Universidad Nacional de Cordoba

Aplicación de las técnicas de Ingeniería de Software para un proyecto de programación

Informe del Trabajo Final

Grupo: "Group Club"
Integrantes:
Almendros, Lucas Matías.
Amaya, María Florencia.

Contents

1. Introducción:

1.1 Resumen:

En este proyecto se implementarán métodos y políticas de desarrollo de Software, aprendidos en el transcurso de la materia, para hacer un trabajo eficiente y entregar un producto aceptable.

Dentro de las tareas se encuentra la de hacer un modelo original que implemente ciertas características predeterminadas por el código en el cual se desarrollara.

El modelo "PingIt" es un programa para probar y medir la velocidad de conexión de internet con un servidor en particular. El motivo principal por el cual se la desea producir es para tener una herramienta para un producto futuro donde el usuario necesite saber su estado de conexión para mejorar el uso de la aplicación en cuestión (por ejemplo, un videojuego multijugador online).

1.2 Objetivos:

Se quiere lograr un programa final donde el usuario pueda establecer primero la dirección deseada, sin importar que sea una URL o un Ip Adress, luego se establece la frecuencia en la cual desea probar la conexión y se inicia. El ping actual aparece en otra ventana, junto con una barra que hace una animación cada vez que se manda un ping, y demás estadísticas como el ping promedio, el estado y cantidad de los paquetes, la frecuencia actual, etc.
Si se desea se puede aumentar y disminuir la frecuencia por medio de botones durante el ciclo. También hay un botón para detenerlo.
El código del modelo tiene que poder ser flexible para adaptarse no solo a la Vista propia del modelo, sino también a la otra Vista del trabajo.
Por otro lado hay que hacer unos cambios al código citado en el enunciado como la limitación a la clase HeartModel de solo poder crear una instancia.
Por último, se deben poder utilizar los tres modelos a la vez con tres vistas y distintas y, en otra ocasión, en la vista del BeatBar, cambiar el modelo que se desea observar.

2. Nota de Entrega:

Feature Type	Feature	Descripción
Modificación	Instanciamiento	Modificación del HeartModel para que solo pueda ser instanciado una vez.
Nuevo	Modelo PingItModel	Nueva nuevo modelo para el proyecto PingItModel que hace ping a host remotos para saber si están conectadas y la velocidad de envió de paquetes
Nuevo	Vista PingIrVista	Nueva vista para el Ping pudiendo apreciar las nuevas funcinalidades como modificar el host remoto y poder ver diferentes estadisticas asociadas al ping.
Nuevo	Muestra de todas las Vistas	Se muestra los tres modelos trabajando al mismo tiempo con sus diferentes vistas
Modificación	Una vista para todos los modelos	En una vista se puede seleccionar que modelo utilizar
Bug Conocido	Intercambio de modelos en una vista	La funcionalidad de elegir qué modelo se quiere observar en la Vista no está implementada correctamente ya que con cierta combinación de vistas, el programa deja de responder correctamente.

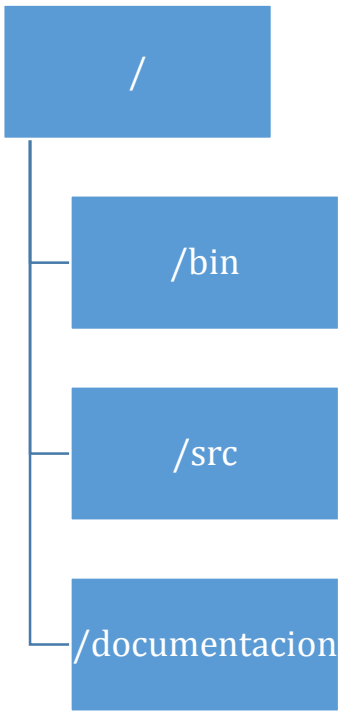
Manejo de las Configuraciones:

2.1 Dirección y forma de accesos a la herramienta de control de versiones:

Para el control de configuraciones vamos a usar la herramienta de control de versiones GIT http://git-scm.com/. GIT es software libre distribuido bajo los términos de la GNU General Public License versión 2.
El repositorio web a utilizar será GItHub, donde se puede almacenar el código de forma pública y gratuita.
Dirección del repositorio: https://github.com/almendrosl/PingIt/
Utilizaremos además el software open source TortoiseGit como un **Windows Shell Interface** para GIT basado en **TortoiseSVN** para facilitarnos la utilización de las operaciones relacionadas con el control de versiones como "Commit", "Pull", "Push" de GIT.

2.2 Esquema de directorios y propósito de cada uno:

/:Raiz del proyecto
/bin: archivos binarios
/src: código fuente en .java



2.3 Normas de etiquetado y de nombramiento de los archivos:

El nombre de los tags que se hará sobre el branch principal ira en formato X.Y.Z donde X es el número de releáse, Y es un número que indica funcionalidades agregadas y Z indica correcciones realizadas. El primer releáse será el 0 por lo que el primer tag es 0.0.0 luego se agregaran funciones y por cada función agregada aumenta Y ej. 0.1.0, y de acuerdo sobre que versión se corrige algún bug se aumenta Z.
El nombramiento de los archivos .java deberá ser en formato CamelCase y el nombre ser lo que representa, en caso de ser un unit test deberá tener la palabra "Test".

2.4 Plan del esquema de ramas a usar (y en uso):

El modelo de branching que se usara es "Early". Porque solo tendremos un releáse. Cada vez que se quiera agregar una nueva funcionalidad o arreglar un bug se hace una rama y antes de hacer Merge se hace rebase para evitar conflictos y agregar toda la funcionalidad que tiene el trunk.

