HOJA 0

45085146

Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba
Asignatura: Ingeniería del Software I
Segundo Cuatrimestre de 2024

1	10
2	+3
3	9
4	9
5	8

Apellido: ACHÁVAL BENZERO
Nombre: TOMÁS
Nro. hojas entregadas: 4

Examen Parcial No. 2

Contestar con lapicera. No usar verde.

~~74% = 6,1~~

78% = 6.7

Ej. 1a. Describa refactorización.

Ej. 2a. Describa el proceso de codificación: Incremental.

Ej. 3a. Describa el modelo de proceso de desarrollo: Timeboxing. Enuncie dos ventajas y la principal desventaja de este modelo.

Ej. 4a.

a) Explique el enfoque ETVX.

b) Describa Programación estructurada.

Ej. 5a.

a) ¿Qué son y para qué sirven los criterios de selección de tests?

b) Dé un ejemplo de un sistema con dos conjuntos de tests donde uno cumpla el criterio de sentencias y no encuentre el error y otro cumpla el criterio de ramificaciones y encuentre el error.

HOJA 1

45085146 A.

TOMÁS ACHAVAL BETETO

EJERCICIO 1:

LA REFACTORIZACIÓN ES UNA ACTIVIDAD QUE SE REALIZA DURANTE EL PROCESO DE CODIFICACIÓN Y CONSISTE EN SIMPLIFICAR/MEJORAR LA ESTRUCTURA INTERNA DEL PROGRAMA SIN ALTERAR SU COMPORTAMIENTO EXTERNO. ESTAS MEJORAS SUELEN TENER LOS OBJETIVOS DE AUMENTAR COHESIÓN, REDUCIR ACOPLAMIENTO, Y MEJORAR LA RESPUESTA AL PRINCIPIO ABIERTO-CERRADO.

ESTA ACTIVIDAD PERMITE UNA CONTINUA MEJORA DEL DISEÑO Y AYUDA A SU COMPRENSIBILIDAD.

COMO SE MODIFICA EL CÓDIGO, UNA REFACTORIZACIÓN GRANDE PODRÍA TRAER MUCHOS ERRORES, POR LO QUE SE RECOMIENDA:

- REFACTORIZAR EN PEQUEÑOS PASOS
- DISPONER DE SCRIPTS DE TEST PARA VERIFICAR QUE EL COMPORTAMIENTO EXTERNO DE CADA MODULO REFACTORIZADO PERMANEZCA INALTERADO.

LAS REFACTORIZACIONES MÁS COMUNES SON:

MEJORAS DE MÉTODOS:
- EXTRACCIÓN DE MÉTODOS: SE REEMPLAZAN PARTES DE UN MÉTODO LARGO POR LLAMADOS A MÉTODOS CUYA SIGNATURA ES CLARA E INDICA LO QUE SE REALIZA. TAMBIEN SE PUEDEN SEPARAR EN DOS LOS MÉTODOS QUE MODIFICAN EL ESTADO DE UN OBJETO Y A LA VEZ DEVUELVEN UN VALOR.
- AGREGAR/ELIMINAR PARÁMETROS DEJANDO ÚNICAMENTE LOS NECESARIOS.

MEJORAS DE CLASES:
- DESPLAZAMIENTO DE MÉTODOS/ATRIBUTOS: SI UN MÉTODO O ATRIBUTO DE UNA CLASE INTERACTÚA MÁS CON OTRA CLASE, LLEVARLO A ELLA.
- EXTRACCIÓN DE CLASES: SI UNA CLASE ENGLOBA MUCHOS CONCEPTOS, LLEVAR CADA CONCEPTO A UNA CLASE INDIVIDUAL. ESTO ES UN EJEMPLO DE UNA MEJORA CON EL OBJETIVO DE AUMENTAR LA COHESIÓN.
- CONVERTIR CONJUNTOS DE ATRIBUTOS EN OBJETOS: A VECES, UN CONJUNTO DE ATRIBUTOS CONFORMA UNA ENTIDAD LÓGICA QUE PUEDE SER MÁS FÁCIL DE MANIPULAR A TRAVÉS DE UN OBJETO.

