Pruebas de software

Tópicos Avanzados en Ing.de Sw

María Antonieta Soto Ch.

2019-1

Material de Sommerville, Ian. Software Engineering, 10th ed. Pearson, 2014

1

Contenido

- ✧ Pruebas de desarrollo
- ✧ Desarrollo conducido por pruebas
- ✧ Pruebas de versión
- ✧ Pruebas de usuario

2

Pruebas de programa

- ✧ Las pruebas intentan demostrar que el software hace lo se intenta que haga y descubrir defectos antes de usarlo.
- ✧ Al probar software, se ejecuta un programa usando datos artificiales.
- ✧ Se revisan los resultados después de realizar la prueba buscando errores, anomalías o información acerca de atributos no funcionales del programa.
- ✧ Puede revelar la presencia de errores, NO su ausencia.
- ✧ Las pruebas son parte de un proceso de verificación y validación más general, que también incluye técnicas de validación estáticas.

3

Objetivos de las pruebas de software

- ✧ Demostrar al desarrollador y al cliente que el software satisface los requisitos.
  - Para el software a medida, esto significa que debería haber al menos una prueba por cada requisito presente en la documentación de requisitos. Para el software genérico, significa que debería haber pruebas para todas la características del sistema, junto con la combinación de estas que se incorporarán en la versión del producto.
- ✧ Descubrir situaciones en las que el comportamiento del software es incorrecto, indeseable o no se ajusta a su especificación.
  - La prueba de defectos se ocupa de erradicar el comportamiento indeseable del sistema tales como caídas, interacciones no deseadas con otros sistemas, cálculos incorrectos y corrupción de datos.

4

Pruebas de validación y de defectos

- ✧ El primer objetivo conduce a la prueba de validación
  - Se espera que el sistema se desempeñe correctamente al usar un conjunto de casos de prueba dado que reflejen el uso esperado del sistema.
- ✧ El segundo objetivo conduce a las pruebas de defectos
  - Los casos de prueba se diseñan para exponer defectos los casos de prueba de defectos pueden ser deliberadamente confusos y no necesitan reflejar cómo se usa normalmente el sistema.

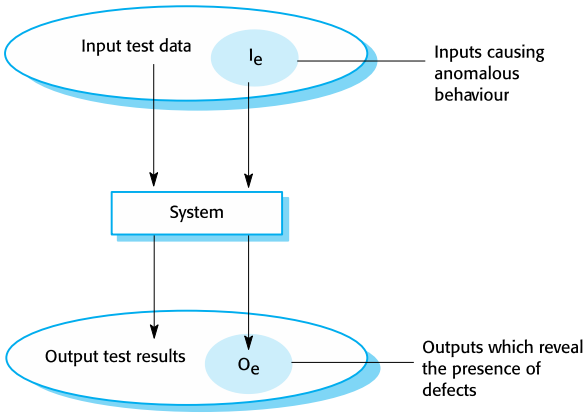
5

Objetivos del proceso de pruebas

- ✧ Pruebas de validación
  - Demostrar al desarrollador y al cliente del sistema que el sistema satisface los requisitos.
  - Una prueba exitosa muestra que el sistema funciona como se espera.
- ✧ Prueba de defectos
  - Descubrir fallas o defectos en el software donde su comportamiento es incorrecto y no se ajusta a su especificación.
  - Una prueba exitosa es una prueba que hace que el sistema se comporte incorrectamente y de ese modo expone un defecto en el sistema.

