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

IIC3745 - Testing (2020 / II)

Actividad 1 - Solución

Diseño de pruebas

Se necesitan pruebas para validar los requisitos :

> Una contraseña se considera segura si posee al menos 1 dígito (0 al 9), una letra mayúscula y 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.

Test esperado:

1. Un test para validar que la contraseña no es segura sólo porque no tiene 1 dígito.
    a. *Input*: aaaaaaaA
    b. *Output*: False

2. Un test para validar que la contraseña no es segura sólo porque no tiene una letra mayúscula.
    a. *Input*: aaaaaaa1
    b. *Output*: False

3. Un test para validar que la contraseña no es segura sólo porque no tiene una letra minúscula.
    a. *Input*: AAAAAAA1
    b. *Output*: False

4. Un test para validar que la contraseña no es segura porque no cumple el largo mínimo.
    a. *Input*: aA419
    b. *Output*: False

5. Un test para validar que la contraseña no es segura porque tiene 4 dígitos consecutivos
    a. *Input*: wW2020Ww
    b. *Output*: False

6. Un test para validar que si cumple todo, debe ser contraseña segura
    a. *Input*: Aa123Aa1
    b. *Output*: True

Esta solución es la más relajada posible. Es posible determinar más test si empiezan a considerar, por ejemplo, casos bordes como: Si hay 2 números consecutivos, una letra y luego otros 2 números ¿La función tendrá un error que considera eso como 4 números consecutivos? Si bien no se esperaba ese tipo de análisis, en caso de hacerlo no se aplica descuento. Lo importante es que cada test esté justificado y cada uno pruebe algo en específico.

Cobertura

Parte A

Los test esperados para ***statement coverage*** son:

1. Tiene dinero suficiente y [es un libro recomendado o es último volumen] (**sólo 1 test donde el OR sea True**). En particular, haremos que es un libro recomendado.
```
book = Book.new("Señor de los anillos", 2, 3)
puts book.can_buy?(["Señor de los anillos", "El Hobbits"], 5000) #true
```

2. Tiene dinero suficiente, no es recomendado, es el volumen 1 y tiene más de 7 volúmenes totales. No se debe hacer un test para que retorne True y otro con False, **sólo 1 test** sin importar la respuesta.
```
book = Book.new("Naruto", 1, 73)
puts book.can_buy?(["Señor de los anillos", "El Hobbits"], 5000) #false
```

3. Tiene dinero suficiente, no es recomendado y [NO es el volumen 1 o tiene menos de 8 volúmenes totales] (**1 sólo test donde el OR sea True)**. En particular, tomaremos que es un volumen 2 y si tiene más de 7 volúmenes.
```
book = Book.new("Naruto", 2, 73)
puts book.can_buy?(["Señor de los anillos", "El Hobbits"], 5000) #false
```

Para 100% de ***branch coverage***, usar los test anteriores y agregar:

1. No tiene suficiente dinero para un libro.
```
book = Book.new("Señor de los anillos", 2, 3)
puts book.can_buy?(["Señor de los anillos", "El Hobbits"], 100) #false
```

Para 100% de ***condition coverage*****,** usar todos los test anteriores y agregar:

1. Tiene dinero suficiente , no es un libro recomendado y si es el último volumen.

```
book = Book.new("Naruto", 73, 73)
puts book.can_buy?(["Señor de los anillos", "El Hobbits"], 5000) #true
```

2. Tiene dinero suficiente, no es recomendado, es el volumen 1 y no tiene más de 7 volúmenes.

```
book = Book.new("Crepúsculo", 1, 6)
puts book.can_buy?(["Señor de los anillos", "El Hobbits"], 50000) #true
```

Parte B

En función del texto indicado, el pseudocódigo puede ser

```
Approve = false
if Tareas >= 4 && Proyecto >= 4:
    Approve = true
if Tareas >= 4.5 && Proyecto >= 3.8 && Asistencia >= 0.9:
    Approve = true
return Approve
```

En base a**l pseudocódigo**, notamos que ambos conjuntos tienen casos donde ambos IFs evalúan *True* y *False*. Dado esto, en ***branch coverage*** ambos tienen **100%**. En relación a ***condition coverage***, podemos definir las siguientes cláusulas:

1. `A = Tareas >= 4`
2. `B = Proyecto >= 4`
3. `C = Tareas >= 4.5`
4. `D = Proyecto >= 3.8`
5. `E = Asistencia >= 0.9`

Dado los predicados y tomando en cuenta que en un **A AND B** no se evalúa B en caso que A sea *False*, podemos analizar cada conjunto:

Conjunto 1

Test	Resultado A	Resultado B	Resultado C	**Resultado D**	Resultado F
1	True	True	True	**True**	False
2	False	No se evalúa	False	**No se evalúa**	No se evalúa

3	True	False	True	**True**	False
4	True	True	True	**True**	True
5	True	False	True	**True**	True
6	True	False	True	**True**	True

