Instituto Tecnológico de Costa Rica
Ingeniería en Computación
Programación Orientada a Objetos
I Semestre 2015
Prof. Mauricio Avilés

Proyecto #1

Estudiantes:

Kenneth Callow
Sara Castro Sáenz

13-04-2014

b. Resumen ejecutivo

Este proyecto consiste en un sistema de manejo de pacientes para entrenadores de fitness, programado en Java para el curso Programación Orientada a Objetos. El fenómeno cultural de los estilos de vida saludables ha creado la necesidad para servicios como estos, y para que llevarlos a cabo eficientemente se pueden aprovechar los sistemas de información disponibles. El proyecto busca satisfacer las necesidades de los entrenadores para mantener todos los datos personales de sus pacientes, además de un registro de sus mediciones a través del tiempo, y sus programas de ejercicios. Les permitirá mantener un registro de todas las máquinas y ejercicios que tienen disponibles sus clientes, y la construcción de programas de ejercicio según estas opciones. En fin, busca empoderar a los entrenadores para que lleven a cabo su trabajo de la mejor manera para ellos y sus clientes.

Este documento busca explicar los detalles del diseño e implementación del proyecto. La forma en que se plantea y analiza el problema, y luego cómo se implementa se cubre en la presentación y análisis del problema. Seguidamente se busca producir conclusiones con respecto a lo que se logró y lo que no, a qué conocimientos obtenidos en la realización del proyecto podrán ser de utilidad a futuro para otros proyectos programados. Basándose en las conclusiones se plasman recomendaciones para aquellos quienes realizarán un proyecto semejante, para que logren seguir las buenas decisiones y evitar las malas.

c. Introducción

Los tiempos actuales han visto la proliferación de una cultura orientada a la salud y el ejercicio. Quizá como producto de la osmosis cultural globalizada, los costarricenses, en especial los de la media y alta clase, buscan adaptar sus estilos de vida para calzar con un ideal de salud nuevo: ya no basta con ir al gimnasio a menudo, sino que es necesario que los ejercicios sean cuidadosamente planeados por conocedores en el tema, los llamados *entrenadores de fitness*. Ellos deben asegurarse que el ejercicio efectuado por sus pacientes concuerde con sus capacidades y necesidades físicas, por lo que deben registrar sus mediciones cada cierto plazo y analizarlas debidamente para definir programas de entrenamiento adecuados.

Un entrenador con deseos de mantener los datos de sus pacientes y sus programas de entrenamiento de forma ordenada y eficiente le vendría de utilidad un sistema de software que satisfaga estas necesidades. Para esto surge este novedoso sistema multiplataforma en Java 8: Doge Fitness. Doge Fitness te permite guardar los datos personales de tus clientes, incluyendo nombre, número de teléfono, fecha de nacimiento y cédula, también permite que guardes un listado de tipos de máquinas que podrás asignar a los tipos de ejercicios que realizarán tus clientes.

Cada cliente tiene su programa de ejercicio, que consiste en varios días almacenados por número, cada día conformado por varios ejercicios que indican sus pesos, repeticiones, series y tiempos de descanso. Los programas de ejercicio tienen fecha de inicio, fin y creación. Cada cliente también tiene una colección de mediciones que definen su estado físico actual. En fin, es un sistema útil que satisfacerá tus necesidades de entrenador de fitness casi completamente.

El proyecto incluye un código fuente en Java 8, dividido en paquetes para la interfaz gráfica y para el almacenamiento interno de los datos, un ejecutable de Java en formato JRE que podrá ser ejecutado en cualquier máquina con la máquina virtual de Java actual instalada, y un archivo de carga que utiliza la aplicación para guardar sus datos, llamado saveFile.txt, precargado con algunos clientes, máquinas, ejercicios de ejemplo.

d. Presentación y análisis del problema

i. ¿Qué se debe resolver?

