

Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería de Software 1

Obligatorio 2

Marco Fiorito

Agustín Hernandorena

Entregado como requisito de la materia Ingeniería de
Software 1

25 de noviembre de 2019

Declaraciones de autoría

Nosotros, Marco Fiorito y Agustín Hernandorena , declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Resumen

El objetivo del obligatorio era realizar una aplicación de escritorio en Java la cual consistía en un EcoShop con dos tipos de usuario (vendedor y comprador), esta aplicación, entre otras funcionalidades, daba la posibilidad de agregar productos, ver gráficas de datos útiles para el vendedor, comprar productos, etc.

Para organizarnos hicimos un análisis previo de todas las cosas que íbamos a tener que hacer y luego procedimos a agregarlas a Trello ordenadas en una columna “ToDo” por prioridad, así siempre tomábamos lo más prioritario.

Además de la columna nombrada en el párrafo anterior, hicimos uso de columnas “In Progress”, “Done” y “Nice to have”.

Las cards creadas fueron distribuidas principalmente por flujo, uno de nosotros hizo principalmente el flujo de Vendedor y otro el flujo de Comprador.

Para el desarrollo decidimos hacer uso de JavaFX junto a JFoenix (librería con componentes de Material Design), esto nos dio la posibilidad de hacer una aplicación más linda estéticamente y con una mayor usabilidad.

Para el control de versiones utilizamos git vinculado con Github siguiendo la metodología de trabajo Git Flow.

En este proyecto aprendimos varias cosas que consideramos importantes en términos de: planeación y organización, versionado y correcto uso de github, esfuerzo en aprender algo nuevo para nosotros como es JavaFX.

Como conclusión quedamos conformes con el trabajo realizado, en cuanto al diseño de la aplicación, su usabilidad y como manejamos git y el control de versiones.

Índice general

1. Versionado	2
1.1. Repositorio utilizado	2
1.2. Elementos de la Configuración del Software (ECS)	2
1.3. Criterios de versionado	2
1.3.1. Uso de ramas	2
1.3.2. Uso de comentarios en commits	3
1.4. Resumen del log de versiones	3
2. Codificación	6
2.1. Estándar de codificación	6
2.2. Pruebas unitarias	8
2.3. Análisis de código	9
3. Interfaz de usuario y usabilidad	11
3.1. Criterios de interfaz de usuario	11
3.2. Evaluación de usabilidad	12
4. Pruebas funcionales	14
4.1. Técnicas de prueba aplicadas	14
4.2. Casos de prueba	14
4.3. Sesiones de ejecución de pruebas	16
5. Reporte de defectos	18
5.1. Definición de categorías de defectos	18
5.2. Defectos encontrados por iteración	18
5.3. Estado de calidad global	21
6. Reflexión	23

1. Versionado

1.1. Repositorio utilizado

La gestión de versiones implica gestionar grandes cantidades de información para asegurar que los cambios en el sistema se registren y se controlen. Las herramientas de gestión de versiones controlan un repositorio de elementos, para trabajar sobre un elemento de la configuración, debe extraerse del repositorio y colocarlo en un directorio de trabajo. Después de hacer los cambios en el software, se introducirá de nuevo en el repositorio creándose automáticamente una nueva versión [1].

En nuestro caso, usamos como software de control de versiones Git, mediante este link: <https://github.com/Marcoo09/FioritoHernandorena> se tiene acceso al repositorio online utilizado.

1.2. Elementos de la Configuración del Software (ECS)

En un sistema de software puede haber muchos módulos de código fuente, documentación, casos de pruebas, librerías, entre otros. Para seguir el registro de toda esta información, se necesita un esquema consistente de identificación para todos los elementos del sistema de gestión de configuraciones [1].

Para ello, antes de iniciar el proyecto, nos centramos en el proceso de planificación de la gestión de configuraciones, es decir, decidir exactamente que elementos del sistema se van a controlar.

En nuestro proyecto, decidimos incluir como elementos de configuración al código fuente (proyecto de NetBeans) y a la documentación.

1.3. Criterios de versionado

1.3.1. Uso de ramas

El uso de ramas, para administrar y diferenciar el trabajo, y a su vez para identificar distintos ambientes (producción, pruebas, desarrollo) son de las principales utilidades que dispone Git [2], esto nos ayuda a llevar un mejor control del código y del proyecto. Una rama se trata de una bifurcación del estado del código que crea un nuevo camino de cara a la evolución del código, en paralelo a otras ramas que se puedan generar.

En nuestro caso, decidimos hacer uso de Git [2] mediante 2 ramas fijas (master develop) y otros 3 tipos de ramas: features (funcionalidades), bugfix (bugs) y refactor.

La rama master la utilizamos cada vez que llegamos a una etapa estable que tuviera algún flujo completamente cerrado. Sería como la rama de producción de nuestro proyecto.

