Colocador de fichas

Plan de pruebas

Versión 1.0

Arburola León, Samantha Patricia
Murillo Vargas, Jose Pablo
Villalobos Rodríguez, Roger Andrés

Marzo 2018

Historial de revisiones

Nombre	Fecha	Modificaciones
revisión 1.0	21/03/2018	Creación
revisión 1.1	30/03/2018	Avance en el punto II y III

I. Introducción

A. Descripción del proyecto

El proyecto a evaluar consiste en un programa que acomoda fichas de un tablero en el menor tiempo posible. El acomodo debe dejar las fichas agrupadas ya sea en una sola fila, una columna o en diagonal.

El tamaño del tablero y las posiciones iniciales de las fichas son especificados en un archivo de texto. El programa va a ser llamado por medio de línea de comando y se especifica el parámetro del nombre del archivo de donde agarrar los datos.

B. Alcance

El plan de pruebas propuesto va a realizar inspecciones sobre los artefactos creados por el grupo. Dentro de las inspecciones se va a estar buscando que se cumpla con las normas de estilos de programación y la presencia de funciones o métodos que cumplan los requerimientos funcionales esperados.

El plan no cubre lo que es la funcionalidad del producto de software ni tampoco va a ser probado contra diferentes conjuntos de entradas. Solo se va a inspeccionar lo que es la documentación y artefactos con el objetivo de detectar e identificar las anomalías del software.

C. Especificaciones Técnicas

El ambiente de desarrollo es CodeBlocks 17.12 el cual fue elegido por el equipo de desarrollo del Grupo 07 de Lenguajes de Programación de CASJ del Semestre I 2018

Las herramientas de software que se utilizan para las pruebas estáticas de código es CppCheck y Splint, seleccionados gracias a las características robustas que presenta, dado el gran espectro de creación de la solución para el problema asignado al equipo de desarrollo, esta herramienta de prueba de código colabora con la cobertura de la mayor cantidad de incidentes que se pueden presentar.

II. Diseño y tipos de pruebas

A. Distribución del Equipo

Se distribuye el equipo según la recomendación de la la IEEE en el Estándar 1028 de 2008.

Inspection leader:	Jose Pablo Murillo Vargas
Recorder:	Roger Villalobos Rodriguez
Reader:	Jose Pablo Murillo Vargas
Author:	Samantha Arburola
Inspector:	Roger Villalobos Rodriguez

B. Pruebas

La primera prueba consiste en usar una serie de listas de chequeo para verificar qué tanto cumple el código fuente en cuanto a lo especificado por el cliente anteriormente. El equipo de trabajo de inspección de calidad definió una lista de chequeo con un conjunto mínimo de funciones que deben estar presentes para satisfacer los requerimientos funcionales. Se buscará dentro del código fuente para la presencia de estas funciones. La segunda lista de chequeo es para verificar requerimientos adicionales definidos por el cliente. Se hará un resumen de hallazgos y las listas de chequeo llenadas por el grupo.

La definición de funciones ayuda a modularizar el código y también así poder asignar una sola responsabilidad para mejorar la mantenibilidad y detección de errores.

Requerimiento funcional	Cumple		
	Si	No	Observaciones
Recibir parámetro de archivo de prueba desde la terminal			
Abrir y validar existencia de archivo de prueba			
Validar datos dentro de archivo de prueba			

Guardar datos de entrada especificados dentro de una estructura de datos			
Realizar acomodo óptimo de fichas			
Mostrar resultado con las fichas acomodadas y el número de movimientos realizados			

Lista de chequeo para requerimientos funcionales del sistema

Requerimiento	Cumple		
	Si	No	Observaciones
No usar variables globales			
Definición y uso de funciones			
Documentación interna para cada función			

Lista de chequeo para requerimientos adicionales del sistema

La segunda prueba es poner a prueba el código fuente ante diferentes estilos de programación sugeridos usando la inspección de los integrantes de grupo. El propósito de seguir un estilo de programación es para mejorar la legibilidad de código y facilitar hacerle mantenimiento, mejoras y agregar más código.

