IIC 2143 Ingeniería de Software
Examen - Semestre 2 /2021
Profesores M. Peralta y J. Navón

1. (30 pts) A continuación, encontrarás una lista de 30 afirmaciones que no son verdaderas. Cada una de ella tiene algún problema que la invalida. Tu misión es encontrar el principal problema y explicarlo en un máximo de dos líneas (idealmente una sola línea, mientras mas corto mejor). No incluyas la afirmación en tu respuesta.

1. Por lo general la forma más usada de ajustar cuando el equipo se ha dado cuenta antes del fin de desarrollo que todo indica que ni podremos cumplir es un ajuste por alcance

Esa es la forma mas deseable pero la mas utilizada es un ajuste por calidad (muchas veces en forma implícita)

2. El elemento fundamental que incorpora Scrum para que el producto final no se desvíe demasiado de lo deseado es la reunión de retrospectiva (sprint retrospective)

Es el sprint review y no el sprint retrospective la reunión orientada a este fin

3. Los niveles de planeación que ustedes como alumnos pudieron experimentar son los de iteración y producto

Se trata de los niveles de iteración y release (un producto incorpora varios releases)

4. En Kanban se limita el número de items por hacer en cada iteración a un número prefijado que puede ser ajustado

Lo que se limita no es el número de items en la iteración sino el número de items en una actividad

5. Al trabajar con Kanban el "lead time" corresponde al tiempo que está tomado desde que comienza a trabajarse en una tarea hasta que está lista para ser deployada

Eso corresponde al cycle time. El lead time corresponde al tiempo desde que la tarea entra al backlog hasta que es deployado

6. En la descripción de un caso de uso un flujo alternativo corresponde a una situación en que no se llegaría a cumplir con el objetivo del actor primario

No necesariamente ya que los flujos alternativos pueden tener un fin exitoso, solo son situaciones que se desvían del happy path

7. Cuando el proyecto definitivamente va a terminar costando mucho más que lo planeado (por lo general por tomar mas tiempo) se habla de una deuda técnica

Nada que ver con plata. Se trata de asumir que estamos ganando algún tiempo hoy (diseño simple, recorte en pruebas) que tendremos que pagar mañana con intereses

8. La ventaja de los story points sobre las líneas de código como métrica de esfuerzo es que puede ser estimada en forma temprana

No los story points ni las líneas de código son métricas de esfuerzo sino de tamaño

9. Podemos confiar en una pieza de software que ha pasado por testing con cobertura superior al 80%

Ni siquiera con un 100% de cobertura se puede confiar ya que ello solo dice que se ejecutó todo el código en los tests (puede haber otros inputs que lo hagan fallar)

10. Una de las maneras mas efectivas de encontrar bugs sin tener que hacer tests es el análisis estático del código con herramientas automatizadas

El análisis estático es bastante limitado en cuanto a encontrar bugs, es la revisión formal (hecha por personas) la técnica que resulta muy efectiva en esto

11. Lo que va a influir mayormente en el número de casos de prueba a utilizar es el rango de inputs que son posibles

No. El rango del input suele ser muy grande o infinito casi siempre. Lo que influye es el número de clases de equivalencia que pueden definirse respecto al input

12. El test de regresión y el test de integración son la misma cosa cuando se está usando integración contínua

El test de regresión puede hacerse cada vez que se hacen cambios importantes, el de integración solo en el momento de construir el build

13. El smoke test y el sanity test estan destinados a aumentar la confiabilidad del software que estamos desarrollando

No. Ambos tests rápidos están orientados a ahorrarse los costosos tests de regresión cuando se hacen cambios

14. El test de carga consiste en someter al sistema a una carga varias veces mayor a la planificada en cuanto a número de usuarios o transacciones simultáneas

Eso corresponde al test de esfuerzo. En el test de carga el sistema se somete a la carga máxima estipulada para evaluar la respuesta

15. Un equipo de desarrollo de gente brillante en que hay una alta competitividad entre ellos por lo general exhibirá características de alto rendimiento

Probablemente no. La competitividad interna atenta contra la complementariedad y la confianza entre los miembros del equipo