La rama develop es donde mergeamos cuando una de las ramas features/bugfix/-refactor fue revisada y no tiene errores (por lo menos al probar manualmente el desarrollador). Sería como la rama de pruebas para que el cliente pueda ver la última versión del producto y pueda probar.

Cada tarea en la cual trabajamos a lo largo del obligatorio la podemos catalogar bajo una de las tres tipos de ramas comentadas más arriba.

Para cada nuevo trabajo que realizamos, creamos una rama que partiera de develop y cuando el trabajo está finalizado y testeado, subimos el branch y creamos un pull request, por lo general acompañado de una breve descripción y un enlace a la card de trello correspondiente.

Luego está el proceso de validación del pull request creado, el miembro del equipo que no trabajo en esa funcionalidad se encarga de hacer una revisión del código, y si considera necesario puede dejar comentarios, que le pueden servir de ayuda al desarrollador para realizar cambios en pos de mejoras en cuanto a estándares de codificación, detección de errores, entre otros. En caso de que considere que todo está funcionando correctamente, aprueba el pull request y se hace un merge de la rama sobre develop. Luego el flujo se reinicia.

En cuanto a las ramas refactor, las creamos con los objetivos de: mejorar la facilidad de comprensión del código y eliminar código muerto. En el proceso de refactorización, no arreglamos errores ni incorporamos funcionalidades, sino que alteramos la estructura interna del código sin cambiar el comportamiento externo.

1.3.2. Uso de comentarios en commits

Un commit es la acción de guardar o subir archivos a un repositorio. Los mensajes de commit son una herramienta muy útil en el desarrollo del software, ya que nos permiten encontrar cambios concretos en la historia de nuestro repositorio.

Cuando realizamos un commit, nos enfocamos en que el mensaje sea conciso y claro, de forma que brinde la suficiente información como para determinar que cambios se realizaron en el commit.

1.4. Resumen del log de versiones

Dado que no nos pareció la mejor forma de representar nuestro trabajo el log de Github decidimos mostrar los Pull Request que hemos abierto y cerrado a lo largo del proyecto.

Como hemos comentado, utilizamos distintas ramas para representar que tipo de valor estabamos agregando ya sea a la rama master o a develop.

A su vez para no quedarnos solo en el comentario del commit, tratamos de agregar una descripción corta y el link a la card de Trello [3] correspondiente para su mayor entendimiento.

Estas capturas fueron tomadas días antes de terminar la implementación, por lo tanto la cantidad de Pull Request aumentó.










<input type="checkbox"/>	 0 Open ✓ 42 Closed	Author ▾	Labels ▾	Projects ▾	Milestones ▾	Reviews ▾	Assignee ▾	Sort ▾
<input type="checkbox"/>	 jacoco integration #48 by Marcoo09 was merged 2 hours ago							
<input type="checkbox"/>	 the unit test in whole application #47 by agustinh2000 was merged 2 hours ago							
<input type="checkbox"/>	 advance in unit tests #46 by agustinh2000 was closed 2 hours ago							
<input type="checkbox"/>	 Bugfix/buy is allowed without products in cart #45 by Marcoo09 was merged 3 hours ago							
<input type="checkbox"/>	 arrows and new logic #42 by Marcoo09 was merged 6 hours ago							
<input type="checkbox"/>	 Feature/resume screen #41 by Marcoo09 was merged 7 hours ago							
<input type="checkbox"/>	 add a dashbord with charts in seller section #40 by agustinh2000 was merged yesterday							
<input type="checkbox"/>	 table with most reused packages #39 by agustinh2000 was merged 2 days ago							

Figura 1.1: Pagina 1 Pull Request











<input type="checkbox"/>	 improvements in client and in read only tab #38 by Marcoo09 was merged 3 days ago	
<input type="checkbox"/>	 Refactor/improvements in navigation #37 by Marcoo09 was merged 3 days ago	
<input type="checkbox"/>	 client registration window #36 by agustinh2000 was merged 3 days ago • Approved	 1
<input type="checkbox"/>	 Point of sale buy flow #35 by Marcoo09 was merged 3 days ago	
<input type="checkbox"/>	 Map improvements #34 by Marcoo09 was merged 3 days ago	
<input type="checkbox"/>	 advance in map and fix bugs in the purchase process #33 by agustinh2000 was merged 3 days ago • Approved	 4
<input type="checkbox"/>	 improvements buy flow #29 by agustinh2000 was merged 3 days ago	
<input type="checkbox"/>	 Feature/seller and buyer connection #28 by agustinh2000 was merged 4 days ago	

