Nombre _____

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

5 de junio de 2018

IIC2143 – Ingeniería de Software

Interrogación 3

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. (1.0 pts)

Originalmente el sistema computacional de *Spotify* consistía en una gran red de *P2P* (*peer-to-peer*) para proveer música a sus millones de suscriptores. Esta red creció hasta ser una de las más grandes del mundo. Sin embargo, el año 2014 la empresa anunció que cambiaría su arquitectura para proveer directamente su música a cada uno de sus clientes.

Datos de la empresa: *Spotify* es una empresa que provee música a sus clientes en dos modalidades: una gratis y otra pagada. La empresa no tiene directamente los derechos de autor sobre la música.

a. Justifique la decisión original de la empresa para utilizar una arquitectura *P2P*.

Con una gran red *P2P* se puede facilitar el manejo de un creciente número de usuarios sin la necesidad de incrementar tan dramáticamente la infraestructura. También provee baja latencia a sus clientes.

b. ¿Qué motivos podrían haber llevado a la empresa a realizar este cambio en su arquitectura?

En términos de seguridad de los datos es posible para los usuarios descargar los "archivos" dado que éstos estarían en los clientes. Esto llevaría a las compañías dueñas de la música a retirar sus "productos" de la aplicación. Por otra parte, también es relevante considerar que el servicio provisto debe cumplir con ciertos estándares de calidad. También se podrían considerar la dificultad para:

Nombre _____

- predecir la música que se escuchará para guardar los datos pertinentes en los clientes y así evitar guardar tanta música en ellos
- descargar los datos que se requieren del archivo (el segmento de la canción)
- uso de red (subida y bajada de contenido) en los clientes

2. (0.6 pts) Explique 3 razones por las cuáles se podría explicar que "buenos desarrolladores" generen "mal código" (*code smells*).

- Cumplir con presupuesto o plazos muy exigentes
- Se realizan cambios en el código sin revisar/actualizar el diseño
- Se extiende código *legacy* sin mejorarlo ni actualizarlo
- No se consulta la opinión de pares sobre el diseño o implementación
- No se considera el código en su totalidad, solamente partes aisladas

3. (0.8 pts)

   a. Explique 3 ventajas y 3 desventajas de una arquitectura basada en microservicios.

   Ventajas:
   - Facilita desarrollo, *deployment* y escalabilidad del sistema (al ser pequeños servicios independientes)
   - Permite heterogeneidad de herramientas y tecnologías utilizadas
   - Mejor control sobre fallas: si un servicio falla el resto puede seguir funcionando

   Desventajas:
   - Complejidad elevada al registrar y coordinar múltiples servicios
   - Congestión de red y latencia
   - Pensar de manera distribuida y asíncrona

   b. Imagine que su aplicación del proyecto del curso es un éxito, por lo que debe rediseñar su arquitectura. ¿Utilizaría una arquitectura monolítica o una de microservicios? Justifique su respuesta.

   Cualquiera de las 2 opciones puede ser correcta, lo importante es que la justificación tenga sentido y que se expongan ventajas y/o desventajas de las aplicaciones monolíticas para justificar elección frente a microservicios.

Nombre _____

4. (1.2 pts) Considere el siguiente escenario:

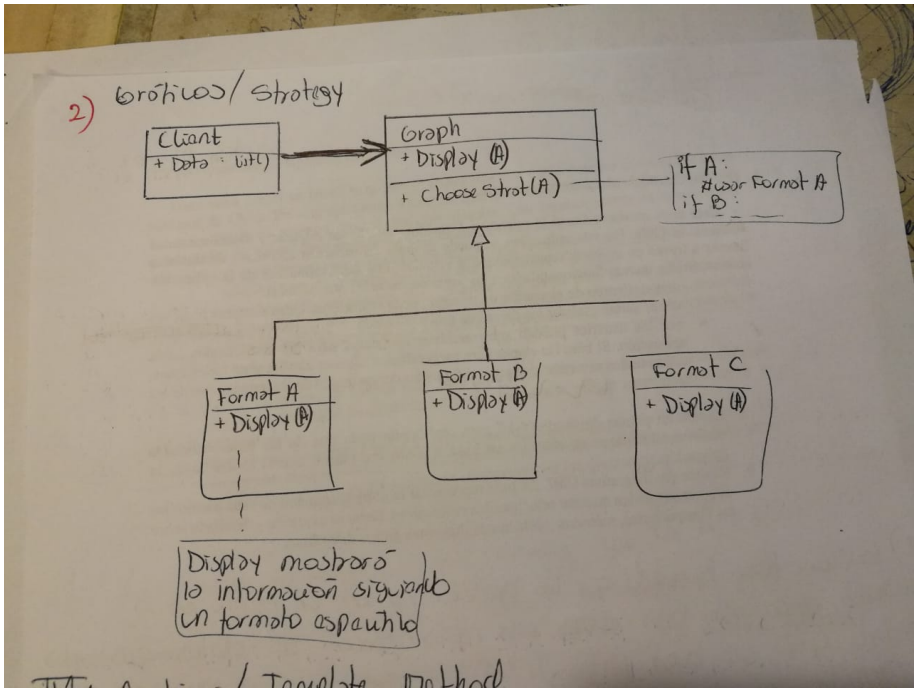
Hace unos meses se lanzó la iniciativa *DataChile* que busca integrar y visualizar datos públicos de Chile. En esta aplicación se puede analizar información con datos de distintas fuentes a través de diversas representaciones visuales. Los desarrolladores de la aplicación quieren ofrecer nuevas funcionalidades a los usuarios, entre las que destacan:

- que en tiempo de ejecución un usuario pueda seleccionar entre distintas opciones cómo quiere graficar los datos que está analizando.
- que los usuarios puedan subir archivos con datos para ser visualizados en la aplicación. Si bien las etapas para cargar documentos es similar para todos estos, estas pueden presentar pequeñas diferencias dependiendo de las extensiones de los archivos.

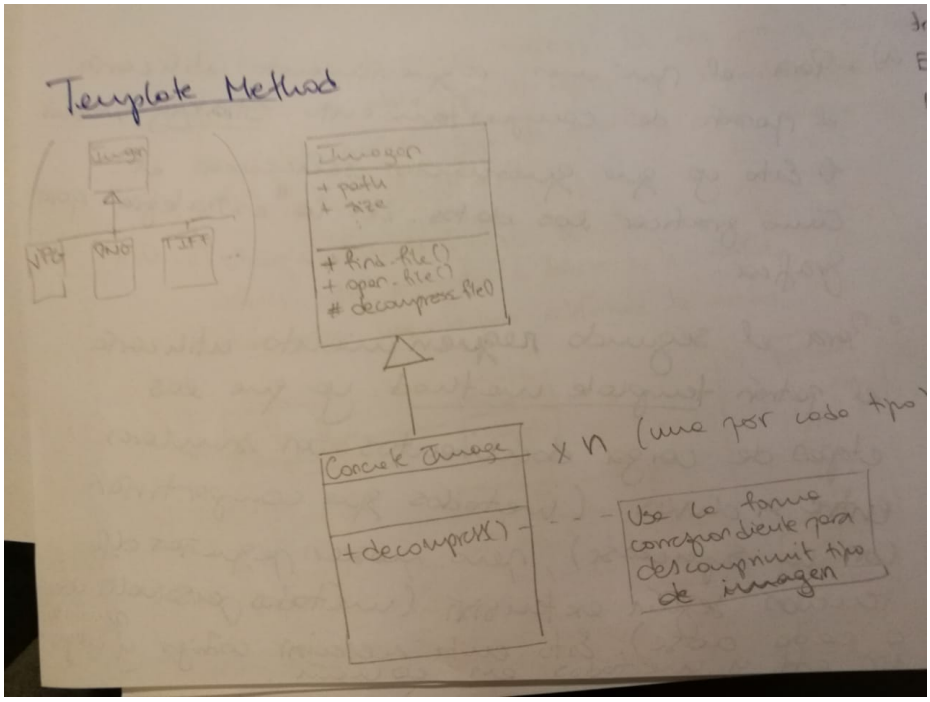
a. Escoja un patrón de diseño *GoF* para representar cada una de las funcionalidades descritas. Justifique su elección en base a cómo ese patrón aporta al diseño de la aplicación.

- Graficar datos - *Strategy*: permite que en tiempo de ejecución los datos se representen de diferentes formas.
- Cargar archivos – *Template Method*: como las etapas para subir archivos son similares, se puede seguir una misma lógica diferenciando algunos pasos.

b. Realice un diagramas *UML 2.0* para representar la implementación de cada patrón (los diagramas pueden no estar relacionados entre ellos). Debe se explícito y detallado sobre los componentes, métodos y relaciones presentes en los diagramas.



Nombre _____



5. (1.0 pts) Para cada una de las siguientes afirmaciones comente justificadamente si está de acuerdo o en desacuerdo:

    a. El porcentaje de *coverage* de los *tests* de una aplicación está directamente relacionado a la calidad de esta.

