ANÁLISIS DE LA TÉCNICA INSPECCIÓN DE CÓDIGO COMO TÉCNICA PARA MEJORAR LA CALIDAD DEL SOFTWARE

SANDRA JOHANNA RÍOS HORTÚA

TRABAJO DE GRADO

NORMAN DANILO CASTRO TELLEZ
COORDINADOR ÁREA DE INGENIERÍA DE SOFTWARE

INSTITUCIÓN UNIVERSITARIA POLITÉCNICO GRANCOLOMBIANO
FACULTAD DE INGENIERÍA Y CIENCIAS BÁSICAS
BOGOTA
2014

1. DEDICATORIA

Dedico este trabajo a DIOS que me ha dado la vida, la sabiduría y todo su favor para llegar hasta donde estoy.

A mi madre que es mi ejemplo de mujer guerrera, mi pilar y que me ha dado su apoyo incondicional siempre.

2. AGRADECIMIENTOS

Quiero agradecer al Politécnico Grancolombiano por brindarme las herramientas y el conocimiento para llevar a cabo el experimento para la investigación que hoy se traduce en este documento.

Al profesor Danilo Castro, quien me asesoró y apoyo durante el proceso de investigación, a todos y cada uno de quienes compartieron conmigo estos años de estudio y que de una u otra forma aportaron conocimientos y experiencias a mi vida profesional y personal.

A mi amiga incondicional Alexandra, Gracias por estar.

3. CONTENIDO

4. INTRODUCCION

A medida que pasa el tiempo y el mundo se moderniza con el uso de nuevas tecnologías se ve la necesidad de implementar aplicaciones de software que apoyen el funcionamiento de las diferentes organizaciones, para estar a la vanguardia y tener altos niveles de competitividad en los diferentes mercados.

Por esta razón la industria del software debe trabajar arduamente para proveer desarrollos de software impecables, asegurando la calidad de su producto, el cual será usado por un usuario que debe estar satisfecho porque se cumplieron sus requerimientos y expectativas con respecto a una aplicación implementada para una labor especifica.

Para que dicha calidad se pueda garantizar es necesario hacer uso de las buenas prácticas, procesos o metodologías que actualmente existen en la industria del software para asegurar la calidad de los artefactos de ingeniería de software que son entregados como producto final.

Para llevar a cabo esta investigación se trabajó con estudiantes de último semestre de ingeniería de sistemas, los cuales tenían conocimientos en el desarrollo de software y debían hacer una serie de retos (Implementaciones cortas) en sesiones y espacios estipulados en cuales se podía controlar los tiempos de ejecución de tareas como es el caso de las inspecciones de código, actividad para la cual se les dio instrucción y capacitación dado que no conocían la técnica. Esto permitió evaluar los tiempos de construcción, defectos encontrados en cada inspección, el tiempo de mantenimiento y la funcionalidad de cada reto impuesto.

La metodología del presente estudio consta de la revisión bibliográfica pertinente, estructuración y desarrollo del anteproyecto que contempla la consulta, exploración, recopilación, proceso, resultados y análisis a partir del estudio de las inspecciones de código como técnica para mejorar la calidad del software.

Objetivo de la investigación: se establece la finalidad de la investigación.

Problema: muestra cual es la situación que conlleva a la investigación y cuál es la hipótesis de la misma.

Justificación: Expone el propósito y la importancia de esta investigación

Marco Teórico: se presentan los planteamientos de diferentes autores con respecto a las inspecciones de código y las diferentes formas de realizarlas.

Metodología: Detalla la forma en que se llevó cabo la investigación.

Resultados: muestra el análisis de los resultados obtenidos durante la investigación.

Conclusiones: Muestra que se concluyó luego de validar los resultados obtenidos.

5. OBJETIVOS

5.1. Objetivo General

Definir cómo las inspecciones de código minimizan la cantidad de defectos en un software para mejorar la calidad del producto entregado al cliente.

5.2. Objetivos específicos

- Evaluar si la metodología seleccionada permite el uso de menos recursos para el desarrollo del software.
- Validar si es posible hacer un diagnóstico más acertado de los defectos encontrados en el software.