2.5 Políticas de fusión de archivos (o sea mergeo) y de etiquetado de acuerdo al progreso de calidad en los entregables:

El mergeo sobre la rama principal se hará solo cuando se pasen todos los unit test y se deberá primero contar con la última versión del programa del trunk del repositorio web en su workspace para hacer un mergeo con su versión ramificada.
El etiquetado será como se explicó anteriormente.

2.6 Forma de entrega de los "releases", instrucciones mínimas de instalación y formato de entrega:

Se entregaran los releases en formato JAR es un ejecutable no instalable llamado PingIt.jar pero necesita para funcionar Java Runtime Environment(JRE) que puede ser descargado del siguiente link: https://www.java.com/es/download/.
El software solo puede ser instalado en el sistema operativo Windows.
El comando para ejecutar el código es `java –jar PingIt.jar`

2.7 Listado y forma de contacto de los integrantes del equipo:

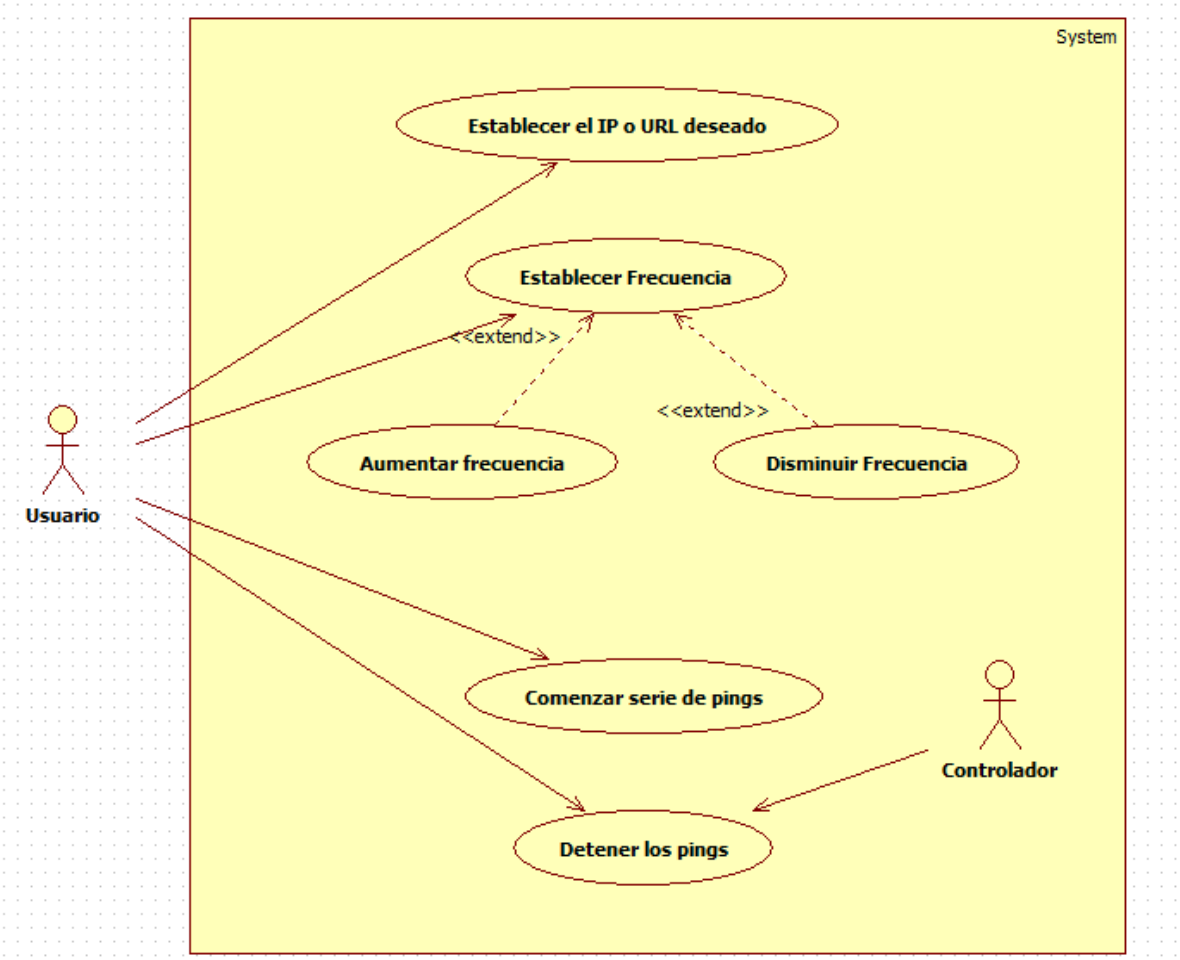
Integrantes	Contacto	Roles
Almendros, Lucas Matías	alucas17m@gmail.com 0351-153724780	Colaborador general. Administrador de Configuraciones. Tester. Desarrollador. Diseñador.
Amaya, María Florencia	floppies.amaya@gmail.com 0266-154222806	Colaborador General. Arquitecto. Desarrollador. Gestor de Proyectos.