Llegar a producir el programa deseado es un problema que podría dividirse en algunos pasos puntuales:

- Establecer cuáles clases van a corresponder a los datos que se desean almacenar. En este caso, se deben crear clases que puedan guardar un tipo de ejercicio, un tipo de máquina, los datos personales de un paciente, un programa de entrenamiento, un día de un programa, y un ejercicio de un día. En este paso sólo es necesario saber cuáles clases se necesitan. Modelado UML es una buena forma de realizar este paso, y el siguiente.

- Definir con exactitud cómo se van a relacionar estas clases. Por ejemplo, un paciente posee un programa de entrenamiento y una o más mediciones, un programa de entrenamiento consiste en uno o más días, un día consiste en uno o más ejercicios, un ejercicio corresponde a un tipo de ejercicio, un tipo de ejercicio posee nombre, descripción y un tipo de máquina, etc. Si al final las relaciones producen un diseño excesivamente complejo o poco claro, y que podría simplificarse mediante herencias, implementaciones y uso de genéricos, se debería intentar rediseñarlo con esto en mente. Así, las relaciones entre clases pueden sugerir la creación de clases e interfaces nuevas.

- El diseño de ventanas puede ocurrir paralelamente al diseño de clases. Diseñar ventanas no requiere de un diagrama de clases tanto, sino que es más importante pensar en términos de los requerimientos funcionales del programa, con tal de que cada ventana corresponda a la estructura implícita en los casos de uso dados. En este paso se deben definir cuáles ventanas se van a construir, cuáles serán sus contenidos, en qué contextos desplegarán ciertos contenidos y en cuáles no, cuáles datos modifican, qué opciones abren otras ventanas, etc.

- Con un buen diseño que busque la simplicidad y comprensión ya hecho, se puede comenzar a implementarlo en el lenguaje deseado, en este caso Java. Las clases deberían ser construidas de abajo para arriba, o sea desde las clases que no dependan de ningunas otras, hasta las que dependan de todas, en forma de árbol de dependencias. Por ejemplo, la clase correspondiente a tipo de máquina no depende de ninguna otra, así que debería implementarse primero. Luego, las clases para tipo de ejercicio y el contenedor de tipos de máquina dependen únicamente de tipo de máquina, por lo que deberían implementarse de siguiente. Con tipo de ejercicio implementado, se puede implementar la clase que guarde un ejercicio de un día, además del contenedor de tipos de ejercicio. Así, paso por paso, el programa se construye hasta llegar a la clase principal de la aplicación. Las clases deben de ser probadas en el momento de implementarse, para asegurar que su comportamiento sea el deseado. Java incluye una clase Main en cada clase, la cual puede usarse para poner las pruebas unitarias de cada clase.

- Se debe considerar en la implementación de cada clase la forma en que va a ser almacenada en un archivo. Cada clase debe tener un método que la convierta en un formato que pueda ser leído desde un archivo. La clase principal de la aplicación debe tener métodos que permitan cargar y guardar las clases desde un archivo. Estas deben de funcionar correctamente antes de comenzar a unir los datos con las ventanas, para hacer que las pruebas de las ventanas sean eficientes.

- En paralelo a la implementación de las clases, se pueden implementar las ventanas, pero sin funcionalidad de datos. O sea, que carguen e interactúen entre ellas, mostrando datos “de mentiras”, pero todavía sin relacionarse con los datos. Las opciones que abran nuevas ventanas deberían hacerlo dado este paso, igual que las que cambien los datos mostrados dentro de una sola ventana. La funcionalidad de las ventanas debe probarse después de implementarse cada una. También es sugerible implementarlas de abajo para arriba, para que todas las dependencias se satisfagan al irse construyendo.

- Ya al implementarse todas las clases del diseño, se integran las ventanas con los datos. Este paso puede ser el más tedioso, dado que implica unificar el trabajo de diferentes programadores, y también por el hecho de que los elementos de las ventanas, como las tablas, no representan los datos “nativamente”, en la forma de objetos en que fueron definidos. sino que estos por lo general aceptan arreglos o matrices como argumento, y los datos deben ser convertidos a este formato. Es necesario en este paso que la información desplegada corresponda a los casos de uso de cada ventana, y que la información se actualice cada vez que es modificada. Las ventanas que existen para agregar, modificar y borrar elementos deben implementar esta funcionalidad y probar que funcionan.