16. No hay un número ideal de personas para un equipo de desarrollo ágil ya que dependiendo el proyecto puede variar enormemente

El número ideal de personas en el equipo va entre 5 y 8. Si el proyecto es grande se necesita un esquema de cooperación entre equipos al estilo de Spotify que vimos en clases

17. Es posible constituir un equipo de alto rendimiento si elegimos a las mejores personas que están disponibles en el momento

Los equipos de alto rendimiento pasan por 4 fases de crecimiento y si se arman equipos nuevos en cada oportunidad no se da la oportunidad de que maduren

18. Tanto sobreestimar como subestimar el esfuerzo o el tiempo de desarrollo son problemas igualmente graves

A pesar que ambas cosas son malas, subestimar es muchísimo peor en cuanto al impacto en el desarrollo del proyecto

19. El punto de partida al hacer estimaciones para planificar un proyecto de software es estimar el tiempo de desarrollo

El punto de partida siempre es el tamaño. Con el tamaño se puede estimar esfuerzo y tiempo que es lo que realmente interesa

20. Siempre un diseño con nulos acoplamiento y cohesión será preferible a uno con acoplamiento y cohesión no nulos

Un diseño con nula cohesión siempre será muy malo, probablemente peor que cualquier esquema alternativo

21. La manera que tiene Ruby de asegurar que exista una sola instancia de un objeto es implementar una fábrica abstracta

Nada que ver. La manera es declarar privado el método de clase new

22. El patrón comando permite que los objetos subscritos no tengan que estar permanentemente haciendo polling para averiguar si es necesario actualizar

Eso corresponde al patrón observador

23. El patrón fábrica abstracta se parece mucho al patrón método plantilla

El patrón fábrica abstracta se parece al patrón estrategia y no al método plantilla

24. No es fácil inventar un nuevo patrón de diseño

Los patrones de diseño no se inventan, se descubren

25. En el modelo C4 de Brown el nivel 4 corresponde al código fuente mismo

El nivel 4 se llama nivel de código, pero no veremos código sino diagramas UML que representan al código (de clases principalmente)

26. Es posible tener una arquitectura de varios tiers que contienen layers y tambien una de varios layers que contienen tiers

Que un tier se implemente en layer tiene sentido, el inverso no lo tiene

27. La principal diferencia entre la arquitectura de servicios conocida como SOA con la de microservicios es el tamaño de los servicios

La principal diferencia es que no se requiere de un middleware que actúe como un bus de servicios

28. La arquitectura de microservicios tiende a producir soluciones de mejor desempeño que las monolíticas

Por lo general no es así porque hay mucho costo de comunicación (http) entre estas componentes (servicios) débilmente acopladas

29. Se espera que a lo menos un 40% del tiempo de desarrollo total sea dedicado a producir una muy buena documentación

Mucho menos. No debería ser más de un 10% o 20% a lo sumo. Lo más importante sigue siendo el código.

30. El código no es parte de la documentación del producto

El código no puede ser TODA la documentación pero indudablemente es parte importante de ella y por eso se pone énfasis en escribir código fácil de entender.

Notas de Pauta

- Cada respuesta vale 1 punto.
- No asignar medios puntos, solo 0 o 1 punto
- Se otorga el punto si la justificación coincide en esencia (aunque no en palabras) con la de la pauta

2. (15 pts) Un par de alumnos armaron su propia Pyme para vender sus productos (distintos tipos de salsas). Los clientes están caracterizados por su nombre, RUT, email y tipo (grande, mediano, pequeño). Según el tipo de cliente se les hace un 5%, 2% y 0% de descuento en sus ordenes respectivamente. Las ordenes de compra son colocadas por los clientes y pueden incluir una serie de items (líneas de la orden), cada uno de los cuales hace referencia a un producto y a la cantidad pedida de el. Una orden ser vería como algo así:

Orden Nº 12345
Cliente: Jorge Perez
RUT: 8.876.897-3

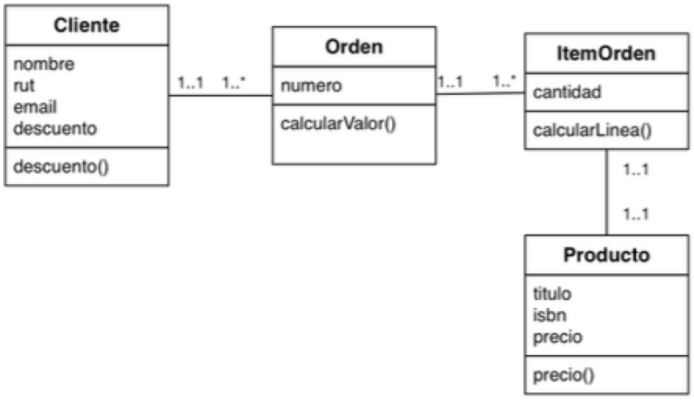
Codigo	Descripción	Cantidad	Precio	Monto
2342	salsa especial	2	1200	2400
1321	salsa de coco	1	2000	2000

Para cada producto se maneja un código, una descripción y un precio que es el mismo en cualquier orden en que aparezca. Al momento de emitir la factura asociada a una orden dada (por ejemplo la orden 12345 anterior) se debe calcular el monto total. Este corresponde a la suma de los montos de cada línea de la orden menos un posible descuento que le corresponde al cliente según su tipo y mas el IVA (19%). Al momento de hacer este cálculo los montos de cada línea deben calcularse usando los precios de cada producto (podría haber un error en la orden o un cambio de precio del producto).

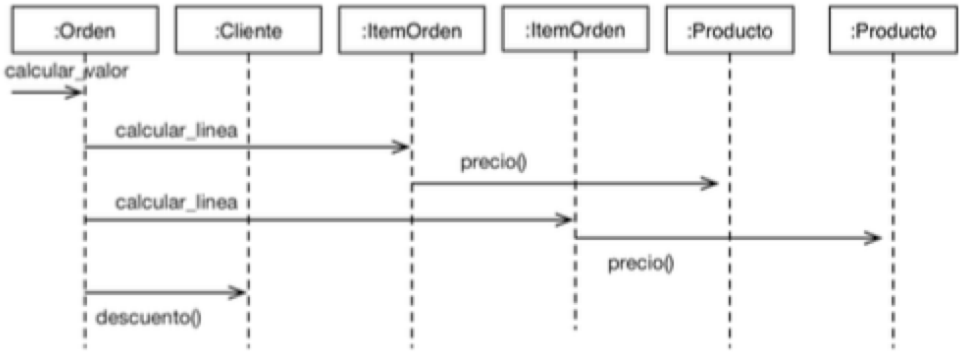
a) (5 pts) Haz un modelo de clases para lo descrito mas arriba (incluya cardinalidades, atributos y métodos). Incluye en la clase Orden un método calcular_valor que hace el cálculo del monto total según se ha descrito antes.
b) (10 pts) Haz un diagrama de secuencia que muestre lo que sucede cuando un objeto de esa clase recibe un mensaje calcular_valor (puedes suponer para este diagrama que la orden tiene solo las 2 líneas del ejemplo)

Solución

a)



b)



Notas de Pauta

- En la parte a) podría haberse usado una asociación de composición entre Order e ItemOrder (rombo lleno) y de agregación entre ItemOrder y Producto (rombo vacío). Un rombo lleno en este último es un error. Entre Cliente y Orden no debe haber rombo. Es importante que aparezcan correctamente las cardinalidades de la asociación (1..1, etc)
- En la parte b) debe aparecer de alguna manera que la orden envía un mensaje por cada item (Debe aparecer el mensaje de la orden al cliente para obtener el descuento)
- Los mensajes de la parte b) deben corresponder con los nombres de los métodos de las clases de la parte a)

