Verificación y Validación de Software

Proceso de Desarrollo

Relevamiento y especificación de requisitos → Diseño del software → Traducción del diseño a lenguajes de programación

- ¿Para qué otra etapa más? ¡Ya tenemos el software!

Procesos Humanos

- Aún adoptando los mejores métodos y siendo extremadamente cuidadosos, todos los proceso humanos pueden introducir errores.
  - Puentes, galletitas, suelas de zapatos, clavos, etc.
- Los productos deben ser verificados en todas sus etapas:
  - el plan inicial (¿es algo razonable?)
  - el proceso de desarrollo (¿se lleva a cabo el plan?)
  - el producto final (¿cumple con las necesidades?)

Verificación de Software

- La verificación del software es el proceso a través del cual se corrobora que el software satisface sus objetivos.
- Existen autores que marcan la diferencia entre verificación y validación de software:
  - verificación: el software es correcto, está implementado de acuerdo con sus especificaciones;
  - validación: el software es útil para satisfacer las necesidades del usuario.

Objetivos de la Verificación

- Prueba de pequeños programas:
  - se usa el programa como en un caso típico y se constata si se comporta de la manera esperada.
- Esta práctica solamente puede hacerse para programas muy chicos y no críticos:
  - no da ninguna garantía sobre el comportamiento del sistema en casos distintos de los probados.

Importancia de la Calidad

Programa doméstico con fines recreativos → Producto de software crítico o comercial

CALIDAD:

Técnicas de control de calidad.

Elección de los casos de prueba.

Aseguramiento de la calidad.

Todo debe ser verificado

- Todas las cualidades relevantes del sistema deben ser verificadas:
  - corrección: la implementación se comporta de acuerdo con las especificaciones;
  - portabilidad,
  - modificabilidad,
  - performance.
- La verificación debe realizarse por distintas personas durante distintas etapas del desarrollo del software;
  - el implementador o un verificador independiente.

7

Resultado de la Verificación

- Después de hacer una serie de pruebas de verificación no se llega a una conclusión tajante:
  - “el producto está bueno y podemos comercializarlo”;
  - “el producto está malo y debemos desecharlo”.
- Más bien se llega a tener una idea conceptual de la calidad del software.
- Supuestos sobre la verificación de software:
  - la presencia de defectos no puede evitarse en grandes sistemas de software,
  - a veces, algunos defectos pueden tolerarse,
  - la corrección es relativa y difícil de evaluar.

8

Verificación Objetiva o Subjetiva

- Algunas pruebas del sistema son objetivas:
  - dar una entrada y verificar la corrección de la salida,
  - medir el tiempo de respuesta de una transacción.
- Otras cualidades no pueden medirse tan objetivamente:
  - portabilidad: sólo puede decirse algo acerca de cuán difícil sería portar el software a un ambiente particular;
  - modificabilidad: el sistema será más o menos modificable de acuerdo al cambio a aplicar.
- Es deseable sin embargo poder decir algo más general sobre las cualidades del software: estimaciones subjetivas.
  - portabilidad: el software está desarrollado en java, o sea que podrá correr en cualquier máquina.

9

Ejemplo: Modificabilidad

- Fabricante A:
  - fabrica editores de pantalla;
  - el editor de A corre actualmente en el monitor de B.
  - A supone que sus editores son adaptables y portables a muchos monitores.
- Fabricante B:
  - fabrica monitores y está buscando un editor para sus monitores;
  - desea que el editor sea adaptable a futuras versiones de monitores.
- ¿Cómo puede saber **B** si el editor de **A** será adaptable?
- Experimento:
  - **B** proporciona un nuevo monitor y da a **A** una semana para adaptar su editor.

10

Más Cualidades Subjetivas

- Usabilidad.
- Comprensibilidad.
- Parece ser ineludible que finalmente casi todos los juicios que pueden hacerse acerca del software son subjetivos y opinables.
- Algunas soluciones:
  - comités de evaluación,
  - delegar la decisión en alguien en quien todos confían.

11

Cualidades Implícitas

- Existen muchas cualidades que por obvias son dejadas fuera de la especificación de los requisitos. Son un juicio de sentido común.
  - una consulta en un cajero automático no puede tardar más de 1 minuto en el 95% de los casos.
- La mantenibilidad de los sistemas es una cualidad muy deseable y raramente especificada como parte de los requisitos;
  - es muy difícilmente cuantificable;
  - solamente anticipando los posibles cambios, uno puede evaluar el costo de aplicarlos.

