

Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Diseño de Aplicaciones 1

Obligatorio 1 – Documentación

Sofía García Nro.: 187588

Facundo Revello Nro.: 178000

Grupo N4A

Docente: Ramiro Visca

Octubre 2018

1. Índice

1. Índice	1
2. Introducción	2
3. Descripción general	3
3.1. Herramientas utilizadas.....	3
3.2. Cobertura de pruebas.....	3
4. Descripción funcionalidades.....	4
4.1. Usuarios.....	4
4.2. Productos.....	4
4.3. Data sets	4
4.4. Visualización	4
4.5. Estadísticas	4
5. Notas sobre las funcionalidades.....	5
5.1. Usuarios.....	5
6. Descripción y justificación de diseño.....	6
6.1. Diagrama de paquetes.....	6
6.2. Diagrama de clases (UML)	7
6.2.1. Diagrama proyecto dominio	7
6.2.2. Diagrama proyecto lógica.....	7
6.2.3. Diagrama proyecto accesoDatos.....	7
6.2.4. Diagrama proyecto UI.....	8
6.3. Justificación de diseño.....	9
7. Cobertura de pruebas.....	11
8. Anexos	¡Error! Marcador no definido.

2. Introducción

El objetivo de este documento es el de informar el estado de la aplicación, junto con la explicación de cómo fueron realizadas las funcionalidades, el diseño de la aplicación y aspectos generales del trabajo en la misma. Se agregaran imágenes para facilitar la explicación del diseño de la aplicación junto con evidencias en cuanto a la cobertura de las pruebas y el estado final de las funcionalidades.

3. Descripción general

El trabajo fue realizado por los dos integrantes del grupo en igual cantidad y tiempo de esfuerzo en el tiempo que llevo la realización del proyecto.

3.1.Herramientas utilizadas

Se utilizó la herramienta Github para poder trabajar en simultáneo sin tener problemas con las versiones. El repositorio que contiene el obligatorio se encuentra en:

URL: <https://github.com/frevello/Ob-178000-187588>

Usuarios: frevello (Facundo Revello) y sofiagarcia14 (Sofía García)

Utilizamos la herramienta de Github para Visual Studio 2015, para poder crear ramas, realizar operaciones de push y pull, realizar commits, etc.

Para poder analizar la cobertura de las pruebas, utilizamos la herramienta OpenCover en el Visual Studio 2015.

Para poder hacer el UML se usó la herramienta Visual Paradigm Enterprise.

3.2.Cobertura de pruebas

Decidimos fijarnos como objetivo que una vez terminado el sistema deberíamos tener al menos un 95% de cobertura de pruebas la solución.

4. Descripción funcionalidades

A continuación se informara el estado de las funcionalidades y la explicación de cómo fueron realizadas las mismas.

4.1. Usuarios

- **Alta usuario:** la siguiente funcionalidad agrega un nuevo usuario de tipo desarrollador al sistema, funciona correctamente, durante las pruebas no surgieron inconvenientes o se detectaron bugs.
- **Baja usuario:** esta funcionalidad elimina a un usuario del sistema, funciona correctamente, al realizar las pruebas no se encontraron errores o bugs.
- **Modificar usuario:** la funcionalidad permite realizar modificaciones a los datos de los usuarios, el funcionamiento es correcto, no se detectaron errores en las pruebas realizadas.
- **Visualizar datos usuarios:** la siguiente funcionalidad permite visualizar los datos de los usuarios en el sistema, funciona correctamente, en las pruebas no se encontraron errores o bugs.

4.2. Productos

- **Alta producto:** esta funcionalidad permite agregar un nuevo producto al sistema, junto con una versión por defecto, el funcionamiento es correcto, no se detectaron errores en las pruebas.
- **Modificar producto:** la funcionalidad permite realizar modificaciones a los datos de los productos, su funcionamiento es correcto sin errores detectados en las pruebas realizadas.
- **Alta Versión:** la siguiente funcionalidad agrega una nueva versión a un producto ya ingresado en el sistema, el funcionamiento es correcto, durante las pruebas no se detectaron bugs.
- **Modificar versión:** la funcionalidad es la encargada de modificar los datos de las versiones correspondientes a un producto, funciona correctamente y no hay errores en la misma.