- Ya terminado esto, el programa se debe probar para determinar que todas las funcionalidades sean correctas. Se debe recorrer entre todos los casos de uso y llevar a cabo cada uno de ellos, asegurándose que se cumplan los casos normales y excepcionales. Si no es el caso, se debe buscar el error y corregirlo. Dado que cada clase debería tener sus pruebas unitarias, el error no debe ser tan difícil de encontrar. Asegurados que funciona, el programa ya se puede considerar un producto final.

ii. ¿Cómo se va a resolver el problema?

Esta sección se basa en la falsa premisa de que fue escrita antes de realizarse el proyecto. La realidad es que la documentación fue realizada de último, después de un proceso desordenado y desesperado de desarrollo, el cual produjo un producto funcional pero menos que lo que pudo haberse hecho con un mejor planteamiento y análisis.

El programa estará constituido por dos paquetes, uno para la interfaz gráfica y otro para el manejo de los datos. El paquete para los datos poseerá la capacidad de cargar y guardar todos los datos de la aplicación. Cada una de las clases de datos posee un método que la convierte a un formato de texto que puede ser leído por la aplicación. Se implementarán las clases de datos:

- ProyectoFitness: el la clase principal de la aplicación. Implementa los métodos para guardar y cargar todo. Utiliza un archivo de cargas *saveFile.txt* para cargar y guardar. Implementa un *shell* mediante el cual se accede a la funcionalidad del programa mediante texto. Los métodos para cargar envían todas las líneas del saveFile a este shell para que lo interprete. De esta forma, el saveFile puede considerarse un shell script, y los métodos de cada clase para conversión a archivo de cargas como comandos del shell. Igualmente, ProyectoFitness contiene los contenedores para pacientes, tipos de máquina, tipos de ejercicio y tipos de medición, e implementa los métodos por los cuales las ventanas pueden acceder y modificar los datos.
- Pacientes, Maquinas, TiposEjercicio: son los contenedores para Paciente, tipos de máquina, y TipoEjercicio. Cada uno implementa métodos para agregar, borrar y modificar sus contenidos. También para convertir sus contenidos a comandos del shell. TiposEjercicio debe implementar un método para borrar todos los ejercicios que dependan de una máquina que ya fue borrada. También implementa un método que actualiza un nombre de máquina dentro de los ejercicios, por si éste fue actualizado por el usuario. Todos heredan de HashMap, ya que esta clase relaciona una única llave con un valor, lo cual es útil en este contexto: los Pacientes tienen una cédula única, las máquinas y los tipos de ejercicio un nombre único.
- TiposMedicion: también es un HashMap, pero no implementa métodos para guardar ya que sus valores permanecerán constantes para toda ejecución del programa. Cada tipo tiene un nombre único.

Nota: las clases Maquina, TipoMaquina y TipoMedicion no existen. Debido a que uno de sus campos se volvió llave de un HashMap, el único campo restante fue más lógico implementarlo como String.

- Paciente: posee nombre, sexo, fecha de nacimiento, teléfono, y correo. También puede tener algunos conjuntos de mediciones y algunos programas de entrenamiento. Su cédula no es parte de la clase, sin embargo está guardada como llave, con la clase envoltorio Cedula, en el contenedor Pacientes. El sexo se guarda como un String (“F” para femenino, “M” para masculino), la edad con LocalDate, teléfono y correo con clases envoltorio Telefono y Correo, que validan los datos. Cedula también valida.
- ConjuntoMediciones: guarda el conjunto de series de mediciones, relacionando cada serie con su fecha de creación.
- Mediciones: guarda una serie de mediciones, relacionando cada medición con su nombre. Sólo admite agregar mediciones que están en TiposMedicion. Posee la fecha en que se realizaron.
- ProgramaEntrenamiento: Es un HashMap que relaciona números de día con un conjunto de días.
- Dia: Guarda un día con sus ejercicios.
- Ejercicio: posee tipoEjercicio, tiempoDescanso, series repeticiones, peso 1, 2 y 3 que se guardan como entero.

