Universidad Nacional de Córdoba

Informe Trabajo Final

Ingeniería de Software

Aplicación de las técnicas de Ingeniería de Software sobre un proyecto inconcluso.

Autores:
Cristian Gutierrez (35.596.222)
Esteban Morales (35.104.714)
Fabiola Campos (34.243.721)
Gianfranco Barbiani (36.372.693)

Supervisor:
Ing. Martín Miceli

5 de julio de 2014

Índice

Resumen

En el presente informe se expone el desarrollo del trabajo final para la cátedra Ingeniería de Software que se cursa en el primer cuatrimestre del cuarto año de la carrera Ingeniería en Computación en la Facultad de Ciencias Exáctas, Físicas y Naturales de la Universidad Nacional de Córdoba, Argentina. Este trabajo está basado en el ejemplo del Libro Head First design Patterns en la página 526 a 548 (DJView).

1. Consignas

- Modificar la clase `HeartModel` para que sólo se pueda crear una instancia (usando el patrón *Singleton*) y extender la ventana de control del `BeatController` para "tratar" de generar nuevas instancias cada vez que se clickea en el botón >>. La ventana de la `BeatBar` debería mostrar en texto el número de intentos de creación de un nuevo `HeartBeatModel` en el texto donde se mostraba la frecuencia cardíaca.
- Crear un nuevo modelo con su controlador específico que pueda usarse para verse desde la vista `DJView` en la ventana `BeatBar`. Generar un java main class para poder ejecutar tal modificación llamándolo `My<modelName>TestDrive.java` (similar to `HeartTestDrive`). Se proveerán puntos adicionales por la originalidad del modelo creado.
- Generar una vista propia que permita usar su modelo sin modificar el código existente del ejemplo pero que permita mostrar los cambios gráficamente y por medio de texto (similar al `BeatBar` que muestra el ritmo con una barra y la frecuencia en texto).
- Generar un `TestDrive` que permita mostrar a los tres modelos trabajando al mismo tiempo (se esperarán ver al menos 3 ventanas `BeatBar` con los 3 modelos andando simultáneamente.
- Modificar la vista `BeatBar` para que permita cambiar gráficamente en tiempo de ejecución (ej. Mediante un dropdown box) el modelo usado (el `BeatModel`, el Nuevo modelo y el `HeartBeatModel`). Por favor use para tal implementación el patrón *Strategy*. Generar un `TestDrive` que permita ejecutar tal acción.

2. Nota de Entrega

Se entrega junto a este informe:

- El código fuente del proyecto de software.
- Los modelos UML en formato imagen y proyecto de VisualParadigm®.
- Dos (2) archivos ejecutables: para Windows y Linux.
- Una guía de instalación y ejecución dentro del Readme.md del Repositorio.
- La presentación con diapositivas en Reveal.js y su equivalente en PDF.

2.1. Listado de Funcionalidad

Ver detalle en sección 6.

2.2. Pass/Fail Ratio del Sistema (PFR)

Ver detalle en sección 7.

2.3. Vínculo a las fuentes del proyecto

La dirección web del Repositorio del Proyecto es: *https://github.com/Andresteve07/FinalIngDeSoftware*.

3. Manejo de Configuraciones

3.1. Plan de Manejo de las Configuraciones

El plan de manejo de las configuraciones trata y controla:

- La elaboración de código fuente por varios desarrolladores simultáneamente.
- El seguimiento del estado de las versiones y sus cambios.
- La conducción de la integración de las partes del software en un solo producto de software.

Para la realización de SCM hay diferentes herramientas. Pero herramientas que pretenden ofrecer una solución total al problema a menudo no cumplen con los requisitos técnicos como:

- Apoyo a diferentes plataformas.
- Iniciar el proceso de build.
- Conexión a los bancos de datos existentes.
- Integración a la organización existente.

Por esa razón ofrece una mayor flexibilidad una solución que integre herramientas parciales que sean más fáciles de integrar en el proceso existente.

3.2. Herramienta de Control de Versiones

Para el control de versiones del software que se implemento como consigna del siguiente trabajo final se uso la herramienta de versiones distribuidas y manejo de código fuente

3.3. Sobre los Directorios

El repositorio cuenta con tres (3) directorios:

- FinalIngSoft: Este directorio es donde se aloja el proyecto de software desarrollado en la
- Informes: En este directorio se encuentra el código fuente escrito en LaTeXdel informe para el trabajo final, un archivo de compilación en PDF y un subdirectorio de imágenes utilizadas.
- Modelos: Este directorio contiene los proyectos que se realizaron para el modelado UML del software implementado mediante el uso de la herramienta de pago `Visual Paradigm`®.

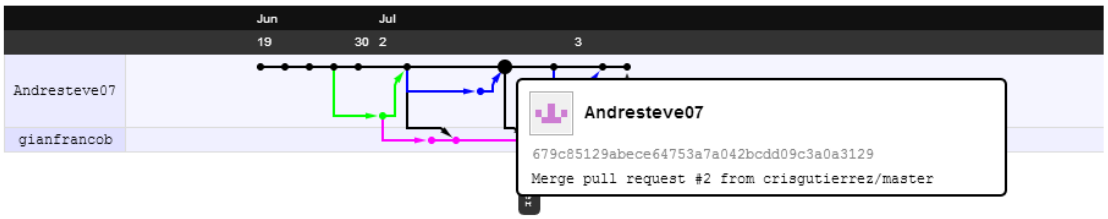
3.4. Etiquetado y Nombramiento de Archivos

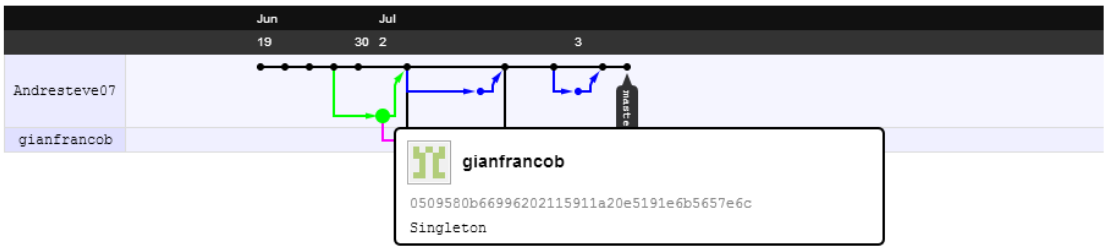
Se decidió del siguiente estándar para el etiquetado y nombramiento del archivos para el proyecto de software:

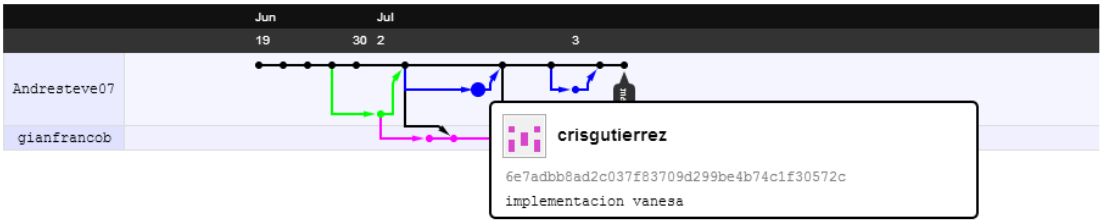
- Para los archivos de código fuente escritos en JAVA se adhiere a la costumbre de iniciales en mayúsculas sin espacios.
- Para las etiquetas sobre las versiones del software se usa la plantilla: vX.Y.Z; con X: Major Version, Y:Minor Version, Z:Patches

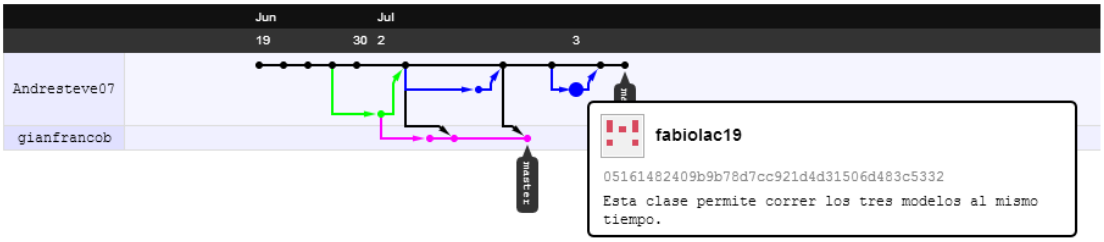
3.5. Plan del Esquema de Ramas

Como política de branching se optó por UCM (rebase before deliver).









3.6. Políticas de Marge y de Etiquetado de Progreso.

Como política de mergeo se decidió que cada desarrollador tenga un fork (o bifurcación, copia) del repositorio original en su cuenta de Github y en su computadora personal, sobre la cual este podrá realizar modificaciones a su conveniencia. Dichos cambios van desde crear branches (ramas) personales hasta etiquetas propias con el objetivo desarrollar la funcionalidad que se le asignó. Una vez terminado su labor, el desarrollador deberá integrar todos los cambios a la rama principal de su fork y recién luego de solucionar los conflictos que hubiere, podrá solicitar un pull request al Configuration Manager (de ahora en más CM) para que la funcionalidad sea integrada el repositorio original.

3.7. Sobre los Releases

Forma de entrega de los releases, instrucciones mínimas de instalación y formato de entrega.

3.7.1. Forma de Entrega

Una vez conseguida una versión estable el CM generará la etiqueta pertinente respetando el estándar mencionado en la sección 3.4.

3.7.2. Instrucciones de Instalación

Dentro del repositorio se encuentra un archivo Readme.txt que explica en forma simple y pocos pasos la instalación y ejecución de un release específico.

3.7.3. Formato de Entrega

Se adjuntará al directorio `Releases/` dentro del repositorio un archivo con el formato `r\_vX.Y.Z.sh` que es un script de ejecución para un intérprete de bash de Linux.

3.8. Integrantes

Listado y forma de contacto de los integrantes del equipo, así como sus roles en la CCB. También incluir periodicidad de las reuniones y miembros obligatorios.

3.8.1. Roles

3.8.2. Sobre las Reuniones

Las reuniones se acordaron mediante comunicación vía e-mail, y mensajería instantánea. En ellas se encontraron la totalidad de los miembros del equipo.

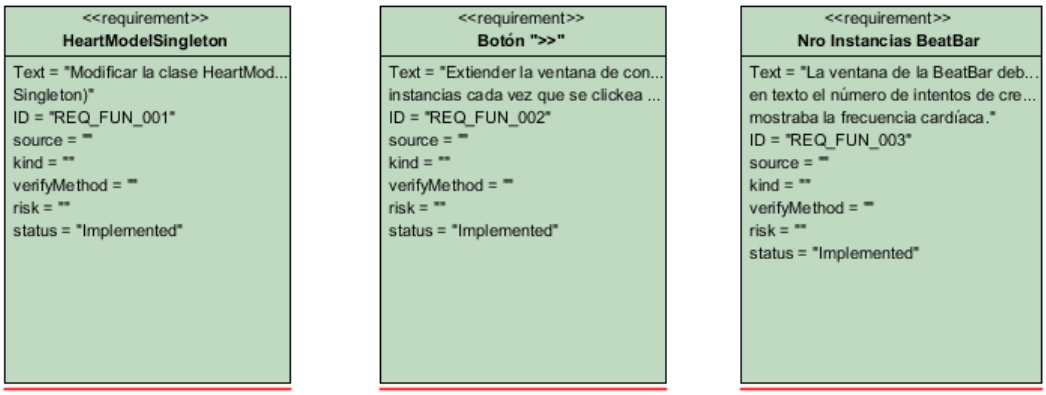
- Frecuencia de las reuniones: Encuentros no periódicos de larga duración (XtremePrograming).