MEJORAS DE JERARQUÍA:
- REEMPLAZAR CONDICIONALES POR POLIMORFISMO: SI UNA OPERACIÓN DEPENDE DE UN CHEQUEO DE TIPOS, REEMPLAZAR LA LÓGICA P/UTILIZAR UNA JERARQUÍA DE CLASES APROPIADA.
- SUBIR MÉTODOS/ATRIBUTOS A CLASES MÁS ALTAS EN JERARQUÍA PARA EVITAR DUPLICACIÓN DE CÓDIGO EN SUB-CLASES.

TAMBIÉN SE DENOMINA POR "MALOS OLORES" A AQUELLAS PARTES DEL CÓDIGO FÁCILMENTE IDENTIFICABLES QUE INDIQUEN UNA POSIBLE NECESIDAD DE REFACTORIZACIÓN. ALGUNOS EJEMPLOS DE MALOS OLORES PUEDEN SER MÉTODOS/CLASES MUY LARGAS, SUB-CLASES CON IGUAL COMPORTAMIENTO QUE SUPER-CLASE, ETC.

Ejercicio N°2

El proceso de codificación incremental consiste en agregar funcionalidades una a una sobre un sistema parcial, partiendo de un sistema base hasta llegar a un sistema completo.

Esto provee un marco que facilita el testing, pues el testing de cada incremento es más simple que el testing de un sistema completo.

También facilita el seguimiento del progreso al saber cuántas funcionalidades fueron implementadas, y cuántas faltan.

Cada incremento debe resultar en un "sistema completo" en sí mismo, es decir, no puede ser una funcionalidad implementada "a medias".

Un modelo de proceso que utiliza codificación incremental es el modelo iterativo (con mejora iterativa) en donde se define una lista de tareas "simples" a realizar y se ejecutan una a una (en orden) hasta vaciar la lista.

HOJA 2

45085146
TOMÁS ACHÁVAL BECERRA

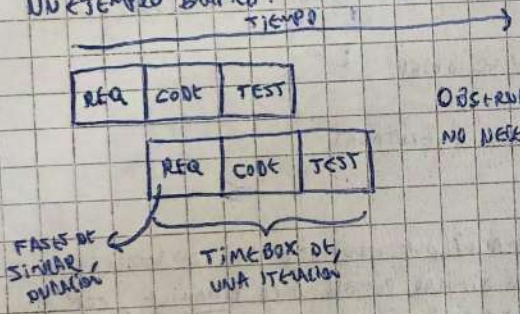
EJERCICIO N°3:

EL MODELO DE PROCESO DE DESARROLLO TIMEBOXING ES UN MODELO ITERATIVO QUE CONSISTE EN DIVIDIR CADA ITERACIÓN EN FASES DE IGUAL DURACIÓN Y ASIGNAR UN EQUIPO DE TRABAJO DISTINTO A CADA UNA DE ELLAS, POR LO QUE SE PUEDE UTILIZAR PIPELINING PARA EJECUTAR ITERACIONES EN SIMULTÁNEO Y REDUCIR AÚN MÁS EL TIEMPO DE ENTREGA DE UN PROCESO ITERATIVO CLÁSICO.

COMO ES ITERATIVO SE DESARROLLAN UNA A UNA Y EN ORDEN TODAS LAS FUNCIONALIDADES DEL PROYECTO) TIENE VENTAJAS COMO PAGOS Y ENTREGAS INCREMENTALES Y EL HECHO DE QUE EL FEEDBACK DE UNA ITERACIÓN PUEDE AYUDAR A ITERACIONES FUTURAS.

EN TIMEBOXING, CADA EQUIPO DE TRABAJO TERMINA SUS TAREAS Y PASA EL/LOS PRODUCTOS DE TRABAJO OBTENIDOS AL EQUIPO ENCARGADO DE LA SIGUIENTE FASE, POR LO QUE INMEDIATAMENTE PUEDEN COMENZAR A TRABAJAR EN LA MISMA FASE DE LA ITERACIÓN SIGUIENTE, RESULTANDO EN PLAZOS DE ENTREGA AÚN MÁS CORTOS.

UN EJEMPLO GRÁFICO:



OBSERVAR QUE EL EQUIPO ENCARGADO DE ANALIZAR Y ESPECIFICAR LOS REQUERIMIENTOS NO NECESITA ESPERAR A QUE SU ITERACIÓN TERMINE P/COMENZAR CON LA SIGUIENTE.