6

Modelo de entrada/salida de una prueba de programa



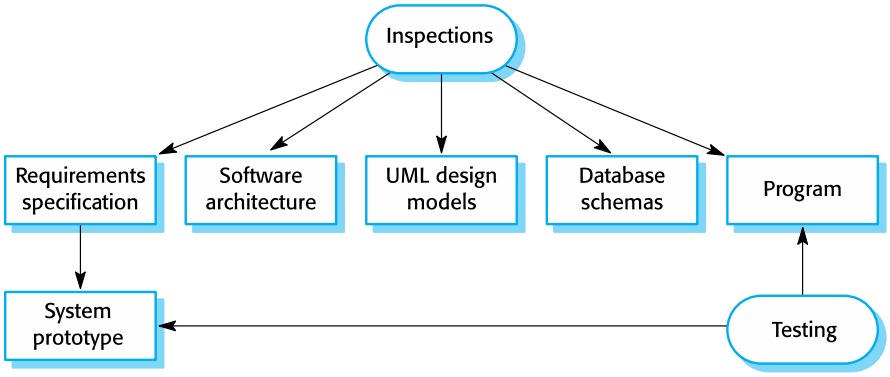
7

Inspecciones y pruebas

- ✧ Inspecciones de software. Se ocupa del análisis de la representación estática del sistema para descubrir problemas (verificación estática)
  - Se puede complementar con análisis de documentos y código basado en herramientas.
  - Abordado en el capítulo 15.
- ✧ Prueba de software se ocupa de ejercitar y observar el comportamiento del producto (verificación dinámica)
  - El sistema se ejecuta con datos de prueba y se observa su comportamiento operacional.

8

Inspecciones y pruebas



9

Inspecciones de software

- ✧ Estas involucran a personas que examinan la representación en código fuente con el fin de descubrir anomalías o defectos.
- ✧ Las inspecciones no requieren la ejecución de un sistema así que se puede utilizar antes de la implementación.
- ✧ Se pueden aplicar a cualquier representación del sistema (requerimientos, diseño, datos de configuración, datos de prueba, etc.).
- ✧ Han mostrado ser una técnica efectiva para descubrir errores de programa.

10

Ventajas de las inspecciones

- ✧ Durante las pruebas, los errores pueden enmascarar (ocultar) otros errores. Debido a que la inspección es un proceso estático, no hay que preocuparse de las interacciones entre errores.
- ✧ Las versiones completas de un sistema se pueden inspeccionar sin costos adicionales. Si un programa (sw) está incompleto, se necesitará construir resguardos (sw) especializados para probar las partes que están disponibles.
- ✧ Además de buscar defectos de sw, una inspección puede considerar atributos más amplios de calidad de un sw, tales como cumplimiento de estándares, portabilidad y mantenibilidad.

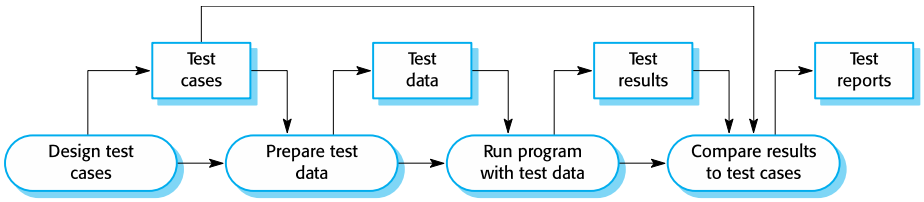
11

Inspecciones y pruebas

- ✧ Las inspecciones y las pruebas son técnicas de verificación complementarias y no opuestas.
- ✧ Ambas se deberían usar durante el proceso de V & V.
- ✧ Las inspecciones pueden verificar la conformidad con una especificación, pero no la conformidad con los requisitos reales del cliente.
- ✧ Las inspecciones no pueden verificar requisitos no funcionales tales como rendimiento, usabilidad, etc.

12

Un modelo del proceso de prueba de software



13

Etapas de prueba

- ✧ Pruebas de desarrollo, donde el sistema se prueba durante el desarrollo para descubrir errores y defectos.
- ✧ Pruebas de versión, donde un equipo de prueba, distinto al de desarrollo, prueba una versión completa del sistema antes de ser entregado a los usuarios.
- ✧ Las pruebas de usuario, donde los usuarios o los potenciales usuarios de un sistema prueban el sistema en su propio entorno.