4.3. Data sets

- **Cargar data sets:** la funcionalidad permite cargar un data set al sistema, realizando previamente las comprobaciones necesarias para verificar si es correcto, el funcionamiento es correcto, no se detectaron errores en las pruebas realizadas.

4.4. Visualización

- **Lista data sets:** la siguiente funcionalidad es la encargada de mostrar la lista de data sets cargados para un producto y una versión y una representación gráfica de los mismos, es correcta y no surgieron errores en la pruebas.

4.5. Estadísticas

- **Estadísticas:** esta funcionalidad permite a los desarrolladores ver estadísticas sobre los data sets par aun producto y una versión, su funcionamiento es correcto y no se detectaron errores en las pruebas realizadas.

5. Notas sobre las funcionalidades

En esta sección pasaremos a explicar y comentar ciertos aspectos sobre las funcionalidades.

5.1. Usuarios

- De acuerdo a la letra del obligatorio el sistema debe cargar la última fecha y hora de conexión de los desarrolladores al sistema, la funcionalidad esta implementada y funcionando, debido a que no hay persistencia de datos, no es fácil ver la implementación de esta funcionalidad.
- Al ingresar al sistema tanto a los desarrolladores como al administrador se les despliega el menú principal y se visualizan los datos de los usuarios por defecto en el menú, igualmente se puede volver a esta opción con el botón correspondiente.

5.2. Producto Versión

- Al dar de alta un producto debe dar de alta la primera versión del producto, pero si los datos de la versión están incorrectos se crea por defecto la versión 1.00.000.

5.3. UI

- En el inicio de sesión al presionar el botón “Cargar Datos” se crean en el sistema 2 usuarios de tipo desarrollador, 3 productos cada uno con sus respectivas versiones.

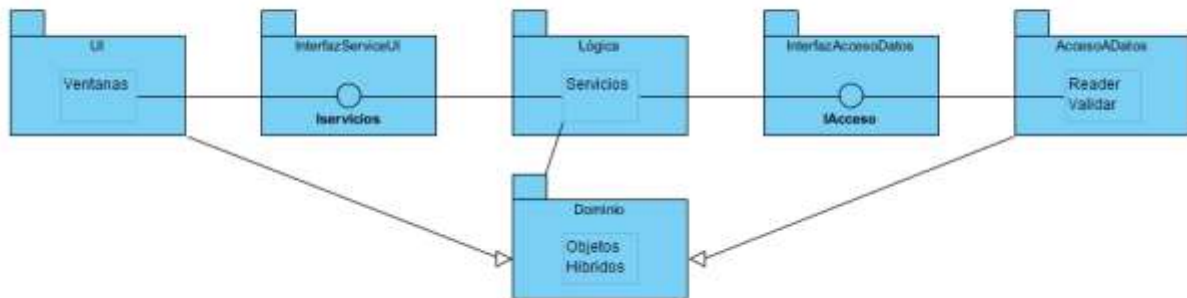
Los usuarios que se crean son:

- Usuario: Sgarcia, Contraseña: 1234
- Usuario: Frevello, Contraseña: 1234

6. Descripción y justificación de diseño

6.1. Diagrama de paquetes

La siguiente imagen muestra el diagrama de paquetes utilizado en el proyecto:

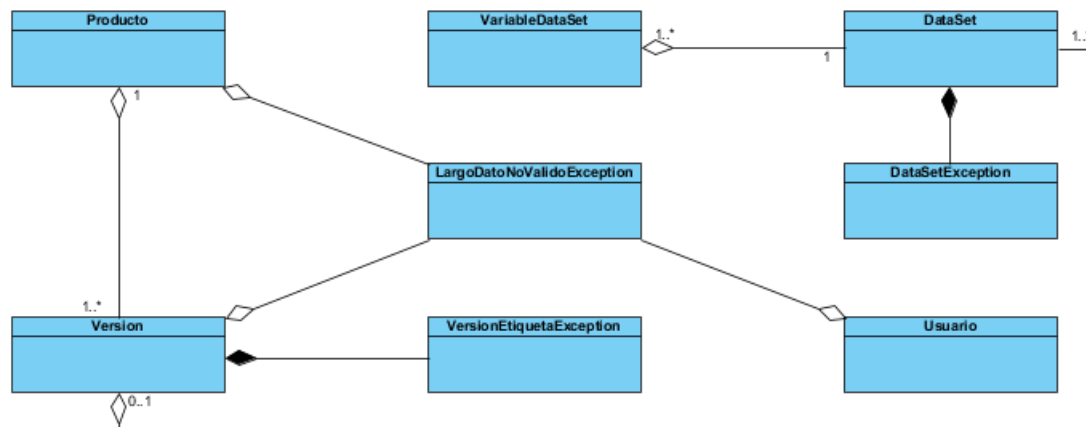


Después de varios cambios en el diseño de la solución, la imagen representa el diseño final del mismo. El diseño final incluye 3 proyectos distintos:

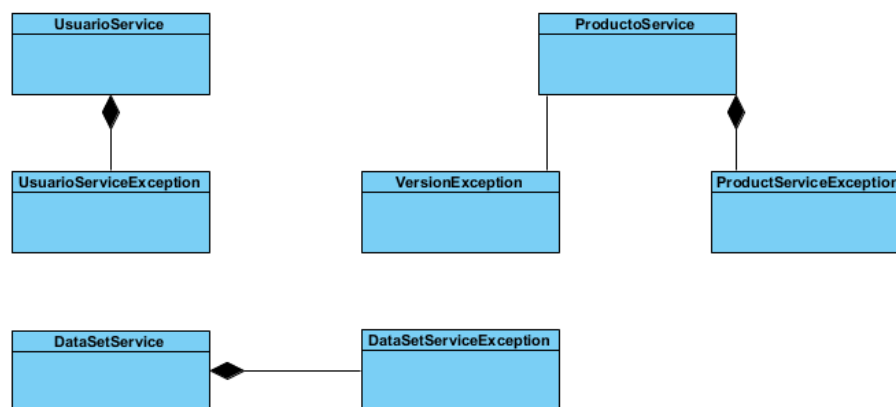
- Proyecto dominio: En el proyecto dominio se encuentran los objetos y los híbridos usados en el sistema, como son usuario y producto o el objeto dataSet. También se encuentran las excepciones correspondientes a esas clases-
- Proyecto lógica: En el proyecto lógica se encuentra toda la lógica de negocio, las clases Service, son las encargadas de realizar las operaciones requeridas por la lógica de negocio, junto con sus correspondientes excepciones.
- Proyecto test: Este proyecto contiene todas las clases de pruebas unitarias correspondientes a las clases de lógica de negocio. En estas clases de pruebas se realizaron las pruebas unitarias antes de realizar los métodos y operaciones en los services.
- Proyecto UI: Este proyecto contiene todas las ventanas y elementos de la interfaz de usuario, es el primer proyecto que se ejecuta al correr el sistema.
- Proyecto interfaz service UI: El proyecto contiene las interfaces que son las encargadas de comunicar la lógica de negocio o sea los services del proyecto lógica con la interfaz de usuario.
- Proyecto interfaz acceso datos: Al igual que el proyecto anterior, este proyecto se encarga de comunicar el acceso a datos, como es el caso del filereader con los demás proyectos.

