**Trabajo Final**

**Ingeniería de Software 2016**

Grupo: Enigma

Integrantes:

Gatica Carolina



Heredia Marco



Vazquez Franco



**INDICE GENERAL**

**Nota de Entrega ..………………………. Pág 3**

**CM Plan ………………………………….. pág 6**

**Requerimientos ................................... pág 17**

**Arquitectura …………………………….. pág 21**

**Diseño e Implementación ……………. pág 25**

**Pruebas Unitarias y de Sistema ......... pág 30**

**Datos Históricos .................................. pág 37**

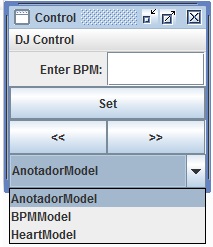
**Información Adicional ......................... pág 37**

**NOTA DE ENTREGA**

Se entrega el ejecutable *"IngSoft-2016-Enigma.jar"*equivalente al realease v1.0.0 de nuestro proyecto.

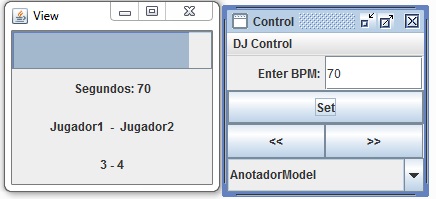
El cual tambien puede ser descargado del siguiente link:[https://github.com/francovqz/IngSoft-2016-Enigma/blob/master/ejecutable/IngSoft-2016-Enigma.ja r](https://github.com/francovqz/IngSoft-2016-Enigma/blob/master/ejecutable/IngSoft-2016-Enigma.jar)

El programa cuenta con 3 modelos entre los cuales se puede elegir al utilizar la drop down box.



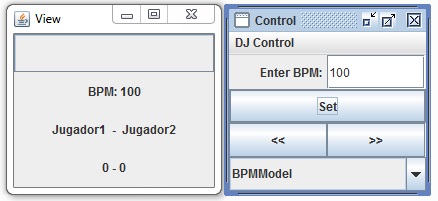
El modelo Anotador tiene la función de ser un anotador para algún juego y ademas cuenta con un temporizador para determinar (en caso de ser necesario) el tiempo de cada turno. Se lo utiliza de la siguiente manera:

* Con el boton << se le suma un punto al jugador 1.
* Con el boton >> se le suma un punto al jugador 2.
* Con el boton Set se inicializa un temporizador con la cantidad de segundos ingresada.
* Al ingresar el valor 0 al Set se reinicia el contador.

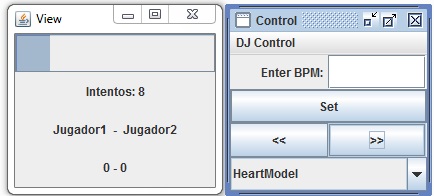


El modelo BPM realiza un sonido con una frecuencia por minuto igual a la que se le coloca al presionar el botón Set, esta frecuencia se inicializa en 90 al iniciar el modelo y se puede modificar con los botones de la siguiente forma:

* Con el boton << se disminuye en 1 la frecuencia
* Con el boton >> se aumenta en 1 la frecuencia



El modelo Heart simula los latidos del corazón, al presionar el boton >> se intenta crear otro modelo igual y debido a que se tiene implementado el patrón Singleton no se puede, en pantalla muestra un mensaje con la cantidad de intentos de creación de un nuevo modelo.



**Pass/Fail Ratio y Bugs conocidos**

El 95% de los test pasaron exitosamente, los test que fallaron están indicados en la sección de Testing, pero son errores menores que no impiden el funcionamiento correcto del modelo desarrollado.

Con estos Test pudimos encontrar que existen Bugs cuando se quiere ingresar un valor que no es entero en el tiempo para el temporizador. Este error solo tilda el programa por un tiempo muy pequeño debido a las excepciones que lanza el mismo. Este bug, no esta corregido en el momento de la entrega. Esto esta indicado en la herramienta Issues.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

http://www.htmlpublish.com/newTestDocStorage/DocStorage/e39d1379494044c3956142098924bcdc/Informe_images/Informe2x1.jpg

**PROYECTO "ANOTADOR"**

**CONFIGURATION MANAGEMENT PLAN**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Autores: Grupo Enigma**

* Gatica, Carolina
* Heredia, Marco
* Vazquez, Franco

**Versión del documento: 1.0.0**

**Indice de contenidos**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. Introduccion............................................... | | | pág 1 | | |  |
| 1.1 | Propósito y alcance..................................... | | pág 1 | | |  |
| 1.2 | Propósito de las prácticas de SCM.............. | | pág 1 | | |  |
| 1.3 | Glosario....................................................... | | pág 1 | | |  |
| 2. | Control de versiones.................................. | | pág 2 | | |  |
| 3. | Integracion continua................................... | | pág 3 | | |  |
| 4. | Gestion de defectos................................... | | pág 4 | | |  |
| 5. | Modo de reporte de defectos..................... | | pág 4 | | |  |
| 6. | Esquema de directorios.............................. | | pág 5 | | |  |
| 7. | Normas de etiquetado y nombramiento de archivos............ | | | | | pág 5 |
| 8. | Plan del esquema de ramas a usar............ | | pág 6 | | |  |
| 9. | | Politica de fusion de archivos y etiquetado..pág 6 | | |
| 10. | | Gestion de lanzamiento de releases | | ..........pág7 |
| 11. | | Change Control Board................................ | | pág 8 |
| 11.1 Miembros de la CCB................................... | | | | pág 8 |

11.2 Periodicidad de las reuniones de la CCB....pág 8

**1. Introducción**

**1.1 Propósito y alcance**

Este documento cubre el plan de Configuation Managment(CM) que se usará para el proyecto. Este abarca el seguimiento del control de versiones, integración continua, gestión de defectos, esquema de directorios, normas de etiquetado de archivos, esquema de ramas a usar, configuración de la CCB, entre otras cuestiones.Las herramientas usadas son de código libre, pertenecientes a distintos entes.