3. (20 pts) Desde hace algún tiempo las clínicas y centros de salud han comenzado a proporcionar un servicio de teleconsulta médica. En este esquema, el paciente puede conversar con el médico a través de una sesión de teleconferencia. El paciente puede agendar una hora con un médico en particular o con un médico de la especialidad que el desee consultar. Una vez agendada la hora el paciente debe hacer el pago correspondiente a mas tardar 24 horas antes de la hora médica agendada. Una vez pagado, la clínica envía al paciente un link que le permite ingresar a la teleconsulta con el médico a la hora agendada y conversar con él. Una vez terminada la sesión con el médico el paciente recibe las prescripciones médicas por medio de un nuevo correo electrónico.
Identifica las principales épicas y los principales relatos dentro de cada una de las épicas. Los relatos deben ser descritos en la forma estándar pero no es necesario incluir condiciones de satisfacción. Ojo que revisaremos con cuidado la técnica y no solo la completitud de la captura de requisitos.

Solución

Epica: Agendar una hora con un médico
Relatos:

Revisar médicos disponibles para la especialidad
Yo como paciente necesito ver qué médicos hay en dicha clínica que sean especialistas en mi problema para poder seleccionar quien me parece la mejor opción

Revisar los detalles de un médico específico
Yo como paciente que ya tengo el nombre de un médico especialista en mi problema necesito saber mas detalles sobre el de modo de poder decidir en forma mas segura

Agendar una hora con el médico deseado
Yo como paciente que ya he decidido el médico que me interesa necesito poder revisar las horas que éste tiene disponibles para agendar una teleconsulta el día y hora que me es más cómodo

Epica: Pagar la hora agendada
Relatos:

Ingresar a la plataforma de pagos
Yo como paciente necesito ingresar a la plataforma de pagos de la clínica para pagar la consulta agendada para poder recibir el link de atención.

Realizar el pago
Yo como paciente necesito efectuar el pago de la teleconsulta mediante una tarjeta de débito o de crédito

Epica: Llevar a cabo la teleconsulta
Relatos:

Iniciar a la sesión con el médico
Yo como paciente necesito ingresar a la sesión de video conferencia a través de link enviado por la clínica de modo de establecer la comunicación con el médico

Escribir los detalles de prescripción y tratamiento
Yo como médico necesito completar la ficha con los detalles de las prescripciones y el tratamiento de modo que sea enviado por correo al paciente al cierre de la sesión

Cerrar la sesión
Yo como médico necesito cerrar la sesión con el paciente de modo que las prescripciones sean enviadas a el y yo pueda atender a otro paciente.

Notas de Pauta

I. Correcto uso de la técnica

- Observar que no exista confusión entre épica y relato
- Observar que cada épica y cada relato tenga un nombre que comience con un verbo
- Observar que cada relato incluya las 3 partes (yo como ... necesito ... para ...)
- Las épicas llevan solo título y no una descripción, los relatos llevan título y descripción

II. Completitud en captura de requisitos funcionales

- Independientemente de la forma de agrupar relatos, deben haber relatos que permitan
    - agendar una hora
    - pagar por la consulta
    - efectuar la consulta (perspectiva del médico y del paciente)
- Deberían haber encontrado a lo menos 6 relatos (8 es mejor)
- No es necesario que incluyan relatos que tengan que ver con los administradores de la clínica

4. (15 pts) Acorde a lo visto en clases sobre potenciales errores en el código, identifique el o los code smells asociados y los problemas de cada uno de estos códigos. Además, explique cómo se podrían mejorar.

a) (5 pts)

```
class Calc
 def initialize
   @result = 0
 end
 def add(number)
   @result += number
 end
 def subtract(number)
   @result -= number
 end
 def multiply(number)
   @result = @result * number
 end
 def divide(number)
   unless number == 0
       @result = @result / number
   else
       @result = "Can't divide by 0"
   end
 end
 def result
   puts @result
 end
 def reset
   @result = 0
 end
end

nmbr = Calc.new
nmbr.add(1000)
nmbr.divide(5)
nmbr.subtract(105)
nmbr.multiply(15)
nmbr.result
nmbr.reset
nmbr.add(150)
nmbr.subtract(24)
nmbr.divide(2)
nmbr.result
```

