Principios de las Pruebas y su Psicología

Pruebas de Sistemas

UNIVERSIDAD SIGLO 21

Principios de las pruebas y su psicología

Comenzaremos con la consideración de que entendemos un principio como una verdad aceptada que encarna una idea sobre la disciplina de las pruebas. Creemos que los principios, en cualquiera de los autores que tomemos, surgen a partir de las premisas de las pruebas, de su psicología y de los puntos clave o guía que pueden ser identificados.

Al presentarlos, pueden parecer intuitivamente obvios; sin embargo, son muchas veces pasados por alto y no tenidos en cuenta a la hora de hacer las pruebas.

Según los autores y comunidades de pruebas, la lista de principios cambia tanto en cantidad como en contenido. Hetzel (1988) estableció, por ejemplo, seis principios fundamentales, mientras que otros autores o comunidades señalan que los principios cambian con el correr de los años, ante los cambios de procesos, metodologías y evolución que tiene esta disciplina. Los principios fundamentales de las pruebas no son ajenos a este desarrollo.

Comenzaremos por los principios que datan de la década de los 80, pero también tomaremos algunos más actuales que demuestran la evolución.

Principio N.° 1: La prueba completa no es posible

> De las siguientes expresiones, que frecuentemente escuchamos, como pararé de probar cuando esté seguro de que funciona, o implementaremos el sistema tan pronto como todos los errores sean corregidos, podemos ver que asumen a la prueba como una actividad que puede terminar y que siempre se tiene la certeza de que todos los defectos son removidos. (Uninotas, 2017, https://bit.ly/2Uzfzjz)

Sin embargo, no podemos esperar lograr la prueba completa, que otros autores denominan como **prueba exhaustiva**, por dos razones: una, limitaciones prácticas, y dos, imposibilidad teórica.

Dado cualquier programa, por más pequeño y simple que sea, será imposible ejecutar todos los casos de pruebas posibles. Si tomamos un ejemplo similar al dado por Hetzel (1988), donde un simple programa recorre una lista de palabras e imprime el primero que encuentra bajo la cadena **prueba**, ¿cómo podremos determinar cuándo hemos terminado de probarlo completamente?

¿Qué probaremos?, ¿dos casos únicamente? Uno, con una lista que tenga la palabra *prueba* y otro, con una lista que no la contenga.

¿Probaremos diferentes longitudes de listas? ¿Importa que probemos con la palabra *prueba* en diferentes partes de la lista?

Todas estas son variables que podrían afectar el funcionamiento apropiado de un programa. Entonces, ¿cuántas pruebas son suficientes?

> Los probadores o *testers* deben seleccionar un pequeño *set* de pruebas que incluye estas posibilidades. Y ese *set* debe ser lo suficientemente grande para proporcionar un nivel satisfactorio de confianza, pero no tan grande, de modo que resulte impráctico de administrar. (Uninotas, 2017, https://bit.ly/2Uzfzjz)

Y, si no podemos testear todo, ¿qué podemos hacer?

- Gestionar y reducir el riesgo.
- Priorizar los casos para enfocar las áreas principales de riesgo.
- Distribuir el tiempo de pruebas según el grado de los riesgos involucrados.
- Entender el riesgo del negocio.

Principio N.° 2: El trabajo de pruebas es creativo y difícil

Existen falsas creencias como las siguientes:

- Hacer pruebas es fácil.
- Cualquiera puede hacer pruebas.
- No se requiere ni experiencia previa ni entrenamiento.
- Hacer pruebas es aburrido y rutinario.
- No está involucrado personal *senior*.

- El personal de pruebas no tiene crecimiento profesional, a menos que pase a desempeñarse como programador.

Esto no es así, ya que, para probar efectivamente, se debe entender el sistema en su totalidad y los sistemas no son simples ni sencillos de comprender. No se puede comenzar a diseñar y desarrollar una prueba efectiva hasta que no se haya entendido detalladamente lo que se supone que hace el sistema. Mientras que para esto se destinan a los mejores analistas de negocios, se asume equivocadamente que los probadores o *testers* deben tener este conocimiento como prerrequisito.