**1.2 Propósito de las prácticas de SCM:**

●Asegurar consistencia en las actividades prácticas de SCM

●Definir los organismos competentes para apoyar las prácticas de SCM

●Garantizar que no se realicen cambios de forma inapropiada

●Informar a los grupos e individuos involucrados sobre el estado del proyecto

●Garantizar integridad del producto a lo largo de su ciclo de vida

●Crear un historial de los estados actuales y anteriores de los productos en desarrollo

●Evitar problemas por falta de estandarización de las distintas actividades

●Mejorar procesos

**1.3 Glosario**

●CCB: Change Control Board

●CM: Configuration Managment

●SCM: Software Configuration Manager

●PCM: Project Configuration Management

●SCM: Software Configuration Management

●Tag: Etiqueta

●CI: Configuration Item

**2. Control de Versiones**

Como su nombre lo indica, le concierne el control de versiones de un componente o de un producto o programa. Se trata con conjuntos de items elementales que están bajo el control de configuracion, con el fin de producir un componente o producto coherente. En consecuencia esta actividad esta basada en una generación controlada de un número de versiones para items del proyecto.

En nuestro caso utilizaremos Git y GitHub como la herramientas para el control de versiones, que no es necesario confiar todo en un repositorio central lo cual nos permite que se pueda trabajar sobre copias de trabajo o hacer nuevos modelos de trabajo a partir de dicho repositorio como si fuera el proyecto general.

Se accedera al repositorio de trabajo creando una cuenta de usuario en GitHub y luego se obtendran los permisos de edicion correspondientes para poder editar el proyecto a traves de la siguiente direccion URL:

<https://github.com/francovqz/IngSoft-2016-Enigma>

**3. Integracion Continua**

La integracion continua constituye una parte importate al Sofware Configuration Management ya que se asegura que cada cambio realizado sea integrado al proyecto principal lo antes posible lo cual proporciona una informacion viable sobre la salud del proyecto.

La idea basica es integrar nuevas partes de codigo tan rapido como sea posible y testear las modificaciones en el contexto del proyecto en su totalidad. Para alcanzar este objetivo un servidor de integraion continua recupera la fuente del codigo del proyecto despues de cada cambio producido, lo compila y corre los tests correspondientes para verificar la funcionalidad.

Se utilizara Travis CI ya que es bastante convenido con GitHub. Tiene varias ventajas, entre ellas; no es necesario tener proporcionado un server de IC dedicado ademas de que el servicio es gratuito

para proyectos de open source.

Para usar Travis , el usuario debe entrar en su cuenta de GitHub, luego la pagina de perfil de Travis permite al usuario activar IC para cualquiera de sus proyectos. Despues de la activacion, un build sera ejecutado cada vez que se produzcan cambios (push) al repositorio.

Para decirle a Travis como construir el proyecto, un archivo config llamado .travis.yml debe ser agregado al directorio del repositorio. La configuración contiene el lenguaje del programa que es usado en el proyecto asi como tambien todas las versiones con las que el sofware sera testeado.

La direccion del servidor de Integracion continua de Travis es la siguiente:

<https://travis-ci.org/francovqz/IngSoft-2016-Enigma>

Junto con Travis se utilizará la herramienta Gradle para la realización de los builds y su comprobación. El link de descarga de Gradle es el siguiente:

<http://gradle.org/gradle-download/>

Se deberá incluir en el repositorio los archivos necesarios para que gradle realice los builds. Debido a que en el proyecto no utilizamos Test Unitarios por lo cual gradle será configurado unicamente para realizar los builds de java.

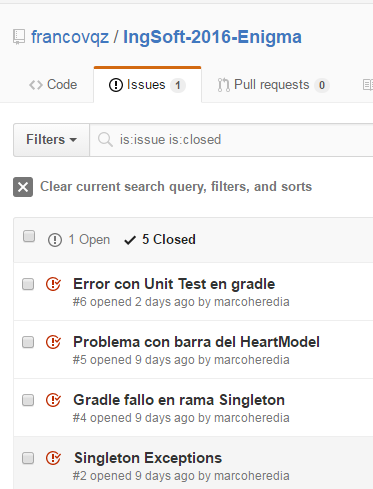
**4. Gestion de defectos**

Para nuestro proyecto utilizaremos la herramienta Issues proporcionada por GitHub para el reporte de defectos y su seguimiento. Para acceder a la misma es necesario entrar al apartado de Issues de GitHub con las cuentas de en las cuales tienen acceso como colaboradores al proyecto principal.

Los permisos de acceso a esta herramienta están incluidos al pertenecer al grupo de colaboradores al repositorio de GitHub.

Es importante revisar el apartado en el cual se publican todos los defectos para no tener problemas con los mismos.

<https://github.com/francovqz/IngSoft-2016-Enigma/issues>



**5. Modo de reporte de Defectos**

En caso de observar algún defecto en el proyecto es importante dirigirse inmediatamente a la herramienta Issues y crear un nuevo reporte explicando detalladamente el defecto que se encontró, el lugar en el cual se encuentra el defecto y una posible solución del mismo. Es importante garantizar que realmente el defecto existe antes de realizar un reporte.

El Issue Coordinator determinará quién se hará cargo de resolver el defecto y realizará un seguimiento del mismo hasta garantizar que el defecto está solucionado. Una vez solucionado se cerrará el Issue en la herramienta dejando activos únicamente los defectos que todavìa no fueron resueltos.

La prioridad de resoluciòn del defecto va a ser determinada por el Issue Coordinator, teniendo en cuenta los problemas que puedan causar el postergar la resoluciòn del defecto.

**6. Esquema de directorios**

El esquema de directorios que se va a usar va a ser el típico de un proyecto java con Maven. Esto seria: un directorio src/main/java en la cual se almacena el código, src/main/resourses para los recursos que se usan en el código java, src/test/java y src/test/resources para el código y recursos de los tests respectivamente . (véase la imagen).

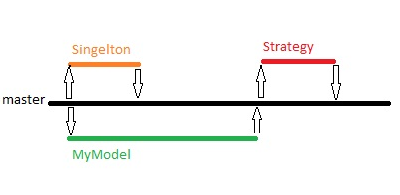


**7. Normas de Etiquetado y Nombramiento de Archivos**

Para el nombramiento de clases y metodos se utilizará la notación CamelCase y el nombre que se les ponga deberá ser un nombre identificatorio de la función que se lleva a cabo en esa clase o metodo. Para las etiquetas se utilizará un nombre identificatorio con la notación CamelCase seguido por una numeración A.B.C, siendo A la que representa cambios mayores en el programa, B la que identifica cambios menores y siendo C la que identifique unicamente a modificaciones menores de la parte grafica que no modifiquen ninguna de las funcionalidades.