Se implementarán estas clases de ventana:

- VentanaPrincipal: Es la ventana principal del programa, en ella está la sección de administrar pacientes, administrar máquina y administrar tipos de ejercicio. Trae una tabla en la que se pueden visualizar los pacientes, las maquinas y los tipos de ejercicio. Los elementos en la tabla son seleccionables.
- VentanaPaciente: Esta ventana tiene varios cuadros de texto en las que se ingresa la información del paciente.
- VentanaTipoEjercicio. La ventana tipo ejercicio tiene cuadros de texto para escribir el nombre y una descripción. Tambien tiene una lista de maquinas de las cuales se puede seleccionar una.

- VentanaMaquina: Esta ventana cuenta con dos cuadros de texto para ingresar la información de la maquina, también tiene un botón modificar para editar los contenidos y borrar para eliminar la maquina.
- VentanaDatosPaciente: En esta ventana se despliegan los datos del paciente. Tiene botones para editar la información asi como para borrar al paciente, cuenta con una tabla en la que se pueden ver las mediciones que tiene el paciente, también puede agregar mediciones y programas de entrenamiento con su respectivo botón para verlo.
- VentanaMedicion Esta ventana cuenta con diferentes espacios para ingresar cada uno de los valores de las mediciones
- VentanaProgramaEntrenamiento En esta ventana se ingresa una descripción del programa o los objetivos, Cuenta con una tabla en donde se pueden ver los días y seleccionarse, también tiene botones para borrar el programa y agregar mas días al entrenamiento.
- VentanaDia En la VentanaDia se crea el dia, después de creado se pueden agregar los ejercicios, los cuales se verán en la tabla que hay en la ventana.
- VentanaEjercicio La VentanaEjercicio sirve para crear los ejercicios, en ella se ingresa la información correspondiente al ejercicio, las series, repeticiones, el tipo de ejercicio, el tiempo de descanso y el peso.

También se implementa una interfaz llamada ModosVentana, la cual declara ciertos métodos que deben usar algunas ventanas para cambiar entre modo de agregar, borrar y modificar.

iii. Análisis crítico de la implementación.