12

Enfoques de Verificación

- Existen dos enfoques fundamentales:
  - Test: experimentar con el comportamiento del sistema;
  - Análisis: comprobar propiedades del sistema.
- Otra clasificación de la verificación:
  - Dinámica: requiere ejecutar el software;
  - Estática: no requiere ejecución.
- Afortunadamente todos los enfoques son complementarios.

13

Testing o Pruebas del Sistema

- La forma más habitual de verificar un software es:
  - ejecutarlo en una serie de situaciones representativas,
  - verificar que su comportamiento es el esperado.
- Hay que elegir un conjunto de datos de prueba apropiado porque no es posible probar absolutamente todas las posibilidades:
  - deben ser los menos posibles para tener menos trabajo,
  - pero deben cubrir la mayor cantidad de casos posibles.
- La tarea de elegir el conjunto de datos de prueba es algo complejo:
  - un puente que mostró soportar 100 toneladas podemos suponer que también soportará 10 toneladas. ¿Analogía?

14

Ejemplo: Búsqueda Binaria

```
procedure busqueda-binaria (clave: in elem; tabla: in elemTabla;
        encontrado: out Boolean) is
begin
        inicio := tabla.primero; fin := tabla.ultimo;
        while inicio < fin loop
                if (inicio + fin) mod 2 <> 0 then
                        medio := (inicio + fin - 1) / 2;
                else
                        medio := (inicio + fin) / 2;
                endif;
                if clave <= tabla (medio) then
                        fin := medio;
                else
                        inicio := medio + 1;
                endif;
        endloop;
        encontrado := clave = tabla (medio);
end busqueda-binaria
```

15

Continuidad

- En general los productos de ingeniería cuentan con la propiedad de continuidad:
  - pequeños cambios en el ambiente provocan sólo pequeños cambios en el comportamiento del producto.
- Los productos de software no cuentan con esta propiedad de continuidad.

16

Testing: Prueba de Software

- La prueba del software es una actividad crítica de la ingeniería de software.
- Debe aplicarse en forma sistemática:
  - explicitar claramente los resultados esperados,
  - describir la forma de obtener los resultados,
  - documentar los resultados obtenidos.
- En la práctica, la prueba del software se hace:
  - en forma desestructurada,
  - sin aplicar criterios claros.

17

Objetivos de la Prueba de Software

- Las pruebas del software pueden usarse para demostrar la existencia de errores, nunca su ausencia.
  [Dijkstra, 1972]

Pruebas con errores → Sistema incorrecto

Pruebas sin errores → ¿?

18

Utilidad de las Pruebas

- Si las pruebas no dan certeza sobre la corrección del software, ¿tienen alguna utilidad?
- Si bien no proporcionan certeza, las pruebas pueden aumentar nuestra confianza en que el sistema se comportará como es esperado.
- Lo esencial de las pruebas es:
  - elegir un conjunto de datos de prueba apropiados,
  - aplicar las pruebas en forma sistemática.

19

Valores Aleatorios de Prueba

```
read (x); read (y);
if x = y then
        z := 2;
else
        z := 0;
endif
write (z);
```

- Los datos aleatorios de prueba suelen no ser los más apropiados.
- Sólo el desarrollador del programa sabe cuáles son los datos sensibles para cada aplicación y cuáles los resultados esperados.
- En el ejemplo no sirven una cantidad de casos de diferentes valores de (x,y), sino solamente dos:
  - un caso con x igual a y,
  - un caso con x distinto de y.

20

Localizable, Repetible y Precisas

- Las pruebas no sólo deben detectar la presencia de errores, sino que deben indicar cuál es el error y dónde se encuentra.
- Las pruebas deben organizarse para que provean información acerca de la localización de los errores.
- Las pruebas también deben ser repetibles: aplicadas dos veces, debieran producir los mismos resultados.
- La influencia del ambiente de ejecución atenta contra la repetibilidad de las pruebas.
- Los resultados esperados deben definirse dependiendo de los datos de entrada y de otros eventos del ambiente.