6. PROBLEMA

6.1. Planteamiento del problema

Actualmente se ha evidenciado que en la industria del software se generan inconvenientes a la hora de garantizar la calidad del software de los productos entregados. Una de las principales causas es que en las metodologías utilizadas no se tienen en cuenta las prácticas para minimizar los defectos de los artefactos, que permitan reducir costos en recurso humano y en tiempos de ejecución. Por esta razón es fundamental indagar sobre técnicas que permitan garantizar la calidad del software y cumplir con las especificaciones entregadas por el cliente.

Teniendo en cuenta el requisito de asegurar la calidad de los productos en la industria de software surge la necesidad de estudiar uno de los procesos ya existentes, como lo son la inspecciones de código, con el fin de analizar sí la ejecución de esta actividad durante el desarrollo puede cumplir con las expectativas de calidad del cliente y minimizar los tiempos para las partes interesadas.

6.2. Hipótesis

Comprobar que las inspecciones de código con listas de chequeo permiten encontrar mayor cantidad de defectos a diferencia de las inspecciones que se realizan sin lista de chequeo.

1. MARCO TEÓRICO

1.1. Generalidades Sobre Calidad del Software

La calidad del software se define como el conjunto de características y cualidades del software que determinan su usabilidad, eficiencia, flexibilidad, corrección, fiabilidad, portabilidad, reusabilidad, interoperabilidad, facilidad de prueba, mantenibilidad, seguridad e integridad. (McCall James Paul K. Richards, 1977). Debido a esto, muchas organizaciones han puesto a la calidad en el desarrollo y mantenimiento del software como uno de sus principales objetivos. (Deming, 1982)

La calidad está directamente asociada con la usencia de defectos en el código, permitiendo que las pruebas sean agiles y con tiempos de ejecución cortos. También se muestra la calidad desde varios puntos de vista: la visión trascendental que puede ser reconocida pero no definida, la visión del usuario como la adecuación al propósito del usuario, la visión del productor como conformidad con la especificación, la visión del producto, basada en las características observables del producto, y la visión basada en el valor que el cliente está dispuesto a pagar. (Garvin, 1984).

(López Lengua, Hacia la identificación de un método universal de calidad de software., 2013) en su tesis de grado, propone varias técnicas y prácticas para el aseguramiento de la calidad del software; entre ellas la programación por parejas, el modelo de pruebas en V, el desarrollo dirigido por pruebas, y las inspecciones de código y que adicionalmente algunas organizaciones han mejorado la calidad del software usando esta última técnica (M, Advances in Software Inspections) , (M, Design and Code Inspections to Reduce Errors in Program Development)

Además de esto, modelos como el CMMI (Capability Maturity Model Integration) (CMMI Product Team;, 2006) TSP (Team Software Process) (W, Team Software Process, 2000), PSP (Personal Software Process) (W, The Personal Software Process, (PSP), 2000), todos ellos desarrollados por el SEI (Software Engineering Institute), han adoptado esta técnica para mejorar la calidad de su código.

1.2. Inspecciones de Código

Esta investigación está enfocada en la técnica de inspecciones de código las cuales fueron definidas por (Fagan, 1976) con el fin de hacer revisiones de código únicamente. Sin embargo, hoy en día se ejecutan durante todas las etapas de la construcción del código para detectar defectos de manera temprana en cualquier artefacto para realizar las correcciones necesarias, y de este modo reducir tiempos, costos y esfuerzos en la implementación del desarrollo de software (Zamuriano Sotés).

Consiste en hacer una revisión por pares, es decir; se definen roles:

- 1 Autor: Desarrollador del producto que será inspeccionado
- 2 revisores: Colegas de otro proyecto
- 1Moderador: Miembro del grupo de aseguramiento de la calidad, quien se encargará de verificar que los artefactos se inspeccionen correctamente

Todos deben tener las habilidades y conocimientos para realizar la práctica de esta técnica. (Casallas G.)

