Lic. Marcelo Vinjoy
Ing Roberto Landaburu

Análisis de Software

TESTING (Pruebas).

La elaboración de un sistema de software requiere de una serie de actividades en las cuales es muy factible que aparezcan fallas. Dichas fallas pueden aparecer desde el primer momento, ya sea en la especificación de los objetivos, en el diseño ó en el desarrollo del proyecto. Debido a al imperfección natural del hombre, el desarrollo de software debe poseer una etapa que garantice la calidad. Dicha etapa suele llamarse *Pruebas* o *Testing*.
Las pruebas de un software garantizan la calidad del mismo. Igualmente, dan pie a una revisión final de las especificaciones, del diseño y de la codificación. El diseño de los casos de prueba de un programa consta de un conjunto de técnicas para la creación de casos de prueba que satisfagan los objetivos globales del testing.
Cada día se crean pruebas más minuciosas y bien planificadas, ya que muchos sistemas se están automatizando y es importante tomar en cuenta el costo de la reparación de una falla en el software elaborado. No es extraño que una empresa de desarrollo de software emplee mayor esfuerzo en la etapa de las pruebas que en la propia implementación del sistema, más si se trata de una programa de suma importancia como los de control de tráfico aéreo, control de reactores nucleares o equipos de medicina. Sin embargo, muchas veces suele subestimarse la importancia de las pruebas del software.

1. Definición del Testing

La Definición de las Pruebas a un determinado software se encuentra encerrada en las siguientes características:

1.- La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.

2.- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.

3.- Una prueba tiene éxito si descubre un error que no se haya detectado anteriormente.

Someter a pruebas (testing) un producto es la actividad que se dedica a encontrar fallas o errores en dicho producto. No se trata de determinar si funciona o no. La diferencia está en la actitud que tome el que elabore las pruebas, ya que no es lo mismo que alguien busque las fallas a que alguien quiera demostrar que todo está bien. Dijkstra recalca esta diferencia diciendo:

Lic. Marcelo Vinjoy
Ing Roberto Landaburu

"*Las pruebas sólo pueden demostrar la presencia de errores, no su ausencia*"

1.1 Fundamentos de la prueba del software

La prueba presenta una interesante contrariedad para el ingeniero de software. Durante las primeras fases de definición y de desarrollo, el ingeniero intenta construir el software partiendo de un concepto abstracto y llegando a una implementación tangible. A continuación, llega la prueba. El ingeniero crea una serie de casos de prueba que intentan *demoler* el software construido. De hecho, la prueba es uno de los pasos de la ingeniería del software que se puede ver (por lo menos, psicológicamente) como destructivo en lugar de constructivo.
Sin embargo, la gente que desarrolla software es constructiva por naturaleza. Para la elaboración de las pruebas es necesario descartar las ideas de predisposición negativa ante *la corrección* del software construido. De igual manera, es importante que se superen todos los conflictos que se generen a la hora de realizar las pruebas.

1.2 Las barreras psicológicas

A la hora de planificar un buen proceso de pruebas, se pueden presentar barreras que impiden la elaboración de un buen testing. Dichas barreras son de orden psicológico y cultural. Estos obstáculos pueden presentarse en cualquiera de las personas vinculadas a la realización del sistema. A continuación se enumeran algunos casos:

- Barreras por ser el programador.
- Barreras por ser miembro del equipo.
- Barreras como empleados de una empresa.