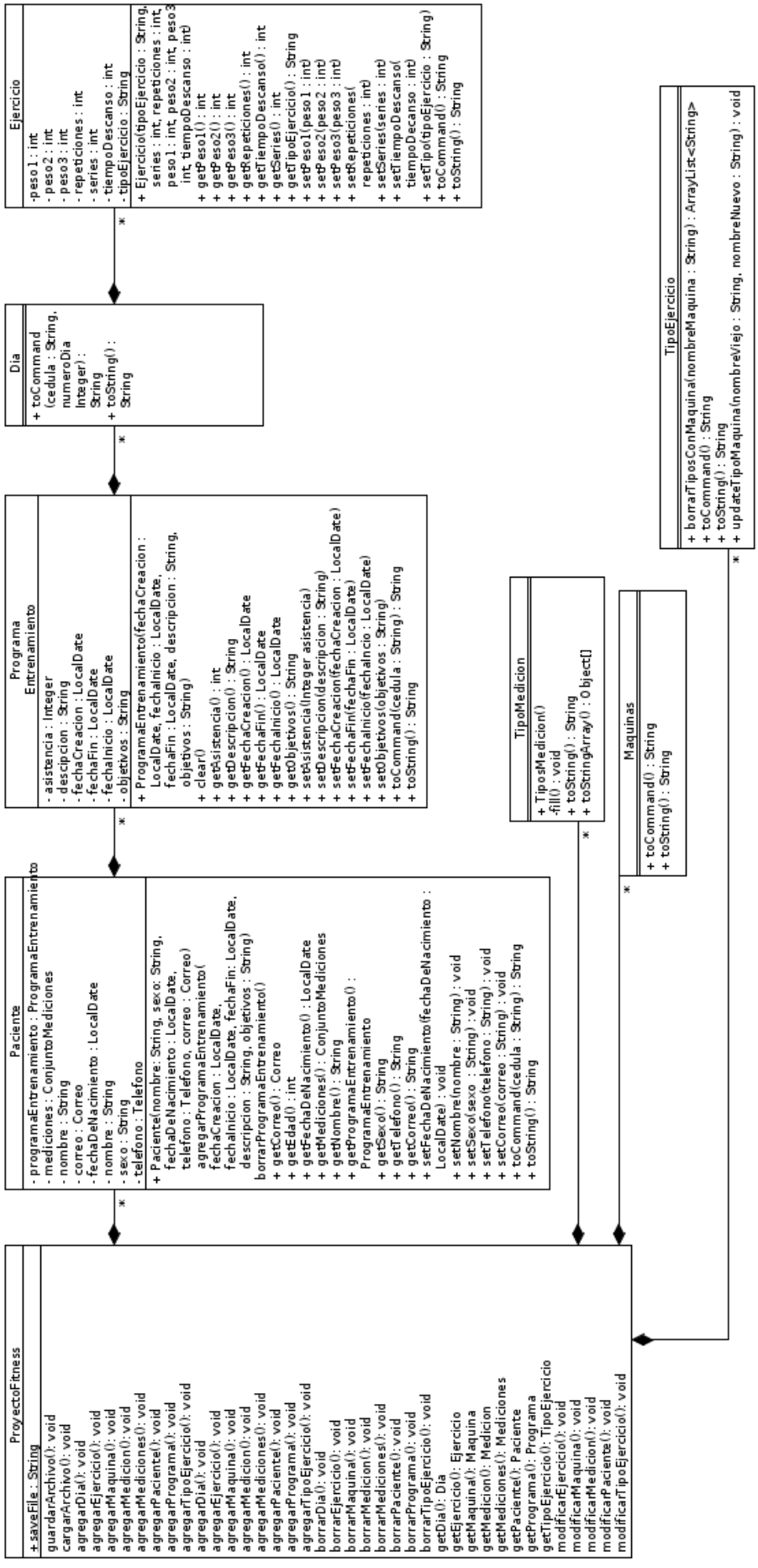
A la hora de implementarse no se hizo un buen análisis de lo que se nos solicitaba, se tuvo que volver a empezar desde cero en una ocasión por que nos dimos cuenta que era un poco más eficiente implementarlo de una forma y no como lo estábamos haciendo.

Cuando se empezó a integrar el trabajo con la interfaz, se tuvo que reescribir código en varias ocasiones por que al no haber pensado bien el funcionamiento del programa y por la falta de comunicación que teníamos, la interfaz requería de métodos o datos que no se habían implementado correctamente en el programa y se tuvo que reescribir código en función de la interfaz.

El programa es muy rigido, por lo tanto cada vez que se cambia algo hay que acomodar todo el código para que funcione, incluyendo la interfaz.

Algunas de las funcionalidades requeridas no lograron completarse. Los pacientes sólo poseen un plan de ejercicios, lo cual es menos de lo pedido. El cálculo de mediciones en la ventana de mediciones presenta algunos problemas. Si se cierra una ventana sus hijas no se cierran.

e. Diagrama UML:



f. Conclusiones

- El modelo que implementamos es inmantenible y rígido. Por haber hecho una clase de datos que fuere la única interfaz por la cual las ventanas podían acceder a los datos, se aumentaron las dependencias sobre esa interfaz, ya que cada vez que se quería modificar la implementación de los datos, se debía cambiar también la de la interfaz. Esto aumenta el trabajo considerablemente, y limita la forma en que pueden tratarse los datos a lo que permite la interfaz.

- El uso de clases de Java para almacenar datos implicó la creación de métodos que convirtieran los datos a un formato utilizable por la interfaz gráfica, y también métodos que facilitaran cargar y guardar los datos desde un archivo. Esto en retrospectiva no fue la forma más eficiente de resolver el problema, ya que existen mejores formas de representar y almacenar los datos. Quizás rompería con la orientación a objetos, pero nuestra opinión es que lo ideal sería guardar los datos en el mismo formato que esperan los elementos de las ventanas (arreglos, no objetos), y así evitarse conversiones innecesarias (ciclos 'for', toStrings, etc) por parte del programa y la programadora. Cargar y guardar pudo haber sido con XML o bases de datos, pero al final lo implementamos con archivos de texto simple.