Figura 1.2: Pagina 1 Pull Request

<input type="checkbox"/>	0 Open ✓ 42 Closed	Author ▾	Labels ▾	Projects ▾	Milestones ▾	Reviews ▾	Assignee ▾	Sort ▾
<input type="checkbox"/>	filtered sales per month #18 by agustinh2000 was merged 7 days ago • Approved							3
<input type="checkbox"/>	add validations when adding a product #17 by agustinh2000 was merged 7 days ago							
<input type="checkbox"/>	left alignment of tables #15 by agustinh2000 was merged 7 days ago							
<input type="checkbox"/>	Feature/related seller windows #14 by agustinh2000 was merged 9 days ago							
<input type="checkbox"/>	Feature/pre sales logic #13 by agustinh2000 was merged 9 days ago • Approved							3
<input type="checkbox"/>	add a pie chart with the amount of organic and inorganic products sold #12 by agustinh2000 was merged 11 days ago • Approved							
<input type="checkbox"/>	Feature/pie chart of organic products #11 by agustinh2000 was closed 11 days ago							
<input type="checkbox"/>	Feature/most sold products logic #10 by agustinh2000 was merged 12 days ago							

Figura 1.3: Pagina 2 Pull Request

<input type="checkbox"/>	improvements menu screen images with custom components #9 by Marcoo09 was merged 12 days ago							
<input type="checkbox"/>	Feature/add product logic #8 by agustinh2000 was merged 12 days ago • Approved							3
<input type="checkbox"/>	Feature/menu screen #7 by Marcoo09 was merged 14 days ago							
<input type="checkbox"/>	Feature/menu screen #6 by Marcoo09 was merged 14 days ago							
<input type="checkbox"/>	Refactor/modify jar route #5 by Marcoo09 was merged 14 days ago							
<input type="checkbox"/>	Home screen #4 by Marcoo09 was merged 14 days ago							
<input type="checkbox"/>	Feature/setting up interfaces #3 by Marcoo09 was merged 14 days ago							
<input type="checkbox"/>	domain #2 by Marcoo09 was merged 14 days ago							
<input type="checkbox"/>	Domain classes #1 by agustinh2000 was closed 14 days ago							

Figura 1.4: Pagina 2 Pull Request

2. Codificación

2.1. Estándar de codificación

En este obligatorio tratamos de emular lo más posible un escenario de un proyecto de la vida real, por eso utilizamos Git Flow como flujo de trabajo y por eso mismo tratamos de utilizar criterios de codificación que se sigan en equipos de empresas de Software.

Lo primero que nos gustaría comentar es que escribimos todo el código, sin ser los strings de la interfaz en inglés, porque nos parece que el código tiene que ser lo más universal posible y que es muy probable que en el ámbito laboral tengamos que trabajar con personas de otras nacionalidades.

Luego seguimos algunos principios que son nombrados en el libro “Clean Code” como lo son:

- Nombres auto descriptivos para tratar de evitar escribir comentarios innecesarios y leer más fácilmente el código, como se observa en la figura 2.4.
- Siguiendo lo comentado en el primer punto, tratamos de evitar lo más posible los comentarios en el código.
- Formatear el código para que sea coherente en cualquier parte del proyecto.
- Tratamos de evitar repetir código, aunque muchas veces por falta de experiencia en JavaFX tuvimos que hacerlo, cosas así como el menú, las navegaciones, entre otras cosas no logramos abstraerlas de manera que fueran fácil de usar desde todas las pantallas.
- Tratamos de seguir el principio de única responsabilidad, principalmente en las clases del dominio y en las funciones que inicializan el estado de cada vista, luego en funciones más complejas de la interfaz muchas veces esto no se cumplió.
- Utilización de diferentes paquetes, con el objetivo de separar dominio de interfaz, imágenes y otros recursos. Los nombres de los mismos fueron escritos

en minúscula. En nuestro caso tenemos los paquetes: components (con componentes de interfaz), domain (clases del dominio), interfaces (ventanas), interfaces.htmlResources (mapa) y resources (imágenes), tal como se observa en la figura 2.1.

- Inclusión de código fuente en las clases del dominio para generar JavaDoc, el cual es el estándar de la industria para documentar clases de Java. Para ello, en cada función indicamos el significado de cada parámetro de entrada y el retorno, como se observa en la figura 2.3 y en el encabezado de la clase, incluimos una descripción de la misma y el nombre de los autores, tal como se observa en la figura 2.2.