1.2.1 Barreras por ser el programador

El hombre por naturaleza crea mecanismos psicológicos de defensa. Es innato de ser humano que nunca va a predicar una falla en algo que le llevó mucho tiempo y esfuerzo. Al contrario, si el error existe, busca esconderlo o evitarlo. El programador que desarrolla un programa ha invertido mucho esfuerzo en tratar de hacerlo bien. Pedirle que busque donde falló es psicológicamente destructivo. Por algo reza el proverbio: " *No hay peor ciego que quien no quiere ver*".
Es posible que algunos de nosotros seamos capaces de afrontar el hecho que cometimos algunos errores, pero seguramente preferiríamos que fueran pocos, con lo cual se dificulta la búsqueda exhaustiva de fallas y errores.
Es interesante destacar que muchas personas crean un lazo de identificación con su trabajo; si el producto presenta fallas, ellos sienten que ellos también tienen fallas y no sirven. Ante esta situación, la persona se sentirá amenazada o atacada y para defenderse apela a una variedad de tácticas:

- Minimizar la importancia de la falla.
- Defender la falla como un "beneficio" o mejora.
- Buscar un chivo expiatorio a quien echarle la culpa.
- Olvidar la falla (y por ende la corrección que hay que hacer).
- Evitar las pruebas (en caso extremo, si no hay pruebas, no se encuentran fallas).
- Angustiarse.
- Disociarse de la responsabilidad por las fallas.
- Contraatacar a quien se percibe como la fuente de la situación ("esta empresa que exige...", "el inservible de mi jefe me obliga...", "las pruebas son una pérdida de tiempo...").

En algunas ocasiones se hace presente un personaje muy popular llamado *"bug"* (bicho), el cual se utiliza para designar un error de programación. Así el programador:

- Minimiza la falla - no es una falla severa, sino un "bichito" irritante.
- Encuentra un chivo expiatorio (son los "bugs" los que se metieron en el programa...).
- Se disocia de sus errores, los cuales cobran existencia "autónoma". Agrede a los "causantes" de los errores, los bichos, los cuales hay que "exterminar".

1.2.2 Barreras por ser Miembro de un Equipo:

Cuando no es el programador quien conduce las pruebas, el equipo humano de desarrollo puede levantar otras barreras.
En algunos medios, al encargado de las pruebas lo hacen sentirse como "el villano de la película". Todos los demás construyen un software, el tester busca destruirlo. El tester pasa a ser contrincante de los programadores. Incluso el gerente de un proyecto puede estar más interesado en "sacar el producto" que "sacar un buen producto", por lo que se une al papel de los programadores. En este sentido, el rol que desempeña el tester ante los programadores es alguien que pone las cosas más difíciles.
Sin embargo, el rol del tester es más afín al rol que debe desempeñar un buen médico durante un chequeo. Uno no le paga a ese médico para que le diga que todo está bien, sino para que busque "fallas" que puedan conducir al diagnóstico temprano de una enfermedad o situación de cuidado.
Vale la pena destacar que a algunas personas se le dificulta enormemente el decir "NO". El que prefiere guardar silencio, "pasarle la pelota" a otro ("a lo mejor si habla con fulanito...") o incluso posponer la decisión indefinidamente antes de responsabilizarse de una negativa. En otras ocasiones presenta excusas y rodeos para evitar decir "no". El problema para el tester es que su rol implica tener que decir no, en la mayoría de los casos debe decir : "el software NO está listo, NO se puede avanzar en el desarrollo, NO se puede entregar al cliente".

1.2.3 Barreras como empleado de una empresa

Algunas veces, la etapa de pruebas, puede estar orientada de acuerdo con la política con que trabaja la empresa. La compañía que desarrolla el software puede tener dos actitudes: la de entregar un buen programa con un mínimo de errores menores o un programa que no cumpla con los requisitos exigidos y presente gran cantidad de fallas. En la cultura de la empresa, es importante evaluar algunos casos:

- Hay una mentalidad corto-placista?. Entonces, en esa empresa siempre resultará más importante terminar rápidamente con un software que terminarlo bien. Por ende, el tester es quien interfiere con el objetivo prioritario ("cobrar por solo decir: "hemos terminado") por estar defendiendo un objetivo percibido por la empresa como mucho menos importante: "satisfacer al cliente". En este tipo de empresa se observa que:
    - Si el proyecto se retrasa, lo primero en sacrificarse son las pruebas (y la documentación).
    - Se argumenta que las pruebas son menos importantes que la programación, porque la programación produce código (el mismo tipo de mentalidad preconiza que un programa muy largo es mejor que uno más compacto, cosa que, como ya sabemos, es completamente falso). Hay que admitirlo, el proceso de prueba no produce código, produce calidad!.
    - Se desestiman las pruebas aduciendo que "sólo" tenemos que asegurar una funcionalidad mínima, no la perfección. El problema radica en saber ¿Cuál es el compromiso con el cliente? A todas estas empresas hay que insistirles que arreglar una falla después de entregado el software puede ser de 60 a 100 veces más caro que arreglarlo en el momento de detectarlo. Los costos adicionales incluyen:

    - Costos en que incurre el cliente por la falla.
    - Costos para la imagen del vendedor.
    - Costos de retirar las copias vendidas.
    - Costos de corregir el problema en el futuro, cuando los desarrolladores del sistema dejaron la empresa o se olvidaron de sus decisiones de diseño y construcción.
    -
- ¿Cómo se tratan las malas noticias?. En la antigüedad se recompensaba al mensajero de buenas noticias y decapitaba al mensajero de malas noticias. Comprensiblemente, los mensajes malos solían perderse por el camino. De igual manera, el personal encargado de las pruebas puede evitar decir los errores para no recibir sanciones o "malas caras". Es importante que la empresa y los programadores tengan un trato justo con el tester y que éste proporcione las

noticias (bien sea buenas o malas) de una forma correcta a las personas involucradas con la elaboración del sistema.

- ¿Cómo reacciona la empresa ante los errores de sus empleados? Por un lado, hay empresas que son bastantes tolerantes con los errores de sus empleados ("errar es de humanos", "uno aprende con sus errores", "equivocarse le proporciona al empleado la oportunidad de crecer en autonomía y responsabilidad"), lo que en ocasiones puede generar con el tiempo una actitud de irresponsabilidad por parte, tanto del empleado, como de la empresa. Por otro lado, hay empresas que responden al error con una serie controles, revisiones, chequeos, penalizaciones y castigos que suelen hacer más daño que bien, ya que existen ocasiones en que se afecta a un personal no involucrado con la falla. Hay empresas que buscan relacionar los sueldos de los empleados a la tasa de errores que cometen. Esto luce bien en teoría, pero suele resultar contraproducente en la práctica. De hecho no es extraño que ante esta situación los empleados se esfuercen ingeniosamente para evitar errores y que sus errores no se vea reflejado en los sueldos. Quizás el caso más extremo es el de cierta empresa que penalizaba a sus programadores según el número de errores que sus compañeros le encontraran. Al cabo de muy poco tiempo, el numero de errores por programador había descendido a cero, ya que los programadores no reportaban los errores que encontraban en el trabajo de sus compañeros.

- ¿Quién asume los errores en la empresa? Cada vez que se producen errores la empresa busca un chivo expiatorio, cuando el usuario reporta su insatisfacción por las fallas del software, los ingenieros invertirán mucho tiempo y esfuerzo cubriéndose las espaldas para evitar cargar con la culpa. Pero una vez que el usuario reclama, ¿qué pasos se toman para determinar el tamaño y el origen del error? ¿ Quién asume la responsabilidad de la corrección del error? ¿Quién da la cara ante los usuarios? A modo de ejemplo, considere él siguiente caso, reportado por la agencia Efe el 24 de Mayo de 1990:

La compañía telefónica Nippon Telegraph and Telephone (NTT), la mayor del mundo, recargó 12.2 millones de dólares a las facturas de sus clientes debido a un error en sus computadoras. Los recargos se descubrieron durante una investigación por todo el país, tras detectarse por reclamos de sus usuarios que la central de Yokohama cargaba en exceso las facturas de sus clientes. El presidente de la firma, Haruo Yamaguchi, presentó públicamente sus excusas y señaló que dichos recargos se produjeron por la errónea introducción de datos en las computadoras en 1985. Yamaguchi añadió que se devolverá el dinero antes del mes de julio próximo y que se gastarán 52 millones de dólares en un nuevo sistema de cobro que evite en lo sucesivo errores similares. Es importante tomar en cuenta los valores que se ponen en practica en situaciones como esta: la *honestidad* de la empresa se ve ratificada cuando la compañía decide devolver el dinero a sus clientes.

