Mantenimiento de Clientes
2. Editar un cliente
3. Modificar datos, pero no grabar todavía
4. Bajar el servidor de base de datos
5. Intentar grabar el cambio del cliente
6. Recibir un error de “timeout”
7. Cerrar la aplicación. Resultado: el programa se cae.

IMPRECISO

1. Ejecutar la aplicación
2. Consultar la base de datos por nuevos registros
3. Abrir el browser
4. Leer noticias en yahoo.com
5. Cerrar el browser
6. Ir al mantenimiento de Clientes
7. Filtrar por Categoría “Nuevos Clientes”
8. Editar un cliente
9. Modificar datos, pero no grabar aún
10. Bajar el servidor de base de datos
11. Intentar grabar el cambio del cliente
12. Recibir un error de “timeout”
13. Cerrar la aplicación.
Resultado: el programa se cae.

En este último ejemplo el tester transcribe todo lo que hizo antes de encontrar el error. Pero no se comprueba si todos los pasos, como leer las noticias de yahoo.com fueron necesarios. Si el tester trabaja para detallar el número de pasos que son absolutamente necesarios, los desarrolladores son menos propensos a decir que el error no se puede reproducir y los Comités de Revisión de Errores son menos propensos a decir que “nadie puede hacer todo esto”.

Pero ¿qué sucede si cada paso fuera necesario? Si los errores solo se manifiestan después de llevar a cabo todos los irrelevantes pasos. Debes escribir “paso necesario” ante tu aparente ilógico paso o debes agregar una nota al comienzo del reporte: **“Todos los pasos descriptos aquí son necesarios para reproducir el error”**. Escribiendo en forma clara los pasos muchas veces ayuda en el momento de verificar la solución del problema, especialmente si será otro tester quién realizará la verificación.

EXPLICAR LOS EFECTOS DEL ERROR, NO SÓLO LOS SÍNTOMAS:

Algunos reportes de errores son engañosos. Una primera impresión indicaría que no hace mucho daño, pero si se examinan las implicaciones se pueden descubrir problemas muy serios. Si estuvieras en el Comité de Revisión de Errores ¿qué error priorizarías primero?

- 1- Un reporte que dice: “un molesto diálogo te impide cerrar la aplicación”
- 2- Un reporte que dice: “la aplicación se cuelga al salir”

El error es el mismo, la diferencia es la forma en que el tester lo reportó. En este caso “un molesto diálogo” es la ventana que muestra Windows cuando no puede salir de un proceso. El tester encuentra el problema intentando apagar el equipo sin haber cerrado la aplicación. La aplicación no esperaba una entrada del usuario, por eso no había razón para que fallara en la salida. De hecho, este síntoma indica problemas más profundos, problemas que casi se pierden en el primer reporte (molesto diálogo que impide cerrar la aplicación).

Hay dos problemas con el “molesto” reporte del error.

Primero, es impreciso. Si el tester hubiera incluido en su reporte “el molesto dialogo”, quiénes deben solucionar el error hubieran reconocido en el diálogo un serio problema más que una molestia menor.

Segundo, el reporte no indica las consecuencias del error: la aplicación se cuelga.

Conclusión: todos pretendemos que nuestro trabajo marque la diferencia.

Queremos que el sistema liberado sea mejor porque nosotros trabajamos en él. Que podamos hacer una diferencia o no estará muy influido por nuestra habilidad para comunicar los errores encontrados. Cuando escribes un reporte de error, debes **recordar a tu audiencia, elegir un buen título, registrar los pasos en forma clara y explicar las consecuencias del error**. El reporte debe ser bueno por el esfuerzo extra que tú pones en él. Y cuanto mejor el reporte más chances de que la mayoría de los errores reportados sean solucionados. Y en última instancia ese es el punto, tener más errores solucionados antes que puedan afectar a los usuarios.