**8. Plan del esquema de ramas a usar**

Se utilizará una rama para cada uno de los items a desarrollar, como en el proyecto se desarrollan 3 modelos bien diferenciables se va a desarrollar uno para cada uno (Singleton, Strategy y el modelo propio). Cada una de estas ramas llevará el nombre identificatorio sobre el tipo de modelo que proceso que se esta codificando. Una vez finalizada la codificación especifica se hará un merge con el master y se harán los modificaciones especificas para asegurarse que no haya ningún tipo de error con el merge.



**9. Política de fusión de archivos y etiquetado**

La fusión de archivos se realizará a base de los tipos de cambios que se iran realizando en los CI correspondientes.

Luego de las consultas necesarias con los miembros de trabajo y a partir de las necesidades requeridas por los clientes o por la exigencia de reparación de algun inconveniente o actualización del proyecto, se trabajarán en distintas funcionalidades que se desarrollaran en ramas o branchs del thread master inicial para luego acoplarse a éste segun ya esten completos los cambios y las necesidades que se requerian en un principio y que llevaron a cabo su desarrollo.

Los etiquetados, tags o label se realizaran cada vez que el proyecto se encuentre en un estado que cumpla con las exigencias o requerimientos por las que fueron ejecutadas su desarrollo en base a la necesidad originaria, describiendo adecuadamente las caracteristicas de las nuevas versiones producidas, como asi tambien los datos de miembros de trabajo involucrados, respetando un orden cronologico en los nombres correspondientes y teniendo en cuenta de respetar siempre la integridad del proyecto en su conjunto y totalidad.

**10. Gestion de lanzamiento de releases (cambios o agregados al proyecto)**

Este proceso es el responsable de planear, distribuir, y controlar la construccion, el testeo y los lanzamientos de "releases" asi como tambien del envio de nuevas funcionalidades al proyecto que sean requeridas por los clientes que usaran el producto siempre teniendo en cuenta de proteger y mantener la integridad de los servicios y features existentes.

Para asegurar que sean elegidos los CI adecuados para ser "release" y evitar fallas o errores durante el proceso de implementacion, la gestion de release debe verificar la correcta funcionalidad y su uso antes de la entrega mediante pruebas y tests.

El plan para la realizacion de los release debe definir:

Este proceso es el responsable de planear, distribuir, y controlar la construccion, el testeo y los lanzamientos de "releases" asi como tambien del envio de nuevas funcionalidades al proyecto que sean requeridas por los clientes que usaran el producto siempre teniendo en cuenta de proteger y mantener la integridad de los servicios y features existentes.

Para asegurar que sean elegidos los CI adecuados para ser "release" y evitar fallas o errores durante el proceso de implementacion, la gestion de release debe verificar la correcta funcionalidad y su uso antes de la entrega mediante pruebas y tests.

El plan para la realizacion de los release debe definir:

* Alcance y contenido del release
* Evaluacion de riesgo y perfil de riesgo para el release
* Usuarios/clientes afectados.
* Miembros de la CAB (Change Advisory Board) que aprobaron la recomendacion para el release
* Equipo responsable del release
* Estrategia de lanzamiento
* Recursos disponibles

A su vez las actividades del proceso de entrega y desarrollo incluyen:

1. Construir y probar antes de realizar el despliegue de produccion
2. Crear pilotos de prueba
3. Logistica y plan de envio:
4. Como y donde seran enviadas las unidades de release
5. Como son los plazos de entrega
6. Como llevar un rastreo del envio y obtener informacion de confirmacion
7. Metricas para el seguimiento y la determinacion del exito de los esfuerzos de implementacion del release.
8. Desarrollo y testeo de los releases
9. Documentacion del release y su construccion
10. Lanzamiento del packaging del release

**11. Change Control Board**

La CCB es el grupo encargado de verificar y autorizar cada cambio que se quiera realizar sobre el plan, el producto, el código o la documentación del mismo. Ningún cambio que tenga consecuencias o que produzca conflictos con el cliente o con algún miembro del personal que esté trabajando en el proyecto puede ser realizado sin la autorización de la CCB.

Las decisiones deben ser tomadas teniendo en cuenta las posibles consecuencias políticas o económicas que los cambios puedan ocasionar tanto con el cliente como con los miembros del proyecto.

La CCB evaluará cada pedido de cambio que se realice a cualquier miembro de la CCB, este pedido tendrá que ser realizado por escrito detallando los cambios que se quieren implementar.

**11.1 Miembros de la CCB**

**Engineering Manager – CCB Chair:**Su presencia es indispensable para realizar un control de cambios. Es el que dirige la CCB y es el que se encarga de coordinar las distintas áreas involucradas en el proyecto por lo tanto conoce las incidencias que un cambio puede causar en todas las áreas. En caso de no poder asistir tiene que autorizar a otro miembro de la CCB a ocupar su lugar.

**Release Manager – Issue Coordinator:**Su presencia es indispensable para realizar un control de cambios. Es el encargado de coordinar la forma en la cual se hacen y se entregan los realeases. Además se encargar de coordinar y solucionar todos los problemas que puedan surgir una vez realizados los cambios.

**Uber Scrum Team:**Su presencia es indispensable para realizar un control de cambios. Es el encargado de coordinar a todos los Scrum Masters de la empresa, y por lo tanto tiene un conocimiento de cómo afectarán los cambio con otros proyectos.

**Director Grafico:**Su presencia es opcional en caso que se lo requiera. En caso de necesitar realizar cambios que involucren la parte gráfica del proyecto es necesaria su participación.

**Configuration Manager:**Su presencia es opcional en el caso que se lo requiera. En el caso que los cambios a realizar involucren alguna parte del CM Plan es necesario su participación ya que es la persona que lo diseño.