Es impresionante que una empresa tome la decisión de hacerse responsable en un caso como este e invertir dinero en corregir sus errores con la finalidad de reflejar una buena imagen ante sus usuarios.

1.3 Objetivos de la prueba

El objetivo fundamental es diseñar pruebas para encontrar diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo. Si la prueba se lleva a cabo con éxito (de acuerdo con el objetivo anteriormente establecido), se descubrirán errores en el software. Como ventaja secundaria, la prueba demuestra hasta qué punto las funciones del software parecen funcionar de acuerdo con las especificaciones y requisitos de rendimiento. Además, los datos que se van recogiendo a medida que se lleva a cabo la prueba proporciona una buena indicación de la finalidad del software y, de alguna manera, indican la calidad del software como un todo.

1.4 Principios de la prueba

Para la aplicación de métodos para el diseño de casos de prueba efectivos, un ingeniero de software deberá conocer los principios básicos que guían las pruebas del software, entre los que podemos mencionar:

- A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente. Como ya sabemos, el objetivo de las pruebas del software es descubrir errores. Aquellas fallas que impiden que el programa cumpla con sus requisitos son consideradas como defectos graves.
- Las pruebas deberían planificarse mucho antes de que empiecen. Una vez completado el modelo de requisitos, es recomendable comenzar con la planificación de las pruebas; y luego de consolidar el modelo de diseño, podemos comenzar con la definición detallada de los casos de prueba. Por tanto, se puede planificar y diseñar todas las pruebas antes de generar ningún código.
- No son posibles las pruebas exhaustivas. Es imposible ejecutar todas las combinaciones de caminos durante las pruebas. Es posible, sin embargo, asegurarse de que se han aplicado todas las condiciones del diseño de pruebas.

1.5 Lecciones

1. Para ser más efectivas, las pruebas deberían ser conducidas por un equipo independiente. Un programador (o equipo de programadores) no debe hacer sus propias pruebas.

2. Los resultados de las pruebas deben compararse automáticamente con los resultados esperados (para evitar lo de "no hay peor ciego que quien no quiere ver").

3. Hay que esforzarse en ver el lado positivo del rol de las pruebas.

4. Hay que evitar los plus de productividad relacionados con los errores. Si un programador recibe un plus especial por cometer pocos errores ejercerá presión psicológica sobre el tester para que lo evalúe positivamente. También puede ponerse de acuerdo con sus compañeros para realizar pruebas antes de pasarle su programa al tester. En este caso, la empresa paga dos veces por el mismo servicio. Si un tester recibe un plus por número de fallas detectadas, ejercerá presión sobre los programadores para que entreguen su producto antes que realmente está listo. Si a ambos grupos se les da un plus sobre la base de errores cometidos/detectados, se fomenta el rol de contrincantes entre los dos.

5. Hay que aprender a recibir reportes de fallas y hay que aprender a hacerlos.

6. Recuerde que un buen caso de prueba es el que tiene una alta probabilidad de encontrar un defecto. Por ende es conveniente incluir casos de prueba "inesperados".

2 El Proceso de Prueba

El proceso de prueba de un software consta de las siguientes etapas

1. Inspección del análisis (verifica si se cometieron errores o falla en la etapa de análisis).
2. Inspección del diseño (debe ser completo y eficiente).
3. Inspección del código (observar el entendimiento y facilidad del código).
4. Pruebas unitarias (probar cada módulo por separado).
5. Pruebas de integración (probar todos los módulos, verificando que compaginen entre sí).
6. Pruebas de validación de requerimientos (verificar que cumple con todos los requerimientos exigidos por el cliente).
7. Pruebas de sistema (ejecutar el programa para verificar si cumple con los requisitos exigidos).