21

Variables no Inicializadas

```
if x = 0 then
        write (“anormal”);
else
        write (“normal”);
endif
```

- Si la variable x no está inicializada y se tiene este trozo de programa, ¿cuál será el resultado?
  - a veces “anormal”,
  - a veces “normal”.
- Esta prueba no es repetible.
- Algunos lenguajes no chequean que las variables estén inicializadas, por razones de eficiencia.

22

Especificaciones Precisas

- Las especificaciones del software deben ser lo suficientemente precisas para guiar las pruebas:
  - “como consecuencia del estímulo X, el sistema debe producir la salida Y en menos de Δt”.
- ¿Qué prueba debe aplicarse?
  - Dar el estímulo X al sistema,
  - medir el tiempo hasta que produzca la salida,
  - comprobar que la salida sea Y y el tiempo sea menor o igual a Δt.
- Pero, ¿qué sucede si ocurren otros eventos después de dar el estímulo X? ¿Cómo se prueba un sistema concurrente?

23

Especificaciones Formales

- Si las especificaciones establecen formalmente los resultados de una operación, será más fácil probarla.
  - $\forall\ i \mid 1 \leq i \leq N \Rightarrow a(i) \leq a(i+1)$
  - si el programa produce un array a tal que satisface esta fórmula, el software será correcto con respecto a esta especificación.
- Las especificaciones formales pueden ser útiles también para generar datos de prueba automáticamente.
- Estas pruebas estructuradas aumentan la confiabilidad del software.

24

Bases Teóricas de las Pruebas

- Definición formal:
  - sea P un programa;
  - sea D su dominio de entrada y R su recorrido de salidas (D y R pueden ser infinitos);
  - dado un elemento de D a P, este producirá un elemento de R (función parcial): P: D → R.
  - sean OR (output requirements) los requerimientos sobre los datos de salida especificados formal o informalmente;
- dado d ∈ D ⇒ P(d) es correcto ⇔ P(d) satisface OR
- P es correcto ⇔ ∀ d ∈ D ⇒ P(d) es correcto

25

Defectos, Fallas y Faltas

- La presencia de un error o defecto se demuestra encontrando un d tal que P(d) es incorrecto.
- Una falla es el síntoma de que existe un error; se da durante la ejecución.
- Pero un error puede existir en el código sin causar ninguna falla.
- El objetivo de las pruebas es tratar de que todos los defectos existentes provoquen fallas.
- Una falta es un estado intermedio incorrecto en que entra un programa durante su ejecución.

26

Más sobre Faltas y Fallas

- Una falla solamente ocurre si existe una falta.
- ¿Qué ocurre si un defecto no provoca una falla?
- ¿Qué ocurre si una falta no produce una falla?

27

Casos y Conjuntos de Prueba

- Un caso de prueba d es un elemento del dominio D.
- Un conjunto de prueba T, es un conjunto finito de casos de prueba. T ⊂ D.
- P es correcto para T si es correcto para todos los d ∈ T. Se dice entonces que T es exitoso para P.
- Un conjunto de prueba T es ideal si:
  - P es incorrecto ⇒ ∃ d ∈ T | P(d) es incorrecto;
  - un conjunto de prueba ideal siempre muestra la existencia de defectos si estos existen.
- T es un conjunto ideal y T es exitoso para P ⇒ P es correcto.

28

Criterio de Selección

- C es el criterio de selección de los elementos de D que forman parte del conjunto de prueba T.
- C es una condición que debe cumplir el conjunto de prueba; representa todos los subconjuntos de D que satisfacen dicha condición.
- Ejemplo: si D es el conjunto de los enteros, el conjunto de prueba deberá tener al menos un dato negativo, uno cero y uno positivo.
  - C = {<x1,x2,...,xn> | n ≥ 3 ∧ ∃ i,j,k | (xi < 0, xj = 0, xk > 0)}
- T satisface C si T ∈ C.

29

Propiedades de C

- C es consistente ⇔
  - T1 ∈ C ∧ T2 ∈ C ⇒ T1 es exitoso ⇔ T2 es exitoso
- C es completo ⇔
  - P es incorrecto ⇒ ∃ T ∈ C | T no es exitoso
- O sea que si se dispone de un criterio C consistente y completo, cualquier conjunto de prueba será útil para determinar la corrección de P.
- Lamentablemente no existe un algoritmo o método para determinar si un criterio de selección de un conjunto de prueba es consistente o completo.

30