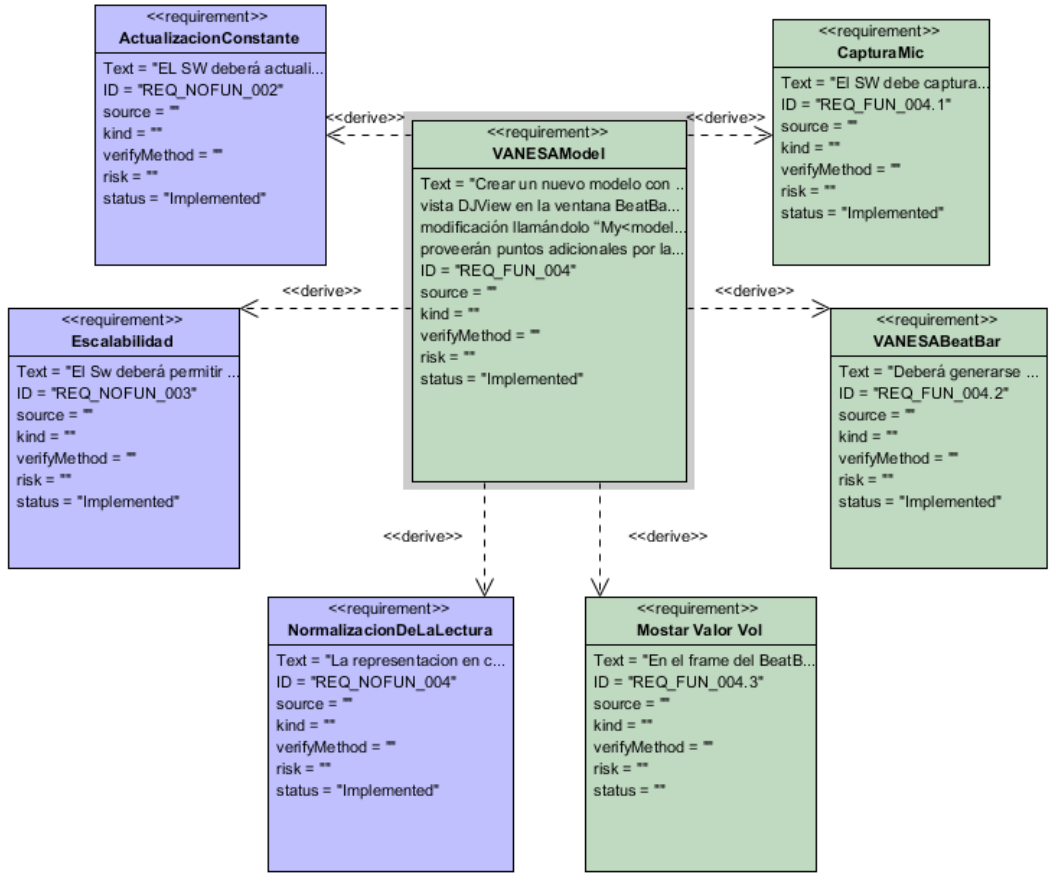
3.9. Herramienta de Seguimiento de Errores

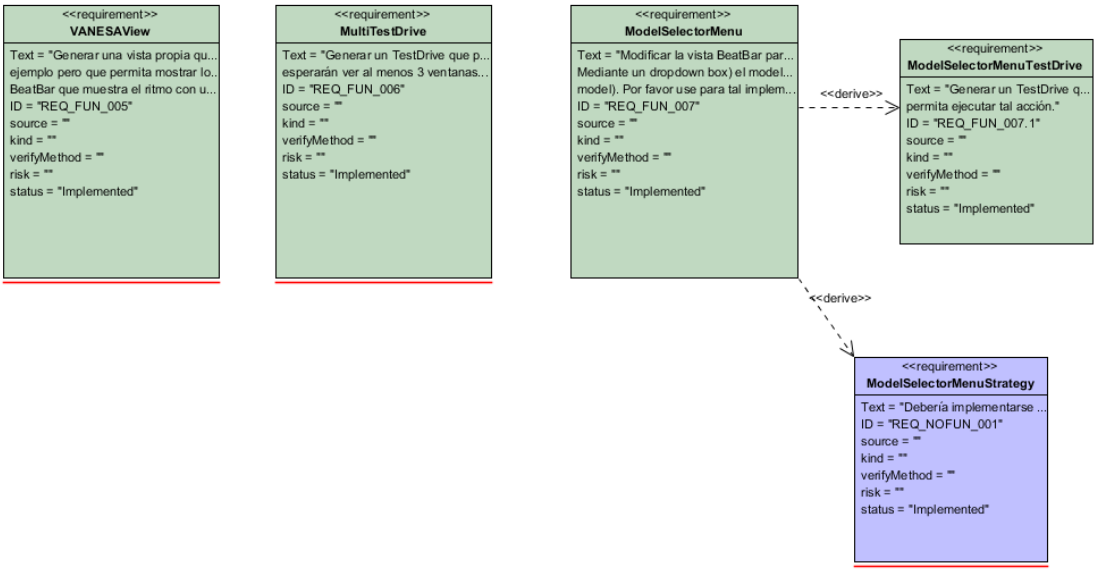
Para el seguimiento de los errores se utilizó el sistema de gestión de issues de provee Github mediante el cual es posible reportar defectos descubiertos, posibles mejoras, detectar código duplicado y realizar preguntas sobre rutinas confusas.

4. Requerimientos

Como parte del `REQ\_FUN\_004`, denominamos a la nueva clase `My<modelName>TestDrive.java` como `VANESATestDrive` donde VANESA es un acrónimo para: A continuación se presentan tres (3) imágenes que exponen lo requerimientos funcionales y no funcionales detectados y su estado al final de la implementación:

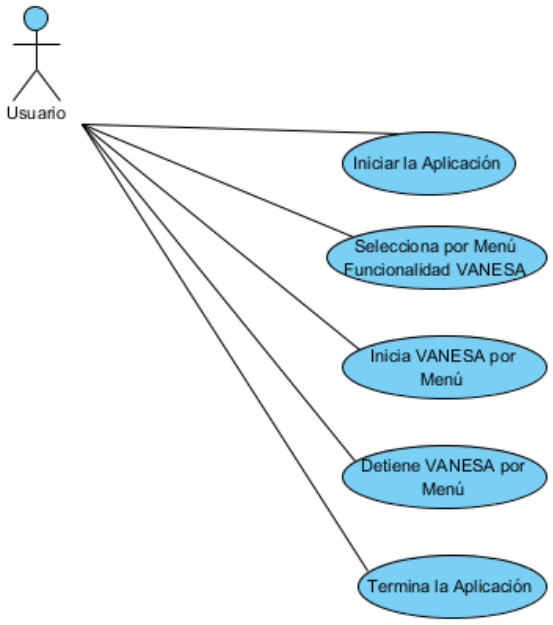

<<requirement>>
HeartModelSingleton
Text = "Modificar la clase HeartMod...
Singleton)"
ID = "REQ_FUN_001"
source = ""
kind = ""
verifyMethod = ""
risk = ""
status = "Implemented"
<<requirement>>
Botón ">>"
Text = "Extiender la ventana de con...
instancias cada vez que se clickea ...
ID = "REQ_FUN_002"
source = ""
kind = ""
verifyMethod = ""
risk = ""
status = "Implemented"
<<requirement>>
Nro Instancias BeatBar
Text = "La ventana de la BeatBar deb...
en texto el número de intentos de cre...
mostraba la frecuencia cardíaca."
ID = "REQ_FUN_003"
source = ""
kind = ""
verifyMethod = ""
risk = ""
status = "Implemented"


