Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

9 de noviembre de 2018

IIC2113 – Diseño Detallado de Software

Interrogación 2

Instrucciones:

- Sea preciso: no es necesario escribir extensamente pero sí ser preciso.
- En caso de ambigüedad, utilice su criterio y explicite los supuestos que considere convenientes.
- Responda y entregue cada pregunta en hojas separadas. Si no responde una pregunta debe entregar de todas formas la hoja correspondiente a la pregunta.
- Indique su nombre en cada hoja de respuesta.
- Esta interrogación fue diseñada para durar 120 minutos.

1. (0.8 pts) Relacione los nombres de los siguientes patrones *GoF* con su descripción correspondiente (utilice la letra que lista cada patrón):

a. Visitor
b. Template Method
c. Strategy
d. Interpreter
e. Mediator
f. State
g. Chain of Responsability

__ Encapsula la implementación de un algoritmo en una clase
__ Dado un contexto, especifica cómo evaluar expresiones
__ Captura y restaura el estado interno de un objeto
__ Agrega comportamiento a un objeto de manera dinámica
__ Altera el comportamiento de un objeto cuando cambia su estado
__ Delega la implementación de los pasos de un algoritmo a subclases
__ Define una comunicación simplificada entre clases
__ Encapsula una acción como un objeto
__ Define una nueva operación para una clase sin modificarla
__ Captura y restaura el estado interno de un objeto
__ Accede de forma secuencial los elementos de una colección
__ Genera una jerarquía dinámica de objetos capaces de procesar una solicitud

2. (0.8 pts) Considere el siguiente artículo sobre GitLab escrito por Aricka Flowers:

Why we use Ruby on Rails to build GitLab

When our Co-founder and Engineering Fellow Dmitriy Zaporozhets decided to build GitLab, he chose to do it with Ruby on Rails, despite working primarily in PHP at the time. GitHub, a source of inspiration for GitLab, was also based on Rails, making it a logical pick considering his interest in the framework. GitLab CEO Sid Sijbrandij thinks his co-founder made a good choice:

"It's worked out really well because the Ruby on Rails ecosystem allows you to shape a lot of functionality at a high quality," he explained. "If you look at GitLab, it has an enormous amount of functionality. Software development is very complex and to help with that, we need a lot of functionality and Ruby on Rails is a way to do it. Because there's all these best practices that are on your happy path, it's also a way to keep the code consistent when you ship something like GitLab. You're kind of guided into doing the right thing."

Depending on useful gems

Ruby gems play an integral role in the building of GitLab, with it loading more than a thousand non-unique gems, according to Sid. Calling the Ruby on Rails framework "very opinionated," he thinks it's a strong environment in which to build a complex app like GitLab.

"There's a great ecosystem around it with gems that can make assumptions about how you're doing things and in that regard, I think the Ruby on Rails ecosystem is still without par," he says. "If you look at our Gemfile, it gives you an indication of how big the tower is of dependencies that we can build on. Ruby on Rails has amazing shoulders to stand on and it would have been much slower to develop GitLab in any other framework."

Overcoming challenges

All of this is not to say there haven't been challenges in building GitLab with Ruby on Rails. Performance has been an issue that our developers have made strides to improve in a number of ways, including rewriting code in Go and using the Vue framework. The latter is being used to rewrite frequently accessed pages, like issues and merge requests, so they load faster, improving user experience.

Go is being used to address other issues affecting load times and reduce memory usage.

"Ruby was optimized for the developer, not for running it in production," says Sid. "For the things that get hit a lot and have to be very performant or that, for example, have to wait very long on a system IO, we rewrite those in Go … We are

still trying to make GitLab use less memory. So, we'll need to enable multithreading. When we developed GitLab that was not common in the Ruby on Rails ecosystem. Now it's more common, but because we now have so much code and so many dependencies, it's going to be a longer path for us to get there. That should help; it won't make it blazingly fast, but at least it will use less memory."

Adding Go to GitLab's toolbox led to the creation of a separate service called Gitaly, which handles all Git requests.

Building on GitLab's mission

The organized, structured style of Ruby on Rails' framework falls in line with our core mission. Because Rails is streamlined, anyone can jump into GitLab and participate, which made it especially attractive to Sid from the start.

"Our mission is that everyone can contribute," he explains. "Because Ruby on Rails is really opinionated about which pieces go where, it's much easier for new developers to get into the codebase, because you know where people have put stuff. For example, in every kitchen you enter, you never know where the knives and plates are located. But with Ruby on Rails, you enter the kitchen and it's always in the same place, and we want to stick to that.

"I was really encouraged when I opened the project and saw it for the first time a year after Dmitriy started it. I opened it up and it's idiomatic Rails. He followed all the principles. He didn't try to experiment with some kind of fad that he was interested in. He made it into a production application. Dmitriy carefully vetted all the contributions to make sure they stick to those conventions, and that's still the case. I think we have a very nice codebase that allows other people to build on top of it. One of our sub-values is boring solutions: don't do anything fancy. This is so that others can build on top it. I think we've done that really well … and we're really thankful that Ruby has been such a stable, ecosystem for us to build on."

a) En base a la lectura, comente ventajas y desventajas al utilizar un *framework* en un proyecto de *software*.

b) Comente en base a conceptos vistos en el curso la siguiente frase: "Ruby was optimized for the developer, not for running it in production."