Es importante tener en cuenta que las inspecciones de código son de mejora incremental, esto quiere decir que los defectos deben reducirse en el tiempo.

A continuación se relacionan los objetivos de las inspecciones (Zamuriano Sotés) (Casallas G.):

- Encontrar tempranamente los defectos.
- Prevenir el mal funcionamiento de los procesos o planes establecidos.
- Proporcionar mejoras en la fiabilidad, disponibilidad, y la facilidad de
- mantenimiento del software.
- Descubrir continuamente la información técnica, asociada con las funciones,
- formularios y actividades internas que aseguran el producto.
- Continuar el mejoramiento del proceso de desarrollo.
- Establecer una igualdad de conocimiento dentro de los desarrolladores para la buena práctica de los estándares y técnicas de desarrollo.

Continuando con el modelo propuesto por (Fagan, 1976) , en el cual dice que el 60% de defectos pueden encontrarse haciendo inspecciones, es necesario mencionar los pasos a tener en cuenta para la inspección

- Planificación: Se establece cuando el desarrollador termina su producto, se forma el equipo de inspectores y se designa un moderador que debe asegurarse que se satisfaga el criterio de la inspección.

- Familiarización: Solamente se hace si los inspectores no conocen el desarrollo del proyecto, en este paso se detalla el contexto a cubrir en las inspecciones.

- Preparación: Los miembros del equipo estudian de forma individual los artefactos para prepararse para satisfacer los papeles asignados. En este paso se recomienda el uso de listas de chequeo para encontrar defectos comunes.

- Inspección: El equipo realiza una reunión de inspección para encontrar defectos, y registrarlos. El propósito de la reunión de la inspección es la detección de los defectos o de violaciones de estándares, y cualquier tentativa de encontrar soluciones. El moderador debe asegurarse que todos los inspectores se encuentren preparados.

- Correcciones: El autor realiza las correcciones de los defectos encontrados en la reunión de inspección.

- Seguimiento: la corrección de cada uno de los defectos reportados es verificado por el moderador, para garantizar que no se presentan nuevos defectos.

De acuerdo con Fagan, las inspecciones son para evaluar el código y no a los desarrolladores. El objetivo es mejorar la calidad de un producto, teniendo en cuenta que las correcciones son realizadas por el autor fuera de las reuniones, no durante el tiempo de las inspecciones, dado que las reuniones no deben durar más de dos horas y se deben revisar máximo 250 líneas de código por sesión, esto para no disminuir la productividad

en la detección de defectos. Esta disminución se da por el cansancio que se puede presentar en el equipo de inspectores.

Es importante mencionar que al modelo de Fagan se le han propuesto modificaciones las cuales difieren unas de otras: (IEEE, 1997) (Zamora Hernández, 2011).

Inspecciones: evalúan el código centrándose principalmente en los documentos, los modelos y el código. Se pueden entender como un repaso detallado y formal del trabajo en proceso a través de grupos de trabajadores, los cuales estudian el producto de forma independiente, y posteriormente se reúnen con el fin de examinar el trabajo en detalle. Los productos no serán considerados óptimos hasta que estén completas la inspección y las correcciones necesarias.

Walkthroughs: Inspecciones, pero en ellas el análisis es presentado por el propio desarrollador y el mecanismo para conducir la inspección es el desarrollo de un escenario sobre el código.

Revisión técnica del software: es una forma de revisión de par, en la cual un equipo cualificado examina la conveniencia del producto software e identifica las diferencias con las especificaciones y los estándares.

Auditorias: Revisión del software la cuál es llevada a cabo por uno o más interventores, los cuales no pertenecen a los miembros de la organización. Se trata de un examen independiente de un producto, de un proceso, o de un sistema software con el fin de determinar su conformidad con las especificaciones, los estándares, los acuerdos contractuales u otros criterios.

Modelo de Humphrey

Es similar al proceso de Fagan pero los roles del equipo de inspectores cambia, ahora son el asesor, productor y revisor, pero la diferencia real del proceso radica en que permite que el autor participe de las inspecciones grupales. (Zamora Hernández, 2011).