El primer estilo al que se va a poner en prueba es una serie de prácticas “obligatorias” sugeridas por la página Tecnowired.com. Esta serie de reglas toma en cuenta la estructura del programa, las variables y comentarios. Lo que se va a revisar incluye:

- Indentación dentro de funciones, registros y estructuras de control
- Uso de llaves en toda estructura de control
- Escribir sólo una instrucción por línea

- Incluir un espacio después de las comas
- Separar los operadores binarios con espacios a ambos lados
- Las variables de índices dan comienzo con i
- Uso de formato CamelCase para variables con nombre compuesto
- Uso de #define para variables constantes
- Cada bloque de código comienza con un bloque de comentario donde explica el propósito, parámetros de entrada, posibles valores de salida,

El segundo estilo que se va a usar como base para la inspección es el estilo GNU popularizado por Richard Stallman. Los aspectos que se van a revisar son:

- Usar, a lo sumo, 79 caracteres por línea de código
- Poner el corchete del cuerpo de una función en la primera columna debajo de la declaración de este
- Comenzar los nombres de funciones a declarar en la columna uno
- Un comentario inicial en el archivo que explica su propósito
- Incluir comentarios en cada función de qué hace, que parámetros recibe y qué valores puede retornar
- Declaración explícita del tipado de todos los objetos
- Uso de minúsculas para la declaración de variables
- Uso de guión bajo para nombres compuestos

La tercera prueba consiste en una revisión en cuanto a la documentación incluida con los archivos de código fuente. Aspectos sobre los documentos que se esperan revisar son:

A. Informe escrito sobre el proyecto
    a. Revisión de estrategia planteada ante lo codificado en el archivo fuente
    b. Documentación o referencia en cuanto a uso particular de funciones o algoritmos
    c. Explicar qué uso de paso de parámetros usaron
    d. Evidencia de pruebas realizadas
    e. En caso de existir, justificación de aspectos de la especificación que no fueron desarrollados
    f. Bibliografía

B. Archivo readme.txt
    a. Descripción de ambiente de desarrollo utilizado
    b. Descripción de cómo correr el programa

A cada punto se le dará una calificación de acuerdo al siguiente rubro de completitud:

0	0.5	1
El punto no está presente	El punto está presente pero el texto carece de claridad	El punto está presente y se presenta de manera clara

La cuarta prueba consiste en el uso de diferentes herramientas de análisis estático de código para poder detectar errores potenciales que se pueden presentar dentro del código.

La primera herramienta a utilizar se llama cppcheck. Esta herramienta se encarga de detectar errores que usualmente no se encuentran a la hora de compilar. Se enfoca en encontrar comportamiento indefinido, lo cual cubre pero no se limita a:

- Punteros muertos
- División entre cero
- Overflow de enteros
- Operaciones de shift inválidas
- Conversiones inválidas
- Manejo de memoria
- Chequeo fuera de rango
- Variables sin inicializar

La segunda herramienta se llama Splint. La herramienta se encarga de encontrar vulnerabilidades de seguridad y errores de codificación.

C. Métricas

Completitud del sistema

a. Uso de listas de chequeo para medir qué tanto de lo codificado cumple con la especificación del sistema a desarrollar.

Se saca un promedio de los requerimientos cumplidos sobre el total para sacar el resultado de puntaje de la prueba.

Cumplimiento con estándares de codificación

a. Prácticas sugeridas por Tecnowired
b. Normas de codificación GNU

De los puntos que se van a revisar en cuanto a los estilos de codificación, se hará una inspección al archivo fuente por parte de los integrantes del equipo. Se reportan las faltas realizadas en el código fuente usando como base los puntos previamente definidos.

Cada 5 errores reportados equivale a restar 1 punto sobre una calificación de 10. Después de realizar una prueba con ambos estilos, se saca un valor promedio. No se va a seguir reportando errores después de los 20 pero si se restan de la calificación.

Calidad de documentación

a. Completitud en cuanto a contenido de documentación de acuerdo a lo establecido en la parte de “Pruebas”

Para esta métrica se va a sacar un promedio de completitud y claridad de los elementos de la documentación sobre el total posible.