6.2. Diagrama de clases (UML)

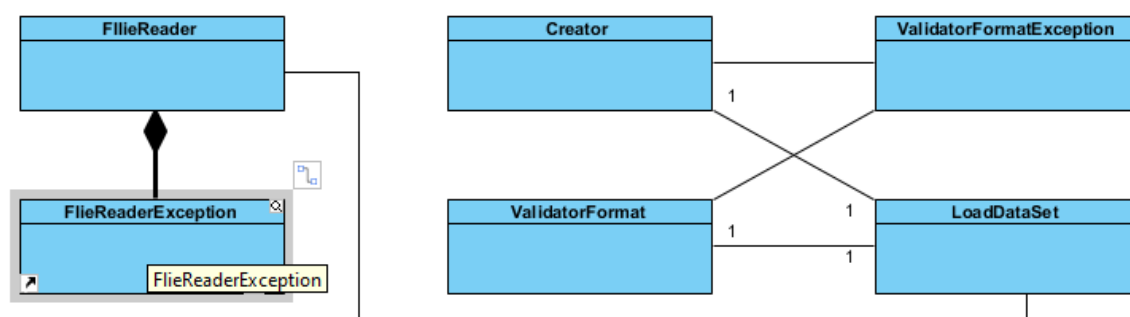
6.2.1. Diagrama proyecto dominio



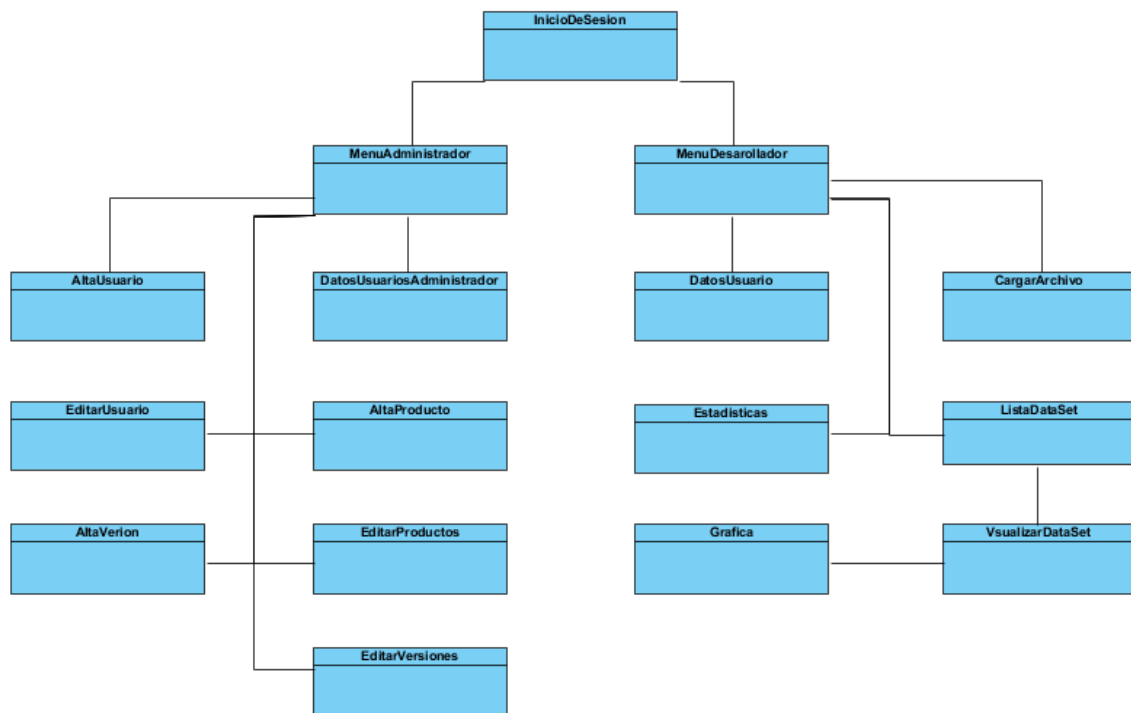
6.2.2. Diagrama proyecto lógica



6.2.3. Diagrama proyecto accesoDatos



6.2.4. Diagrama proyecto UI



6.3. Justificación de diseño

Como se mencionó en el diagrama de paquetes, el diseño de la solución sufrió cambios a lo largo del trabajo, debido a que fueron surgiendo mejores maneras de ir mejorando el diseño para asegurar un alto nivel de mantenibilidad.

El proyecto Dominio, finalizó como se planeó originalmente, como un proyecto que incluyera todas aquellas clases que representarían los objetos y estructuras de datos del sistema. El primer paso fue identificar que clases precisaríamos para nuestra solución, a partir de la letra se identificó rápidamente la necesidad de una clase usuario para definir los usuarios y sus datos y una clase producto, para poder definir los productos y sus datos correspondientes. A la hora de pensar cómo se modelarían las versiones y data sets, tuvimos más dificultades ya que son elementos más complejos de representar, la solución a la que llegamos fue la de definir las siguientes clases, versión, dataSet y variablesDataSet que sería la encargada de definir las variables contenidas dentro de un dataSet.