3. Requerimientos:

3.1 Diagramas:

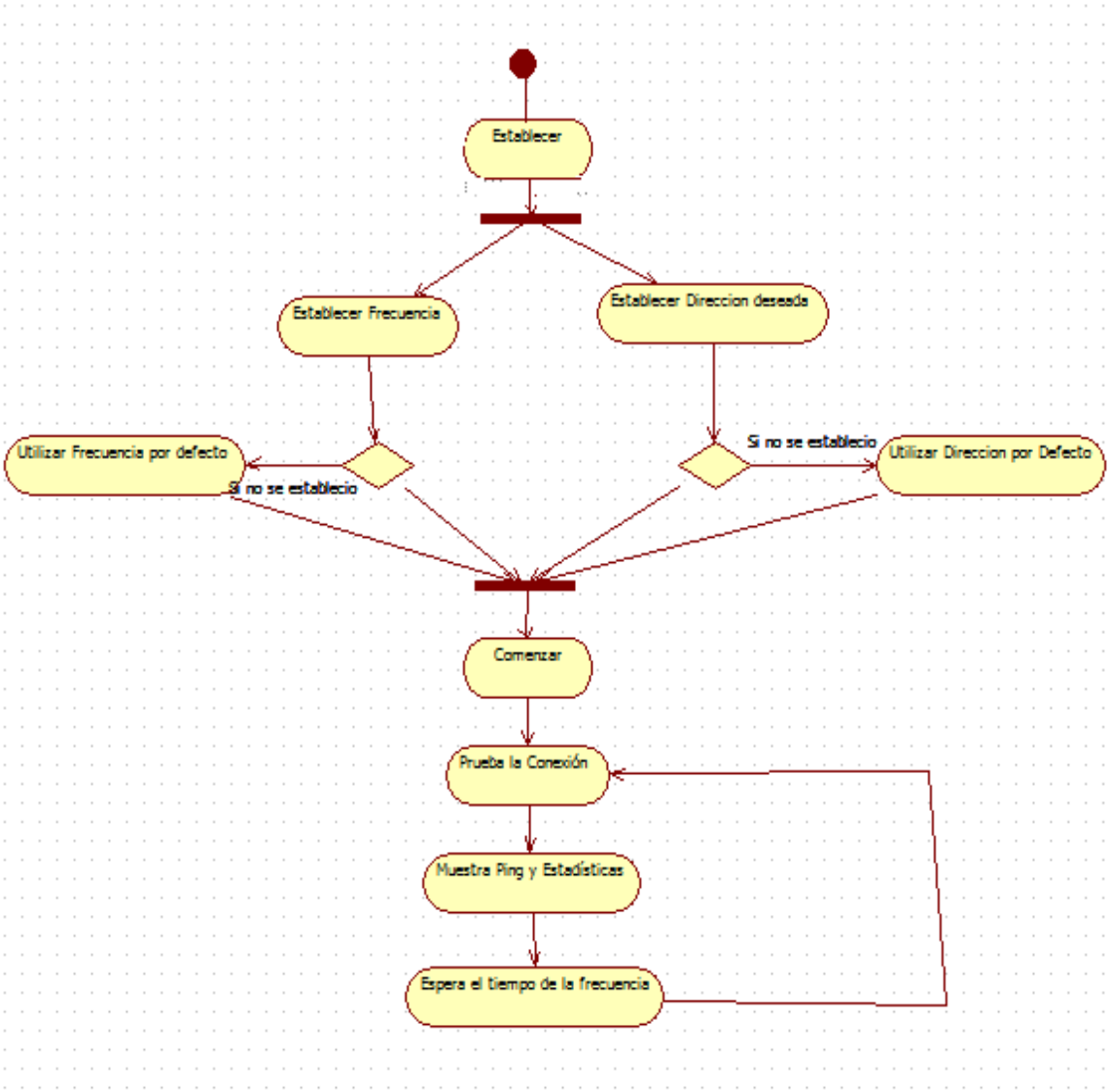
- Diagrama de Casos de Uso:

  A continuación mostramos los Casos de Uso para la aplicación de nuestro modelo.

