Conceptos básicos

Alumno:

Paisano Flores Alejandro

Materia:

Computación tolerante a fallas

Sección:

D05

Profesor:

Michel Emanuel Lopez Franco

Contenido

Introducción

En esta investigación observaremos múltiples herramientas para poder prevenir la creación de errores dentro del desarrollo de software. De la misma forma, también llegaremos a observar en que momento se deben de usar estos métodos y a que área de la producción de software pertenecen.

También se debe de mencionar que, aunque estos métodos previenen errores, estos no son solo errores en el código, si no también fallos que existan en manuales de usuario, interfaz de usuario y de diseño de lógica del programa

Desarrollo

Primeramente, debemos de definir los tipos de errores que puede haber durante el desarrollo de software, como deben de ser tratados, que los origina y el si pueden ser prevenidos o resulta un esfuerzo inútil.

Bugs de implementación: Estos son bugs que aparecen principalmente durante la fase de programación, usualmente se debe a que no se ha definido de forma adecuada la lógica de trabajo, no se han definido bien los requisitos o a errores de escritura de parte del programador. Usualmente son totalmente prevenibles

Bugs de especificación: Se deben primordialmente a que no se definieron bien las respuestas que debe dar el software a determinadas situaciones o a que los requisitos están incompletos. Estos bugs son algo difíciles de detectar, debido a que no dejan errores tan claros y se captaran cuando el software se lance.

Bugs de especificación ausente: Surgen de falta de consideración de ciertos factores durante la producción de software, usualmente solo se resuelven cuando el problema es traído a la luz y tienden a afectar gravemente a la producción. Estos se notan en su mayoría en la etapa de lanzamiento y mantenimiento.

Estos bugs aparecen durante situaciones particulares como:

Inyección/ regresión: Cuando se introduce nuevo código que afecta al funcionamiento de otras partes.

Testeo inadecuado: Ya sea por falta de personal, pruebas o tiempo. El software no recibirá las pruebas adecuadas y por ello tendrá mas problemas que se dejaran pasar.

Falta de comunicación.

Planeación inapropiada: El problema más visto en equipos grandes junto a la falta de comunicación, los errores surgidos en esta fase pesaran de forma constante al programa.

Prevención

Desarrollo guiado por pruebas: la solución al testeo inadecuado. Diseñar una serie de pruebas las cuales están hechas para se cumplidas para el programa final, lo que permite diseñar un programa con el mínimo de fallos llegando al usuario y que permite enfocar el tiempo de diseño del software

Integración continua de testeo continuo: Solo se usa en casos en los que es posible automatizar las pruebas de software, permitiéndonos corregir errores surgidos por inyección/regresión.

Comportamiento enfocado al desarrollo: Se refiere al uso de lenguaje y canales de documentación estándar para poder facilitar la comunicación entre diferentes departamentos de desarrollo.

Especificación en managment y revisiones: Parte del proceso de desarrollo de software, sobre todo en el área administrativa, es la evolución y corrección constante de los objetivos ofrecidos. De esta forma podemos evitar errores originados por especificaciones que hayan quedado ambiguas u obsoletas.

Comunicación clara: Por último, que todos los equipos tengan la confianza, y los canales/ representantes para comunicarse adecuadamente es lo más importante, así se evitaran errores en los que partes del código queden obsoletas o se introduzca nuevos datos que afecten a otras partes del código.

Razones para prevenir o no estos errores

Valor del esfuerzo: En ocasiones los errores son demasiado complicados con relación a cuanto afectan al programa como para tratarlos.

Tolerancia del usuario a los bugs: como en todo programa, es posible que los usuarios se adapten a los bugs o no noten sus efectos negativos.

Estructura del equipo y dinámicas: Puede darse la situación donde x equipo no puedan realizar una tarea por cómo están conformados.