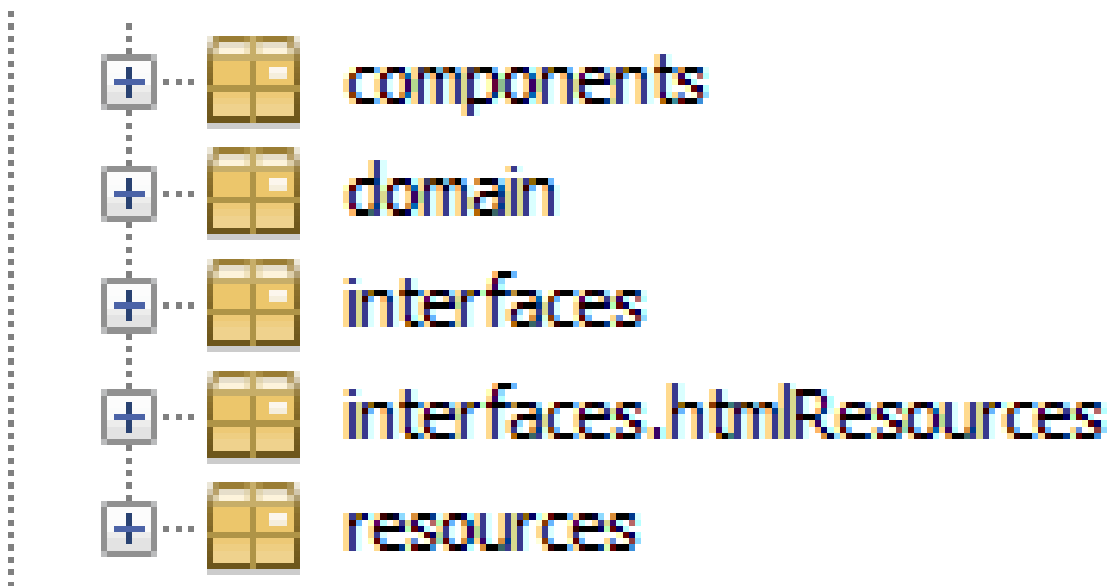


Figura 2.1: Utilización de diferentes paquetes.

```
/**
 * This class contains all the data collections: packages, sales, pre-sales,
 * points of sale.
 *
 * @author Agustin Hernandorena and Marco Fiorito
 */
public class System {
```

Figura 2.2: Encabezado de clase.

```

/**
 * Method that returns an array, in position 0 the quantity of organic
 * products sold and in position 1 the quantity of inorganic products sold.
 *
 * @return An array, in position 0 the quantity of organic products sold and
 * in position 1 the quantity of inorganic products sold.
 */
public int[] quantityOfOrganicProductsSold() {
    //In position 0 put the organics and in 1 the inorganics.
    int[] organicAndInorganic = new int[2];
    ArrayList<Pair> list = totalQuantitySoldPerProduct();
    for (int i = 0; i < list.size(); i++) {
        Product aProduct = (Product) list.get(i).getKey();
        int quantity = (int) list.get(i).getValue();
        if (aProduct.isOrganic()) {
            organicAndInorganic[0] += quantity;
        } else {
            organicAndInorganic[1] += quantity;
        }
    }
    return organicAndInorganic;
}

```

Figura 2.3: Detalle de parámetros y retorno en función.

```

private Client client;
private ArrayList<Package> packagesList;
private ArrayList<PointOfSale> salePoints;
private ArrayList<Product> products;
private ArrayList<Sale> sales;
private ArrayList<Sale> preSales;
private boolean isInPreSaleMode;

```

Figura 2.4: Nombres de atributos autodescriptivos.

2.2. Pruebas unitarias

En esta parte del obligatorio, primero que nada, nos enfocamos en hacer un análisis previo de cada clase a probar, como fruto de este análisis logramos tener claro cual es el dominio de cada función, cuales son los casos bordes y que datos de prueba vamos a necesitar generar para probar cada función.

Luego empezamos a realizar las pruebas en cada clase del dominio. En cada prueba, se probaron casos de borde, casos dentro del dominio, y casos por fuera del mismo.

Si en alguna función estaba presente un condicional, probamos con entradas que cumplan el condicional y otras que no. Con el objetivo de encontrar los problemas rápidamente, cada vez que finalizamos un test, lo ejecutamos, y chequeamos si pasa la prueba correctamente. Consideramos que el no ejecutar los test con relativa frecuencia puede hacer que tardemos mas en encontrar los problemas que vayan surgiendo.

Otro aspecto que consideramos, es el de realizar los test de forma independiente, para que el resultado de un test no condicione el de otros.

Para realizar el análisis de cobertura de pruebas, utilizamos el plugin de NetBeans TikiOne JaCoCoverage [4], el detalle de cobertura se encuentra en la figura 2.5.

domain

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Miss
System.new Comparator() {...}		0%		0%	4	4	6	6	
System		100%		98%	1	54	1	122	
Sale		100%		100%	0	32	0	72	
Product		100%		100%	0	20	0	44	
PointOfSale		100%		n/a	0	15	0	35	
Client		100%		n/a	0	13	0	31	
Package		100%		n/a	0	9	0	20	
Total	32 of 1.246	97%	5 of 68	93%	5	147	3	326	

Created with [JaCoCo](#) 0.7.6.201602180812

Figura 2.5: Detalle de cobertura de pruebas

Nos hubiera gustado usar la metodología TDD [5] pero dado que no tenemos experiencia en JavaFX no quisimos utilizar mucho tiempo en aplicarla y nos enfocamos más en el desarrollo.