14

Pruebas de desarrollo

15

Pruebas automatizadas

- ✧ Siempre que sea posible, se deben automatizar las pruebas de unidad de manera que las pruebas se ejecuten y verifiquen sin intervención manual.
- ✧ En las pruebas de unidad automatizadas, se puede usar un framework xUnit para automatización de pruebas (tal como JUnit) para escribir las pruebas de software.
- ✧ Los frameworks de pruebas de unidad proporcionan clases de prueba genéricas que se pueden extender para crear casos de prueba específicos. Luego, estos pueden ejecutar todas las pruebas que se han implementado e informar, a menudo a través de alguna GUI, el éxito o no de las pruebas.

16

Componentes de pruebas automatizadas

✧ En las pruebas se distinguen tres secciones:

```
public class NameNormalizerTests {
    @Test
    public void FirstLetterUpperCase() {
        // Arrange
        String name = "pablo rodriguez";
        NameNormalizer normalizer = new NameNormalizer();

        // Act
        String result = normalizer.FirstLetterUpperCase(name);

        // Assert
        assertEquals("Pablo Rodriguez", result);
    }
    …
}
```

1
2
3

Herramientas de pruebas

✧ Hay distintos tipos de herramientas de prueba:

- Herramientas de gestión de pruebas
  - Tarantula, Testcube, QABook, Testopia, QaManager, Radi,…
- Herramientas de automatización de pruebas para sw móvil
  - Appium, Calabash, Frank, MonkeyTalk, Robotium,…
- Herramientas de prueba de diseño responsivo
  - ResponsiveTest, Responsinator, Responsive, Screenfly,…
- Herramientas de prueba de aplicaciones web
  - Jmeter, Grinder, Multi-Mechanize, Selenium, Capybara,…

✧ http://www.testingexcellence.com/

Elegir casos de prueba de unidad

- ✧ Los casos de prueba deben mostrar que, cuando se usan como se esperaba, el componente que se está probando hace lo que se supone debe hacer.
- ✧ Si hay defectos en el componente, estos se deberían revelar mediante los casos de prueba.
- ✧ Esto conduce a 2 tipos de casos de prueba de unidad:
  - El primero debe reflejar la operación normal de un programa y debe mostrar que el componente funciona como se espera.
  - El otro tipo de caso de prueba debe considerar la aparición de problemas comunes. Debe usar entradas anormales para verificar que estas se procesan adecuadamente, sin colapsar el componente.

19

Estrategias de prueba

- ✧ Pruebas de partición, en la que se identifican grupos de entradas que tienen características comunes y se deberían procesar del mismo modo.
  - Se deben elegir pruebas de cada uno de estos grupos.
- ✧ Pruebas basada en lineamientos, en la que se usan lineamientos para escoger los casos de prueba.
  - Estos lineamientos reflejan la experiencia previa de los tipos de error que los programadores comenten a menudo cuando desarrollan componentes.

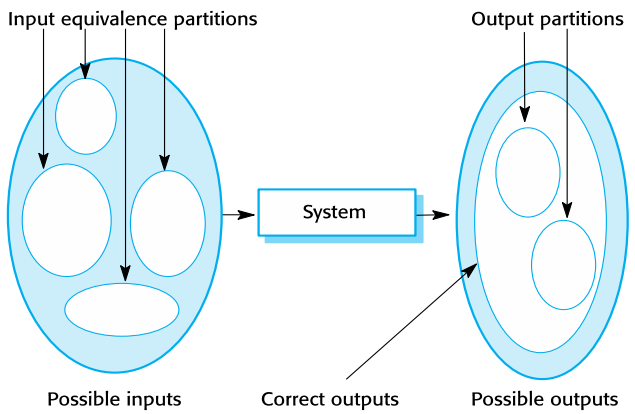
20

Pruebas de partición

- Los datos de entrada y los resultados (salida) suelen caer en diversas clases. Todos los miembros de una clase se relacionan.
- Cada una de estas clases es una partición de equivalencia o dominio donde el programa se comporta de manera similar con cada miembro de la partición.
- Los casos de pruebas se deben elegir desde cada partición.