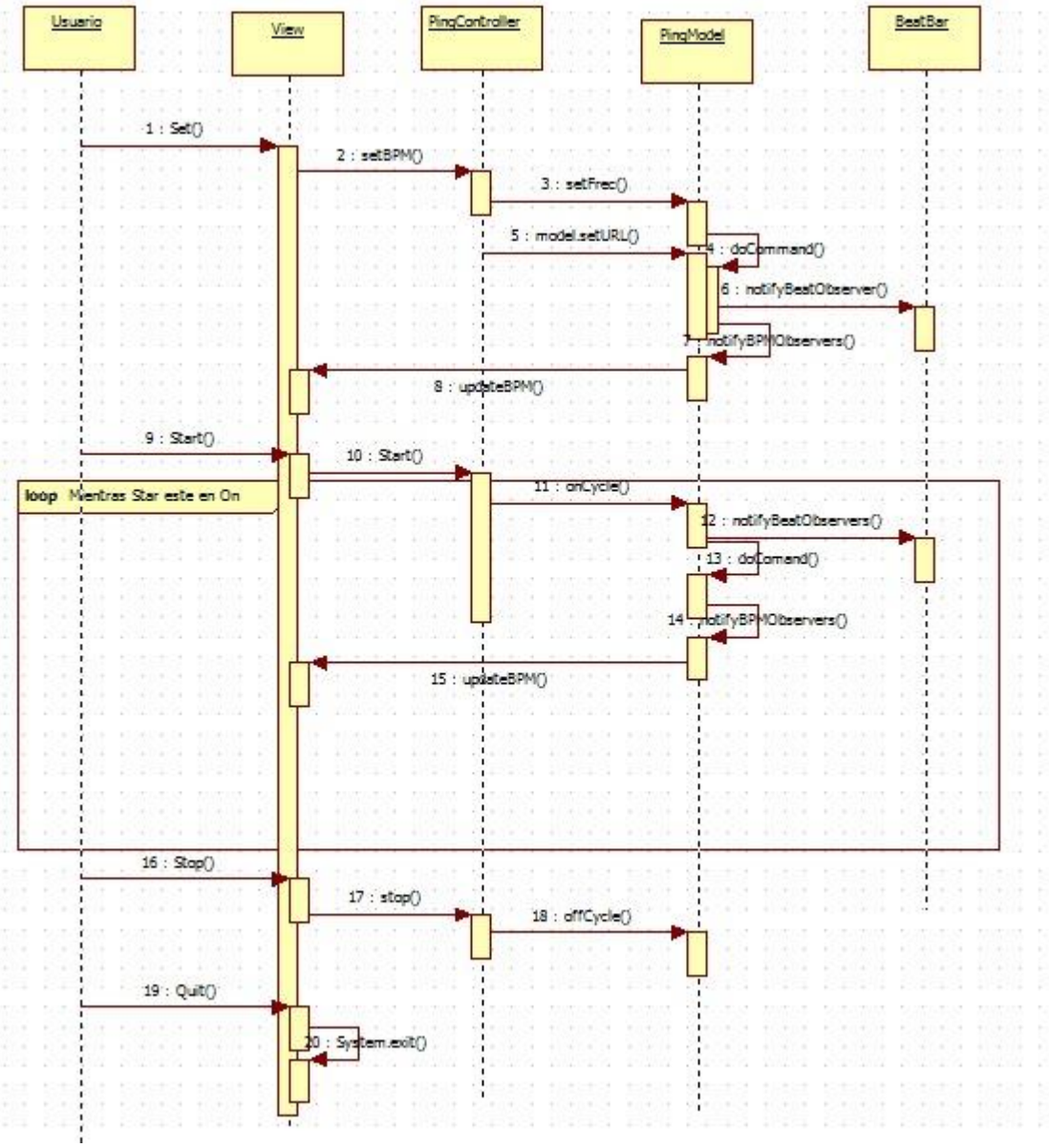


- Diagrama de Actividades:

  La actividad que se ve aplicada es cuando se usar el modelo haciendo un ciclo para ver el ping cada cierto tiempo, es decir teniendo en cuenta una frecuencia.



- Diagrama de Secuencia:



3.2 Requerimientos funcionales:

F.1.1) El Heart Model solo se puede instanciar una vez.
F.1.2) En la ventana de control del Beat Controller se debe tratar de generar nuevas instancias cada vez que se hace click en ">>".
F.1.3) La ventana del BeatBar debe mostrar la cantidad de intentos de creación de instancias donde la frecuencia cardíaca.
F.2.1) Se debe poder ejecutar el modelo en una vista similar a la que tienen el HeartTestDrive.

F.2.2) En “Enter BPM” se establece la frecuencia a la que se quiere realizar pings a la dirección fija que previamente se desea.
F.2.3) Al apretar “Start” si no se estableció previamente, se hacen pings con una frecuencia por defecto de 1 cada 1 seg. Se debe poder disminuir con “<<” y “>>” con una magnitud de 0.1 segundo.
F.2.4) El ping se debe observar en el BeatBar.
F.2.5) Cuando se presiona el botón “Set” además de establecer la frecuencia se realiza un ping unitario.
F.3.1) Con la nueva vista se debe poder establecer una dirección diferente para probar la conexión.
F.3.2) Se debe poder detener el proceso de mandar paquetes.
F.3.3) Se debe poder ver la frecuencia (pings por seg) y el tiempo de respuesta en milisegundos.
F.3.4) Se pueden observar datos como el promedio de Ping, el estado de los paquetes.
F.3.5) Si no se establecen ni la frecuencia ni el ping el programa utiliza los que se tiene por defecto.
F.4.1) Se deben poder ejecutar y ver simultáneamente cada modelo.
F.5.1) En otro TestDrive se debe poder elegir qué modelo se quiere ver en la BeatBar usando un menú en el BeatBar.

3.3 Requerimientos no funcionales:

NF.1.1) Debe haber una frecuencia de pings mínima porque el tiempo de respuesta puede llegar a ser más alto que la frecuencia de pedido. Además estaría ocupando de una manera fútil la conexión. (**Eficiencia**)
NF.1.2) Los tres modelos deben poder funcionar conjuntamente. (**Eficiencia**)
NF.1.3) La interfaz de usuario debe ser de fácil comprensión. (**Usabilidad**)
NF.1.4) El programa podrá usarse en computadoras con Windows que tengan al inglés como lenguaje predeterminado. (**Adaptabilidad**)
NF.1.5) El código del modelo nuevo debe ser adaptable y flexible para poder agregarlo a otros sistemas que necesiten su funcionalidad principal. (**Mantenibilidad**)
NF.2.1) El programa debe detenerse solo después de dos horas para evitar el uso no adecuado, como ataques ICMP. (**Ética**)