1. METODOLOGÍA

1.1. Metodología de Trabajo

El experimento realizado se hizo para tratar de evidenciar que a través del uso de las inspecciones de código, se puede minimizar la cantidad de defectos en el desarrollo de software.

Para dicho experimento fueron seleccionados 9 estudiantes de la facultad de ingeniería de sistemas del Politécnico Grancolombiano, los cuales estaban cursando práctica aplicada y por lo tanto tenían conocimientos en desarrollo de software.

Las actividades para este experimento se iniciaron en febrero de 2014 y terminaron en mayo de 2014, es decir; con una duración de 3 meses en los cuales se tenían 2 sesiones de trabajo por semana, los jueves con una hora treinta minutos y los sábados con tres horas, veinte minutos.

La selección se hizo con un cuestionario enfocado en conocimientos den JAVA, bases de datos, manejo de archivos, interfaces gráficas, bases de datos JDBC, y la experiencia de cada uno en el desarrollo de software a nivel académico y profesional. Luego los estudiantes se clasificaron usando una calificación 1 a 4, donde 1 correspondía a los que no tenían menos experiencia, o no tenían experiencia profesional y 4 a los que tenían más experiencia o si la tenían a nivel profesional.

Se armaron 2 equipos de trabajo, uno de 4 y otro de 5 estudiantes respectivamente, el equipo de 4 estudiantes realizaba tareas correspondientes a diseño y aplicación de metodología TDD. El equipo de 5 integrantes era el encargado de realizar el diseño y las inspecciones de código, técnica en la cual se enfoca este documento.

Durante este tiempo se hicieron seis (6) desarrollos de software (retos), los cuales tenían diferente grado de dificultad y diferentes instrucciones para su construcción las cuales incrementaban de acuerdo al número de reto así:

N. RETO	REQUERIMIENTO	INSTRUCCIONES
1	Construir un lector de palabras reservadas	N/A
2	Construir un lector de inventarios	• Realizar inspecciones de Código • Elaborar informe de las inspecciones
3	Construir un sistema de cobros de parqueadero	• Realizar inspecciones de Código • Elaborar informe de las inspecciones
4	Construir un sistema de facturación para un restaurante	• Realizar el diseño cual debe ser aprobado para iniciar la construcción del software. • Realizar inspecciones de Código • Elaborar informe de las inspecciones
5	construir un lector de extractos bancarios	• Realizar Diseño, el cual debe ser aprobado para iniciar la construcción del software. • Realizar inspecciones de Código con lista de chequeo
6	Construir un administrador de torneos	• Realizar Diseño, el cual debe ser aprobado para iniciar la construcción del software. • Realizar inspecciones de Código con lista de chequeo

Tabla 1: Retos e Instrucciones

Los retos sólo se desarrollaron durante las sesiones de trabajo pactadas, en las cuales se solucionaron dudas y se hizo seguimiento de los tiempos empleados para cada una de las actividades de análisis de requerimientos, diseño, implementación, mantenimiento e inspecciones.

1.2. Cuerpos Metodológicos Básicos

1.2.1. Personal Software Process (PSP)

Teniendo en cuenta la importancia del trabajo individual de los ingenieros de sistemas, Watts Humphrey desarrollo este proceso en la década de los 80´s, el cual aplicó inicialmente a estudiantes de maestría de ingeniería de software de la universidad Carnegie Mellon. (W, 2005)

PSP ayuda a mejorar las estimaciones de esfuerzo y tiempo para la implementación del desarrollo de software; este proceso también ayuda en la mejora de calidad del producto entregado disminuyendo los defectos. El propósito es la aplicación de métodos avanzados para las tareas que ejecutan los ingenieros en su día a día. Cada ingeniero debe ser responsable de su producto, identificar como mejorar su propio trabajo, estimación del tamaño del producto, planificación, gestión de calidad y diseño. Los ingenieros deben verificar la calidad del producto que están entregando y garantizar el menor número de defectos.

