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

13 de septiembre de 2016

IIC2113 - Diseño Detallado de Software

Interrogación 1

Instrucciones: Sea preciso: no es necesario escribir mucho pero sí ser preciso. En caso de ambigüedad, utilice su criterio y explicite los supuestos que considere convenientes. Esta interrogación fue diseñada para durar 80 minutos, pero se puede extender a 100 de ser necesario.

1. (1.2 pts) Nombre los principios S.O.L.I.D y describa brevemente a qué se refiere cada uno.

   Single-responsibility principle: Una clase debería tener una y solo una razón para cambiar. Esto quiere decir que una clase debería tener un solo deber.
   Open-closed principle: Los objetos o entidades deberían estar abiertos a extensiones, pero cerrados a modificaciones.
   Liskov substitution principle: Sub-clases deberían se sustitutos para sus clases base.
   Interface segregation principle: Varias interfaces especificas son mejores que una sola con un propósito general.
   Dependency inversion principle: Se debe depender de las abstracciones, no de las implementaciones.

2. (1.0 pts) Comente qué describe cada una de las vistas del modelo 4 + 1 propuesto por Philippe Kruchten.

   Vista lógica: Describe la estructura y funcionalidad del sistema. Los interesados son los usuarios finales.
   Vista de procesos: Trata los aspectos dinámicos del sistema, explica los procesos y cómo se comunican. Se enfoca en el comportamiento del sistema en tiempo de ejecución.
   Vista de implementación: Ilustra el sistema desde la perspectiva de los desarrolladores, y está enfocada en la administración de los artefactos de software.

Vista física: Describe el sistema desde el punto de vista de un ingeniero de sistemas. Ilustra la topología de componentes de software en la capa física, así como las conexiones entre ellos.

Vista de escenarios: Descripción holística que se utiliza para identificar y validar el diseño de la arquitectura.

3. (0.8 pts) Git:
   a. (0.2 pts) Nombre 3 motivos que justifiquen el uso de un sistema de control de versiones.

      - llevar un registro de múltiples versiones de un archivo,
      - colaborar con otros desarolladores en un mismo proyecto,
      - ver cuánto ha trabajado cada uno: quién, dónde y cuándo,
      - experimentar con alguna nueva funcionalidad,
      - revisar el historial de cierto código,
      - seguir bugs, analizando qué fue lo que rompió tu código.

   b. (0.3 pts) Imaginemos que usted acaba de ejecutar el comando *git commit*, después de haber añadido varios archivos al *staging area*. Sin embargo, se percató de que ese *commit* no es atómico, por lo que le gustaría volver en el tiempo para quedar justo en el momento anterior a la ejecución del comando. De forma cándida, usted escribe *git uncommit*. Lamentablemente, ese comando no existe.

      ¿Cómo haría usted para emular ese comando?

      - git reset –soft HEAD^

   c. (0.3 pts) Con respecto al modelo *gitflow*, nombre los 5 tipos de *branches* que se proponen. Describa brevemente la finalidad de cada una.

      - master: rama que representa al código que está en producción.
      - develop: rama que contiene el código que será publicado en la siguiente entrega del proyecto.
      - feature: ramas para desarrollar nuevas características de la aplicación.
      - hotfix: ramas para corregir errores que están presentes en el código de producción.
      - release: ramas para preparar la siguiente entre de código a producción.

4. (3.0 pts) Archi-TECH

Dada la popularidad de la realidad virtual (*Virtual Reality*), lo han contratado para un emprendimiento de tecno-arquitectos. Estos emprendedores quieren desarrollar un producto que les permita realizar maquetas 3D de manera colaborativa.

De esta forma, una funcionalidad básica para el proyecto es la capacidad de que varios arquitectos trabajen sobre un mismo diseño de forma simultánea (similar a *Google Docs*). Para minimizar el uso de internet, se planea que solo se haga *broadcast* de las modificaciones que realiza cada arquitecto en la maqueta. De esta forma, en vez de estar enviando constantemente el estado completo de una maqueta, se pueda enviar pequeñas actualizaciones de su estado.

Por otra parte, como toda herramienta de edición, se tiene que permitir la opción de deshacer y rehacer los últimos cambios al proyecto, considerando que estos cambios están asociados a la persona que los realizó.

Otra funcionalidad importante para el proyecto es la capacidad de poder crear y utilizar "bloques estructurales" para realizar maquetas 3D. Los bloques más primitivos son hexaedros, esferoides, pirámides y cilindros. A partir de estos se pueden crear objetos de más alto nivel, tales como paredes y puertas, que a su vez se reutilicen para poder crear objetos más elaborados como casas y puentes.

Además, se le pide la posibilidad de agregar extensiones futuras a la aplicación (*plugins*). La idea de esto es que puedan aplicar algoritmos que extiendan la funcionalidad de un bloque (ya sea primitivo o de alto nivel). Algunos ejemplos son:

- Modificar el color de un bloque
- Extensiones para métricas y propiedades de un bloque
- Algoritmos de compresión por bloque

También, se espera que las creaciones de los arquitectos puedan ser sometidas a simulaciones de pruebas estructurales en otras aplicaciones. Es así como la aplicación debe ser capaz de exportar las creaciones en distintos formatos, para que puedan ser reutilizadas por otros programas.

Por último, dadas las constantes modificaciones a las que se ve sometida una maqueta, se ve la necesidad de guardar versiones de un proyecto. De esta forma, se podría almacenar el estado de una maqueta, y volver a ese estado en cualquier momento.

Preguntas:

a) (1.5 pts) Identifique 5 patrones de diseño que podrían ser utilizados en una solución para este proyecto. Para cada uno debe justificar su elección con respecto a un requisito de la aplicación.

- Mediator: Trabajar sobre un proyecto de forma simultánea
- Observer: Broadcast de las modificaciones
- Comand: Herramienta de edición con deshacer y rehacer
- Memento: Herramienta de edición con deshacer y rehacer. Guardar versiones de los proyectos
- Composite: 'Bloques estructurales' 3D
- Decorator: Plugins para los bloques
- Builder: Exportar los proyectos en distintos formatos

b) (1.5 pts) Escoja 2 patrones de diseño y realice un diagrama UML 2.0 que refleje los participantes en el contexto de la aplicación descrita. Se debe realizar un solo diagrama que refleje la relación entre ambos patrones. Debe señalar claramente dónde se implementa cada patrón en el diagrama y explicar la responsabilidad de cada participante.