b) (10 pts)

```
class Player
 def initialize(health, attack, speed)
   @health = health
   @attack = attack
   @speed = speed
 end
  def set_attack(attack)
   @attack = attack
 end
  def set_speed(speed)
   @speed = speed
 end
  def set_health(health)
   @health = health
 end
  def get_attack
   return @attack
 end
  def get_speed
   return @speed
```

```
 end
  def get_health
   return @health
 end
end

class RoleGame
 def initialize(player1, player2)
   @player1 = player1
   @player2 = player2
   @winner = "unknown"
 end
  def battle
   while @player1.get_health > 0 and @player2.get_health > 0
     rand1 = rand
     rand2 = rand
     crit1 = rand1 > 0.3
     crit2 = rand2 > 0.3
     if @player1.get_speed > @player2.get_speed
       health2 = @player2.get_health
       att1 = @player1.get_attack
       if crit1
         att1 = att1 * 2
       end
       health2 -= att1
       @player2.set_health(health2)
       if @player2.get_health <= 0
         @winner = "Gano el jugador 1"
         break
       else
         health1 = @player1.get_health
         att2 = @player2.get_attack
         if crit2
           att2 = att2 * 2
         end
         health1 -= att2
         @player1.set_health(health1)
         if @player1.get_health <= 0
           @winner = "Gano el jugador 2"
           break
         end
       end
     else
       health1 = @player1.get_health
       att2 = @player2.get_attack
       if crit2
         att2 = att2 * 2
       end
       health1 -= att2
       @player1.set_health(health1)
       if @player1.get_health <= 0
         @winner = "Gano el jugador 2"
         break
       else
         health2 = @player2.get_health
         att1 = @player1.get_attack
         if crit1
           att1 = att1 * 2
         end
         health2 -= att1
         @player2.set_health(health2)
         if @player2.get_health <= 0
           @winner = "Gano el jugador 1"
           break
         end
```

```
      end
    end
  end
 end
 def show_winner
   puts @winner
 end
end
p1 = Player.new(15, 3, 5)
p2 = Player.new(12, 5, 3)
role = RoleGame.new(p1 ,p2)
role.battle
role.show_winner
```

Solución

a) El código implementa una clase que es una calculadora básica para hacer operaciones matemáticas. Sin embargo éstas se pueden realizar con los operadores matemáticos propios del lenguaje. Además si se quiere calcular un nuevo valor hay que hacer un reset del valor actual de la calculadora, sin tener la posibilidad de iniciar con un valor distinto a 0.

Code smell: Lazy class y Large class. Esta en una clase innecesaria, solo agrega complicaciones para una operación sencilla del lenguaje. Es muy difícil de extender, no escala bien y no es una solución flexible. Para mejorarlo es mejor eliminar la clase y apoyarse en las operaciones existentes del lenguaje

- 3 ptos: Por reconocer alguno de los dos code smells y los problemas.
- 2 ptos: Por dar la solución de eliminar la clase

b) El código implementa un juego de rol. En este caso hay una clase Player y una clase RoleGame y el juego consiste en una batalla simple donde cada jugador tiene estadísticas de velocidad ataque y vida. Puede ocurrir un golpe crítico con 30% de probabilidad que lo que hace es duplicar el daño de un jugador. La pelea continúa hasta que un jugador llegue a tener vida menor que 0.

Code smell: Data Class, Long method y código duplicado. La clase player lo único que hace es tener métodos getter y setter para los atributos iniciales del jugador por lo que la convierte en una Data Class. Por otro lado, el método battle es muy largo y tiene código repetido. Cada parte del if else grande vuelve a repetir la lógica. Hay muchas posibles soluciones. Podemos deshacernos de la clase Data Class y usar variables para guardar los datos, mientras que debemos extraer lógica repetida del método battle y colocarlo en otro método. Así, battle define qué jugador será el que recibe el golpe, y llama a un solo método con toda la lógica repetida. Finalmente battle también verifica cuando la batalla termina. Otra solución es aprovechar la clase Player para mover toda la lógica de recibir un golpe y actualizar su estado. Así el método battle puede ser más pequeño y sin repetición de código. Cualquiera de las dos soluciones está bien

- 5 ptos: Por reconocer los tres code smells y los problemas.
- 5 ptos: Por dar una de las dos soluciones.