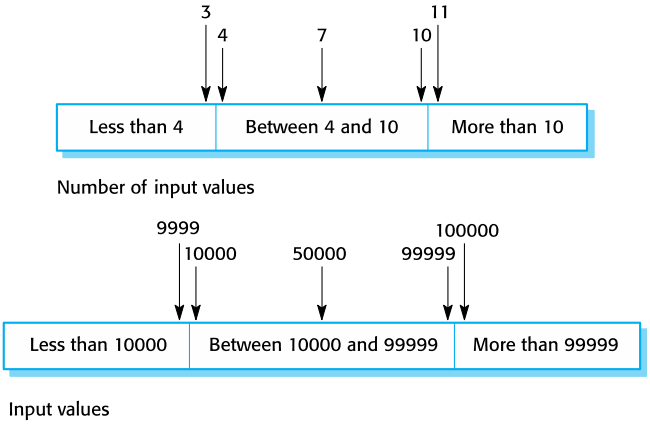
21

Partición de equivalencia



22

Partición de equivalencia



23

Lineamientos (secuencias) de pruebas

- ✧ Probar el software con secuencias que tienen un único valor.
- ✧ Usar secuencias de tamaños diferentes en distintas pruebas.
- ✧ Derivar pruebas de manera que se acceda al primero, intermedio y último elemento de la secuencia.
- ✧ Probar con secuencias de largo cero.

24

Lineamientos generales de prueba

- ✧ Elegir entradas que obliguen al sistema a generar todos los mensajes de error
- ✧ Diseñar entradas que desborden los buffers de entrada
- ✧ Repetir la misma entrada o series de entradas numerosas veces
- ✧ Forzar la generación de salidas inválidas
- ✧ Forzar resultados de cálculo demasiado grandes o demasiado pequeños.

Pruebas de componentes

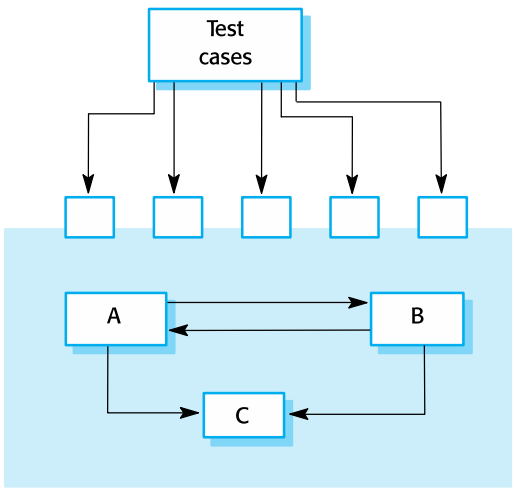
- ✧ Los componentes de software son a menudo componentes compuestos constituidos por varios objetos en interacción.
  - Por ejemplo, en el sistema weather station, el componente de reconfiguración incluye objetos que tratan con cada aspecto de la configuración.
- ✧ El acceso a la funcionalidad de estos objetos es a través de la interfaz definida del componente.
- ✧ Probar los componentes compuestos debe, por lo tanto, enfocarse en mostrar que la interfaz del componente se comporta de acuerdo a su especificación.
  - Se puede asumir que las pruebas de unidad a los objetos individuales dentro del componente se han realizado.

Pruebas de interfaz

- Los objetivos son detectar fallas debido a errores en la interfaz o suposiciones no válidas acerca de las interfaces.
- Tipos de interfaz
  - Interfaces de parámetro. Los datos se pasan de un procedimiento o método a otro.
  - Interfaces de memoria compartida Un bloque de memoria se comparte entre procedimientos o funciones.
  - Interfaces de procedimiento Un subsistema encapsula un conjunto de procedimientos que pueden ser llamados por otros subsistemas.
  - Interfaces de pasaje de mensajes Subsistemas solicitan servicios de otros subsistemas.

