Métodos Formales en Ingeniería del Software

Tarea 1

Juan José Méndez Torrero
i42metoj@uco.es

Universidad de Córdoba

25 de febrero de 2019

1. Pregunta 1

Explica por qué las inspecciones de programas son una técnica efectiva para descubrir errores en un programa. ¿Qué tipos de errores probablemente no sean descubiertos a través de las inspecciones?

Las inspecciones de programas son una técnica efectiva para descubrir errores ya que nos permite encontrar fallos con una sola pasada sin tener que preocuparnos por la interferencia de otros fallos.

Esta rapidez a la hora de encontrar fallos es debida a que para una inspección se reúnen muchos profesionales de varios ámbitos para buscar estos fallos todos a la vez, siendo este método mucho más efectivo que si tuviera que buscar los fallos el creador del programa.

Además, estos profesionales prueban detalladamente todas las funcionalidades del programa, ya que así serán capaces de encontrar más fallos como por ejemplo los fallos de entendimiento.

Por último, los errores que probablemente no sean descubiertos por estas inspecciones son los errores de dominio o errores en las especificaciones. Estos errores no serán encontrados a no ser que haya un profesional de esos ámbitos en la inspección.

2. Pregunta 2

Sugiera por qué una organización podría probablemente encontrar difícil introducir las inspecciones de programas como una técnica de V & V.

Una organización, probablemente no introducirá las inspecciones de programas como una técnica de V&V debido a que estas inspecciones, como hemos explicado en la pregunta anterior, se encargan de encontrar errores dentro de un programa, con lo que el autor de dicho programa puede no querer que otros profesionales de su sector corrijan su código, ya que esto les expone a la competencia. Como nota, los mejores programadores no quieren que sus programas sean inspeccionados.

Otra dificultad que puede encontrar una organización a la hora de introducir las inspecciones de programas es que, como las inspecciones son una agrupación de profesionales en busca de errores, podría llegar a existir una competición entre ellos para ver quién encuentra más fallos. Aunque en muchas ocasiones esto puede ser positivo, en este caso es improductivo, ya que si un inspector no es capaz de encontrar ningún fallo, los demás, en vez de ayudarle, preferirán no hacerlo para así poder ellos ser los mejores.

3. Pregunta 3

Utilizando tus conocimientos de Java, C++, C o cualquier otro lenguaje, deriva una lista de comprobación de errores comunes(no errores sintácticos) que podrían no ser detectados por un compilador, pero que podrían ser detectados en una inspección de programas.

En C o C++, un ejemplo claro de un error que el compilador no detectaría sería que una función devuelva un valor nulo, ya que podría ser que esa variable no haya sido inicializada correctamente. Otro error no detectable por el compilador podría ser que, sobre un vector, se intente acceder a una posición que no existe, este fallo no sería detectado por el compilador pero no funcionaría el programa.

4. Pregunta 4

Genera una lista de condiciones que podrían ser detectadas por un analizador estático en Java, C++ u otro lenguaje de programación que utilices. Comenta esta lista comparándola con la lista de la siguiente figura.

Clase de defecto	Comprobación de inspección
Defectos de datos	¿Se inicializan todas las variables antes de que se utilicen sus valores? ¿Tienen nombre todas las constantes? ¿El límite superior de los vectores es igual al tamaño del vector o al tamaño 21? Si se utilizan cadenas de caracteres, ¿tienen un delimitador explícitamente asignado? ¿Existe alguna posibilidad de que el búfer se desborde?
Defectos de control	Para cada sentencia condicional, ¿es correcta la condición? ¿Se garantiza que termina cada bucle? ¿Están puestas correctamente entre llaves las sentencias compuestas? En las sentencias case, ¿se tienen en cuenta todos los posibles casos? Si se requiere una sentencia break después de cada caso en las sentencias case, ¿se ha incluido?
Defectos de entrada/salida	¿Se utilizan todas las variables de entrada? ¿Se les asigna un valor a todas las variables de salida? ¿Pueden provocar corrupciones de datos las entradas no esperadas?
Defectos de interfaz	¿Las llamadas a funciones y a métodos tienen el número correcto de parámetros? ¿Concuerdan los tipos de parámetros reales y formales? ¿Están en el orden correcto los parámetros? Si los componentes acceden a memoria compartida, ¿tienen el mismo modelo de estructura de la memoria compartida?
Defectos de gestión de almacenamiento	Si una estructura enlazada se modifica, ¿se reasignan correctamente todos los enlaces? Si se utiliza almacenamiento dinámico, ¿se asigna correctamente el espacio de memoria? ¿Se desasigna explícitamente el espacio de memoria cuando ya no se necesita?
Defectos de manejo de excepciones	¿Se tienen en cuenta todas las condiciones de error posibles?

Figura 1: Insertar una tarea y su duración

Para este ejercicio crearemos la siguiente lista de posibles errores detectables por un analizador estático:

- **Defectos de datos:**
  - Que toda variable antes de ser usada ha de ser inicializada correctamente.

    - Toda variable que vaya a hacer uso de la memoria, ha de ser reservada con anterioridad.
    - Nunca acceder a una posición de una variable que sea mayor que el tamaño total.
- **Defectos de control:**
    - La condición ha de cumplirse, ya sea una variable booleana, numérica o tipo string.
    - Toda función que no sea nula, debe devolver un valor concreto.
- **Defectos de entrada/salida:**
    - El programa siempre ha de mostrar una salida, y en el caso de no poder hacerlo, que al pasar cierto tiempo se cierre el programa.
    - No deben existir variables que sean inicializadas pero no usadas.
- **Defectos de la interfaz:** La interfaz ha de ser legible y usable para el usuario final.

Como vemos, la lista anterior de condiciones necesarias para una inspección nos da una guía de lo que cualquier lenguaje de programación que admita un analizador estático ha de cumplir.

5. Pregunta 5

Explica por qué puede ser rentable utilizar métodos formales en el desarrollo de sistemas software de seguridad críticos. ¿Por qué piensas que algunos desarrolladores de este tipo de sistemas están en contra de los métodos formales?

Normalmente, el coste de arreglar los errores en los sistemas software de seguridad críticos suele ser muy caro, con lo que el uso de los métodos formales puede hacer bajar ese coste a pesar de que se tendrá que añadir el coste de su utilización. Por esto, el uso de métodos formales puede llegar a ser rentable si el coste de arreglar los errores en el software es muy caro.

El porqué de que algunos desarrolladores no utilicen los métodos formales puede ser debido al no conocimiento de estas técnicas. Es decir, estos métodos podrían dar una imagen falsa de corrección a los desarrolladores si estos no entienden cómo funcionan estos métodos.

6. Pregunta 6

Un gestor de una empresa decide utilizar los informes de las inspecciones de programas como entrada para el proceso de valoración del personal. Estos informes muestra quién hace y quién descubre los errores del programa. ¿Es éste un comportamiento de gestión ético?¿Podría ser

ético si el personal fuese informado con antelación de que esto podría ocurrir?¿Qué diferencia se podría generar en el proceso de inspección?

No sería un comportamiento ético, ya que esa información podría hacer que la competitividad de los trabajadores aumentara, siendo este hecho algo que no es bueno a la hora de trabajar en equipo.

En el caso de que los empleados hubieran sido avisados de que esa situación pudiera ocurrir, podría hacer que los empleados pudieran rendir más, pero al fin y al cabo, entrarían en una situación de competitividad no deseable, ya que comenzarían a no ayudarse entre ellos y no compartir información que pudiera ser de ayuda para otro compañero.