"Hetzel (1988) establece los siguientes ingredientes esenciales para una buena prueba:

- Creatividad e intuición.
- Conocimiento de negocio.
- Experiencia en pruebas.
- Metodología de pruebas". (Uninotas, 2017, https://bit.ly/2Uzfzjz)

A través de un libro, un curso o una materia, se pueden aprender las metodologías de pruebas y también tener una perspectiva para aprender de la experiencia colectiva. Mientras que el conocimiento del negocio se aprende en el trabajo, la creatividad e intuición no se pueden enseñar, sino que dependen del individuo.

Principio N.° 3: Una importante razón de las pruebas de sistemas es prevenir que ocurran las deficiencias

Si bien muchos profesionales no ven a las pruebas como una actividad de prevención de defectos y sí como la actividad de encontrarlos, este principio está enfocado en una de las últimas visiones, que entiende a las pruebas como algo que se realiza a lo largo del ciclo de vida de desarrollo y no como algo que ocurre en una fase.

Hay muchos entregables asociados a cada fase de desarrollo sobre los cuales se pueden hacer pruebas que ayuden a detectar de manera temprana las deficiencias que podrían estar presentes. De este modo, no resulta tan costosa su corrección y se pueden prevenir los problemas posteriores derivados. El mejor proceso de pruebas es el que da un rápido *feedback*.

También abarca la creencia de que pensar primero los casos de pruebas con los que se va a probar, no bien se han definido los requerimientos, ayuda a evitar ciertos errores o darse cuenta de que existen deficiencias, además de incrementar nuestro conocimiento sobre el sistema. Algunos autores llaman a este principio *pruebas tempranas*.

La tarea de diseñar los casos de pruebas es un mecanismo muy efectivo para la prevención de errores. El proceso de crear buenos casos de pruebas puede descubrir y eliminar problemas de cualquiera de las etapas de desarrollo (Beizer, 1983).

Principio N.° 4: La prueba se basa en el riesgo

Una buena prueba es aquella basada en el riesgo… ¿En el riesgo de qué? En el riesgo de falla.

¿Cuántas pruebas se está dispuesto a realizar si se tiene una baja probabilidad de ocurrencia de una falla? De la misma manera, ¿cuántas pruebas se está dispuesto a realizar para encontrar un defecto que costaría la vida encontrarlo?

La cantidad de pruebas que deberíamos hacer depende directamente en el riesgo involucrado. Sistemas con alto riesgo requieren más casos de pruebas y más énfasis en las pruebas; por el contrario, programas con bajo riesgo o con un impacto de falla limitado no justifican la misma prueba.

Este principio es corolario del primer principio: si no podemos probar todo, debemos priorizar y adecuarnos a los recursos disponibles. El riesgo debe ser usado como la base para asignar el tiempo disponible para pruebas y para ayudarnos a hacer la selección de qué pruebas hacer y por dónde priorizar con las pruebas.

Principio N.° 5: La prueba debe ser planeada

Si bien todos parecen estar de acuerdo con este principio, no todos están disciplinados en planificar las pruebas. Este principio también es corolario del principio número 1 (que dice, en otras palabras, que no se puede probar todo), e indica que debe hacerse una selección y una planificación cuidadosa de las pruebas. Según la selección que se haga, se obtendrá una buena prueba o una prueba pobre.

La etapa de planificación dentro de un proceso de pruebas es así:

- Pensar en un enfoque global.
- Diseñar las pruebas.
- Establecer los resultados esperados.

Como salida de esta etapa de planificación, está el **plan de pruebas**, que define los objetivos y el enfoque de las pruebas. Puede ser formal, si es escrito y está a disposición de manera permanente como un documento del proyecto, o informal, cuando existe en notas temporales o en la cabeza

de alguien, sin la intención de ser grabado o revisado. Como se puede apreciar, se considera la actividad de diseño de los casos de pruebas dentro de la etapa de planificación, cuando otras bibliografías lo consideran como dos tareas separadas. A los efectos de explicar este principio, no tiene importancia.