    Falso: el porcentaje de *coverage* indica la porción de código que la batería de pruebas ejecuta, pero no refleja la calidad de los *tests* ni del sistema.

    b. Si una aplicación cuenta con *tests* de aceptación e integración, entonces no es necesario realizar pruebas unitarias.

    Falso: las pruebas unitarias otorgan una granularidad mayor para determinar el origen de los errores.

    c. Bajo una cultura de *DevOps* todo el código de una aplicación debe tener pruebas automatizadas.

    Falso: no es necesario que todo el código tengas pruebas automatizadas, solamente las funcionalidades vitales para la aplicación.

    d. Realizar *refactoring* no aumenta la calidad de un sistema porque los cambios no son percibidos por los usuarios.

Falso: *refactoring* permite mejorar la calidad interna del *software*.

e. Un código enredado e incomprensible corresponde al anti-patrón *Big Ball of Mud*.

Falso: la definición se refiere a *Spaghetti code*.

6. (0.6 pts)

GitHut, un popular servicio de repositorios remotos de *git*, anunció recientemente que será adquirido por *Macrosoft*. Algunos desarrolladores espantados por tan gran empresa han empezado a migrar su código a la competencia: *GitFactory*.

En *GitFactory*, cuando un desarrollador quiere agregar una nueva funcionalidad a su proyecto éste debe crear un *pull request*. Esta solicitud no puede ser aceptada hasta que pasen todas las pruebas definidas en el repositorio. Los criterios utilizados para aceptar una solicitud son que los *tests* automatizados estén pasando, que exista un mínimo de *coverage* sobre estos, que todas las líneas editadas respeten la guía de estilo del código y que las dependencias del proyecto estén con sus versiones al día. Además, un miembro del equipo de desarrollo que haya trabajado en líneas similares es automáticamente asignado para revisar las líneas editadas, y solamente él puede aceptar la solicitud (o solicitar cambios). Por último, cuando los cambios son integrados a la *branch* principal del repositorio se vuelven a ejecutar las pruebas automatizadas en un ambiente de *staging*.

Indique las medidas adoptadas por *GitFactory* para evitar la aparición de errores en los proyectos. Por cada medida identificada, señale una ventaja y un problema asociado.

- *tests* automatizados aprobados
  - ventaja: se garantiza un funcionamiento mínimo del código en cada cambio.
  - problema: batería de *tests* requiere mantención y mayor tiempo para ejecutarse
- respetar la guía de estilo del código.
  - ventaja: código que respeta un solo estilo es más fácil de entender y modificar.
  - problema: dependiendo de qué tan exigente se haya definido el estilo del códgio se puede reducir la productividad del equipo (ya que tendrían que dedicar esfuerzos en respetar la guía, pasando el desarrollo funcionalidades a segundo plano).
- dependencias del proyecto con versiones al día
  - ventaja: se incluyen parches de seguridad de las dependencias.
  - problema: las últimas versiones de las dependencias pueden ser incompatibles con la aplicación.
- revisión de pares del código modificado
  - ventaja: se crea un conocimiento compartido de la aplicación, en base al cual los desarrolladores pueden aprender unos de otros.
  - problema: si la revisión no es realizada “a conciencia” esta no agregaría valor (o se integran grandes cambios que no son claros).
- ejecutar pruebas en ambiente de *staging*

- ventaja: se detectan errores que podrían ocurrir en ambiente de producción que no se previeron antes.
- problema: mayores costos de ejecución y tiempos de puesta en producción.

7. (0.8 pts) El siguiente modelo de *Rails* representa a un grupo de personas. Cada persona tiene un identificador, un nombre y una fecha de nacimiento.

```
class Group < ActiveRecord::Base
  has_many :people, class_name: 'Person'

  def most_common_name
    # Retorna el nombre más repetido entre las personas del
grupo.
    # En caso de empate retorna un Array de los empatados.
  end

  def average_age
    # Retorna la edad promedio de la gente en el grupo.
  end
end
```

Especifique todas las pruebas que realizaría sobre esta clase. Para ello indique el método a probar, los contextos en los que se pondrá y el *output* que se espera (no es necesario escribir código).

- Verificar que la clase *Group* tenga la asociación *people*
- most_common_name:
  - Grupo sin personas: retorna *nil*
  - Grupo con una persona llamada "Pepe": retorna "Pepe"
  - Grupo con dos personas llamadas "Juana" y "Pablo": retorna ["Juana", "Pablo"]
  - Grupo con tres personas llamadas "Juana", "Juana" y "Pedro": retorna "Juana"
- average_age:
  - Grupo sin personas: retorna *nil*
  - Grupo con una persona de edad 23: retorna 23
  - Grupo con dos personas de 23 y 25 años: retorna 24