27

Pruebas de interfaz



28

Pruebas de sistema

- ✧ Las pruebas de sistema durante el desarrollo incluyen la integración de componentes para crear una versión del sistema y luego probar el sistema integrado.
- ✧ El objetivo de las pruebas de sistema es probar las interacciones entre los componentes.
- ✧ Las pruebas de sistema verifican que los componentes son compatibles, interactúan correctamente y transfieren los datos correctos en el momento adecuado a través de sus interfaces.
- ✧ Las pruebas del sistema prueban el comportamiento emergente de un sistema.

29

Pruebas de sistemas y de componentes

- ✧ Durante las pruebas de sistema, los componentes reusables que se han desarrollado por separado y los sistemas comerciales se pueden integrar con componentes desarrollados recientemente. Entonces se prueba el sistema completo.
- ✧ Los componentes desarrollados por diferentes miembros o grupos de miembros del equipo se pueden integrar en esta etapa. La prueba de sistema es un proceso colectivo más que individual.
  - En algunas empresas, las pruebas de sistema implican un equipo de pruebas independiente, sin la participación de diseñadores, ni programadores.

30

Pruebas de sistema usando casos de uso

- ✧ Los casos de uso desarrollados para identificar las interacciones del sistema se pueden usar como base para las pruebas de sistema.
- ✧ Cada caso de uso generalmente implica varios componentes del sistema, de modo que probar el caso de uso obliga a que estas interacciones ocurran.
- ✧ Los diagramas de secuencia asociados con el caso de uso documentan los componentes y las interacciones que se están probando.

31

Políticas de pruebas

- ✧ Las pruebas exhaustivas de un sistema son imposibles, por lo que se pueden definir políticas de prueba que definen la cobertura requerida de pruebas de sistema.
- ✧ Ejemplos de políticas de prueba:
  - Se deben probar todas la funciones del sistema que se acceden desde menús.
  - Se deben probar todas las combinaciones de funciones (p.e. formateo de texto) que se acceden a través del mismo menú.
  - Se deben probar todas las funciones a las que se le proporcionen datos del usuario, usando entradas correctas e incorrectas.

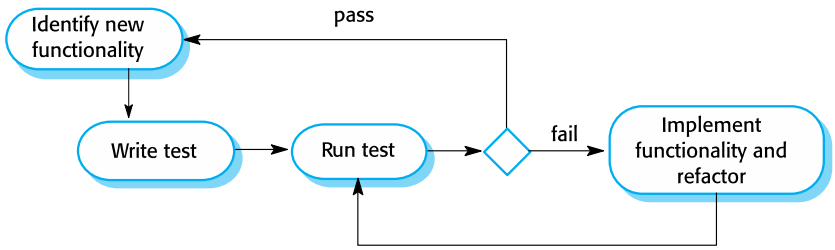
32

Desarrollo dirigido por pruebas (TDD, Test-driven development)

Desarrollo dirigido por pruebas

- ✧ TDD es un enfoque para desarrollar software en el que se entrelazan el desarrollo de pruebas y código.
- ✧ Las pruebas se escriben antes de codificar y “pasar” las pruebas es el factor clave del desarrollo.
- ✧ Se desarrolla código incrementalmente, junto con una prueba para ese incremento. No se avanza al siguiente incremento hasta que el código desarrollado pase las pruebas definidas para dicho código.
- ✧ TDD se introdujo como parte de las métodos ágiles tales como Programación Extrema. Sin embargo, se puede usar también en procesos conducidos por plan.