Principio N.° 6: La actividad de pruebas requiere independencia

Cuando se requiere una medición objetiva, se requiere también una persona objetiva e imparcial que haga dicha medición. Lo mismo ocurre con las pruebas, por dos motivos:

1. Las personas tienden a pasar por alto sus propios defectos. Entonces, los desarrolladores corren el riesgo de no reconocer defectos evidentes.

2. El desarrollador nunca va a analizar su "creación" (el código generado) de forma imparcial, ya que tiene con él un apego afectivo, aunque conozca el objeto de prueba mejor que nadie.

Esto puede resultar contradictorio con la premisa de que no se puede probar lo que no se conoce. Sin embargo, las personas a cargo de las pruebas deben estar involucradas en el proceso de desarrollo desde sus inicios, pero, al momento del realizar las pruebas, estas deben ser aplicadas por personas imparciales y objetivas.

Los programadores entienden el sistema y, por ello, son condescendientes con él; fundamentalmente, están dirigidos por el entregable. Por el contrario, los probadores o testers deben lograr entender el sistema para probarlo, pero, en su tarea de probar, intentarán provocar fallas y, lo que es muy importante, están dirigidos por la calidad. Deben estar comprometidos con la búsqueda de errores y no con la defensa del sistema.

Hay distintos niveles de independencia según las organizaciones, que van desde los niveles de cero independencia, hasta los niveles más altos. Entonces, las pruebas puede hacerlas:

- quien escribe el código;
- otra persona;
- una persona de diferente sección;
- otra persona de diferente organización.

A la selección de casos no la hace una persona, sino una herramienta.

Además de estos principios, presentaremos aquí ciertas premisas que pueden tomarse como principios o derivados de los que ya hemos presentado, pero que resulta interesante tenerlos en cuenta.

- Los casos de pruebas deben ser escritos tanto para condiciones de entradas válidas y esperadas, como para inválidas e inesperadas. (Vega Fernandez, 1999, https://bit.ly/2GaE2c0)

Como corolario de lo anterior, se dice que una parte de las pruebas es probar un programa para ver si no hace lo que se supone que hace, y la otra parte es para probar si un programa hace los que se supone que hace.

Myers (1979), entre sus varios principios, aporta este, que resulta interesante en la explicación que da, pero que deja de estar vigente debido a la magnitud que hoy ha tomado el mantenimiento de *software*.

- Evita desechar los casos de pruebas, a menos que el programa sea ya un verdadero programa.

Aún es frecuente ver en la práctica que quienes realizan las pruebas se sientan en una terminal, inventan casos de pruebas en el aire y comienzan a ejecutarlos sobre el programa. La cuestión es que los casos de pruebas son una valiosa inversión que, en la situación planteada, desaparece después de que las pruebas son ejecutadas. Entonces, cuando el programa requiere ser probado nuevamente, reinventar los casos de pruebas requiere una cantidad de trabajo que ciertamente pocas veces se hace. Esto lleva a que el *retesting* del programa raramente es tan riguroso como la prueba original, con todo lo que esto trae aparejado.

Este principio carece de toda importancia si se tiene un proceso de pruebas maduro donde el diseño de casos de pruebas se realiza y queda registrado y la ejecución se hace contra ese diseño.

- Nunca planificar el esfuerzo de pruebas y suponer que no se encontrarán errores.

Suele ser un error frecuente entre los gerentes de proyectos. Además, refleja una incorrecta visión de las pruebas, que consiste en asumir que son el proceso de mostrar que el programa funciona correctamente.

- La probabilidad de existencia de mayor cantidad de errores en una sección de un programa, es proporcional al número de errores ya encontrados en esa sección.

Hay innumerables ejemplos medidos que datan del nacimiento de este principio, allá por 1979, cuando Glendford J. Myers lo escribe en su libro

The art of software testing. Actualmente, se aplica la conocida ley de Paretto (con cierto grado de verdad), que dice que el 80 % de los defectos se encuentra en el 20 % del código base. Por ende, si una aplicación tiene 1000 errores conocidos y se mira el detalle de cómo se distribuyen, quizás la mayoría esté agrupada en el 20 % de los módulos del sistema.