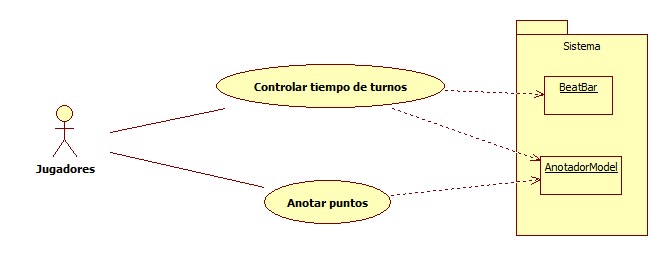
**11.2 Periodicidad de la reuniones de la CCB**

Las reuniones de la CCB se van a realizar 2 veces por semana durante el desarrollo del proyecto, con la posibilidad de llamar a reuniones especiales en caso de ser necesario avisando por email o por teléfono con 24hs de antelación.

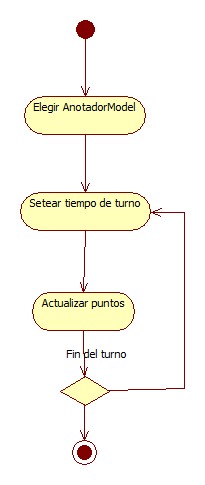
Requerimientos

**Se muestran a continuación los diagramas correspondientes a los requerimientos**

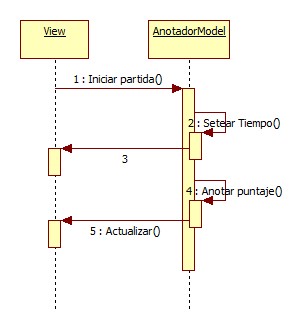
**Diagrama de Casos de Usos:**

[](https://github.com/francovqz/IngSoft-2016-Enigma/blob/master/resources/Caso%20de%20usos%20nuevo.jpg)

### **Diagrama de Actividades:**

[](https://github.com/francovqz/IngSoft-2016-Enigma/blob/master/resources/ActivityDiagram1.jpg)

### **Diagrama de Secuencias:**

[](https://github.com/francovqz/IngSoft-2016-Enigma/blob/master/resources/Diagrama%20de%20secuencia-requerimientos.jpg)

**Se presentaran a continuacion una enumeracion de los requerimientos funcionales y no funcionales que cumplen y satisfacen nuestro proyecto en su composición, luego de haber descrito el plan de desarrollo**

### **Requerimientos funcionales**

-Al inicio de la ejecución, se debe permitir seleccionar que modelo utilizar a partir de un dropdownbox

-Para que comience a funcionar bien el modelo requerido, primero es necesario seleccionarlo en el dropdownbox

-No se permite crear más de un modelo a la vez

-La temporización del modelo creado debe permitir por lo menos colocar una hora por turno

-El modelo no provee la emisión de una señal al terminar la temporización

-No se permite el pausado del temporizador

-Cada vez que se requiera recomenzar el temporizador se debe presionar el botón ‘set’ cargado con el tiempo (en segundos) que corresponda

-Para resetear el puntaje es necesario reiniciar la ejecución “seteando” un ‘0’ en el botón de set, ya que no es posible descontar la puntuación.

**Requerimientos no funcionales**

-El programa estará activo en promedio, el 95% del tiempo de ejecución

-El programa solo se ejecutara en plataforma Java

-La probabilidad de falla del sistema no debe ser superior a 0,05

-El programa no proporciona mensajes de error informativos

-La aplicación se realizara aplicando patrones de diseño de programación

-La aplicación es compatible con todas las versiones de Windows, desde Windows XP.

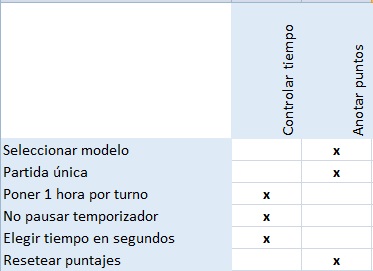
-La aplicación consume hasta 300 MB de memoria RAM

-El tiempo de respuesta para las interacciones de usuario con los botones del programa no supera los 10 milisegundos

-Las pruebas de software se gestionaran usando el testing correspondiente.

**Matriz de Trazabilidad**

**A partir de los requerimientos y los casos de usos, se establece la siguiente matriz:**

[](https://github.com/francovqz/IngSoft-2016-Enigma/blob/master/resources/Matriz%20trazabilidad.jpg)

**Arquitectura**

**Se ha utilizado el patrón de arquitectura MVC ya que brinda al proyecto distintas ventajas que justifican su aplicación:** Principalmente, MVC es un patrón de arquitectura que proporciona mantenibilidad de aplicación a través de la separación de la interfaz de usuario y la lógica del sistema, lo cual permite tener un códigomásfácil de leer y modificar (la modificación de una capa no afectara a las otras), y así acomodarse mejor para cambios futuros necesarios manteniendo la operatividad del sistema.