Una vez identificadas las clases que necesitaríamos en dominio para poder realizar el sistema, surgió la duda de definir el tipo de las mismas. Decidimos que la clase Data Set sería de tipo objeto debido al tipo de validaciones que hace esta clase. Las clases usuario, producto, versión y variablesDataSet serían de tipo híbrido, entendiéndose híbrido como una mezcla de objeto y estructura de datos, ya que no es una estructura de datos propiamente dicha debido a que hacen algunas validaciones como por ejemplo que atributos no sean vacíos. Para finalizar incluimos las clases excepción correspondientes para definir las excepciones que generarían estos elementos. Decidimos para facilitar la implementación de la solución que los productos tendrían una lista de versiones, los versiones una lista de dataSets y a su vez los dataSets tendrían una lista de variables correspondientes a ese data set.

El proyecto Test fue otro de los proyectos que no sufrió grandes cambios a lo largo del proyecto ya que fue uno de los primeros en ser pensado e implementado. Para poder identificar que pruebas unitarias deberíamos hacer, se pensó en qué tipo de clases precisaríamos para la lógica del negocio, para ver qué tipo de pruebas deberíamos realizar. Al identificar que precisaríamos varios servicios para poder representar la lógica del negocio se llegó a la conclusión que necesitaríamos por lo menos dos clases para representar esta lógica, estas fueron la clase productService y la clase userService, que como dicen sus nombres se encargarían de manejar los servicios correspondientes a los productos y usuarios. A partir de la letra del obligatorio pudimos identificar el tipo de métodos que estarían en estos servicios y por lo tanto las pruebas que cada uno de ellos necesitaría. A medida que fuimos armando estas pruebas y avanzando con el proyecto surgió la idea de agregar dos clases más para poder leer los archivos y otra para poder cargar los datos, por lo tanto tuvimos que primero agregar dos clases más de prueba para estas clases a agregar.

El proyecto lógica al igual que el dominio y el Test no sufrió cambios significativos a lo largo del proceso, se pensó como un proyecto que incluyera todas aquellas clases que tuvieran la lógica de

negocio. A partir del proyecto Test tuvimos una idea clara de que servicios deberíamos agregar y las operaciones que tendría cada uno. Inicialmente se decidimos implementar las siguientes clases: `productoService` y `usuarioService`, pero debido a que `productoService` tenía muchas responsabilidades en un refactorio se decidió separar lo que es servicios de productos de los de `dataSets`. Quedando así implementadas las clases de `ProductoService`, `UsuarioService` y `DataSetService`. `ProductoService` es la encargada de manejar las operaciones de ingreso de productos nuevos al sistema y la modificación de productos en el sistema, junto con otras operaciones de utilidad a la hora del acceso a datos, etc. Por último se decidió que este servicio tendría una lista de productos junto con sus operaciones correspondientes. `UsuarioService` sería la encargada de manejar las operaciones de ingreso de nuevos usuarios al sistema, modificar y eliminar usuarios existentes junto con operaciones de acceso a datos, este service al igual que el `productoService` tendría una lista con los usuarios, junto con sus operaciones correspondientes. Por último el servicio `dataSet` sería el encargado de manejar las operaciones de los `dataSets`.

El siguiente proyecto a definir fue el de la UI (Interfaz de usuario), este proyecto simplemente contiene todas las ventanas y paneles de la UI necesarias para que los usuarios pueden usar el sistema y acceder a los datos.

El proyecto `accesoADatos` es el proyecto que se le dio la responsabilidad de poder abrir los archivos, leerlos, validar el formato del mismo a partir de las restricciones en la letra del obligatorio y una vez pasado las restricciones poder crear un `dataSet`.

El proyecto `InterfazUI` fue el próximo proyecto definido y surgió algunas modificaciones a medida que fue pasando el desarrollo del sistema. Este proyecto se pensó como un proyecto que manejara interfaces encargadas de comunicar la lógica de negocio en nuestro caso los services en el proyecto lógica, de esta forma la UI no se estaría comunicando directamente con los servicios si no que la comunicación se haría a través de las interfaces siendo estos un intermediario y así protegiendo el proyecto lógica.