Desarrollo dirigido por pruebas



35

Actividades del proceso

- ✧ Comenzar por identificar el incremento de funcionalidades que se necesita. Este debería ser, por lo general, pequeño y posible de implementar en unas pocas líneas de código.
- ✧ Escribir una prueba para esta funcionalidad e implementarla como prueba automatizada.
- ✧ Ejecutar la prueba, junto con otras pruebas que se han implementado. inicialmente, la funcionalidad no se ha implementado así que la nueva prueba fallará.
- ✧ Implementar la funcionalidad y volver a ejecutar la prueba.
- ✧ Unas vez que se ejecutan todas la pruebas, se pasa a implementar la siguiente funcionalidad.

36

Test Unitarios y xUnit

- ✧ xUnit representa a los diferentes frameworks basados en el original SUnit.
- ✧ SUnit fue creado por Kent Beck para la plataforma SmallTalk y se ha portado a una gran variedad de lenguajes y plataformas
- ✧ Algunos xUnit existentes son:
  - Java (JUnit), .Net (NUnit), Python (PyUnit), Ruby (Rubyunit), Perl (PerlUnit), C++, (CppUnit), etc.

Beneficios del desarrollo dirigido por pruebas

- ✧ Cobertura de código
  - Cualquier segmento de código que se escriba tiene, al menos, una prueba asociada, de modo que “todo” el código escrito tiene una prueba.
- ✧ Pruebas de regresión
  - A medida que se desarrolla una aplicación se desarrolla, incrementalmente, un conjunto de pruebas de regresión.
- ✧ Depuración simplificada
  - Cuando una prueba falla, debería ser evidente dónde yace el problema. El código recién escrito necesita ser revisado y modificado.
- ✧ Documentación del sistema
  - Las pruebas mismas son una forma de documentación que describe lo que el código debería hacer.

Pruebas de regresión

- ✧ Las pruebas de regresión prueban el sistema para verificar que los cambios no hayan introducido bugs al código desarrollado previamente.
- ✧ En un proceso de pruebas manual, las pruebas de regresión son caras, pero con las pruebas automatizadas, es simple y directo. Todas las pruebas se vuelve a ejecutar cada vez que se realiza un cambio en la aplicación.
- ✧ Las pruebas se deben ejecutar de manera exitosa antes de que el cambio se acepte.

39

Pruebas de versión

40

Pruebas de versión

- ✧ Las pruebas de versión son el proceso de probar una versión particular de un sistema que se pretende usar fuera del equipo de desarrollo.
- ✧ El objetivo principal del proceso de pruebas de versión es convencer al proveedor del sistema de que este es suficientemente bueno para usarse.
  - Las pruebas de versión, por lo tanto, tienen que mostrar que el sistema brinda la funcionalidad, rendimiento y confiabilidad especificados, y que no falla durante el uso normal.
- ✧ Las prueba de versión, por lo regular, son un proceso de prueba de caja negra, donde las pruebas se derivan a partir de las especificaciones del sistema.

Pruebas de versión y pruebas de sistema

- ✧ Las pruebas de versión son una forma de prueba de sistema.
- ✧ Diferencias importantes:
  - Un equipo independiente que no intervino en el desarrollo del sistema debe ser el responsables de las pruebas de versión.
  - Las pruebas del sistema por parte del equipo de desarrollo deben enfocarse en el descubrimiento de bugs en el sistemas (pruebas de defecto). El objetivo de las pruebas de versión es comprobar que el sistema cumpla con los requisitos y sea suficientemente bueno para uso externo (pruebas de validación).

Pruebas basadas en requisitos

- Pruebas basadas en requisitos implica examinar cada requisito y desarrollar una prueba o pruebas para él.
- Requisitos del sistema Mentcare (MHC-PMS):
  - Si se sabe que un paciente es alérgico a algún fármaco en particular, entonces la prescripción de dicho medicamento debe dar como resultado un mensaje de advertencia que se emitirá al usuario del sistema.
  - Si quien prescribe ignora la advertencia de alergia, debe proporcionar una razón para ello.

