Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de Computación
IIC3745 – Testing (II/2020)

IIC3745 - Testing (2020 / II)

Actividad 1 - Diseño de pruebas y cobertura de código

Diseño de pruebas

Los usuarios del juego *League of Legend* se han quejado que otras personas han logrado acceder a su cuenta y acusan a los dueños del juego por tener vulnerabilidades en su sistema de registro/login. Los desarrolladores, por otro lado, creen que el problema radica en que los usuarios utilizan contraseñas muy fáciles de adivinar. Por lo tanto, deciden agregar un validador de contraseña **`secure_password(password)`** que se encarga de verificar que la contraseña ingresada en el registro sea segura.

Una contraseña se considera segura si posee al menos 1 dígito (0 al 9), una letra mayúscula o minúscula (a-z;A-Z), tiene un largo mínimo de 8 caracteres y no posee 4 dígitos consecutivos, es decir, si hay 3 dígitos consecutivos, el siguiente carácter no puede ser otro dígito.

a. Diseñe casos de prueba con su resultado esperado para la función **`secure_password(password)`**. Es decir, escriba todos los casos de prueba que le serían útiles para implementar correctamente esta función. Para cada prueba debe explicitar sus parámetros de entrada y salida, además de una pequeña descripción de lo que se está probando.

Disclaimer: Este ejercicio no considera criterios de cobertura, sino que la calidad de sus pruebas. Por lo tanto, no se enfoque en diseñar casos de prueba para cumplir con cierto criterio de cobertura. Sólo piense en casos que permitan testear las diversas características de una contraseña segura.

Cobertura

Code coverage corresponde a una métrica de *testing* que indica el porcentaje de código que fue ejecutado dada una batería de *tests*. Este porcentaje está sujeto a diferentes criterios con los que se quiera medir. Para efectos de esta actividad solamente se tendrán en cuenta 3 criterios de cobertura:

- ***Statement Coverage***: Cantidad de líneas de código ejecutadas sobre la cantidad total de líneas de código.
- ***Branch Coverage***: Cantidad de ramas condicionales (e.g. if/else) ejecutadas sobre la cantidad total de ramas en el código.
- ***Condition Coverage***: Cantidad de condiciones booleanas evaluadas sobre la cantidad total de posibles valores que pueden tomar las condiciones booleanas. **Disclaimer:** Este tipo de cobertura se verá en más detalle en las próximas clases del curso. No obstante, un objetivo transversal de las actividades es fomentar la investigación, y esta es una buena oportunidad para contrastar distintos criterios de cobertura.

Parte A

Uno de sus ayudantes es fanático de comprar libros, por lo tanto dispone de un método para saber si puede o no comprar un libro en particular. Este método toma como *input* una lista de strings (`recommended`) que representa las sagas recomendadas y el dinero que dispone el ayudante (`balance`). Este método retorna `True` si es posible comprar el libro o `False` en caso contrario. A continuación se presenta el detalle de su implementación:

```ruby
class Book
  attr_reader :saga, :volume, :total_volume
  @@price = 5000
  def initialize(saga, volume, total_volumen)
    @saga = saga
    @volume = volume
    @total_volumen = total_volumen
  end

  def can_buy?(recommended, balance)
    buyable = false
    if balance >= @@price
      if recommended.include?(saga) || volume == total_volumen
        buyable = true
      else
        ## Get total price to complete the saga since this volume
        price_until_complete = total_volumen * @@price - (volume - 1) * @@price
        if volume == 1 && total_volumen >= 8
          buyable = price_until_complete / 2 < balance
        else
          buyable = price_until_complete < balance
        end
      end
    end
    return buyable
  end
end

LoR = Book.new("Señor de los anillos", 2, 3)
puts LoR.can_buy?(["Señor de los anillos", "El Hobbits"], 19000) #true
```

Analice el código y diseñe los casos de prueba **mínimos** para alcanzar un *code coverage* del 100% bajo los siguientes criterios:

- *Statement Coverage*
- *Branch Coverage*
- *Condition Coverage*

En su respuesta debe explicitar para cada criterio qué requisitos de prueba están asociados a qué casos de prueba, para así justificar que su batería de pruebas es mínima.

Parte B

Un curso del DCC ha decidido tener 2 tipos de evaluaciones: tareas y proyecto.. Además, en este curso se controla la asistencia de los alumnos a clases. En función de estos 3 indicadores se han decidido los siguientes criterios de aprobación:

1. Si ambas notas son >= 4, el alumno aprueba el curso.
2. Si la nota de tareas es >= 4.5, la nota de proyecto es >= 3.8 y su asistencia es >= 0.9: el alumno aprueba el curso.
3. En cualquier otro caso, el alumno reprueba el curso.

Se desea probar una función que indica si un alumno aprueba el curso. Para esto, se utilizarán los siguientes dos conjuntos de alumnos:

- Conjunto 1

Alumno	Tareas	Proyecto	Asistencia
1	5.0	5.0	0.8
2	3.0	3.0	0.2
3	5.0	3.9	0.5
4	6.0	7.0	0.9
5	4.5	3.8	1.0
6	5.0	3.9	1.0

- Conjunto 2

Alumno	Tareas	Proyecto	Asistencia
1	5.0	5.0	0.8
2	3.0	3.0	0.2
3	6.7	3.7	0.9
4	5.0	3.8	1.0
5	4.3	3.9	0.8

Con la ayuda de un pseudocódigo que represente esta subrutina y en base a su grafo de control de flujo correspondiente responda: ¿cuál de los 2 conjuntos de casos de prueba presenta mayor *condition coverage* y cuál presenta un mayor *branch coverage*? Justifique sus respuestas.

Preguntas conceptuales

Para cada uno de las siguientes preguntas **justifique** su respuesta en un máximo de 5 líneas:

1. ¿Diseñar pruebas de tipo *Black-box testing* garantiza la correcta ejecución de una funcionalidad?
2. ¿Qué variable determina que sea sensato desarrollar pruebas automatizadas? ¿Cómo se explica que en general no se realicen pruebas automatizadas en los trabajos universitarios?
3. ¿Si tuviese que escoger entre utilizar una librería con 50% de coverage y otra con 80%, cuál elegiría?
4. ¿Qué enfoque para sus tests sería de suma importancia si actualiza un servicio en producción con usuarios que lo consumen? ¿Qué niveles tradicionales de *testing* serían necesarios de incluir?