1.2.2. Team Software Process (TSP)

El TSP desarrollado por Watts Humphrey en 1996, y tiene como prerrequisito conocer PSP. Estos dos procesos tienen como objetivo las mejoras del tiempo, la calidad y el tamaño del producto, adicionando fechas de terminación de cada una de las tareas pero desde la perspectiva de equipo y además se incluyen roles: (López Lengua, 2013)

- Líder del equipo
- Gerente de desarrollo
- Gerente de Planeación
- Gerente de Calidad y Proceso
- Gerente de Soporte

2. RESULTADOS

A partir de los datos recopilados, se obtuvieron los siguientes datos:

2.1. RETO 1: LECTOR DE PALABRAS RESERVADAS

El equipo debía construir una solución que a partir de dos archivos de texto (uno con palabras reservadas y otro con un texto) encontrara la cantidad de palabras reservadas y el total de palabras en el archivo, validando que se ingresara al menos una palabra reservada y que no tuviera números ni espacios en blanco y que el archivo con el texto no estuviera vacío.

El tiempo total usado para este reto fue de 230 minutos en los cuales el grupo solo entregó una versión del código que no fue funcional.

Para este reto solamente se hicieron pruebas unitarias, no se aplicó ninguna técnica formal o metodología para el aseguramiento de calidad del software.

Con esta primera actividad se midieron los conocimientos de los integrantes del equipo de trabajo, lo cual permitió ver la falta de comunicación entre ellos para hacer actividades como la distribución de tareas. Adicionalmente, no solicitaron aclaración de las dudas generadas para resolver el problema planteado, demostrando falta de liderazgo en el equipo y bajo nivel de conocimiento técnico.

Realizan el diseño basados en pruebas de concepto encontradas en internet, debido a su bajo nivel técnico.

2.2. RETO 2: LECTOR DE INVENTARIOS

En el segundo reto se debía leer un archivo de texto en donde cada línea estaba separada por comas y correspondía a un producto, cantidad en inventario, cantidad máxima en inventario y el precio unitario de venta. Se debía validar valores vacíos, cantidades y precios no negativos y mostrar por pantalla si algún producto se debía pedir (si había menos del 20% de existencias en el inventario con respecto al máximo) y el precio modificado a la mitad, en caso que un producto no se vendiera (si había el 70% o más del producto con respecto al máximo).

Se dio la instrucción al equipo, que debía aplicar la técnica de inspecciones de código y hacer el informe de los defectos encontrados por cada unos los retos hasta finalizar el experimento.

Teniendo en cuenta que no conocían la técnica, se les dio una inducción a través de un ejercicio práctico.

Con respecto a la primera actividad se les dio más tiempo y se evidenció un poco de trabajo de equipo; hicieron análisis del requerimiento solicitaron aclaración de dudas al respecto así como el diseño de nivel medio solicitando aclaración de dudas técnicas sobre validación de datos.

Se tomaron 270 minutos para hacer el desarrollo, de los cuales 60 minutos fueron usados para hacer inspección de código; se entregaron 4 versiones del código dando como resultado otro reto no resuelto con 9 defectos en la inspección sin lista de chequeo.

2.3. RETO 3: COBRO TARIFA DE PARQUEADERO

En el tercer reto se solicitó hacer un sistema que permitiera hacer los cobros de un parqueadero, a partir de dos archivos de texto, donde uno tenía las placas que correspondían al convenio con el parqueadero y separadas con coma, las cuales no podían ser vacías, ni contener caracteres especiales y debían cumplir con el formato de

una placa de un auto colombiano (tres letras seguidas de tres números); y otro archivo separado con comas con la placa, fecha y hora de entrada, fecha y hora de salida, no podían ser valores vacíos y las fechas debían ser correctas y corresponder con el formato dd/MM/yyyy HH:mm. El sistema debía calcular y mostrar por pantalla el valor a pagar por concepto de parqueadero para cada uno de las placas del segundo archivo, teniendo en cuenta que el valor del minuto es 60, si llevaba estacionado más de 4 horas o más, debía pagar la tarifa plena de 10000 y si se encontraba la placa en el archivo de convenios se realizaba un descuento del 40% en el valor total a pagar incluyendo la tarifa plena.