5. (20 pts) La PUC se encuentra estudiando la puesta en marcha de un nuevo sistema computacional de asignación de becas que pueda conectarse con los registros de admisión y asignar becas a los estudiantes según distintos tipos de criterios. La base de datos de admisión define objetos del tipo *Student*, donde cada estudiante tiene los siguientes atributos:

- Name (string)
- FamilyIncome (integer)
- PsuAverage (float)

Se te pide desarrollar la clase *ScholarshipAssigner*, que recibe en su constructor el atributo:

- *student_list (array)*: listado de estudiantes entregado por la base de datos de admisión

Además, esta clase define el método *assign_scholarships(number_of_scholarships, strategy) que* imprime el nombre de hasta *number_of_scholarships* alumnos del listado de estudiantes que más ameriten una beca según un determinado criterio definido según el patrón *Strategy*

Admisión se encuentra interesado en considerar los siguientes criterios para la asignación de becas:

- Mejor puntaje PSU: se le asignarán becas a los alumnos con mejor promedio PSU
- Mayor necesidad económica: se le asignarán becas a los alumnos con menor ingreso familiar
- Híbrido: se le asignarán becas a los alumnos con menor ingreso familiar siempre y cuando hayan obtenido igual o más de 700 puntos en la PSU. Si hay más becas que alumnos que cumplan la restricción, no se asignan las becas sobrantes.

Implementa la clase *ScholarshipAssigner* junto con todas las demás clases que estimes necesarias para implementar el patrón *Strategy en una buena solución de este problema*. Puede asumir que la clase *Student* viene dada por lo que puedes instanciarla pero no puedes implementarla.

Recuerda que en Ruby existe el método *sort* definido en la clase *Array*, el cual permite retornar un listado de elementos ordenados según el resultado entregado por un bloque que se le puede pasar como parámetro. Este bloque recibe dos parámetros a y b de input y retorna un número negativo si b viene después de a, un número positivo si a viene después de b, y retorna 0 si a es igual a b.

Puedes asumir también que: *number_of_scholarships < student_list.length.*

Solución

Hay varias formas de implementar el código solicitado. En esta solución propuesta se procede a ordenar los arreglos que contienen los estudiantes según el criterio correspondiente y luego imprimir los N primeros nombres, donde N es igual al *number_of_scholarships*. Para el criterio de mejore puntaje PSU, el orden de la comparación está invertido para que el ordenamiento sea de mayor o menor. Otras alternativas son ordenar de menor a mayor y luego revertir con método reverse, o bien iterar desde atrás hacia adelante.

```
class ScholarshipAssigner
  def initialize(student_list)
    @student_list = student_list
  end

  def assign_scholarships(number_of_scholarships, strategy)
    strategy.assign_scholarships(@student_list, number_of_scholarships)
  end
end

class BestPsuScoreStrategy
  def assign_scholarships(student_list, number_of_scholarships)
    list = student_list.sort { |a, b| b.psu - a.psu }
    (0...number_of_scholarships).each { |i| puts(list[i].name) }
  end
end

class LessIncomeStrategy
  def assign_scholarships(student_list, number_of_scholarships)
    list = student_list.sort { |a, b| a.income - b.income }
    (0...number_of_scholarships).each { |i| puts(list[i].name) }
  end
end

class HybridStrategy
  def assign_scholarships(student_list, number_of_scholarships)
    list = student_list.sort { |a, b| a.income - b.income }
    count = 0
    list.each do |student|
      if (count < number_of_scholarships) && (student.psu >= 700)
        puts(student.name)
        count += 1
      end
    end
  end
end
```

Notas de Pauta

- Debe estar escrito en Ruby. No se asignará puntaje a código Python o cualquier otro
- Si la solución no incluye el patrón estrategia el puntaje será cero
- Es importante que se escriban las 3 clases que implementan cada una de las 3 estrategias. Es posible también definir una clase abstracta estrategia de la cual se derivan las otras 3 como subclases
- Es crucial que el método assign_scolarships reciba como parámetro un objeto con la estrategia