2.3. Análisis de código

Con respecto a este tema, como herramienta de análisis de código utilizamos Find Bugs [6], la cual provee la funcionalidad de inspeccionar el código seleccionado, así como ofrecer un conjunto de opciones sobre que tipo de errores se desea detectar.

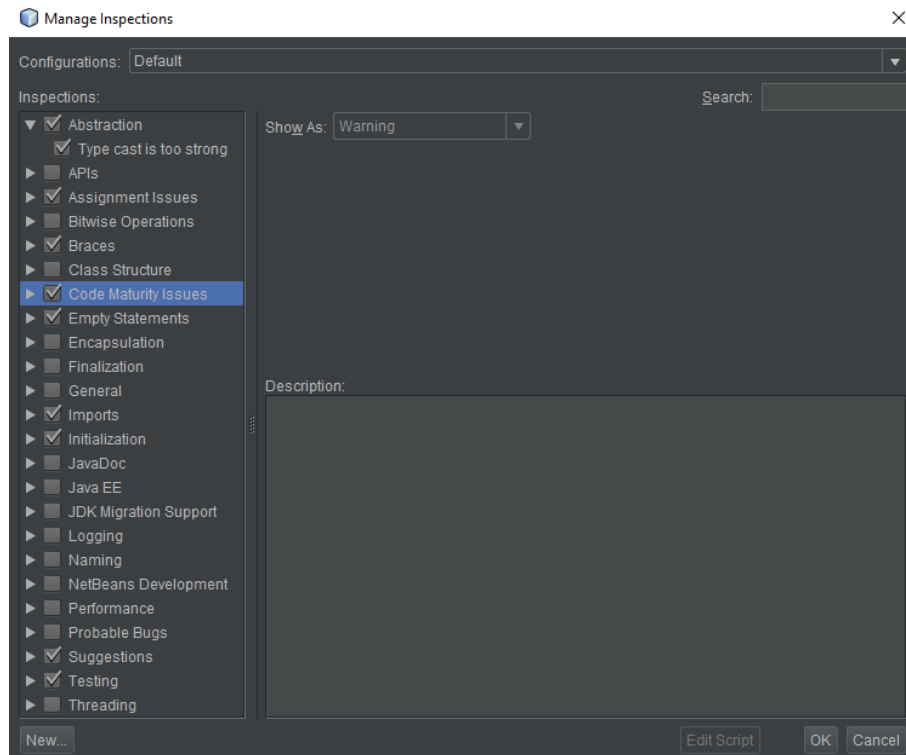


Figura 2.6: Inspecciones de find bugs.

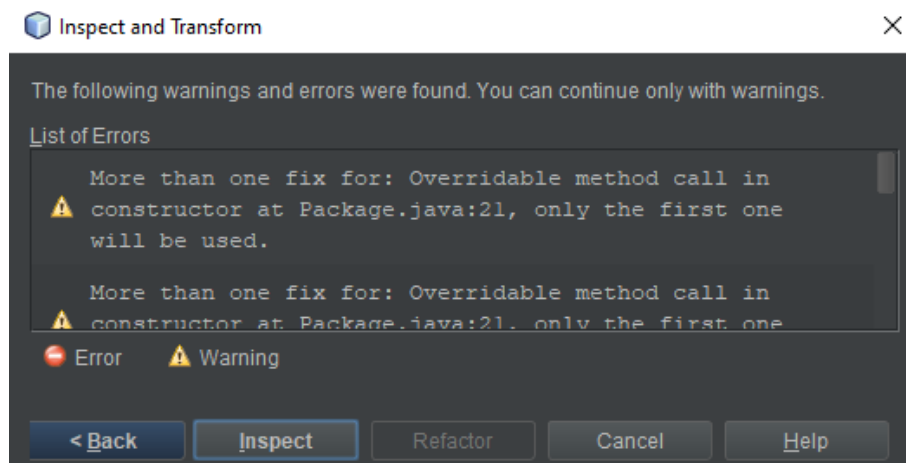


Figura 2.7: Inspecciones de find bugs.

Luego de realizar los pasos anteriores, seleccionamos en Inspect y luego en Refactor para aplicar los cambios sugeridos por la herramienta. Por último en el pull request correspondiente vimos en específico que cambios realizó la herramienta y nos parecieron correctos por lo tanto procedimos a mergear. Las principales mejoras sugeridas fueron relacionado a que en varios casos comparabamos strings con `==` y no con `equals`, código mal formateado, variables sin usar e importaciones que no estaban siendo usadas.

3. Interfaz de usuario y usabilidad

3.1. Criterios de interfaz de usuario

La aplicación fue desarrollada con la herramienta JavaFX, una plataforma que permite a los desarrolladores de la aplicación crear e implementar fácilmente aplicaciones de Internet enriquecidas (RIA) que se comportan de la misma forma en distintas plataformas [7]. Enfocándonos en la interfaz de usuario, la misma fue desarrollada siguiendo los principios de Material Design [8], para ello hicimos uso de la librería JFoenix [9], la cual incluye distintos componentes: botones, calendario, ventanas de diálogo, que siguen la normativa de diseño antes mencionada.