Finalmente el proyecto `InterfazAccesoADatos` fue quizás el proyecto más difícil que tuvimos que diseñar debido a que originalmente no estaba pensado ya que el proyecto `accesoADatos` estaba pensado para comunicarse directamente con los servicios y las clases de dominio. Leyendo la letra identificamos posibles errores en este diseño ya que en futuros cambios en el sistema este diseño no era el mejor. Por eso decidimos que necesitaríamos al igual que con la UI una interfaz para poder comunicar el `accesoADatos` con los servicios y la lógica, previendo un eventual cambio y una posible conexión con una base de datos que haría más simple la adaptación del sistema.

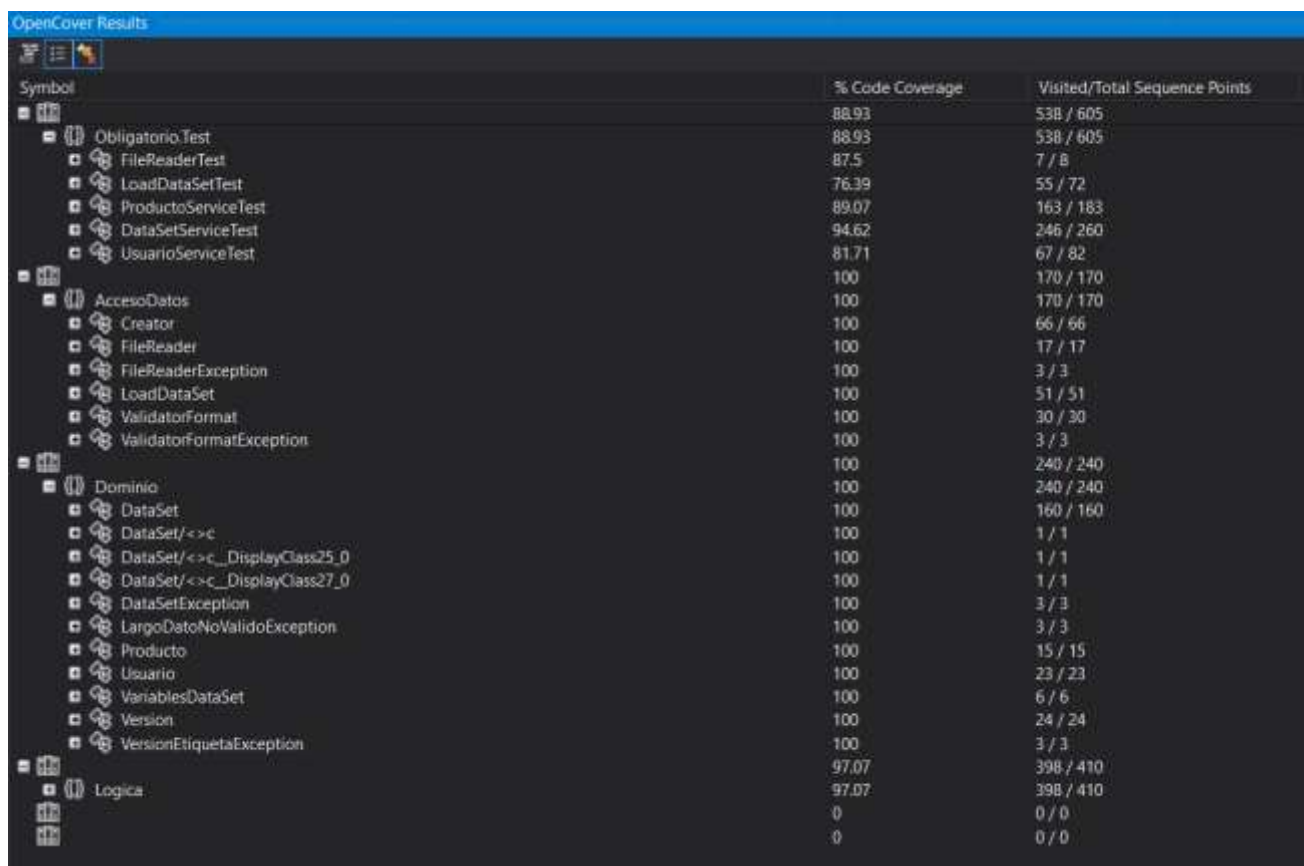
7. Cobertura de pruebas

A continuación se mostrar el estado de cobertura de las pruebas del sistema y la evidencia de las mismas.