LA PRINCIPAL DESVENTAJA DE ESTE MODELO ES QUE SE REQUIEREN EQUIPOS DE TRABAJO GRANDES CUYA ORGANIZACIÓN Y SINCRONIZACIÓN ES COMPLEJA.

Ejercicio Nº4

a) El enfoque ETVX (Entry-Task-Verification-Exit) consiste en que cada fase del proceso de desarrollo esté compuesta y definida por:

- Criterio de entrada: Establece cuándo y qué documentos necesita la fase para comenzar.
- Tarea: Tarea a realizar.
- Verificación: Especifica los métodos de verificación/validación/revisión que se ejecutarán sobre el producto de trabajo de la fase.
- Criterio de salida: Especifica cuándo se puede considerar que la fase ha finalizado.

De esta forma, se facilita la conexión entre fases y el seguimiento del progreso en el proyecto.

b) Para describir la programación estructurada haré referencia a los siguientes conceptos:

- Estructura estática: Orden lineal de las sentencias en el código.
- Estructura dinámica: Orden en el que las sentencias son ejecutadas.

La programación estructurada es un principio de programación que consiste en desarrollar código de manera tal que su estructura estática sea lo más similar posible a la dinámica, con el objetivo de simplificar la comprensión y el razonamiento (formal e informal) sobre el mismo.

Este principio parte del hecho de que la corrección de un programa depende exclusivamente de su estructura dinámica, sobre la cual es difícil razonar (o se requieren herramientas avanzadas).

Los constructores de la programación estructurada tienen una única entrada y salida y deben ser simples de comprender.

HOJA 3

M5085146

TOMÁS ACHÁVAL BEZERRA

EJERCICIO Nº5

a) COMO SUELE SER IMPOSIBLE DEFINIR CONJUNTOS DE CASOS DE TEST QUE SEAN PERFECTOS (NO FAIL ⇒ NO HAY DEFECTOS) Y A LA VEZ REDUCIDOS (POR COSTOS), SURGE LA NECESIDAD DE DEFINIR CRITERIOS DE SELECCIÓN DE TESTS.

UN CRITERIO DE SELECCIÓN DE TEST ESPECIFICA LAS CONDICIONES QUE LOS CASOS DE TEST DEBEN CUMPLIR CON RESPECTO AL PROGRAMA O SU ESPECIFICACIÓN.

SIRVEN PARA ENFOCAR EL TESTING EN ASPECTOS ESPECÍFICOS Y ASÍ, ANTE LA IMPOSIBILIDAD MENCIONADA ANTERIORMENTE, APUNTAR AL MAYOR CUBRIMIENTO POSIBLE PARA CADA CRITERIO.

(I)

b) SISTEMA: DIVIDE ∩ (GLOBAL) POR X SI X ES MAYOR A 0, DEVUELVE FALSE EN CUALQUIER OTRO CASO.

```
1> FUNCTION DIVIDEIFPOS(INT X):
2>     IF (X ≥ 0) RETURN ∩/X
3>     RETURN FALSE
```

CONJUNTO DE TEST QUE CUBRE CRITERIO DE SENTENCIAS: $\{((X=-1); FALSE)\}$

CONJUNTO DE TEST QUE CUMPLE CRITERIO DE RAMIFICACIONES: $\{((X=0); FALSE), ((X=7); \cap), ((X=-1); FALSE)\}$

COMO SE PUEDE OBSERVAR, EJECUTAR EL CÓDIGO CON EL PRIMER CONJUNTO CUBRE TODAS LAS SENTENCIAS (1,2,3) Y NO DETECTA EL ERROR.

EN CAMBIO, EL SEGUNDO CONJUNTO CUBRE TODAS LAS RAMIFICACIONES (ENTRAR Y NO ENTRAR AL IF) Y A LA VEZ DETECTA EL ERROR (LA FUNCIÓN SÓLO DEBERÍA DIVIDIR POR X SI X ES MAYOR A 0) AL EJECUTAR EL PRIMER CASO, EL CUAL ESPERA UNA RESPUESTA 'FALSE' SIGUIENDO LA ESPECIFICACIÓN, PERO OBTIENE UN ERROR ARITMÉTICO CAUSADO POR UN DEFECTO EN EL CÓDIGO.