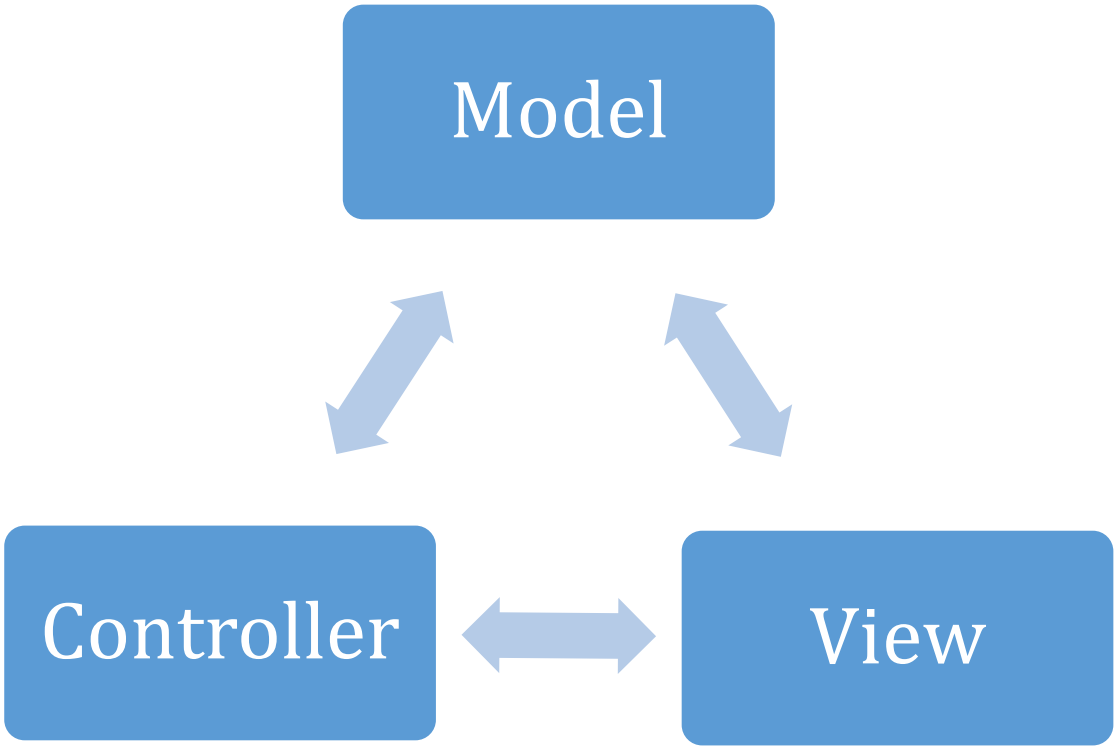
3.4 Matriz de Trazabilidad:

Se muestra la matriz de trazabilidad preliminar que relaciona los casos de uso con los requerimientos. Hay que tener en cuenta que nuestros casos de uso son exclusivos para el uso del modelo creado por nosotros.

	Establecer dirección deseada	Establecer frecuencia	Aumentar frecuencia	Disminuir frecuencia	Comenzar	Detener
F.1.1						
F.1.2						
F.1.3						
F.2.1		X	X	X	X	X
F.2.2		X			X	
F.2.3		X	X	X		
F.2.4					X	
F.2.5		X				
F.3.1	X					
F.3.2						X
F.3.3					X	
F.3.4					X	
F.3.5					X	
F.4.1						
F.5.1						
NF.1.1		X	X	X		
NF.1.2						
NF.1.3						
NF.1.4						
NF.1.5						
NF.2.1						X

3.5 Diagrama de Arquitectura Preliminar:

En la figura siguiente se muestra un diagrama simplificado de la arquitectura de nuestro modelo. Nos basamos en el patrón de Arquitectura MVC (Model View Controller):



4. Arquitectura:

Debido a que los requerimientos no funcionales de mantenibilidad y reutilización de código los más importantes y ya que se usan varias funcionalidades de un software sobre la misma interfaz del ejemplo y sobre una interfaz original se decidió hacer el patrón de arquitectura MVC (Model View Controler) que separa elementos de un sistema, permitiéndoles cambiar de forma independiente.
Por ejemplo, agregar una nueva vista o cambiar una vista existente puede hacerse sin modificación alguna a los datos subyacentes en el modelo.
El MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Resumiendo los conceptos:

- **Modelo**: Se encarga de los datos, generalmente (pero no obligatoriamente) consultando la base de datos. Actualizaciones, consultas, búsquedas, etc. todo eso va aquí, en el modelo.

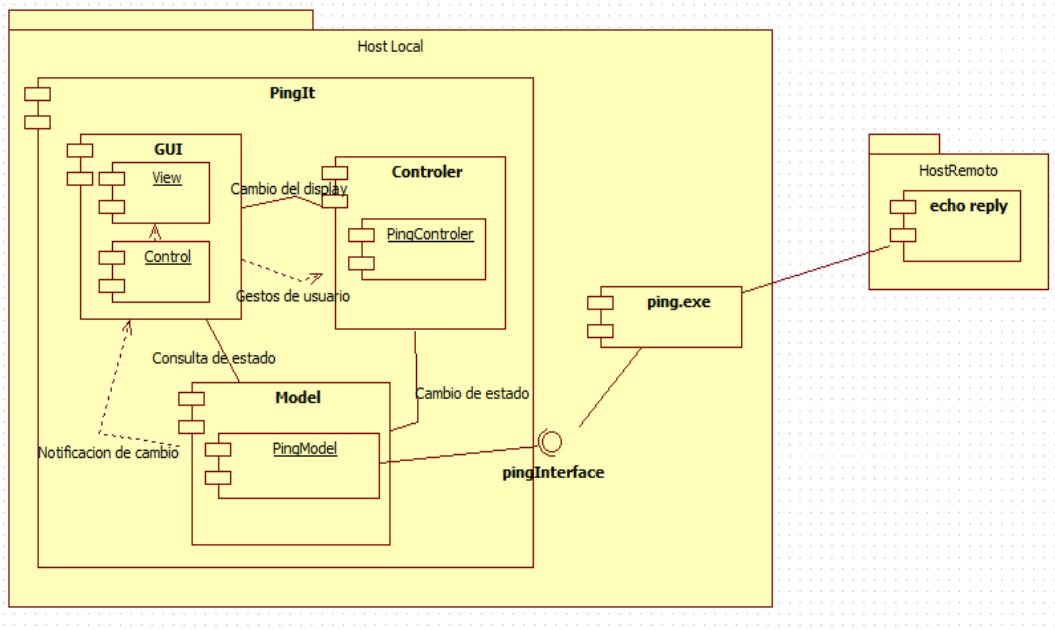
- **Controlador**: Se encarga de controlar, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.