<<requirement>>
ActualizacionConstante
Text = "EL SW deberá actuali...
ID = "REQ_NOFUN_002"
source = ""
kind = ""
verifyMethod = ""
risk = ""
status = "Implemented"
<<derive>>
<<requirement>>
VANESAModel
Text = "Crear un nuevo modelo con ..
vista DJView en la ventana BeatBa...
modificación llamándolo "My<model...
proveerán puntos adicionales por la...
ID = "REQ_FUN_004"
source = ""
kind = ""
verifyMethod = ""
risk = ""
status = "Implemented"
<<derive>>
<<requirement>>
CapturaMic
Text = "El SW debe captura...
ID = "REQ_FUN_004.1"
source = ""
kind = ""
verifyMethod = ""
risk = ""
status = "Implemented"
<<derive>>
<<requirement>>
Escalabilidad
Text = "El Sw deberá permitir ...
ID = "REQ_NOFUN_003"
source = ""
kind = ""
verifyMethod = ""
risk = ""
status = "Implemented"
<<derive>>
<<requirement>>
VANESABeatBar
Text = "Deberá generarse ...
ID = "REQ_FUN_004.2"
source = ""
kind = ""
verifyMethod = ""
risk = ""
status = "Implemented"
<<derive>>
<<derive>>
<<requirement>>
NormalizacionDeLaLectura
Text = "La representacion en c...
ID = "REQ_NOFUN_004"
source = ""
kind = ""
verifyMethod = ""
risk = ""
status = "Implemented"
<<requirement>>
Mostar Valor Vol
Text = "En el frame del BeatB...
ID = "REQ_FUN_004.3"
source = ""
kind = ""
verifyMethod = ""
risk = ""
status = ""



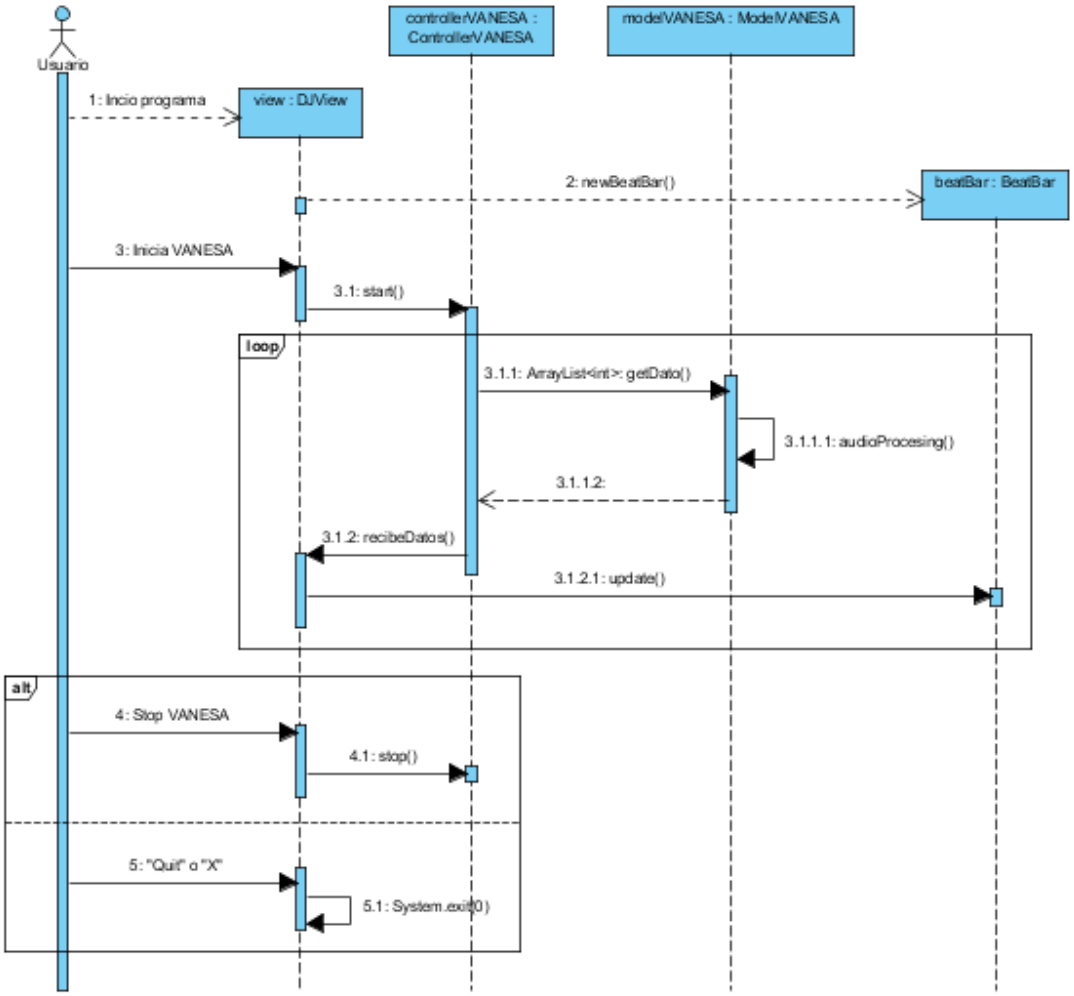
4.1. Diagramas de Casos de Uso

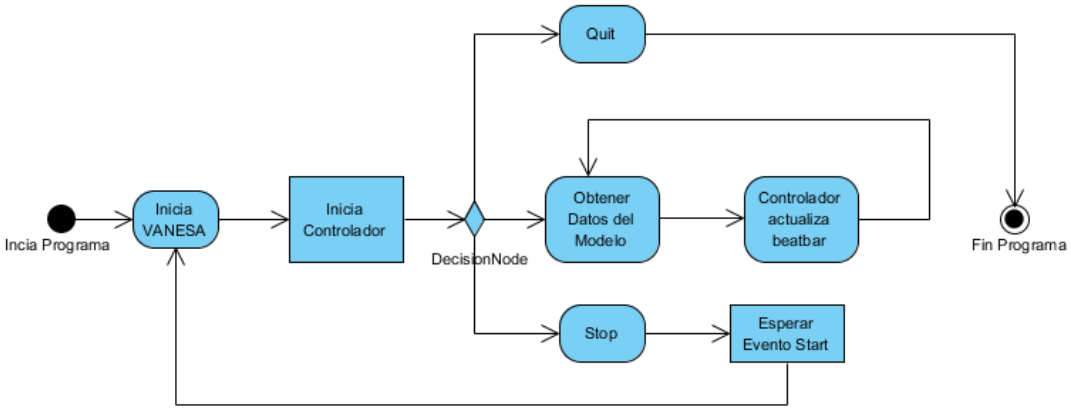
En esta sección se muestra el diagrama de casos de usos que surgió de la interpretación preliminar de la consigna:



4.2. Diagramas de Secuencia y Actividad

Las siguientes imágenes ilustran las representaciones preliminares del diagrama de secuencia y el de actividad para la ejecución del modelo consigna del trabajo:



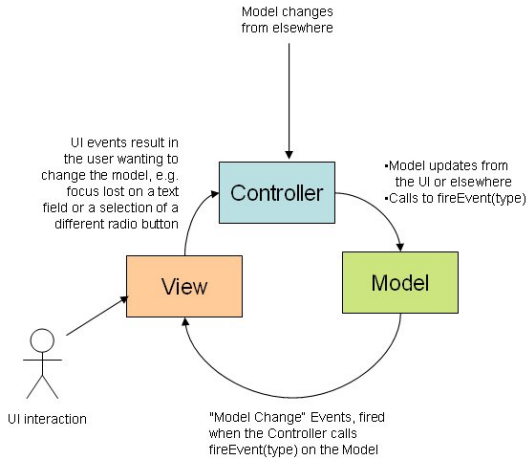


5. Arquitectura

El software implementa una estricta arquitectura de MVC (Model View Controller). En las siguientes subsecciones se indicará con mayor detalle cuáles partes del código la representan.

5.1. Gráfico de Arquitectura

El Patrón de arquitectura MVC es generalmente representada como:



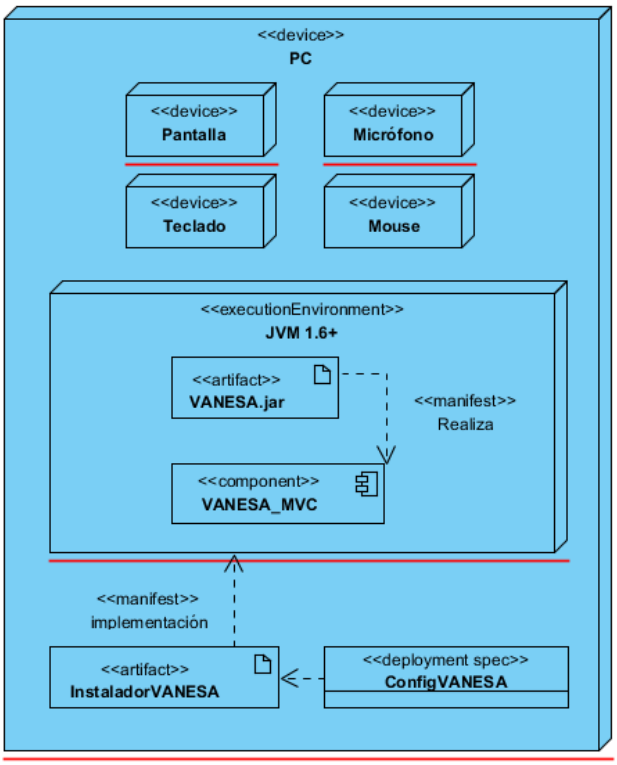
5.2. Patrón de Arquitectura

El marco de trabajo MVC propuesto originalmente en la década de los 80 como una aproximación al diseño de GU que permitió múltiples presentaciones de un objeto y estilos independientes de interacción de cada una de estas presentaciones.El marco MVC soporta la presentación de los

datos de diferentes formas e interacciones independientes con cada una de estas presentaciones. Cuando los datos se modifican a través de una de las presentaciones, el resi de las presentaciones son actualizadas. Los marcos de trabajo son a menudo instanciaciones de varios patrones. Por ejemplo, el marco MVC incluye el patrón Observer, el patrón Strategy relacionado con la actualización del modelo, el patrón Composite y otros patrones descritos por Gamma y colaboradores(Gammaetal.,1995). Las aplicaciones construidas utilizando marcos de trabajo pueden ser las bases para una posterior reutilización a través del concepto de líneas de productos software o familias de aplicaciones. Debido a que estas aplicaciones se construyen utilizando un marco, se simplifica la modificación de miembros de la familia para crear nuevos miembros. El problema fundamental con los marcos de trabajo es su complejidad inherente y el tiempo que lleva aprender a utilizarlos. Pueden requerirse varios meses para comprender completamente el marco de trabajo, por lo que es muy probable que, en organizaciones grandes, algunos ingenieros de software se conviertan en especialistas en marcos de trabajo. No hay duda de que ésta es una aproximación efectiva para la reutilización, pero es muy elevado el coste que supone introducirla en los procesos de desarrollo del software.