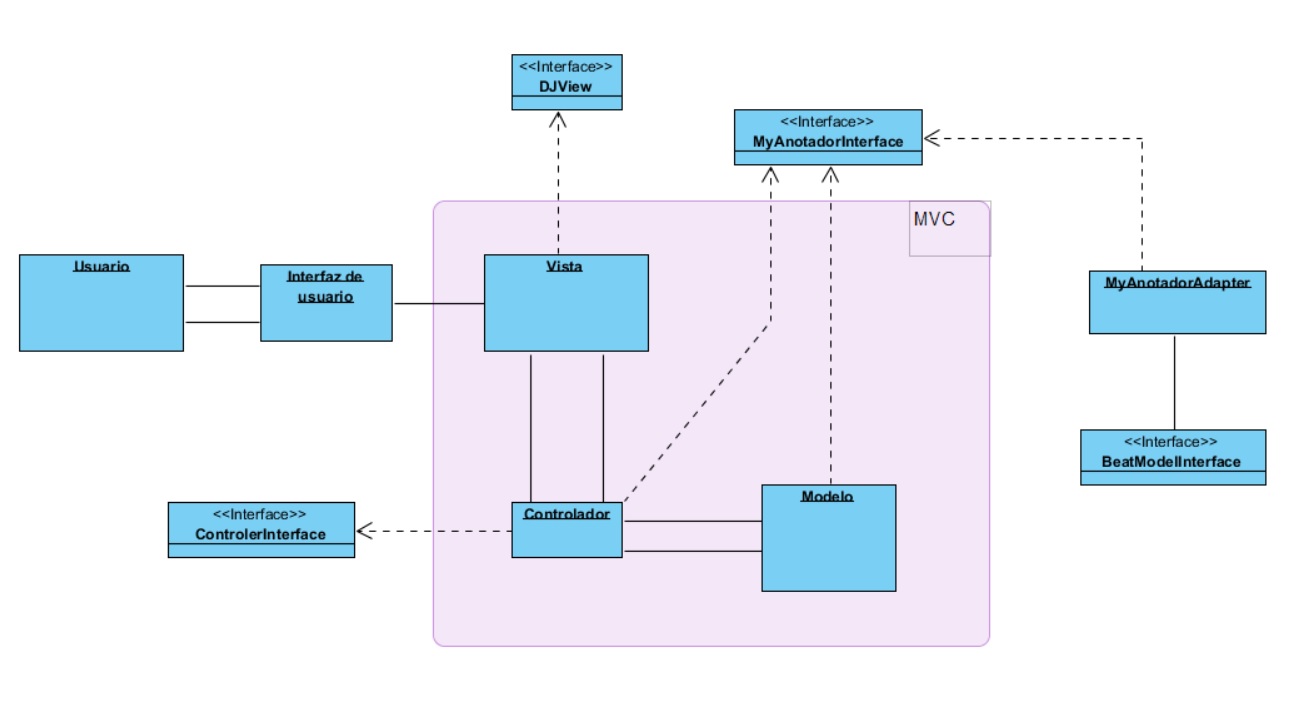
Con esta independencia de "capas" (MVC) se proporciona la habilidad de cambiar de la implementación de un modelo a otro sin problemas. En términos de interacción, la interfaz de usuario puede mostrar múltiples vistas de la misma información y al mismo tiempo.

Otra ventaja del patrón es que al separar el sistema en tres piezas de funcionalidad (MVC) se facilita el uso de Unit Testing pudiendo testear libremente el modelo del controlador y la vista, y viceversa.

Se debe tener en cuenta que el modelo puede tener un costo de dificultad en su desarrollo pero en cambio se tienen las ventajas que se enunciaron anteriormente, por lo tanto es posible que MVC no sea adecuado para aplicaciones pequeñas ya que habrá consecuencias contraproducentes de rendimiento y diseño.

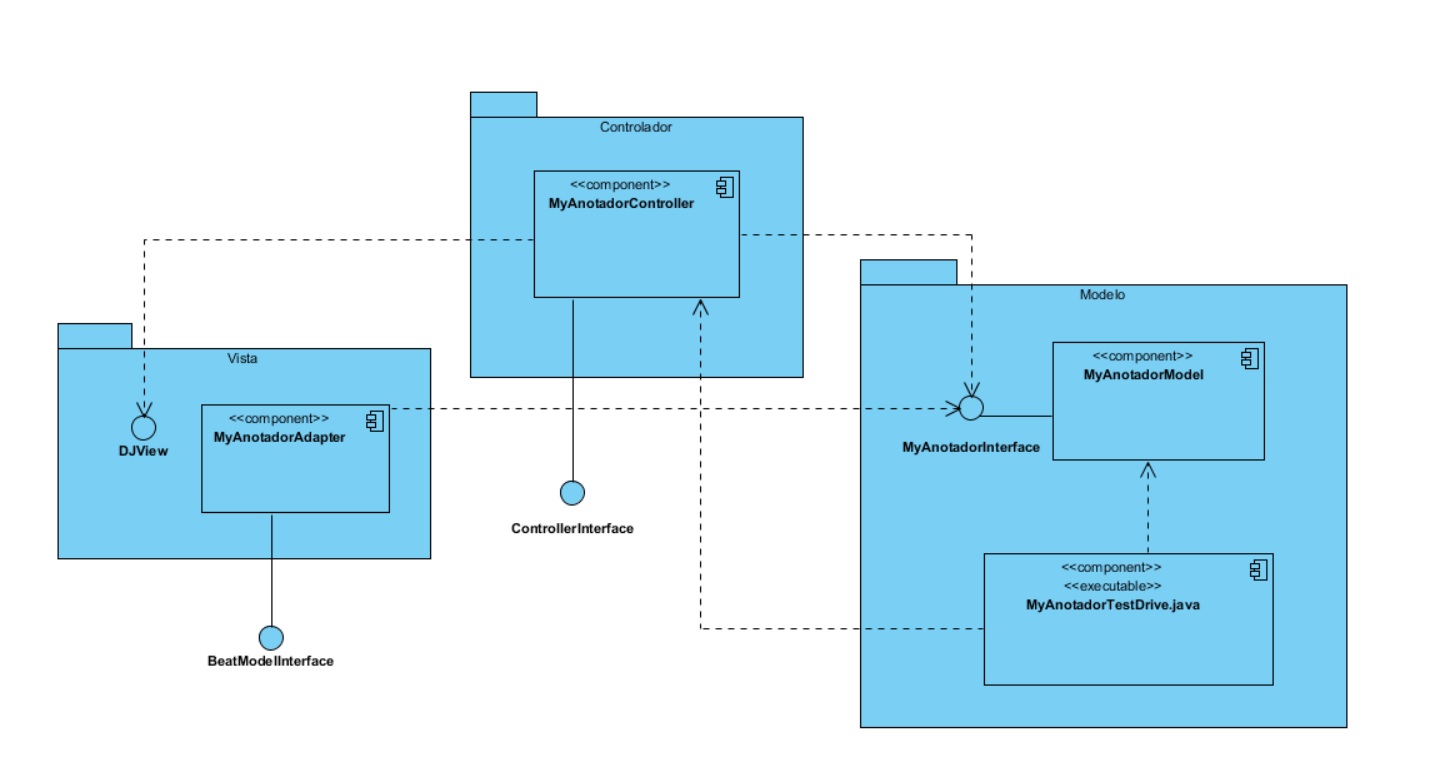
A continuación se enseña el diagrama de arquitectura general del modelo desarrollado que muestra a grandes rasgos la interacción entre los elementos más importantes