Hasta aquí hemos desarrollado los principios de Bill Hetzel (1988) y algunos más. A continuación, listaremos los siete principios más actuales, dados por el International Software Testing Qualifications Board (ISTQB) (2011) y desarrollamos todo aquello que consideramos que se agrega a lo anterior.

1. **El proceso de pruebas demuestra la presencia de defectos:** este principio atiende a la psicología misma de las pruebas. Dicho de manera contraria: no se tiene que probar para demostrar que funciona. Esto tiene que ver con el último principio de la independencia de las pruebas, ya que demostrar que funciona es una acción natural del programador. Si las pruebas reducen la probabilidad de presencia de defectos que permanecen sin ser detectados, entonces se deben seleccionar datos para los casos de pruebas que pueden detectar errores.

2. **No es posible realizar pruebas exhaustivas**: este es idéntico al principio número 1, que dice que la prueba completa no es posible.

3. **Pruebas tempranas**: está encubierto dentro del principio número 3.

4. **Agrupamiento de defectos (*defect clustering*)**: este principio merece la explicación, pues este concepto no fue contemplado junto a los principios de Hetzel (1988). Sostiene que los defectos aparecen agrupados como hongos o cucarachas: donde se encontró un defecto, se encontrarán otros "cerca". De alguna manera, requiere cierta flexibilidad de los probadores, ya que, si siguen un diseño de pruebas, es importante considerar el rumbo de las pruebas posteriores. La identificación o localización de un defecto puede necesitar ser investigada con un mayor grado de detalle; por ejemplo, realizar pruebas adicionales o modificar pruebas existentes.

5. **Paradoja del pesticida**: pareciera que le fue asignado un nombre de fantasía. ¿Qué es la paradoja del pesticida? Si siempre se ejecutan las mismas pruebas de forma reiterada, no se podrán encontrar nuevos defectos, es decir, que es como si el sistema se hiciera inmune a los casos de pruebas elegidos. Lo que indica este principio es que repetir pruebas en las mismas condiciones no es efectivo, de modo que se necesita revisar o modificar las pruebas regularmente para los distintos módulos.

6. **Las pruebas dependen del contexto**: simplemente, objetos de prueba diferentes son probados de forma diferente. Este principio indica,

además, que, si bien siempre habrá desviaciones entre el entorno de pruebas y el entorno de producción, se deben tratar de minimizar dichas diferencias, ya que estas desviaciones pondrán en tela de juicio las conclusiones que se obtengan tras las pruebas.

7. **La falacia de la ausencia de errores**: este principio es similar en su explicación al principio número 4 de Hetzel (1988), aunque indica claramente que, en la mayoría de los casos, el proceso de pruebas no detectará todos los defectos del sistema, pero los defectos más importantes deberían ser detectados, y que esto por sí solo no prueba la calidad del software. No se puede introducir la calidad a través de las pruebas, sino que tiene que construirse desde el principio.

Referencias

Beizer, B. (1983). *Software Testing Techniques.* New York, US: Van Nostrand-Reinhold.

Hetzel, B. (1988). *An Introduction. The Complete Guide to Software Testing* (2nd ed). Massachusetts, US: QED Information Science.

International Software Testing Qualifications Board (ISTQB). (2011). *Fundamentals of Testing*. *Certified Tester. Foundation Level Syllabus*. Recuperado de https://www.istqb.org/downloads/send/2-foundation-level-documents/3-foundation-level-syllabus-2011.html

Institute of Electrical and Electronics Engineers (IEEE). (1990). *Standard 610: Glossary of Software Engineering Terminology*. Recuperado de https://ieeexplore.ieee.org/document/159342/

Myers, G. (1979). *The Art of Software Testing.* New York, US: John Wiley & Sons.

Uninotas (2017). *Prueba, implementaciòn y mantenimiento del sistema.* Recuperado de: https://www.uninotas.net/prueba-implementacion-y-mantenimiento-del-sistema-3/

Vega Fernandez, J. (1999). *"Pruebas de Código".* Recuperado de: https://slideplayer.es/slide/141487/