Una vez que el usuario ingresa al sistema, visualiza la pantalla principal que cuenta con dos opciones: una para ingresar en modo comprador y otra en vendedor.

Estas pantallas cuentan con una side navigation bar que permite acceder en cualquier momento a todas las funcionalidades del sistema de manera muy eficiente, esta barra de navegación esta compuesta por diferentes opciones, cada una de ellas conformada por un icono extraído de la web oficial Material Design y un breve texto vinculado a la funcionalidad. Cuando el usuario se encuentra dentro de una determinada opción, el texto correspondiente a esa opción se muestra resaltado, para que el usuario pueda ver el status del sistema y saber en que opción esta en ese momento. En cuanto a la legibilidad, la interfaz esta compuesta por un lenguaje simple, con una tipografía sencilla y clara, que ayuda a la rápida lectura por parte del usuario, así como también a que se acostumbre mas rápido a las diferentes funciones del sistema dado que todas las pantallas son coherentes entre sí.

En cuanto a la paleta de colores, decidimos utilizar una paleta basada en azules, grises y blanco la cual le da un look and feel ameno y consistente en todo el sistema. En cuanto a la gestión de errores, la interfaz ayuda a los usuarios a reconocer, diagnosticar y recuperarse de los errores. Por ejemplo, en la función de agregar un producto al sistema, si el usuario deja algún campo sin completar el sistema muestra un mensaje en un lenguaje claro, indicando exactamente el problema.

Nos gustaría comentar también que tratamos de realizar una aplicación de escritorio que sigue los principios de Responsive Design situando los componentes en la pantallas con distancias relativas a los bordes, y a medida que se aumenta o disminuye el tamaño de la pantalla, los componentes se adaptan al nuevo tamaño, nos hubiera gustado también ocultar ciertos componentes (tal como la side navigation) al tener un tamaño de pantalla pequeño pero no nos dió el tiempo.

Otro aspecto que tuvimos en cuenta fue la consistencia, en lo que refiere a la uniformidad en apariencia, colocación y comportamiento de los componentes de la interfaz, consideramos que la aplicación de este criterio es importante porque ayuda a reducir el esfuerzo del usuario para aprender, así como las habilidades requeridas para su aprendizaje.

Decidimos incluir en varias secciones gráficos, de barra y circular que le permiten al EcoShop obtener una visualización rápida de estadísticas de la empresa en cuanto a ventas por mes y cantidad de productos orgánicos vendidos.

3.2. Evaluación de usabilidad

Para medir la usabilidad del sistema usamos diferentes técnicas, una de ellas es la prueba con usuarios. En estas pruebas con usuarios, tratamos de simular un entorno semejante al de trabajo en situaciones reales, y medimos tres aspectos:

- **Eficacia:** Observamos algunos aspectos como: si el usuario puede encontrar las funcionalidades básicas de la aplicación, si puede hacer uso de las funcionalidades sin cometer errores.
- **Eficiencia:** Observamos el tiempo que les llevaba poder realizar cada funcionalidad. Tomamos en cuenta: el tiempo invertido en el primer intento, el tiempo requerido para realizar una determinada funcionalidad comparado con el tiempo que le lleva a una persona acostumbrada a utilizar el sistema, y el tiempo invertido en subsanar errores cometidos.
- **Satisfacción:** Observamos la impresión del usuario respecto de la GUI, y que tan motivado se encuentran con su uso.

Otra técnica que utilizamos para evaluar la usabilidad, fue la de ver las heurísticas de usabilidad, en donde se analiza el sistema utilizando una serie de guías. Para ello, nos basamos en las heurísticas de Nielsen [10] que se describen a continuación.

1. **Visibilidad del estado del sistema:** El sistema muestra al usuario el estado en que se encuentra, en nuestro caso al completar un formulario se muestra un mensaje que se ha agregado correctamente o en caso de error se muestra un mensaje de error.
2. **Utilizar el mismo lenguaje que el usuario:** El sistema utiliza el lenguaje de los usuarios, con palabras y términos que le sean conocidos.
3. **Control y libertad para el usuario:** En caso de que el usuario elija una opción del sistema por error, este debe ser capaz de dar opciones de deshacer o rehacer de forma que el usuario nunca pierda el control total del sistema. En nuestro caso, el comprador tiene la opción de llenar su perfil, y una vez

completado, no es necesario volver a llenar los datos cuando desea comprar un producto.