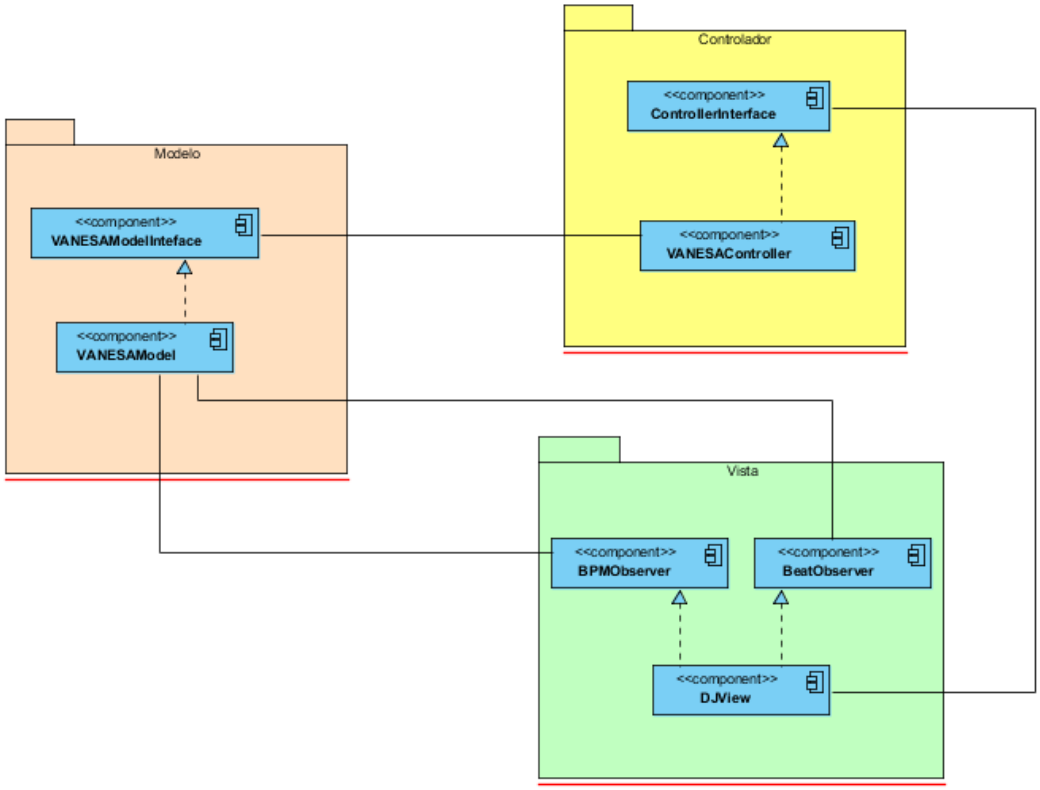
5.3. UML de Despliegue

En esta sección se muestra el diagrama de despliegue que surgió de la interpretación preliminar de la consigna:



5.4. UML de Componenetes

A continuación se presenta una imagen que modela el diagrama de componentes tal y como se detectaron al comienzo de la implementación:



6. Diseño e Implementación

6.1. Diagrama de Clases

Con el objetivo de modularizar el código y procurar el encapsulamiento de cada parte, se decidió la implementación de las siguientes clases:

- Controladores
  - `/FinalIngSoft/src/Controladores/BeatController.java`
  - `/FinalIngSoft/src/Controladores/ControllerInterface.java`
  - `/FinalIngSoft/src/Controladores/HeartController.java`
  - `/FinalIngSoft/src/Controladores/VANESAController.java`

- FinalIngSoft
  - `/FinalIngSoft/src/FinalIngSoft/DJTestDrive.java`
  - `/FinalIngSoft/src/FinalIngSoft/HeartTestDrive.java`
  - `/FinalIngSoft/src/FinalIngSoft/TestDrive.java`
  - `/FinalIngSoft/src/FinalIngSoft/VANESATestDrive.java`
  - `/FinalIngSoft/src/FinalIngSoft/VANESAViewTest.java`
- Modelos
  - `/FinalIngSoft/src/Modelos/BeatModel.java`
  - `/FinalIngSoft/src/Modelos/BeatModelInterface.java`
  - `/FinalIngSoft/src/Modelos/HeartAdapter.java`
  - `/FinalIngSoft/src/Modelos/HeartModel.java`
  - `/FinalIngSoft/src/Modelos/HeartModelInterface.java`
  - `/FinalIngSoft/src/Modelos/VANESAModel.java`
  - `/FinalIngSoft/src/Modelos/VANESAModelInterface.java`
- Testing
  - `/FinalIngSoft/src/Testing/VANESAModelTest.java`
- Vistas
  - `/FinalIngSoft/src/Vistas/BeatBar.java`
  - `/FinalIngSoft/src/Vistas/BeatObserver.java`
  - `/FinalIngSoft/src/Vistas/BPMObserver.java`
  - `/FinalIngSoft/src/Vistas/DJView.java`
  - `/FinalIngSoft/src/Vistas/VANESAView.java`

Se muestra a continuación el diagrama de clases y luego se desarrollará una breve descripción de las nuevas clases e interfaces implementadas y se detallarán sus métodos más relevantes.

6.2. VANESAController.java

Esta clase está encargada de tomar los datos de la View, (de acuerdo al modelo MVC), los procesa y con ellos ejecuta funciones del `VANESAModel`, logrando así la interacción desde el View, pasando por el Controller hasta el Model.

6.3. TestDrive.java

Esta clase fue implementada para poder satisfacer el `REQ\_FUN\_006`, denominado "Multi-TestDrive". Su objetivo, es ejecutar y mostrar simultáneamente los tres (3) modelos que posee el proyecto de software.

6.4. VANESATestDrive.java

Esta clase crea un modelo del tipo `VANESAModel`, el cual luego asocia a un controlador específico (VANESAController), y con los cuales mostrará las vistas correspondientes solicitadas.

6.5. VANESAModel.java

Esta clase es un hilo que representa el comportamiento descrito por el `REQ\_FUN\_004` y sus requerimientos derivados. Sus clases mas importantes son:

- `calculateRMSLevel : int:` Calcula y devuelva el valor eficaz de un vector del tipo byte.
- `getAudioFormat : AudioFormat:` Define la frecuencia de muestreo, el tamaño en bits del vector, el número de canales y otras variables y devuelve un nuevo objeto AudioFormat con el cual se realizará la captura de audio del micrófono.
- `run : void:` Lee la información de audio de la línea de entrada del buffer. Luego se calcula el valor RMS del arreglo de bytes y con él genera una lectura que representa el volumen de la entrada por micrófono. Finalmente se notifica a todos los observadores. Cabe aclarar que al momento de notificar al BeatBar, este se hace cada un período mayor cuanto menor sea el volumen leído.

6.6. VANESAModelTest.java

El comportamiento de esta clase sera detallado en la sección 7.

6.7. DJView.java

Es una clase que se encontraba ya implementada por el código fuente original. Sin embargo, se le realizaron las siguientes modificaciones a fin de satisfacer los requerimientos pertinentes:

- Se creó el método `setModel : void:`, el cual es el encargado de determinar qué modelo se mostrará por la vista. Para hacerlo eficientemente, primeramente remueve los observadores antiguos, refresca el modelo actual al deseado, y finalmente vuelve a registrar los observadores.
- Se modificó la función `createView : void` para que crear un menú nuevo (con sus correspondientes items)dentro del frame del BeatBar, así con el permitir seleccionar entre los distintos modelos.