Además de hacer inspección de código, el equipo debía hacer un diseño de alto nivel, el cual debía ser aprobado para que pudieran continuar con la implementación.

El tiempo total usado para este reto fue de 400 minutos de los cuales 120 fueron para hacer inspecciones; el grupo entregó dos versiones del código. En este reto se identificaron 7 defectos sin lista de chequeo.

2.4. RETO 4: FACTURA DE VENTA RESTAURANTE

En este reto el equipo debía implementar un sistema que leyera un archivo de texto con valores separados por comas, en donde se encontraban pedidos en un restaurante, cantidad de pedidos solicitados, y valor unitario, se debía validar que el nombre del producto no estuviera en blanco, y que los valores fueran numéricos positivos mayores a 0, que no estuvieran vacíos y que no contuvieran letras, el objetivo final era imprimir un archivo de texto en el cual se encontraba la factura de venta de los productos, calculando el IVA, propina del 10% y el total.

El tiempo total para este reto fue de 250 minutos, usando 100 para inspecciones en las cuales se encontraron 11 defectos sin lista de chequeo. El grupo entregó 2 versiones del código.

2.5. RETO 5: EXTRACTO BANCARIO

Para el reto 5, el equipo debía generar un extracto a partir de un archivo de texto con el movimiento de una cuenta bancaria separado por comas, el cual contenía la fecha y hora en formato dd/MM/yyyy HH:mm:ss, el código del tipo de movimiento (1: retiro, 2: consignación, 3: pago en línea) y el valor del movimiento. El sistema debía validar que la fecha tuviera el formato correcto, movimientos numéricos positivos, retiros menores o iguales a millón y medio. En caso de encontrar errores, debía indicar el error exacto que se presentaba y debe continuar el procesamiento en caso de poder hacerlo. El sistema debía generar por pantalla, para cada mes, la cantidad de retiros realizados en el mes y el valor total de retiros, la cantidad de consignaciones en el mes y el valor total de consignaciones, la cantidad de pagos en línea y el valor total de dichos pagos y el saldo al final de cada mes

El tiempo total para este reto fue de 450 minutos, 100 se usaron para hacer inspecciones con lista de chequeo, donde se reportaron 10 defectos. El grupo entregó 3 versiones del código. El formato utilizado para la lista de chequeo se puede evidenciar en el Anexo 1.

2.6. RETO 6: ADMINISTRADOR DE TORNEOS

Para finalizar el experimento el equipo debía construir un administrador de torneos donde se ingresaban 2 archivos de texto: en el primer archivo se encontraban los nombres de los equipos participantes en el torneo separados por comas y en el segundo los resultados de los encuentros entre equipos. Se debía validar en el primer archivo que no existieran nombres de equipos en blanco y que existan exactamente 8 equipos en el archivo. En el segundo archivo se debía verificar los marcadores numéricos, enteros positivos, que no existieran partidos de equipos contra sí mismos, que se cumpliera con el formato especificado y que los emparejamientos de los equipos fueran entre equipos cargados con anterioridad. En caso que alguna de las validaciones anteriores fallará, el sistema debía presentar un mensaje de error indicando el error exacto. El sistema debía generar