Conjunto 2

Test	Resultado A	Resultado B	Resultado C	**Resultado D**	Resultado F
1	True	True	True	**True**	False
2	False	No se evalúa	False	**No se evalúa**	No se evalúa
3	True	False	True	**False**	No se evalúa
4	True	False	True	**True**	True
5	True	False	False	**No se evalúa**	No se evalúa

Podemos ver que el predicado D sólo se evalúa como False en el conjunto 2, lo cual le da mayor ***condition coverage.***

Por otro lado, el pseudocódigo también podría verse de la forma:

```
Approve = false
if Tareas < 4:
    Approve = false
else
    if Proyecto >= 4:
        Approve = true
    else
        if Tareas >= 4.5 && Proyecto >= 3.8 && Asistencia >= 0.9
            Approve = true
return Approve
```

Otra opción podría ser:

```
Approve = (Tareas >= 4 && Proyecto >= 4) || (Tareas >= 4.5 && Proyecto >=
3.8 && Asistencia >= 0.9)
return Approve
```

Por lo tanto, lo importante es que **el grafo sea consistente con el pseudocódigo** y la respuesta esté fundamentada en dicho grafo. Incluso, pueden notar que si le cambiamos el orden los predicados en los IFs y seguimos tomando en cuenta que en un **A AND B** no se evalúa B en caso que A sea *False*, podemos analizar cada conjunto de la siguiente forma:

```
Approve = false
if Proyecto >= 4 && Tareas >= 4:
    Approve = true
if Proyecto >= 3.8 && Tareas >= 4.5 && Asistencia >= 0.9:
    Approve = true
return Approve
```

Conjunto 1

Test	Resultado B	Resultado A	Resultado D	Resultado C	Resultado F
1	True	True	True	True	False
2	False	No se evalúa	False	No se evalúa	No se evalúa
3	False	No se evalúa	True	True	False
4	True	True	True	True	True
5	False	No se evalúa	True	True	True
6	False	No se evalúa	True	True	True

Conjunto 2

Test	Resultado B	Resultado A	Resultado D	Resultado C	Resultado F
1	True	True	True	True	False
2	False	No se evalúa	False	No se evalúa	No se evalúa
3	False	No se evalúa	False	No se evalúa	No se evalúa

4	False	No se evalúa	True	True	True
5	False	No se evalúa	True	False	No se evalúa

En este caso, al conjunto 1 le falta revisar el caso de *False* en el predicado A y C, mientras que el conjunto 2 sólo le falta revisar el caso de *False* en el predicado A, lo que concluye que el conjunto 2 tiene mayor ***condition coverage***. Nuevamente, este resultado fue gracias al pseudocódigo generado, puede variar y por eso es vital justificar esta pregunta con el pseudocódigo y el grafo.

Preguntas conceptuales

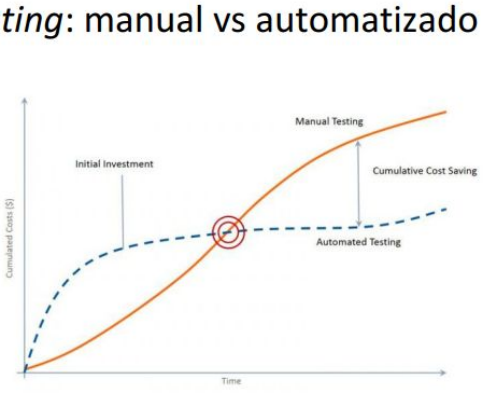
Para cada uno de las siguientes preguntas **justifique** su respuesta en un máximo de 5 líneas:

1. ¿Diseñar pruebas de tipo *Black-box testing* garantiza la correcta ejecución de una funcionalidad?

   Depende: una prueba puede **pasar por suerte**, y por ello, **no garantiza la correcta** ejecución de la funcionalidad, o bien **podría garantizar la correcta** ejecución si se trata de una **funcionalidad sencilla** y/o hay control sobre los **casos bordes**.

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?

   Tal como se ve en la imagen sacada de la clase 2, la variable que determina que es sensato desarrollar pruebas automatizadas es la madurez del proyecto.



En general los trabajos universitarios **no logran llegar al punto de intersección** que se ve en la imagen, donde los costos de realizar pruebas de manera manual pasan a ser mayores que el desarrollo de pruebas automatizadas debido la cantidad de código desarrollado.

3. ¿Si tuviese que escoger entre utilizar una librería con 50% de coverage y otra con 80%, cuál elegiría?

   El porcentaje de cobertura no es suficiente para determinar cuál de las dos librerías conviene, ya que el porcentaje de cobertura en sí mismo no determina necesariamente una mejor o peor calidad de código. Por ello, la respuesta es que para poder elegir **se necesita buscar más información** como: criterio de cobertura, cantidad de *commits*/colaboradores, tiempo desde último *commit*, entre otros.

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?

   Es de suma importancia tener un enfoque de **pruebas de regresión** porque debe existir control sobre las funcionalidades que se exponen al momento de actualizar el servicio. Los niveles tradicionales necesarios podrían ser:

   - **Pruebas de integración** para tener certeza de que la actualización se incorpore de manera correcta al resto del sistema.
   - **Pruebas de sistema** para comprobar que la aplicación aún cumple con el mismo nivel de servicio.