Parches de software: Los errores menores o raros siempre pueden ser atendidos después del lanzamiento del software

Orthogonal defect classification (ODC)

El ODC es un método para la identificación y corrección de errores que fue creado en 1992. Según este método es posible usar múltiples categorías de fases y que es posible medir con estas fases en que momento se producen mas errores y cuando se deben de atender.

El autor del método propone ocho tipos de errores:

Función.

Interfaz.

Chequeo.

Asignación.

serialización

build.

Documentación.

Algoritmo.

Con todo esto, el método ODC ha evolucionado hasta darle una clasificación a los atributos de los defectos, esto con el objetivo de trabajarlos y registrarlos de una mejor manera. Estos atributos son los siguientes:

Actividad: La acción especifica que origina el error

Gatillo: El ambiente o condición que debe de existir para que el error se manifieste.

Impacto: El efecto que tendría el erro en caso de continuar en el programa o presentarse ante un usuario.

También existen varios atributos que definen el arreglo del error:

Objetivo: Que se está arreglando.

Tipo de error: El tipo de error que se está corrigiendo.

Origen: La fuente en codigo o diseño que causo el error.

Edad: Historia del error, desde su descubrimiento, soluciones aplicadas, parches, hasta su solución definitiva.

Conclusión

Prevenir bugs y otro tipo de fallas dentro de un software resulta tan, si no es que en ocasiones incluso mas importante que la misma realización del producto. Cuantas veces no se ha visto grandes programas con ideas innovadoras o con nuevas formas mas eficientes para realizar tareas, solo para caer ante una gran cantidad de errores.

Y no solo es parchear de forma infinita todo lo que ocurra en el programa, debido a que probablemente nos terminemos viendo muy limitados en cuanto a lo que podemos hacer por tiempo y personal. Por esto nace la necesidad de prevenir los errores antes de que ocurran.

Siendo una persona que ha hecho muy pocos proyectos que se hayan visto desde un punto de vista mas profesional que solo una tarea, he de admitir que yo también soy culpable de no prevenir errores por falta de comunicación.

No solo eso, si no que en más de una ocasión he tenido problemas debido a la falta de cuidado en cuanto a errores que se detectan solo cuando el programa se ejecuta.

Con eso dicho, puedo decir que me veo bastante interesado en estas herramientas, sobre todo las relacionadas al testeo, pues tiendo a encontrar que muchos de mie errores surgen en su mayor parte por la falta de consideración de ciertos errores muy específicos.

También debo de resaltar que, aunque exista mucha documentación relacionada al OCD, debo de decir que encontré muy poca información relacionada a los tipos de errores, siendo que la mayoría de los textos actuaban asumiendo que el lector conoce los tipos de errores.

Bibliografía

Richa Agarwal. (25 de mayo de 2022). testsigma .Techniques To Prevent Software Bugs. https://testsigma.com/blog/techniques-to-prevent-software-bugs/

Izzet Yildirim. (24 de abril de 2019). DEV. Error prevention: One of the Usability Heuristics. https://dev.to/izzet/error-prevention-one-of-the-usability-heuristics-2l4b

Frederid Palacios. (7 de febrero de 2023). Intertec international. The Importance of Error Prevention in Software Development. https://blog.intertecintl.com/the-importance-of-error-prevention-in-software-development

IBM. (03/09/2023). Orthogonal Defect Classification (Archival). https://researcher.watson.ibm.com/researcher/view_group_subpage.php?id=481

N.A. (03/09/2023). Flylib.com Orthogonal Defect Classification https://flylib.com/books/en/1.428.1/orthogonal_defect_classification.html

Abdul Kadir. (marzo de 2013). Star Journal. Software Excellence Augmentation through Defect Analysis and Avoidance. file:///D:/Osvaldo/Documents/Dibujos/Semestre%209/Tolerante%20a%20fallas/articulos/ajol-file-journals_508_articles_98848_submission_proof_98848-6001-259757-1-10-20131217.pdf