Symbol	% Code Coverage	Visited/Total Sequence Points
Obligatorio.Test	88.93	538 / 605
AccesoDatos	100	170 / 170
Dominio	100	240 / 240
Logica	97.07	398 / 410
	0	0 / 0
	0	0 / 0

A partir del OpenCover obtuvimos la cobertura de las pruebas, debido a que para algunas clases y proyectos no fueron realizadas pruebas como es el caso de las interfaces o la UI. No vamos a tener un 100% de cobertura a nivel de la solución. Pero en los proyectos que si llevaban pruebas como es el caso de la lógica vemos que nuestra cobertura está en un 97% promedio, alcanzando el objetivo que nos habíamos planteado.



Symbol	% Code Coverage	Visited/Total Sequence Points
Obligatorio.Test	88.93	538 / 605
FileReaderTest	87.5	7 / 8
LoadDataSetTest	76.39	55 / 72
ProductoServiceTest	89.07	163 / 183
DataSetServiceTest	94.62	246 / 260
UsuarioServiceTest	81.71	67 / 82
AccesoDatos	100	170 / 170
Creator	100	66 / 66
FileReader	100	17 / 17
FileReaderException	100	3 / 3
LoadDataSet	100	51 / 51
ValidatorFormat	100	30 / 30
ValidatorFormatException	100	3 / 3
Dominio	100	240 / 240
DataSet	100	160 / 160
DataSet/<>c	100	1 / 1
DataSet/<>c__DisplayClass25_0	100	1 / 1
DataSet/<>c__DisplayClass27_0	100	1 / 1
DataSetException	100	3 / 3
LargoDatoNoValidoException	100	3 / 3
Producto	100	15 / 15
Usuario	100	23 / 23
VariablesDataSet	100	6 / 6
Version	100	24 / 24
VersionEtiquetaException	100	3 / 3
Logica	97.07	398 / 410
	0	0 / 0
	0	0 / 0

En la imagen anterior se puede ver con más claridad la cobertura de los proyectos Dominio, AccesoDatos y Test. El proyecto Test no tiene un 100% de cobertura como los otros proyectos ya que hay algunos casos en que como está planteada la solución y la letra del obligatorio no se accedería a algunas partes del código y de ahí la diferencia con el resto de los proyectos.

Logica	97.07	398 / 410
DataSetServiceException	100	3 / 3
DataSetService	92.79	103 / 111
DataSetService/<>c_DisplayClass7_0	100	1 / 1
DataSetService/<>c_DisplayClass10_0	100	1 / 1
ProductoService	97.81	179 / 183
ProductoService/<>c_DisplayClass5_0	100	1 / 1
ProductoService/<>c_DisplayClass8_0	100	1 / 1
ProductoService/<>c_DisplayClass9_0	100	1 / 1
ProductoService/<>c_DisplayClass10_0	100	1 / 1
ProductoService/<>c_DisplayClass13_0	100	1 / 1
ProductoService/<>c_DisplayClass14_0	100	1 / 1
ProductoService/<>c_DisplayClass15_0	100	1 / 1
ProductoService/<>c_DisplayClass17_0	100	1 / 1
ProductoService/<>c_DisplayClass24_0	100	1 / 1
ProductoService/<>c_DisplayClass28_0	100	1 / 1
ProductoService/<>c_DisplayClass30_0	100	1 / 1
ProductoService/<>c_DisplayClass32_0	100	1 / 1
ProductoService/<>c_DisplayClass33_0	100	1 / 1
ProductoServiceException	100	3 / 3
UsuarioService	100	84 / 84
UsuarioService/<>c_DisplayClass5_0	100	1 / 1
UsuarioService/<>c_DisplayClass6_0	100	1 / 1
UsuarioService/<>c_DisplayClass15_0	100	1 / 1
UsuarioService/<>c_DisplayClass17_0	100	1 / 1
UsuarioService/<>c_DisplayClass18_0	100	1 / 1
UsuarioServiceException	100	3 / 3
VersionException	100	3 / 3

Por ultimo tenemos el proyecto lógica, vemos que tenemos una buena cobertura de pruebas, y tanto en DataSetService como en ProductoService no se pudo obtener un 100% de cobertura debido al mismo motivo que en el proyecto Test no se pudo lograr un 100% de las pruebas.