**Diagrama general de arquitectura del sistema**

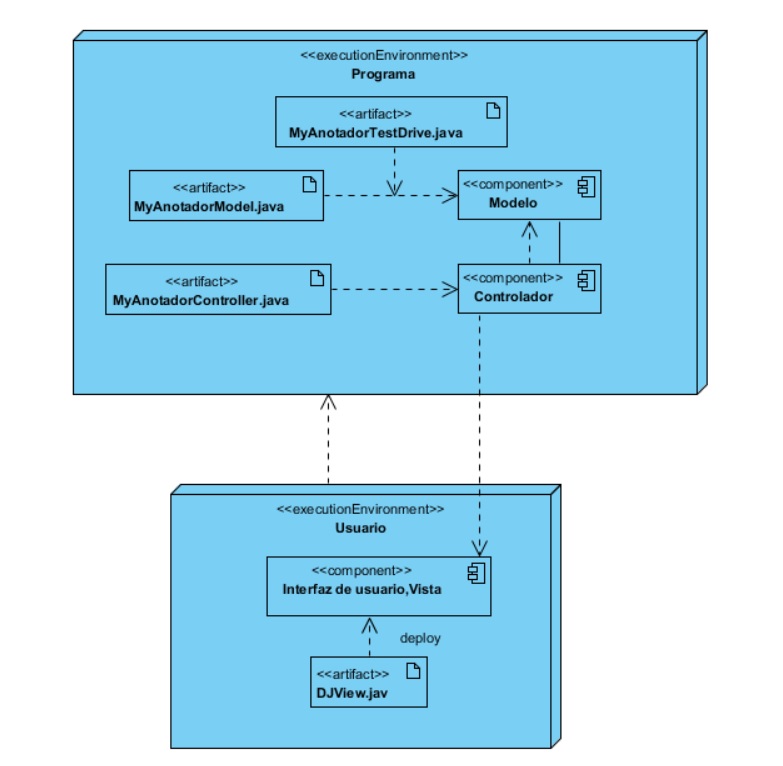


**Se ha tenido en cuenta solo el proyecto creado por el grupo en cuestión para no acomplejar los siguientes diagramas con los detalles de los modelos prerealizados de HeartModel y BeatModel.** A partir del patrón de arquitectura utilizado se detallan los diagramas de componentes y de despliegue del trabajo.

#### Diagrama de Componentes



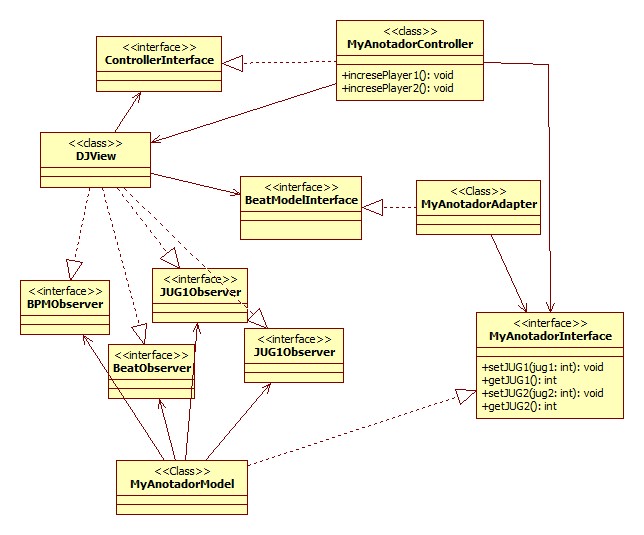
#### Diagrama de Despliegue



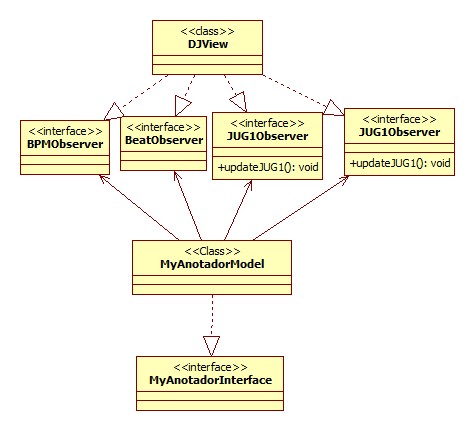
**DISEÑO E IMPLEMENTACION**

**Diagramas de clases**

**Diagramas de clases:**

****

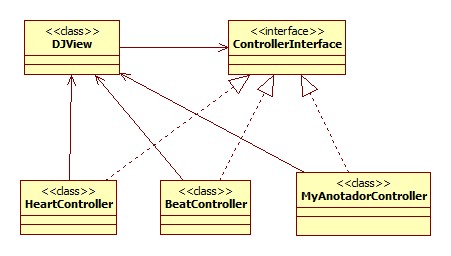
Se incluye el siguiente diagrama general de clases, que describe la relación del DJView y nuestro proyecto "Anotador".



Se utilizó el patrón observer en el desarrollo de nuestro proyecto, ya que nuestro anotador debe notificar a distintos observadores cuando cambie el estado del marcador de los jugadores.

En este caso, el observador concreto es el DJView; los observadores son el BPMObserver, el BeatObserver, el JUG1Observer y el JUG2Observer; el sujeto es MyAnotadorInterface y el sujeto concreto MyAnotadorModel.

Se usó también el patrón Strategy, ya que se puede intercambiar entre las distintas aplicaciones de uso, dentro de las cuales se incluye la nuestra.

****

**Diagrama de paquete:**

Como muestra la siguiente figura, dividimos las clases en cinco categorías:

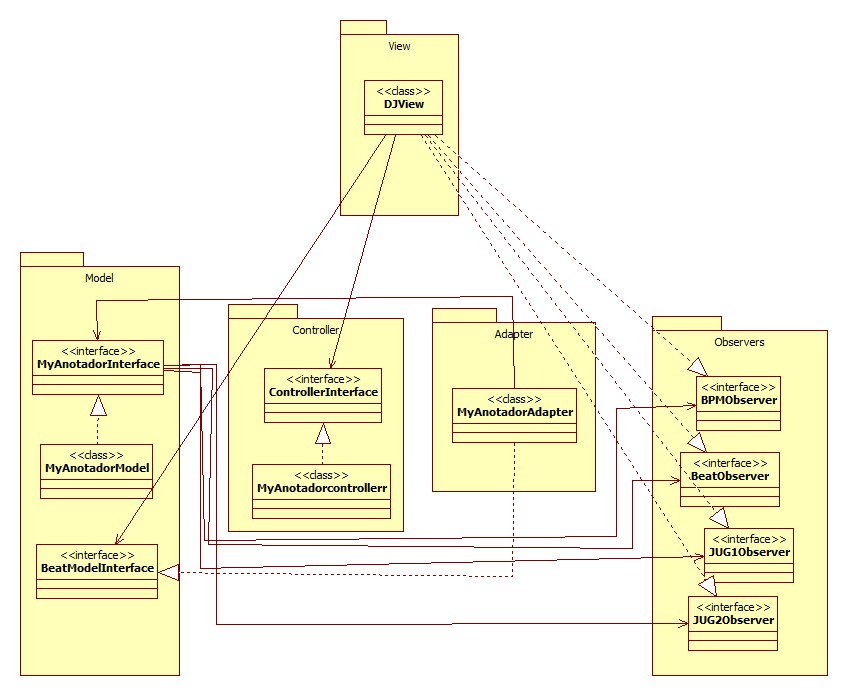
1.View

2.Model

3.Controller

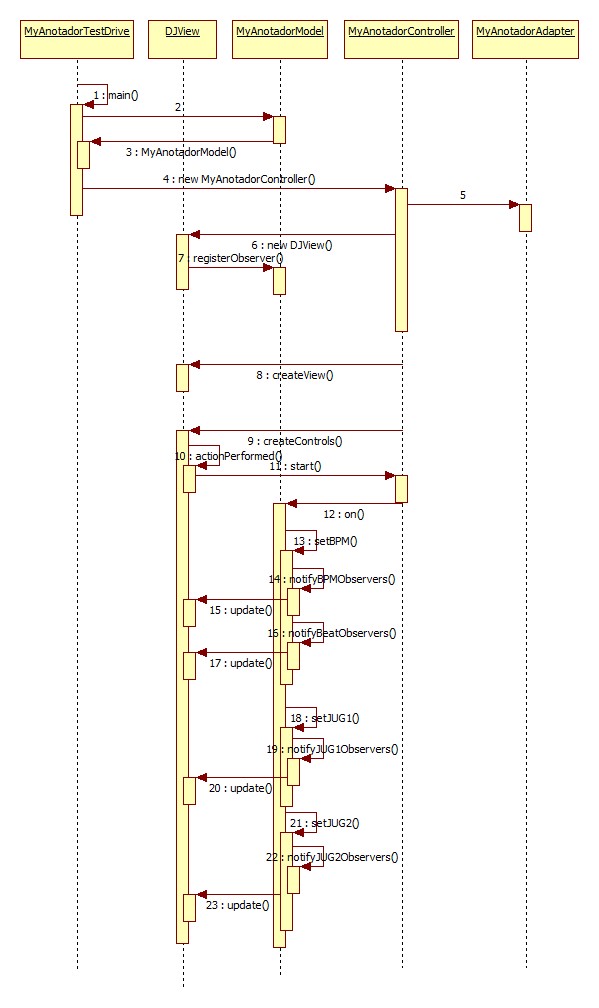
4.Adapter

5.Observers



### **Diagrama de Secuencia**

En el siguiente diagrama se muestran las principales interacciones entre las distintas clases que sigue nuestra aplicación.

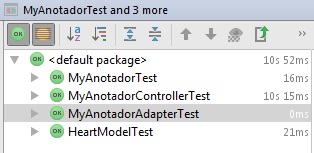


# PRUEBAS UNITARIAS Y DE SISTEMA

### Test Unitarios

Se realizó Unit Test sobre todo el código creado por nosotros. Para ello se utilizo JUnit 4 y Coverage para medir la cobertura de los tests integrados a IntelliJ.

Para correr los test se necesita tener un copilador de Java y ejecutarlos desde la misma, de esa forma saldrá un mensaje detallando los Test que fallaron y los test que fueron exitosos. En nuestro caso se utilizo la aplicación IntelliJ

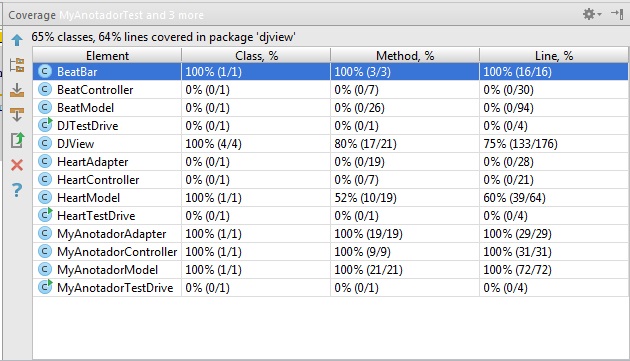


Como se puede observar en la imagen todos nuestros Unit Test pasaron de forma exitosa.

Test

La cobertura abarcada por los Unit Test fue de todo el código que implementamos, solo quedaron sin testear los metodos de las clases que ya se encontraban implementados en el programa.

Para medir la cobertura se utilizó la herramienta Coverage. Para utilizarla desde IntelliJ, simplemente había que correr los Unit Test con la opción de medir cobertura y se obtuvo los siguientes resultados.



Como se puede apreciar en la imagen, **se cubrió el 100% de las clases, métodos y líneas del código** implementado por nosotros, a excepción de la clase MyAnotadorTestDrive en la cual solo se encuentra el método main() que inicializa el programa. Pero a excepción de ese método se cubrió el 100% de todo el código implementado.

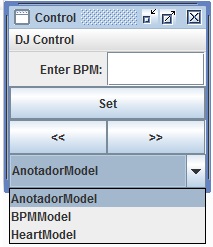
El **Pass/Fail rate de los Test Unitarios fue del 100%** por lo que podemos decir que en esta sección no se encontró ningún bug.