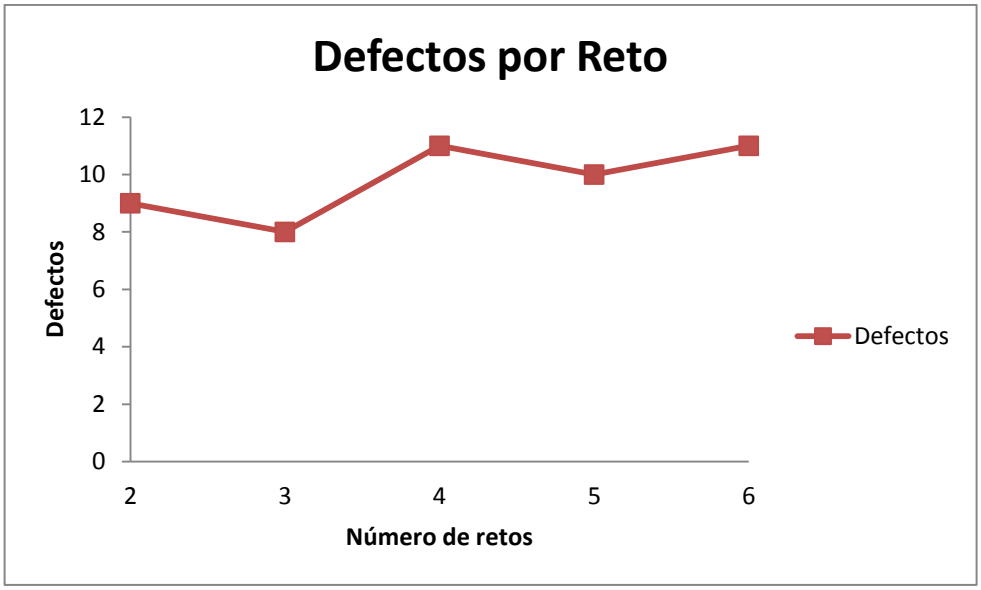
un archivo de texto llamado TablaTorneo.txt, con la tabla de posiciones del campeonato. Los criterios de desempate eran los siguientes: puntos, goles diferencia y goles a favor de visitante. En caso de presentarse empate en los 3 ítems, se debía desempatar por orden alfabético.

El tiempo total para este reto fue de 385 minutos, se usaron 70 para hacer inspecciones con lista de chequeo, donde se encontraron 11 defectos. El grupo hizo entrega de 4 versiones del código.

3. ANÁLISIS DE RESULTADOS

A partir de los resultados encontrados en cada reto, se puede analizar lo siguiente:

Cuando los retos se realizaron bajo lista de chequeo el número de defectos aumentó con respecto a los retos 2 y 3. Sin embargo, en el reto 4 sin lista de chequeo y el reto 6 con lista de chequeo se encontraron el mismo número de defectos, aun cuando el último reto tenía mayor complejidad. Esto puede ser debido a la falta de experiencia del equipo para hacer inspecciones de código. La grafica 1 resume lo anterior.



Gráfica 1: Defectos por reto

4. CONCLUSIONES

Al finalizar el proceso de investigación se puede concluir que:

- No siempre se encuentra mayor número de defectos al emplear lista de chequeo.
- Se encontró que la metodología empleada no fue la apropiada para alcanzar el objetivo de esta investigación dado que no hay un modelo de comparación entre dos equipos de trabajo donde se realicen inspecciones por un equipo experimentado y otro donde se hagan implementaciones de código sin utilizar esta técnica.
- Se observó que las inspecciones de código realizadas no fueron óptimas debido a la inexperiencia del equipo de inspectores que participó en cada reto desarrollado en esta investigación. Se encontró que sin importar el nivel de dificultad y el tiempo empleado para la ejecución de cada uno de los retos, el número de defectos fue similar, sin mostrar un impacto considerable en la minimización de la cantidad de defectos encontrados con o sin uso de listas de chequeo.

5. TRABAJO FUTURO

Dentro de las futuras investigaciones con respecto al uso de técnicas y procesos para garantizar la calidad del software, es importante tener una muestra de retos y un grupo de desarrolladores experimentados más grande. Esto con el fin de comparar los resultados tanto con un equipo experimentado como con otro equipo que sólo implemente el código.

Investigar qué hacer si durante las inspecciones se encuentra que hay líneas de código que no son necesarias y que no afectan el código si están o no, dado que no se ha encontrado literatura que ilustre si es requerido eliminarlas y contarlas como defectos. También se debería tener en cuenta qué hacer si en un código de pocas líneas no se encuentran errores, pues los diferentes autores mencionan que si esto no ocurre, es mejor volver a hacer la implementación o método correspondiente.

Hacer el experimento completo de inspecciones donde se incluya la revisión de todos los artefactos (requerimientos, arquitectura, diseño, datos de prueba) usados en las diferentes etapas para la construcción de desarrollos de software.