- **Vistas:** Son la representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica va aquí. Ni el modelo ni el controlador se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la vista.

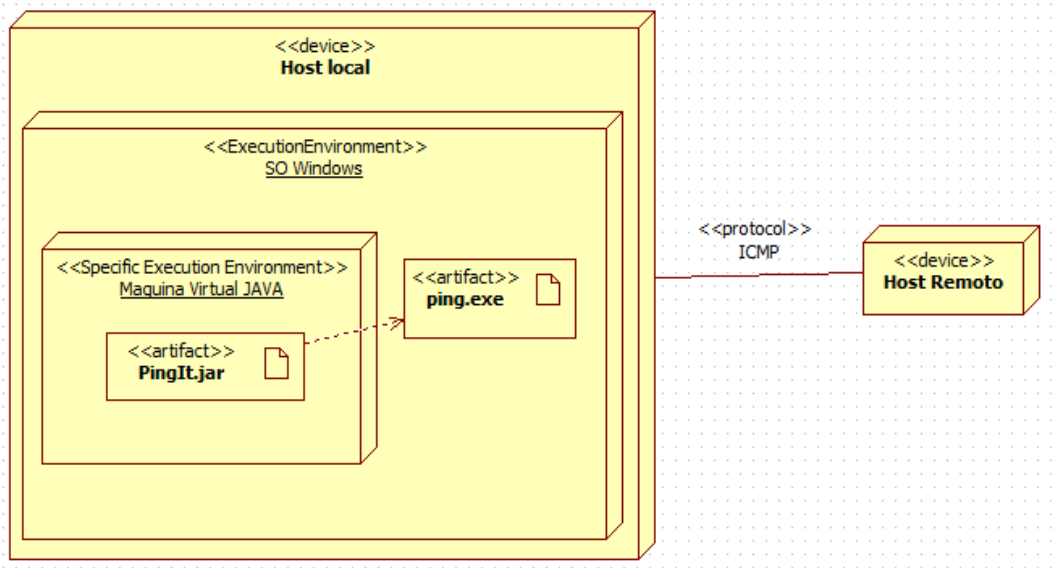
A continuación se encuentran los diagramas de componentes y despliegue.

- ## Diagrama de Componentes:

Se puede apreciar la arquitectura MVC en el Host Local donde el Modelo es el PingModel, el Controlador es el PingControler y también se encuentra la vista. Además vemos que el programa se conecta a un Host Remoto con el fin del envío de paquetes para probar la calidad de la conexión.



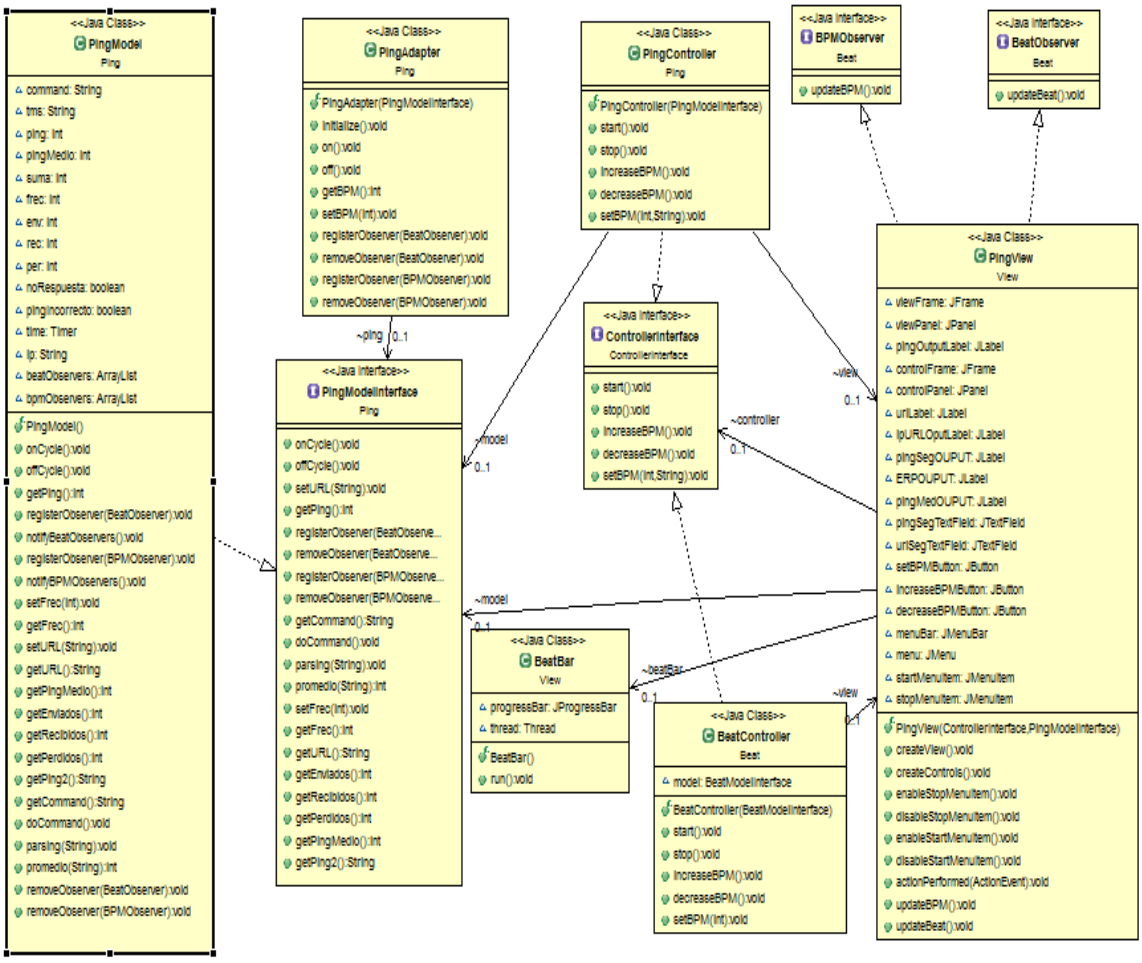
- Diagrama de Despliegue:



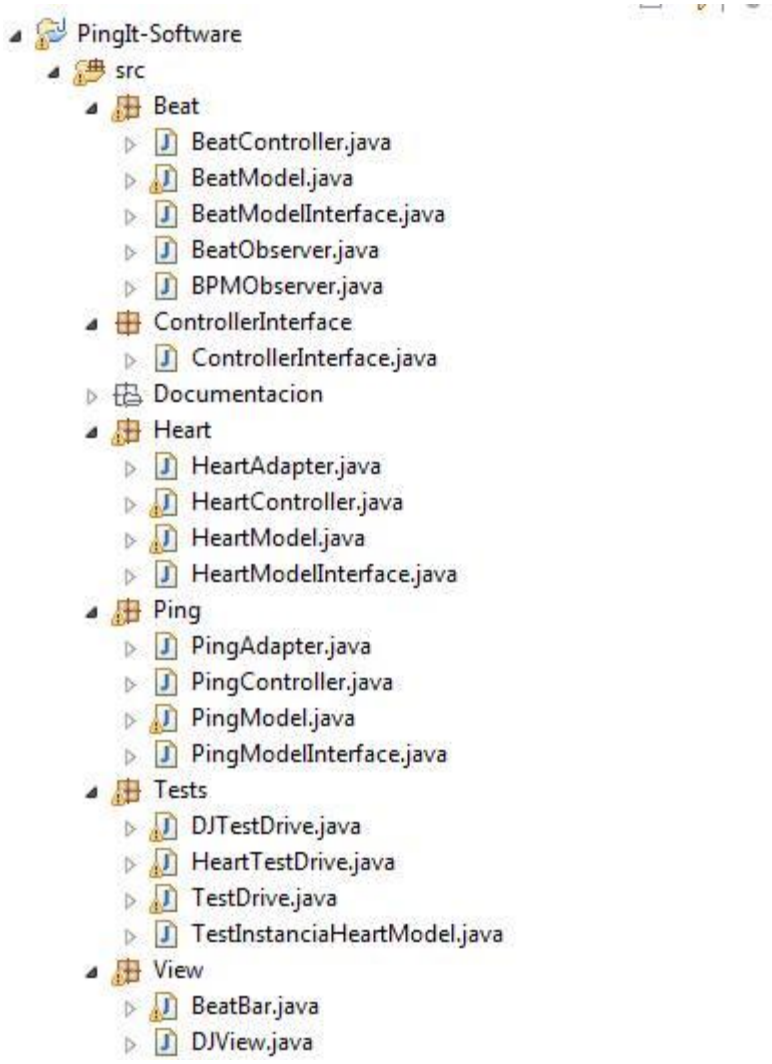
5. Diseño e Implementación:

- Diagrama de Clases:

Diagrama de clases para nuestro modelo PingIt.



- Diagrama de Paquetes:

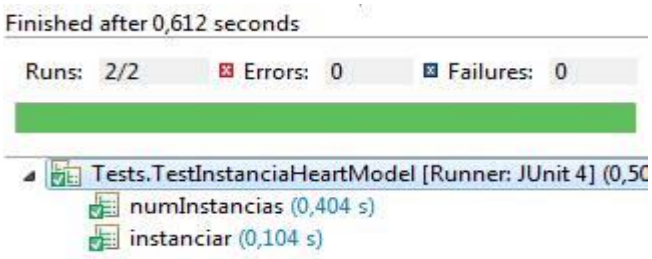


6. Pruebas unitarias y del Sistema:

- Tests Unitarios:

Para el punto 1 se hicieron las siguientes pruebas:

- Test Unitario "numInstancias" (UT1): Prueba que solamente se deja generar una sola instancia de la clase HeartModel.
- Test Unitario "instanciar" (UT2): Prueba que dicha única instancia se está generando efectivamente una instancia.



Para probar las funcionalidades del modelo PingIt se realizaron las siguientes pruebas:

- Test Unitario "testSetIP" (UT3): Prueba que la dirección se fija con éxito.
- Test Unitario "testIPincorrecta" (UT4): Verifica si la salida es un informe de IP Incorrecto al ingresar una dirección equivocada.
- Test Unitario "testSetFrecuencia" (UT5): Prueba que si se establece una frecuencia nueva, la frecuencia del modelo cambia.
- Test Unitario "testPingPorSegundo" (UT6): Verifica que el número de Ping mostrado sea el correcto.
- Test Unitario "testSetURL" (UT7): También prueba que se puede trabajar no sólo con direcciones de IP sino que también con URLs.
- Test Unitario "testPingmenorFrecuencia" (UT8): Verifica que el ping es menor que una frecuencia normal de uso.