Cada etapa pasa por tres fases:

1. Planificación de la etapa.
2. Ejecución de la etapa.
3. Evaluación de la etapa.

En general la fase de planificación de cualquier etapa de prueba requiere las siguientes actividades:

1. Designación del responsable de la etapa.
2. Identificación de los participantes en la etapa y sus roles.

3. Identificación y planificación de los recursos requeridos.
4. Estimación del tiempo y esfuerzo requerido.
5. Documentación del alcance de las pruebas.
6. Documentación de la filosofía de diseño de los casos de prueba.
7. Generación disciplinada y sistemática de casos de prueba, según la filosofía adoptada.
8. Documentación de los casos de prueba. Para cada caso de prueba la documentación debe incluir:
    (a) Justificación de su inclusión.
    (b) Datos de entrada e indicaciones de cómo correr el caso.
    (c) Configuración requerida para la prueba.
    (d) Resultados esperados.
    (e) (En ejecución se agrega) Resultados obtenidos.
    (f) (En ejecución se agrega) Tipo de defecto hallado. Esto puede constar de una indicación de superada o cómo mínimo una indicación de la severidad de la falla (falla grave, seria o menor), una descripción de la falla y una identificación tentativa de la etapa del desarrollo a que se atribuye (Análisis, Diseño, Codificación, etc).
9. Criterio de culminación de las pruebas.

En la fase de ejecución de los casos de prueba se agregan las actividades (e) y (f), indicadas anteriores, teniendo cuidado de indicar quién hizo las pruebas, en qué fecha y sobre qué configuración de máquinas y software. Se reportan los resultados a los desarrolladores y se planifica cómo se llevará a cabo la corrección de los errores detectados y cómo se revisará la versión corregida, de acuerdo con el Criterio de Culminación de Pruebas que haya sido fijado. En la fase de evaluación se busca diagnosticar las características del desarrollo del software. Fue efectivo el desarrollo? Qué etapas tardaron más? Fue exitoso el proceso de pruebas? Qué debilidades descubrió? Hay acciones que vale la pena emprender para mejorar el proceso de desarrollo en el futuro? Si se recogen estadísticas de los errores cometidos por etapa de desarrollo, dichas estadísticas pueden servir de apoyo para propuestas de mejora como la adquisición de una herramienta CASE, mayor entrenamiento en un lenguaje de programación, cambio de un lenguaje de programación, entrenamiento en técnicas de diseño o reestructuración de grupos de trabajo.

3 Pruebas de métodos

Durante las pasadas dos décadas ha habido una evolución de una gran variedad de métodos de diseño de casos de prueba para el software. Estos métodos proporcionaban un enfoque sistemático a la prueba. Más importante aún, los métodos proporcionaban un mecanismo de ayuda para asegurar lo completas que eran las pruebas y para conseguir la mayor probabilidad de descubrimiento de errores del software.

Cada método de una clase puede ser sometido a pruebas para determinar si está correctamente implementado de acuerdo con la especificación que contiene el modelo funcional.
Algunos métodos son lo suficientemente sencillos como para no ameritar muchas pruebas. En general, las estrategias de generación de casos de prueba se dividen en:

1. **Métodos de caja blanca:** La prueba de caja blanca, denominada a veces prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo, ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; ejecuten todos los ciclos en sus límites y con sus límites operacionales, y ejerciten las estructuras internas de datos para asegurar su validez. Con este método se determina cuáles son los casos de prueba a partir del código fuente del software y se utilizan las especificaciones para determinar el resultado esperado del caso. Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa. La prueba de caja blanca del software se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y\o ciclos. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado.

2. **Métodos de caja negra** Con este método los casos de prueba y los resultados se determinan a partir de la especificación funcional. Es decir, la prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. Una prueba de caja negra examina algunos aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica interna del software.

Se puede combinar los atributos de la prueba de caja blanca así como los de caja negra, para llegar a un método que valide la interfaz del software y asegure selectivamente que el funcionamiento interno del software es correcto.