6. RECOMENDACIONES

Para dar continuidad a esta investigación se recomienda tener un equipo de trabajo con experiencia profesional en desarrollo de software y en la ejecución de inspecciones de código. Además se debe usar listas de chequeo desde el primero hasta el último reto. Dichas listas se deben elaborar específicamente para el lenguaje de programación con el cual se trabajará. Adicionalmente, es importante que se trabaje en forma paralela con otro equipo que tenga experiencia en desarrollo de software y que no realice inspecciones de código. De este modo al realizar las pruebas funcionales se podrá saber cuál equipo entrega la aplicación con menos defectos y quién presenta menos versiones del reto solicitado cumpliendo con todos los requerimientos.

7. BIBLIOGRAFÍA

Casallas G., R. (s.f.). Uniandes.edu.co. Recuperado el 25 de 5 de 2014, de http://sistemas.uniandes.edu.co/~csof5101/dokuwiki/lib/exe/fetch.php?media=principal:csof5101-inspecciones.pdf

CMMI Product Team;. (2006). CMMI (Capability Maturity Model Integration). CMU/SEI.

Deming, W. E. (1982). Out of the crisis . Massachusetts: MA:MIT Center For Advanced Engineering Study, Cambridge.

Fagan, M. (1976). Diario de los sistemas de IBM, Vol. 15, no. 3, 1976; Examinando diseños y código del software", Proceso de datos automático, Octubre de 1977; "Avances en inspecciones del software", Transacciones de IEEE en la tecnología de dotación lógica, Vol. 12, no. 7.

Garvin, D. (1984). What Does 'Product Quality' Really Mean. Sloan Management Review , 25(18).

IEEE. (1997). Estándar 1028 de IEEE para las revisiones software.

James McCall Paul K. Richards, a. G. (1977). Factors in Software Quality. NY.

López Lengua, M. R. (2013). Hacia la identificación de un método universal de calidad de software. Bogotá.

M, F. (s.f.). Advances in Software Inspections. IEEE Transactions on Advanced Vol.12.

M, F. (s.f.). Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal , págs. 8, 3.

W, S. H. (2005). PSP(sm): A Self-improvement process for software engineers, . Addison Wesly.

W, S. H. (2000). Team Software Process. SEI CMU/SEI-2000 TR-023.

W, S. H. (2000). The Personal Software Process, (PSP). CMU/SEI-2000 TR-022.

Zamora Hernández, J. (2011). Análisis de los procesos de verificación y validación en las organizaciones software.

Zamuriano Sotés, R. F. (s.f.). Las Inspecciones de Software y las Listas de Comprobación. Recuperado el 20 de 5 de 2014

8. ANEXOS

POLITECNICO GRANCOLOMBIANO

Lista de chequeo para Inspecciones de Código

Fecha:
Autor:
Inspector:

Ítem	SI	NO	N/A	Observaciones
¿La clase se encuentra en el paquete correcto?				
¿Todas las variables están inicializadas?				
¿Se implementa más cosas que las especificadas en el diseño?				
¿Se implementa menos cosas que las especificadas en el diseño?				
¿Los nombres de las clases corresponden a los nombres especificados en el diseño?				
¿Los métodos corresponden con la signatura especificada en el diseño?				
¿Todos los métodos tienen el número correcto de parámetros?				
¿Todos los parámetros de los métodos tienen los tipos correctos?				
¿Todos los métodos retornan el tipo de dato correcto?				
¿Las Excepciones se están capturando correctamente?				
¿Los ciclos implementados terminan?				
¿Los archivos abiertos se cierran correctamente después de ser utilizados?				
¿Los arreglos utilizados son del tamaño suficiente?				
¿Todas las variables declaradas son utilizadas?				
¿Todos los métodos declarados son utilizados?				
¿La variable que recibe el cálculo de tipos numéricos es del tipo adecuado?				
¿Existen muchos ciclos anidados? Máximo 2 ciclos anidados				
¿Hay líneas de código innecesarias?				