4. **Consistencia y estándares:** El usuario no debe estar preguntándose siempre si un determinado botón hará una acción, o si es un botón o un enlace, o si ese elemento es clickeable o no. Consideramos que nuestro sistema es consistente y no hay ambigüedades presentes que hagan dudar y cometer errores al usuario.
5. **Prevención de errores:** En el sistema, no hay acciones predisuestas al error.
6. **Minimizar la carga de la memoria del usuario:** Con la inclusión del side navigation bar, el usuario sabe en donde se encuentra en ese momento y tiene la opción de ir a cualquiera de las restantes opciones solo clickeando en la barra lateral, esto ayuda a que el usuario no tenga que memorizar lo que tiene que hacer para moverse de una ventana a otra, o para realizar determinada tarea.
7. **Flexibilidad y eficiencia de uso:** Consideramos que el sistema se adapta a los usuarios, brindando respuestas rápidas e interacciones amigables.
8. **Estética y diseño minimalista:** Consideramos que el sistema tiene una interfaz sencilla y fácil de entender.
9. **Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de los errores:** Un ejemplo de nuestro sistema que sigue esta regla es en la función de agregar un nuevo producto al sistema. En el caso de que el usuario deje algún campo sin completar, el sistema muestra un mensaje de error claro y conciso indicando exactamente cual fue el problema, que le es de ayuda al usuario para solucionarlo rápidamente.
10. **Ayuda y documentación:** Por las características que presenta el sistema, no consideramos necesario incluir una sección de ayuda y documentación.

4. Pruebas funcionales

4.1. Técnicas de prueba aplicadas

La principal técnica de prueba que aplicamos fueron las pruebas exploratorias, en este tipo de pruebas, simultáneamente se aprende sobre la aplicación, se diseñan casos de prueba, y se ejecutan esos casos de prueba. Con las conclusiones que íbamos obteniendo fuimos aprendiendo mas sobre la aplicación, y utilizamos esa información para diseñar y ejecutar nuevas pruebas. Antes de hacer un test exploratorio, nos planteamos el objetivo que queremos conseguir con ese test, generalmente establecer flujos que podrían seguir los usuarios de la aplicación y probarlos.

4.2. Casos de prueba

Probamos la función de agregar un producto al sistema. Para ello, en primer lugar utilizamos una estrategia basada en escenarios para encontrar todos los escenarios del caso de uso, tal como se observa en la figura 4.1.

Escenario	Nombre	Curso de comienzo	Curso alternativo
Escenario 1	Registra el producto.	Camino básico	
Escenario 2	Campo sin completar.	Camino básico	CA 4.1

Figura 4.1: Escenarios del caso de uso.

Luego, generamos los casos de prueba para los escenarios encontrados, tal como se observa en la figura 4.2.

Caso de prueba	Escenario	Nombre	País de origen	Precio	Material	Orgánico	Con materiales reciclados	Lista de envases disponibles	Resultado esperado
CP 1.1	Escenario 1	V	V	V	V	V	V	V	Registra el producto
CP 1.2.1	Escenario 2	NV	V	V	V	V	V	V	Campo sin completar
CP 1.2.2	Escenario 2	V	NV	V	V	V	V	V	Campo sin completar
CP 1.2.3	Escenario 2	V	V	NV	V	V	V	V	Campo sin completar
CP 1.2.4	Escenario 2	V	V	V	NV	V	V	V	Campo sin completar
CP 1.2.5	Escenario 2	V	V	V	V	V	V	NV	Campo sin completar

Figura 4.2: Casos de prueba para los escenarios encontrados.

Luego, generamos los datos de prueba utilizando la técnica de particiones de equivalencia, tal como se observa en la figuras 4.3 y 4.4.

Condición	Clases válidas	Clases no válidas
Nombre	Carácter alfanumérico. (1)	Nombre vacío. (8)
País de origen	Carácter alfanumérico. (2)	País de origen vacío. (9)
Precio	Carácter numérico mayor a cero. (3)	Precio no numérico. (10)
		Precio vacío. (11)
		Precio menor o igual a cero. (12)
Material	Carácter alfanumérico. (4)	Material vacío. (13)
Orgánico	Cualquier entrada. (5)	Ninguna entrada. (14)
Con materiales reciclados	Cualquier entrada. (6)	Ninguna entrada. (15)
Lista de envases disponibles.	Lista con al menos un elemento. (7)	Lista vacía. (16)

Figura 4.3: Técnica de particiones de equivalencia.

