uc3m | Universidad Carlos III de Madrid

Práctica Final

Desarrollo de Software

Ingeniería informática - Grupo 81

Fecha	**23/05/2022**
Autores	Ángel del Pozo Manglano - 100442123 angel.pozo@alumnos.uc3m.es
	Francisco Antonio Gallardo Fuentes - 100451146 100451146@alumnos.uc3m.es

ÍNDICE

INTRODUCCIÓN

En esta práctica tenemos que aplicar todos los conocimientos adquiridos a lo largo del curso: Clases de Equivalencia y Valores Límites: hallar el rango de valores en los cuales se mueven nuestros parámetros de entrada y sus límites superiores e inferiores.
Pruebas Estructurales: realizar el Diagrama de Flujo para encontrar el número de caminos y por ende, encontrar los casos de prueba.
Pruebas Funcionales: realizar la Gramática y su correspondiente Árbol de Derivación para obtener más casos de prueba.
Proceso de Refactoring: limpieza de código, entendimiento sencillo para aumentar la eficiencia del programa.
Patrones de Diseño .
Todas ellas para desarrollar una nueva función en un código base dado por los profesores, en la cual tenemos que implementar una forma de que un paciente pueda cancelar su cita para la vacuna.
Además de incorporar un nuevo parámetro en la función get_vaccine_date para, en vez de poner una vacuna por defecto para dentro de 10 días, el paciente pueda incluir la fecha de vacunación que le gustaría.
Para la realización de los tests de nuestra función, utilizaremos la técnica de Clases de Equivalencia y Valores Límites y Pruebas Funcionales que vimos relevantes

DECISIONES DE DISEÑO

Para la realización de los tests en nuestra función, hemos optado por realizarlos en base a Clases de Equivalencia y Valores Límites, ya que creemos que es la opción la cual abarca más terreno con respecto a nuestro programa.
Además hemos introducido técnicas de Pruebas Funcionales, en la cual comprobamos al paciente correcto gracias a archivos JSON y otros casos, sin embargo casos de prueba como la escasez de comillas, algún parámetro en la función y crear otros 40 archivos json para abarcar todas las variaciones, lo vemos algo ineficiente, ya que en la mayoría de esos casos se va a devolver un JSON - Wrong Format, y este mismo error ya lo realizamos una comprobación del formato correcto, en el cual solo necesitamos de una función que comprueba todo tipo de variaciones.

CLASES DE EQUIVALENCIA Y VALORES LÍMITES

En esta sección evaluaremos las clases de equivalencia y sus correspondientes valores límites de cada parámetro de la función cancel_appointment:

Nota: LS = Límite superior.
LI = Límite inferior.

Campo	Tipo de Entrada	Clases Válidas	Clases Inválidas	Valores Límites
date_signature	String finito de 64 caracteres	String hex exactamente igual de 64 caracteres	String diferente a 64 caracteres. Variable de tipo distinto a string	LS = LI = 64 caracteres
cancelation_type	Conjunto de valores {Temporal, Final}	Valores: Temporal o Final	Valores distintos a temporal y final. Variable de tipo distinto a string	Temporal o FInal
reason	String entre 2 y 100 caracteres	String mayor o igual a 2 caracteres. String menor o igual a 100 caracteres	Valores inferiores a 2 caracteres. Valores superiores a 100 caracteres. Variable de tipo distinto a string	LS = 100 LI = 2 caracteres

Los casos de prueba se podrán consultar en el archivo xlsx de la carpeta docs sección CE y Valores Límites.

GRAMÁTICA Y ÁRBOL DE DERIVACIÓN

En esta sección aplicaremos la técnica Pruebas Funcionales para nuestra función: cancel_appointment:
En ella tenemos 3 campos: date_signature, que acepta un string hexadecimal de 64 caracteres; cancelation_type, que acepta los valores Temporal o Final y reason, que acepta un string entre 2 y 100 caracteres.

Su forma esquemática sería la siguiente:

```
{
"date_signature": "<String having 64 hexadecimal characters>",
"cancelation_type": “Temporal | Final”,
"reason": "<String with 2 - 100 characters>",
}
```