43

Pruebas de requisitos

- Configurar un expediente de un paciente sin alergias conocidas. Prescribir medicamentos para alergias que se sabe existen. Comprobar que el sistema no emite un mensaje de alergia.
- Crear un expediente de un paciente con una alergia conocida, prescribir el medicamento al que es alérgico y comprobar que el sistema emite la advertencia.
- Crear un expediente de un paciente donde se reporten alergias a dos o más medicamentos. Prescribir dichos medicamentos por separado y comprobar que se emite la advertencia correcta para cada medicamento.
- Prescribir dos medicamentos a los que es alérgico el paciente. Comprobar que se emiten correctamente dos advertencias.
- Prescribir un medicamento que emite una advertencia y pasar por alto dicha advertencia. Comprobar que el sistema solicita al usuario proporcionar información que explique por qué pasó por alto la advertencia.

44

Pruebas de rendimiento

- ✧ Parte de las pruebas de versión puede incluir probar las propiedades emergentes de un sistema, como el rendimiento y la confiabilidad.
- ✧ Las pruebas deben reflejar el perfil de uso del sistema.
- ✧ Las pruebas de rendimiento por lo general implican la planificación de una serie de pruebas en las que se incrementa de forma constante la carga hasta que el rendimiento del sistema se vuelve inaceptable.
- ✧ Las pruebas de estrés es una forma de pruebas de rendimiento, donde el sistema se sobrecarga deliberadamente para probar su comportamiento de falla.

45

Pruebas de usuario

46

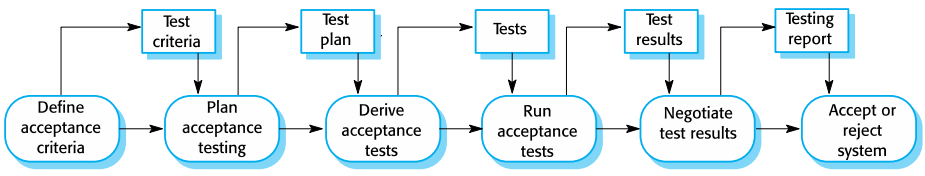
Pruebas de usuario

- Las pruebas de usuario o del cliente son una etapa del proceso de pruebas en la que usuarios o clientes proporcionan entradas y asesoría sobre las pruebas de sistema.
- Las pruebas de usuario son esenciales, aún cuando se hayan realizado pruebas de sistema y versión completas.
  - La razón de esto es que la influenza del entorno de trabajo del usuario tiene gran influencia en la fiabilidad, rendimiento, uso y robustez de un sistema. Lo anterior no se puede replicar en un entorno de prueba del desarrollador.

Tipos de prueba de usuario

- Pruebas alfa
  - Los usuarios del software trabajan con el equipo de desarrollo para probar el software en el sitio del desarrollador.
- Pruebas beta
  - Una versión del software se pone a disposición de los usuarios para permitirles experimentar y suscitar problemas que descubren junto a los desarrolladores del sistema.
- Pruebas de aceptación
  - Los clientes prueban un sistema para decidir si está o no está listo para ser aceptado por los desarrolladores del sistema y desplegado en el entorno del cliente. Principalmente para los sistemas personalizados.

Proceso de pruebas de aceptación



49

Etapas del proceso de pruebas de aceptación

- ✧ Definir los criterios de aceptación
- ✧ Plan de pruebas de aceptación
- ✧ Derivar pruebas de aceptación
- ✧ Ejecutar pruebas de aceptación
- ✧ Negociar los resultados de las pruebas
- ✧ Rechazar/aceptar el sistema

50

Métodos ágiles y pruebas de aceptación

- ✧ En los métodos ágiles, el usuario/cliente forma parte del equipo de desarrollo y es responsable de tomar decisiones acerca de si el sistema es aceptable o no.
- ✧ El usuario/cliente define las pruebas, las que se integran a otras que se ejecutan automáticamente cuando se hacen cambios.
- ✧ No hay un proceso de pruebas de aceptación separado.
- ✧ El problema principal aquí es si o no el usuario incorporado es "típico" y puede representar los intereses de todas las partes interesadas del sistema.

51

52