3. (1.0 pts) Para cada uno de los siguientes *code smells* indique a qué familia pertenecen, explique a qué se refiere y cuáles son los beneficios al evitar su presencia en el código:

    a) *Refused Bequest*

    b) *Shotgun Surgery*

    c) *Data Clumps*

    d) *Inappropriate Intimacy*

    e) *Lazy Class*

4. (1.5 pts) Considere el siguiente código:

```
1    # A movie that can be rented
2    class Movie
3      REGULAR = 0
4      NEW_RELEASE = 1
5      CHILDRENS = 2
6
7      attr_reader :title, :price_code
8
9      def initialize(title, price_code)
10       @title, @price_code = title, price_code
11     end
12   end
```

```
1    # A rental event
2    class Rental
3      attr_reader :movie, :days_rented
4
5      def initialize(movie, days_rented)
6        @movie, @days_rented = movie, days_rented
7      end
8    end
```

```ruby
1    # A customer of the store
2    class Customer
3      attr_reader :name
4
5      def initialize(name)
6        @name = name
7        @rentals = []
8      end
9
10     def add_rental(rental)
11       @rentals << rental
12     end
13
14     def summary
15       total_amount, frequent_renter_points = 0, 0
16       result = "Rental records for #{@name}\n"
17
18       @rentals.each do |rental|
19         this_amount = 0
20         # determine amounts for each line
21         case rental.movie.price_code
22         when Movie::REGULAR
23           this_amount += 2
24           this_amount += (rental.days_rented - 2) * 1.5 if rental.days_rented > 2
25         when Movie::NEW_RELEASE
26           this_amount += rental.days_rented * 3
27         when Movie::CHILDRENS
28           this_amount += 1.5
29           this_amount += (rental.days_rented - 3) * 1.5 if rental.days_rented > 3
30         end
31
32         # add frequent renter points
33         frequent_renter_points += 1
34
35         # add bonus for a two day new release rental
36         if rental.movie.price_code == Movie::NEW_RELEASE && rental.days_rented > 1
37             frequent_renter_points += 1
38         end
39
40         # show detail for this rental
41         result += "\t" + rental.movie.title + "\t" + this_amount.to_s + "\n"
42         total_amount += this_amount
43       end
44
45       # add footer lines
46       result += "Amount owed is #{total_amount}\n"
47       result += "You earned #{frequent_renter_points} frequent renter points"
48       result
49     end
50   end
```

a) Identifique los *code smells* presentes, detallando a qué familia pertenecen. Para cada uno especifique por qué es un problema para el código.

b) Proponga una versión mejorada del código que no presente los *code smells* detectados. Puede utilizar pseudo-código, pero debe ser explicito en lo que quiere expresar.

5. (0.5 pts)

    a) Explique 3 beneficios y 2 costos de utilizar *tests* unitarios para un proyecto.

    b) ¿Cuál es la diferencia entre *mock* y *stub*? ¿Cuál es el propósito principal de aplicar estos conceptos al realizar *tests*?

6. (1.4 pts)

Diseñe *tests* unitarios para las 2 funciones implementadas en el siguiente extracto de código. No es necesario que implemente código, aunque puede utilizar pseudo-código si lo desea. Se pide como mínimo que describa las pruebas con una breve descripción del escenario (qué se quiere probar), un contexto (configuración inicial del escenario) y un resultado esperado. Debe ser explícito en todas condiciones del contexto de sus *tests*. Considere que se descontará puntaje por *tests* redundantes.

```ruby
 1    class UF < ApplicationRecord
 2      def self.get(date = Time.zone.today)
 3        uf = UF.find_by(date: date)
 4        if uf.blank?
 5          published = value_published?(date)
 6          save_from_api(get_from_api(date.year)) if published || UF.empty?
 7          uf = UF.find_by(date: date)
 8          return uf.value if uf
 9          logger.error 'Error: UF not available' if uf.blank? && published
10          uf = UF.where('date <= ?', date).order(:date).last
11        end
12        uf.value
13      end
14
15      def self.value_published?(date = Time.zone.today)
16        today = Time.zone.today
17        date_month = date.month
18        date_day = date.day
19        months_difference = months_difference(today, date)
20        return false if months_difference > 1
21        if date_day > 9 && (months_difference == 1 || months_difference == 0)
22          return false
23        end
24        true
25      end
26
27      def self.get_from_api(year = Time.zone.now.year)
28        # Solicita los valores de la UF publicados en un año
29      end
30
31      def self.save_from_api(ufs)
32        # Guarda los valores de la UF recibidos en el formato de la API
33      end
34
35      def self.months_difference(from, to)
36        # retorna un número que representa la diferencia de meses entre (to - from)
37      end
38    end
```