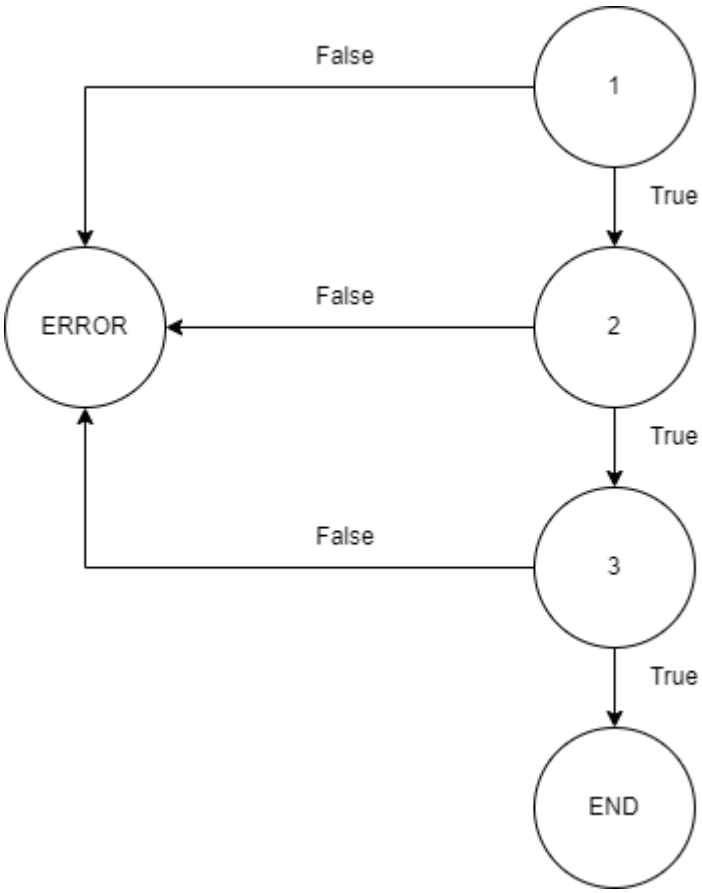
Si desarrollamos la gramática quedaría:

Fichero → CorcheteAbierto Datos CorcheteCerrado
CorcheteAbierto = {
CorcheteCerrado = }
Datos = Campo1 Separador Campo2 Separador Campo3
Separador = ,
Campo1 = Etiqueta1 Igualdad ValorDato1
Etiqueta1 = Comillas ValorEtiqueta1 Comillas
ValorEtiqueta1 = date_signature
ValorDato1 = Comillas Valor1 Comillas
Valor1 = a|b|c|e|f|0|1….|9| {64}
Campo2 = Etiqueta2 Igualdad ValorDato2
Etiqueta2 = Comillas ValorEtiqueta2 Comillas
ValorEtiqueta2 = cancelation_type
ValorDato2 = Comillas Valor2 Comillas
Valor2 = Temporal | Final
Campo3 = Etiqueta3 Igualdad ValorDato3
Etiqueta3 = Comillas ValorEtiqueta3 Comillas
ValorEtiqueta3 = reason
ValorDato3 = Comillas Valor3 Comillas
Valor3 = a|b|c…|z{2-100}
Comillas = “”
Igualdad = :

Finalmente, su forma esquemática es (archivo svg en carpeta docs): Árbol de derivación, y los Casos de Prueba en el archivo xlsx de la carpeta docs, sección Pruebas Funcionales.

DIAGRAMA DE FLUJO

En esta sección aplicaremos la técnica Pruebas Estructurales para la función cancel_appointment.
Para ello hemos de conocer cuál es el Diagrama de Flujo correspondiente, el cual está expuesto a continuación:



En el cual podemos observar que si se cumple un método llamado, pasamos al siguiente, y en caso contrario retornamos excepción.
Podemos observar que hay 5 Nodos y 6 aristas. Dada la fórmula: Caminos = A - N + 2 = 6 - 5 + 2 = 3 caminos, además del camino correcto; en total 4 caminos.
Camino 1 = 1-Error : Fallo en la llamada al primer método.
Camino 2 = 1-2-Error: Fallo en la llamada al segundo método.
Camino 3 = 1-2-3-Error: Fallo en la llamada al tercer método.
Camino 4 = 1-2-3-End: Funciona correctamente.

CONCLUSIONES

Una vez concluida la práctica, podemos decir que al tener que reutilizar todas las técnicas aprendidas, esto nos ha ayudado a fortalecer nuestros conocimientos sobre ellas, y por lo tanto seremos más eficaces a la hora de usarlas en un futuro próximo.
Con respecto al uso de técnicas, creemos que todas son igual de efectivas, sin embargo la técnica de Casos de Prueba y Valores Límites junto a las Pruebas Funcionales son las que más usemos, puesto que la Prueba Estructural nos parece más liosa que entendible, ya que hay que estar pensando los caminos, descomposiciones de la función: puesto que nosotros en un principio hemos hecho la refactorización directamente, sin definir primero todos los métodos dentro de una función, es por ello que nos ha costado construir el Diagrama de Flujo.

Además de dar valor positivo a algunas técnicas de Refactorización, como la creación de clases para los atributos, muy útil a la hora de utilizarlos en varios lugares, pero valor negativo al Patrón Singleton pues aún no entendemos en qué nos puede ayudar a mejorar el código.

En conclusión, creemos que ha sido una práctica muy útil para reforzar los conocimientos anteriormente adquiridos y, además, al haber tenido que editar un código ya construido, esto nos va a ayudar a trabajar en códigos ya hechos con el fin de mejorarlos y hacer que sean más entendibles.