Ausencia o carencia de errores potenciales

Para esta métrica se van a usar las herramientas de análisis sintáctico previamente mencionadas. Se van a ignorar los errores que tengan que ver con el estilo de codificación ya que hay otra métrica para ello. Se va a deducir 1 punto sobre el total de 100 por cada error potencial encontrado por la herramienta y se saca un promedio de calificaciones.

D. Criterio de Aceptación

Se define como Aceptado aquel software que presente únicamente un 4% de errores, es decir, que todas sus métricas den como resultado 9.6 o mayor puntaje para las pruebas.

III. Análisis de evaluación de pruebas

Se analiza el Proyecto 01 *Imperativo Grupo 07 - Pablo - Gerardo - Kevin* del curso de Lenguajes de Programación del Semestre I 2018 de la carrera de Ingeniería en Computación del Tecnológico de Costa Rica en San José.

El equipo de Analistas está conformado por Samantha Patricia Arburola León, Jose Pablo Murillo Vargas y Roger Andrés Villalobos Rodríguez estudiantes del curso de Aseguramiento de Calidad de Software del Semestre I de 2018 de la carrera de Ingeniería en Computación del Tecnológico de Costa Rica en San José.

Para la inspección se programan 3 reuniones de 3 horas cada una, en las cuales se plantean las pruebas por ejecutar, se ejecuta la evaluación y se analizan los resultados para su proceso de documentación de evidencia.

El producto de software inspeccionado es un programa en el lenguaje de programación C, el cual cuenta con un solo archivo de código, un archivo de prueba y tres páginas de documentación.

La entrada para la inspección será el archivo,de código el cual se somete a un análisis sintáctico con OCLint.

El objetivo de la inspección es evidenciar la aplicación del estándar de Código Limpio y la guía para sintaxis de C.

En el reporte de Auditoría se mostrará:

- La lista de anomalías, que contiene cada ubicación, descripción y clasificación de anomalías
- La disposición del producto de software
- Cualquier exención concedida o exención solicitada
- Tiempo de preparación individual y total del equipo de inspección
- El tiempo total de retrabajo

IV. Anexos

Herramienta de Análisis Sintáctico	Características

OCLint	- Posibles bugs
	- Automatiza este proceso de inspección con características avanzadas como:
	  - Carga dinámicamente las reglas en el sistema, incluso en el tiempo de ejecución
	  - Personalización de los comportamientos de la herramienta
	  - Integración continua
	  - Mejor precisión y eficiencia por medio de árboles sintácticos
	- Revisiones bastantes específicas como:
	  - Posibles errores - declaraciones if / else / try / catch / finally vacías
	  - Código no utilizado - variables y parámetros locales no utilizados
	  - Código complicado: alta complejidad ciclomática, complejidad NPath y alta NCSS
	  - Código redundante - declaración if redundante y paréntesis inútiles
	  - Código huele - método largo y larga lista de parámetros
	  - Prácticas incorrectas: lógica invertida y reasignación de parámetros

PC-lint	- Integración en IDEs como herramienta externa - Herramienta multiplataforma FlexeLint - Es capaz de analizar la correcta exclusión mutua en programas multihilo - Analizar el código según un conjunto específico de reglas o recomendaciones
QA-C	- Conformidad con las pautas de codificación - Calcular métricas de código para proyectos con grandes bases de código - Trabaja sobre diferentes IDEs para atender código en mantenimiento y código nuevo - Despliegue de resultados en forma gráfica - Asequible por línea de comandos

Tabla 1. Herramientas de Análisis Sintáctico

V. Referencias

1028-2008 IEEE Standard for Software Reviews and Audits. (2008). NY, USA: IEEE / Institute of Electrical and Electronics Engineers Incorporated, pp.16-25.

Oclint.org. (2018). OCLint. [online] Disponible en : http://oclint.org/ [Consultado el 21 marzo 2018].

Static Analysis and Coding Standards Compliance | PRQA. (2018). Prqa.com. Retrieved 2 April 2018, from http://www.prqa.com

The Leader in Static Analysis for C/C++ -- PC-lint and FlexeLint. (2018). Gimpel.com. Retrieved 1 April 2018, from http://www.gimpel.com/html/index.htm