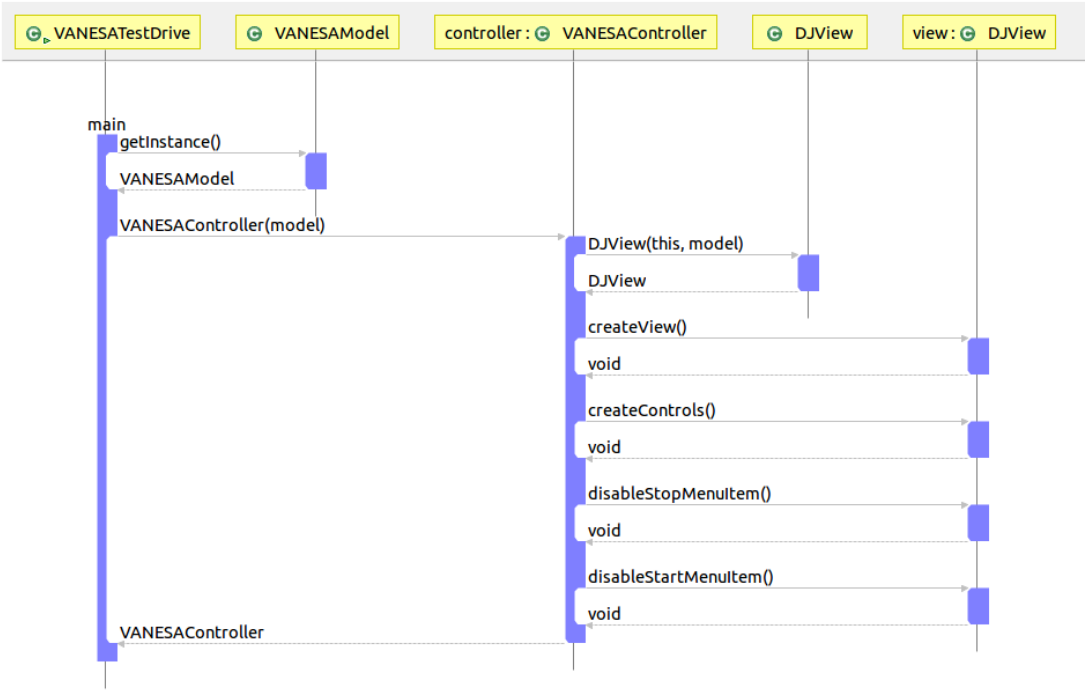
6.8. VANESAView.java

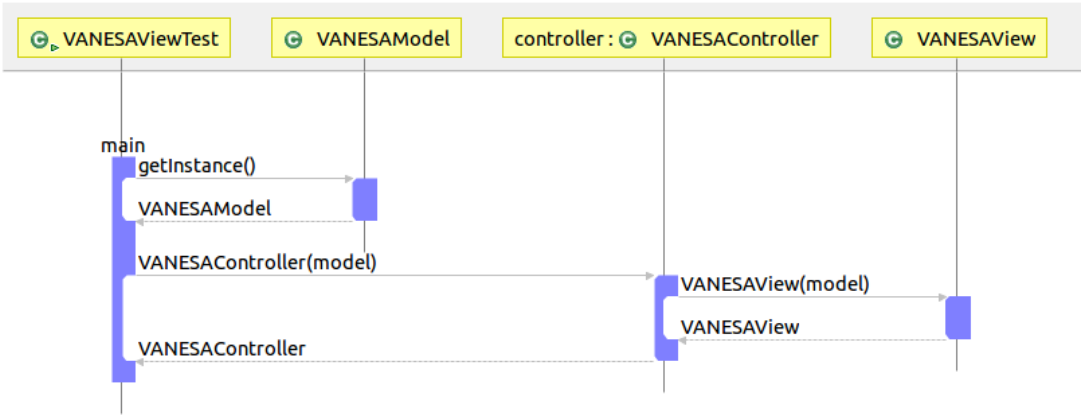
Es la clase encargada de crear una vista para representar el modelo VANESAModel. Sus funciones son:

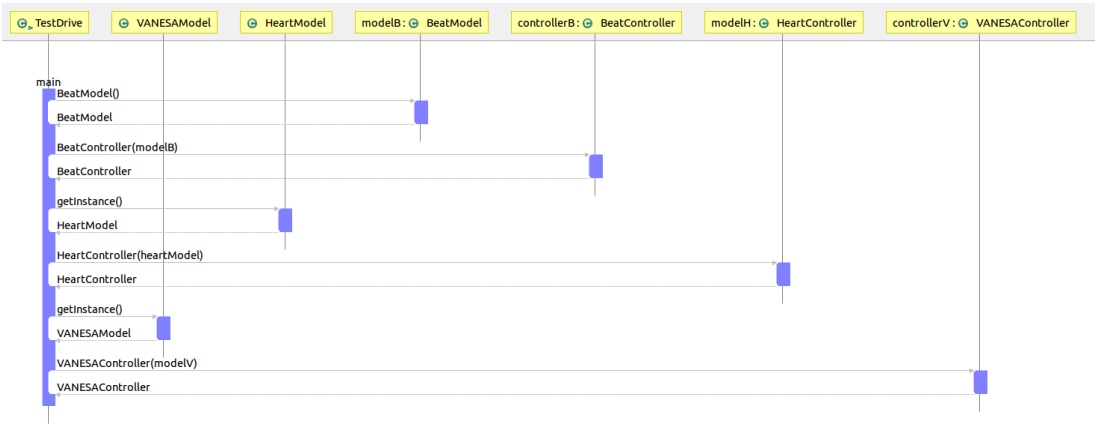
- `VANESAView (constructor):` Registra los observadores (BeatObserver y BPMObserver) al modelo VANESAModel. A continuación crea una barra de progreso (`jp : JProgressBar`) a fin de con ella representar gráficamente el nivel de volumen leído por el micrófono.
- `(@Override) updateBPM : void:` Este método refresca la etiqueta que muestra el nivel de volumen del micrófono.
- `(@Override) updateBeat : void:` En esta función, bajo ciertas condiciones, se "pinta" el valor de volumen del micrófono.

6.9. Diagrama de Secuencia

A continuación se muestran varios diagramas de secuencia de ejecución del proceso:

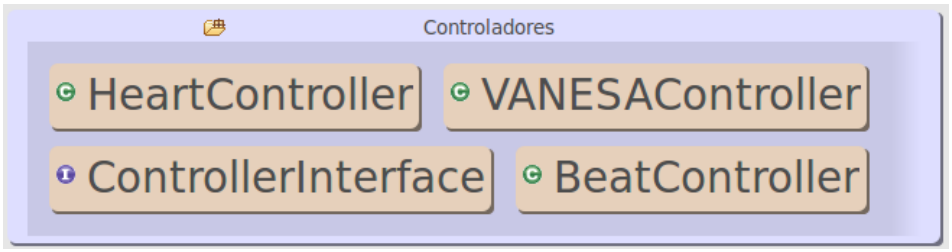


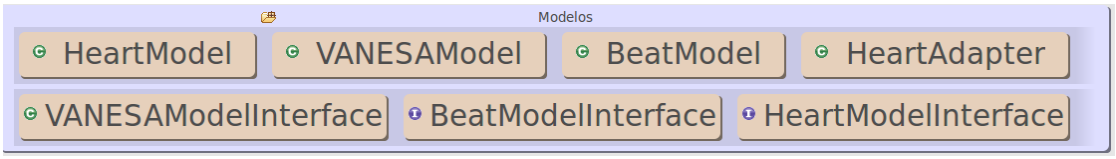


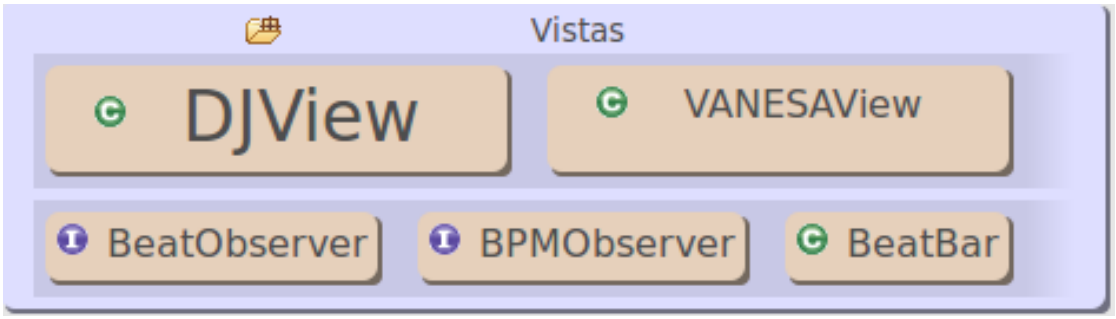


6.10. Diagrama de Paquetes

A continuación se muestran varios diagramas de paquetes que modelan software:







6.11. Patrón de Diseño adicional implementado

Se implementó un patrón de diseño Singleton en la clase VANESAModel.

7. Pruebas Unitarias del Sistema

7.1. Pruebas Automáticas

La clase `VANESAModelTest.java` genera un test unitario para encontrar errores en la clase `VANESAModel`. Realiza pruebas sobre los métodos `calculateRMSLevel` y `getAudioFormat`.

7.2. Matriz de Trazabilidad

Después de realizar la pruebas unitarias de software y compararlas con los requerimientos del sistema se obtuvo la siguiente matriz de trazabilidad:

	Requirements::ActualizacionConstante	Requirements::Boton '>>'	Requirements::CapturaMic	Requirements::Escalabilidad	Requirements::HeartModelSingleton	Requirements::ModelSelectorMenu	Requirements::ModelSelectorMenuStrategy	Requirements::ModelSelectorMenuTestDrive	Requirements::MultiTestDrive	Requirements::NroInstanciasBar	Requirements::Requirement1	Requirements::VANESA Valor vol	Requirements::VANESABeatBar	Requirements::VanesaModel	Requirements::VANESAView
Use Case Model::crear instancias HeartModel - Singleton		↑			↑					↑					
Use Case Model::DJView	↑					↑						↑	↑		
Use Case Model::Heart					↑	↑				↑					
Use Case Model::Selecionar Modelo a usar						↑	↑	↑	↑						
Use Case Model::setearDJView													↑		
Use Case Model::VANESA	↑		↑			↑						↑	↑	↑	↑
Use Case Model::Ver 3 vistas simultaneamente				↑										↑	↑

7.3. Pass/Fail Ratio por tipo de Caso de Prueba

P/F Ratio = 100 %.

7.4. Bugs Identificados

7.4.1. Corregidos

Por un lado se identificaron y corrigieron los siguientes bugs:

- En el DJTestDrive no existía reproducción de sonido. Se corrigió agregando la siguiente linea de código en la posición 85 de la clase BeatModel:
  `sequencer.setMicrosecondPosition(0);`
- En VANESAModel la frecuencia del beatbar no era proporcional al volumen de micrófono, se soluciono haciendo un delay antes de notificar al observador del beatbar, este delay es inversamente proporcional al volumen.
- En DJView en el beatbar, en el menu para seleccionar el modelo a mostrar, cuando se selecciona un modelo cualquiera, este era mostrado en el view actual pero también se creaba una nueva instancia del mismo. Esto se solucionó en el controlador de cada modelo evaluando si ya existía una instancia del view creada por este controlador, en cuyo caso, de ser así, se trabaja sobre esta y no se crea un nuevo view.

7.4.2. No Corregidos

Por el otro lado, los siguientes bugs carecen de solución hasta el momento:

- En VANESAModel el nivel mínimo que toma la lectura del micrófono depende del micrófono, no solucionado.
- En VANESAView la vista la barra creada es de color naranja, cuando corremos este view junto con DJView el beatbar pasa a ser naranja, no solucionado.

8. Datos Históricos

8.1. Cantidad de horas de producción

Fecha de Reunión	Duración [Hs]#1	Total Horas x Persona [Hs]
Vie 13/06	8	32
Mie 25/06	6	24
Sab 28/06	7	28
Lun 30/06	6	24
Mie 02/07	12	48
Jue 03/07	14	56
Vie 04/07	8	24

9. Información Adicional

9.1. Conclusión

Implementar la consigna nos permitió aplicar diferentes patrones de diseño de software. El uso de una herramienta de manejo de la configuraciones nos permitió tener contacto con una experiencia símil profesional de desarrollo de software, así como también aprovechar las ventajas del uso de un sistema automático de versionado. Adicionalmente hicimos uso de un sistema de control colaborativo de revisión y desarrollo de software (Github). El modelado preliminar del sistema nos ayudó a descubrir nuevo requerimientos, prever futuras fallas del software e inferir patrones de diseño que agilizaron el desarrollo del proyecto. Una de las partes mas desafiantes del trabajo fue plantear los casos de pruebas para los componentes mas importantes del sistema. En cuanto a la práctica de la programación, amplió nuestros conocimientos acerca de las librerías que brinda JAVA para el desarrollo de GUI's. Desafortunadamente resulto muy frustrante no poder cumplir con nuestra voluntad inicial de llevar a cabo el proyecto implementando un modelo de desarrollo ágil (SCRUM), debido a que el tiempo y el esfuerzo dedicado tanto a la planificación como a las otras ceremonias, consumían el poco tiempo que disponíamos para la entrega del proyecto.

Referencias

[1] Sommerville, Ian *Software Engineering* 2011, Pearson Education, Inc., publishing as Addison-Wesley

Head First Design Patterns By Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra Publisher: Released: October 2004

[1] Eric Freeman, Bert Bates, Kathy Sierra and Elisabeth Robson *Head First Design Patterns* October 2004