**Test de Sistema**

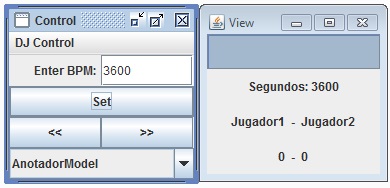
Se realizarán las pruebas de sistemas con respecto a los requerimientos detallados anteriormente:

**Requerimientos funcionales**

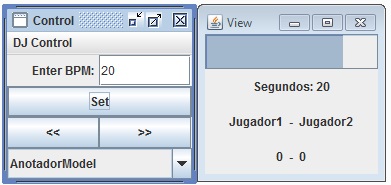
* Al inicio de la ejecución, se debe permitir seleccionar que modelo utilizar a partir de un dropdownbox
* Para que comience a funcionar bien el modelo requerido, primero es necesario seleccionarlo en el dropdownbox



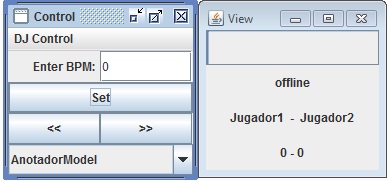
* La temporización del modelo creado debe permitir por lo menos colocar una hora por turno
* El modelo no provee la emisión de una señal al terminar la temporización



* No se permite el pausado del temporizador
* Cada vez que se requiera recomenzar el temporizador se debe presionar el botón ‘set’ cargado con el tiempo (en segundos) que corresponda



* Para resetear el puntaje es necesario reiniciar la ejecución “seteando” un ‘0’ en el botón de set, ya que no es posible descontar la puntuación

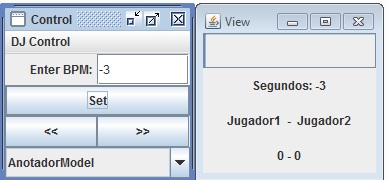


**Requerimientos no funcionales**

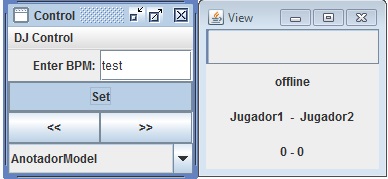
El programa se testeo en Windows 7 y Windows 10 en las cuales se tenía instalado Java JDK1.8.0 sin ningún problema de compatibilidad.

**Pruebas alternativas de valores críticos**

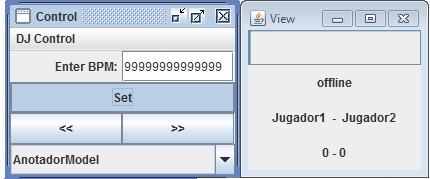
Se intento ingresar valores negativos de tiempo y el programa siguió funcionando normalmente indicando la cantidad de segundos negativa pero sin realizar ningún tipo de conteo en el temporizador.



Se intento ingresar letras en el campo del tiempo y el programa se detuvo durante unos milisegundos, no indica ningún mensaje y al ingresar un tiempo positivo el programa vuelve a funcionar normalmente. Se obtuvieron mensajes de excepción



Al ingresar un numero demasiado grande el programa no realiza el conteo del temporizador, al volver a ingresar un valor correcto vuelve a funcionar normalmente. Se obtuvieron mensajes de excepción.



El **Pass/Fail rate de las Pruebas de Sistema fue del 95%** solo se obtuvieron excepciones al realizar pruebas con valores críticos, pero en general los resultados fueron los esperados.

**Smoke Test**

Como Smoke Test o prueba de humo, la cual es un testing rápido que se realiza sobre aspectos funcionales no tanto para encontrar bugs sino para asegurarse que la funcionalidad básica del software o de una parte del software se encuentre estable y responda al comportamiento esperado se realizó pruebas similares a las que se realizo al probar los requerimientos funcionales. Entre los cuales se destacan los siguientes:

* Se le sumo 3 puntos al Jugador 1 al hacer click 3 veces sobre el botón <<
* Se le sumo 2 puntos al Jugador 2 al hacer click 2 veces sobre el botón >>
* Se inicio el temporizador en 10 segundos al colocar un 10 y seleccionar Set

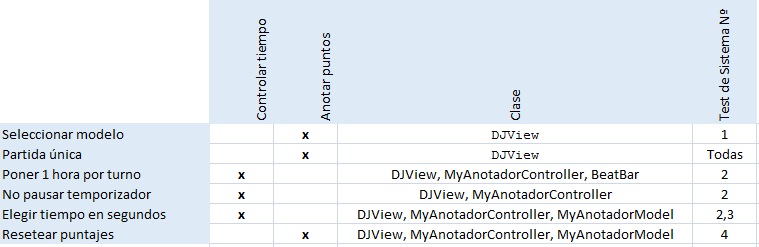
El resultado obtenido fue el siguiente:



Del cual se puede ver que el programa realizo de forma correcta lo que se solicito.

**Matriz de trazabilidad**

La matriz en la cual se relacionan los casos de usos, los requerimientos, las clases y las pruebas de sistema es la siguiente:

[](https://github.com/francovqz/IngSoft-2016-Enigma/blob/master/resources/matriz.jpg)

Para la organización de los bugs o problemas encontrados se utilizó la herramienta Issues incluida en github. El link del mismo es el siguiente:

<https://github.com/francovqz/IngSoft-2016-Enigma/issues>

Se encontraron entre otros los siguientes bugs o problemas:

* Problema con BeatBar: Se veía el HeartModel en la barra por mas que se estuviese en otro modelo. **CORREGIDO**
* Excepciones en tiempo de ejecución al correr la implementación del singleton. **CORREGIDO**
* Errores con gradle al subir el código. **CORREGIDO**
* Errores de travis con los Unit Test: Cuando travis corre el gradle indica que uno de los test falla por un HeadlessException pero al correrlos manualmente los test pasan sin problemas **SIN CORREGIR**
* Al ingresar caracteres que no son enteros positivos el programa deja de funcionar correctamente. **SIN CORREGIR**

**Datos Históricos:**

El principal inconveniente que tuvimos fue encontrar una idea que se adapte a lo requerido en el trabajo.

A partir de plantearnos la idea del anotador, nos fuimos dividiendo las tareas según las aptitudes de cada uno, intentando repartirlas equitativamente y con ello el esfuerzo requerido.

**Información Adicional:**

De cada parte del trabajo fuimos afianzando los conocimientos teóricos de la materia.

Tuvimos errores a la hora de implementar el patrón Observer, ya que en un principio se nos abrían múltiples ventanas por cada ejecución.

A la hora de desarrollar los diagramas, pusimos demasiados detalles que dificultaban la comprensión. Finalmente, intentamos dejar sólo la información que creíamos relevante para el desarrollo de nuestro modelo.