- Dejar la documentación para el final del proyecto no fue una buena decisión. El proyecto se llevó a cabo improvisadamente, sin un modelo suficientemente definido, lo que ocasionó retrabajo y malas decisiones en la implementación. La documentación obliga a tener un diseño bien definido, por lo que es recomendable realizarla desde antes de comenzar a programar.

- No hubo suficiente comunicación entre los miembros del equipo. Por ello, la implementación de datos no correspondía con algunos elementos de la interfaz gráfica. Esto se debió también a una mala lectura de los requerimientos. Es necesario que todo el equipo entienda bien los requerimientos, y que los interprete de la misma manera.

g. Recomendaciones

- Quizá en proyectos grandes separar rígidamente los datos de la interfaz permita mayor orden y comprensión, pero en pequeños como éste sólo complica las cosas. Recomendamos más bien que para proyectos pequeños los datos estén vinculados fuertemente a sus ventanas, o sea que cada clase contenedor de datos esté relacionada a una ventana, sobre la cual tiene control completo y utiliza para visualizar sus datos. El diseño e implementación de las ventanas sería parte del diseño e implementación de datos, por lo que probar las ventanas equivaldría a probar los datos; este modelo de desarrollo evita el tedio de unificar dos partes heterogéneas de un programa (interfaz gráfica y datos), evita tener que crear una única interfaz difícil de mantener entre los datos y las ventanas, permite llevar a cabo pruebas más eficientes, y en general permite subdividir el software en componentes funcionales de mejor manera.

- El uso de clases de Java para almacenar datos es una opción, pero quizá no la más eficiente. Existen incontables sistemas de bases de datos, muchos basados en el lenguaje SQL que permiten un manejo mucho más poderoso y eficiente de datos que simplemente las estructuras de datos incluidas en un lenguaje de propósitos generales como Java. Y no sólo los sistemas de mayor tamaño y complejidad pueden utilizar bases de datos: existen sistemas como SQLite (www.sqlite.org) cuyo enfoque es el de bases simples de un solo archivo, que pueden ser controladas por otros lenguajes como Java mediante librerías. Esto sería ideal para un proyecto pequeño como éste, ya que haría innecesaria la implementación de clases para guardar datos y facilitaría la persistencia de datos. Además, dado que las bases de datos utilizan tablas para almacenar todo, es una representación que es más fácilmente representada dentro de las ventanas del programa, que generalmente piden arreglos o matrices como sus datos y que proveen facilidades para sincronizarse con bases de datos. En fin, el proyecto se reduciría desde implementar varias clases para datos y las ventanas, con muchas líneas de código, a implementar una sola base de datos con sus respectivas tablas y ventanas que actúen sobre esas tablas, con bastantes menos líneas de código. Esta solución presenta simplicidad, eficiencia, mantenibilidad y flexibilidad.

- Documente, después implemente. No se debe dejar la documentación para el final, más bien la presentación y análisis del problema debe ser lo primero, ya que define claramente cómo se debe proceder.

- Además la documentación se realiza con mayor facilidad si es simultánea con el trabajo. Esto porque es más fácil estudiar los elementos del proyecto al mismo tiempo en que surgen, cada detalle está más fresco en las mentes de los integrantes y se evita su olvido. Es más tedioso recordar todo lo realizado a la vez, que irlo registrando poco a poco.

- Se debe procurar tener una buena comunicación entre los miembros del equipo. Todos deben entender bien su papel y cómo calza la parte del proyecto que están realizando con el resto. Los diagramas UML nacieron exactamente para esta necesidad, y se deben construir basándose en un análisis cuidadoso de los requerimientos. Además de tener diagramas definidos, es valioso que los miembros comuniquen los nuevos cambios que introducen al proyecto, para que no hayan malentendidos con respecto al trabajo de los otros.

h. Referencias