- Tests de Sistema:
    - Test de "Uso general" (ST1):
        - Consiste en ejecutar en el modelo *PingIt*, establecer una frecuencia y una dirección, iniciar el ciclo y ver si otorga una respuesta coherente.
        - Estado: Pass.
    - Test de "Frecuencia Mínima y Máxima" (ST2):
        - Se verifica si se aplica algún límite de frecuencia al establecer con "Set" y al cambiar su magnitud con "<<" y ">>".
        - Estado: Fail.
    - Test de "Detención controlada" (ST3):
        - Prueba que si después de 2 horas de iniciado el ciclo de envío de paquetes automáticamente el programa se detiene.
        - Estado: Fail.

- Matriz de trazabilidad:

Casos de Uso	**Test Unitarios**	**Test de Sistema**	**Requerimientos**
Establecer dirección	UT3, UT4, UT7	ST1	F.3.1
Establecer frecuencia	UT5, UT8	ST1, ST2	F.2.1, F.2.2, F.2.3, F.2.5, NF.1.1
Aumentar frecuencia	UT5, UT8	ST1, ST2	F.2.1, F.2.3, NF.1.1
Disminuir Frecuencia	UT5, UT8	ST1, ST2	F.2.1, F.2.3, NF.1.1
Comenzar			F.2.1, F.2.2, F.2.4, F.3.3, F.3.4, F.3.5
Detener		ST3	F.2.1, F.3.2, NF.2.1

- Pass/Fail:

    - Pruebas Realizadas: 11.
    - Pruebas Pasadas: 9
    - Pruebas Fallidas: 2
    - Pass Ratio: 82%
    - Fail Ratio: 18%

- Bugs y Correcciones:

Los bugs fueron corregidos siguiendo el código manualmente, con salidas de mensaje por consola y algunos con la herramienta de corrección debugs de Eclipse.
Los bugs más severos corregidos fueron:

- El seteo de bpm en DJView no generaba el ping.
- No actualizar la View por no hacer notifyBeatObservers() y el notifyBPMObservers().
- Error al mandar String en el bpmOutputLabel() del DJView.
- No mandar cada cierto tiempo un ping cuando se aprieta “Start” en el controlView.

Los bugs más severos sin corregir son:

- No se establece límite de frecuencias por ende, frente a ciertas frecuencias el programa deja de responder correctamente.
- La funcionalidad de elegir qué modelo se quiere observar en la Vista no está implementada correctamente ya que con cierta combinación de vistas, el programa deja de responder correctamente.

7. Datos Históricos:

Informamos las horas invertidas en las diferentes tareas del Trabajo Final:

- Diagramas Actividades, Secuencia; Casos De uso:
  - Esfuerzo: 3hs/Persona.
  - Autor/es: Florencia Amaya.
- Diagrama de Arquitectura Preliminar:
  - Esfuerzo: 1hs/Persona.
  - Autor/es: Florencia Amaya y Lucas Almendros.
- Matrices de Trazabilidad:
  - Esfuerzo: 1hs/Persona.
  - Autor/es: Florencia Amaya.
- Diseño Arquitectura:
  - Esfuerzo: 3hs/Persona.
  - Autor/es: Lucas Almendros.
- Diagramas de Clase Y Objetos:
  - Esfuerzo: 1hs/Persona.
  - Autor/es: Florencia Amaya.
- Diagrama de Paquetes:
  - Esfuerzo: 1hs/Persona.
  - Autor/es: Florencia Amaya.
- Patrones de Diseño:
  - Esfuerzo: 1hs/Persona.
  - Autor/es: Lucas Almendros.
- Generación de Código y pruebas Unitarias:
  - Esfuerzo: 24hs/Persona.
  - Autor/es: Lucas Almendros y Florencia Amaya.
- Redacción del Informe:
  - Esfuerzo: 4hs/Persona.
  - Autor/es: Florencia Amaya.
- Manejo de Configuraciones:
  - Esfuerzo: 4hs/Persona.
  - Autor/es: Lucas Almendros.

8. Información Adicional:

Durante el desarrollo de este proyecto implementamos todos los conocimientos aprendidos durante la materia. De este modo, pudimos apreciar de primera mano las ventajas que ofrece desarrollar software aplicando Ingeniería de Software y sus diferentes métodos.

Al hacer un análisis de requerimientos en un principio y siempre tenerlos en cuenta en el resto de la implementación, nos ayudó a conocer bien nuestros objetivos y a verificar rápidamente si el proyecto era realizable. Cabe decir también que varios fueron modificados en el transcurso ya que nos encontramos con problemas de implementación y de asociación con nuestro modelo original y el resto del código ya escrito.

Una de las herramientas más importantes fue la de manejo de configuraciones y uso de repositorios que agilizó nuestra programación en conjunto y organizó conceptualmente el desarrollo del producto. Aunque tuvimos ciertos problemas al principio con los branches y diversas funciones como merging, cada vez que aumentaba nuestro conocimiento de uso, empezábamos a apreciar más la gran utilidad de esta herramienta.

Otro gran avance fue la utilización de diversos diagramas UML que nos ayudó a darnos cuenta de errores de concepto del proyecto o de funcionalidades que iban a ser imposibles de implementar antes de empezar a diseñar y malgastar tiempo en ellas.

En resumen vemos que la organización y el uso de estos recursos mejoran cuantiosamente el desarrollo de Software aunque es seguro que sus beneficios son directamente proporcionales a la magnitud del proyecto.