Caso de prueba	Escenario	Nombre	País de origen	Precio	Material	Orgánico	Con materiales reciclados	Lista de envases disponibles	Resultado esperado	Clases
CP 1.1.1	Escenario 1	Almendras	Brasil	170	No aplica	V	F	[Tupper hermético, Bollon]	Registra el producto.	(1), (2), (3), (4), (5), (6), (7)
CP 1.2.1.1	Escenario 2	***	Uruguay	190	No aplica	V	F	[Tupper hermético]	Error: Debe ingresar un nombre.	(8)
CP 1.2.2.1	Escenario 2	Nueces	***	120	No aplica	V	F	[Bollon]	Error: Debe ingresar un país de origen.	(9)
CP 1.2.3.1	Escenario 2	Maní	Argentina	***	No aplica	V	F	[Bolsa Ziploc]	Error: Debe ingresar un precio.	(11)
CP 1.2.3.2	Escenario 2	Ajo	Uruguay	-10	No aplica	V	F	[Bollon]	Error: El precio debe ser positivo.	(12)
CP 1.2.3.3	Escenario 2	Almendras con chocolate	Uruguay	0	No aplica	V	F	[Tupper hermético]	Error: El precio debe ser positivo.	(12)
CP 1.2.3.4	Escenario 2	Almendras acarameladas	Brasil	a	No aplica	V	F	[Bolsa Ziploc]	Error: El precio debe ser un número.	(10)
CP 1.2.4.1	Escenario 2	Pasas de uva	Brasil	150	***	F	F	[Tupper hermético]	Error: Debe ingresar un material.	(13)
CP 1.2.5.1	Escenario 2	Nueces	Uruguay	190	No aplica	V	F	[]	Error: Debe seleccionar algún envase.	(16)

Figura 4.4: Datos de prueba.

4.3. Sesiones de ejecución de pruebas

A continuación se detallan las sesiones de ejecución de pruebas:

- **Agregar producto al sistema:**

Tester: Agustín Hernandorena

Fecha: 16/11/2019

Hora: 17:30

Entorno: Entorno de laboratorio.

- **Lista de pre-ventas:**

Tester: Marco Fiorito

Fecha: 17/11/2019

Hora: 22:30

Entorno: Entorno de laboratorio.

■ **Producto mas vendido:**

Tester: Marco Fiorito

Fecha: 18/11/2019

Hora: 21:30

Entorno: Entorno de laboratorio.

■ **Envases reutilizados:**

Tester: Agustín Hernandorena

Fecha: 19/11/2019

Hora: 16:00

Entorno: Entorno de laboratorio.

■ **Ventas por mes:**

Tester: Marco Fiorito

Fecha: 20/11/2019

Hora: 20:30

Entorno: Entorno de laboratorio.

■ **Cantidad de productos orgánicos venidos:**

Tester: Agustín Hernandorena

Fecha: 19/11/2019

Hora: 18:30

Entorno: Entorno de laboratorio.

■ **Dashboard:**

Tester: Agustin Hernandorena

Fecha: 21/11/2019

Hora: 23:00

Entorno: Entorno de laboratorio.

■ **Compra:**

Tester: Marco Fiorito

Fecha: 23/11/2019

Hora: 15:00

Entorno: Entorno de laboratorio.

5. Reporte de defectos

5.1. Definición de categorías de defectos

Los defectos los categorizamos en tres tipos:

- **Compilation errors:** Estos son los menos comunes, ya que generalmente se verifica antes de subir un archivo al repositorio que el mismo compile.
- **Run time errors:** Estos son los que ocurren con mayor frecuencia, se produce cuando esta ejecutando el programa, se denominan excepciones y ocurren en alguna de las instrucciones del programa, por ejemplo: cuando un objeto es null, al realizar una división por cero.
- **Logic errors:** Sucede cuando alguno de los desarrolladores no comprendió completamente como debería haberse implementado una determinada función.⁴

Un ejemplo de nuestro obligatorio puede ser que habíamos implementado el ir al carrito de compra sin hacer la restricción de que el usuario haya seleccionado productos, dando así un comportamiento incorrecto en el programa.

Los defectos tienen asociados una grado de severidad, es decir el impacto que tiene el defecto en el sistema.

Para ello, utilizamos la escala que se indica a continuación:

- **High:** El defecto afecta a una funcionalidad principal del sistema. Hay una solución alternativa, pero no es muy intuitiva.
- **Medium:** El defecto afecta a una funcionalidad menor del sistema. Hay una solución alternativa que es posible implementarla de forma sencilla.
- **Low:** El defecto no afecta a una funcionalidad del sistema. Simplemente se trata de inconvenientes menores: discrepancias menores en cuanto al diseño, aspectos de usabilidad, etc.

5.2. Defectos encontrados por iteración

Los defectos encontrados a lo largo del proyecto, fueron reportados mediante issues en GitHub. Un issue es la unidad de trabajo designada para realizar una

mejora en un sistema informático.

En el título del reporte se indica el tipo de defecto: compilation error, run time error o logic error, junto al grado de severidad: high, medium o low, y una breve descripción del problema.

En el mensaje se indica que pasos se siguieron para llegar al error, para que el miembro del equipo que tenga que resolver el defecto, sepa en que parte del sistema puede estar el error.

Luego cuando se soluciona el error el issue es cerrado y el que lo soluciona comenta que esto quedó solucionado, y comentar en que Pull Request fue solucionado.

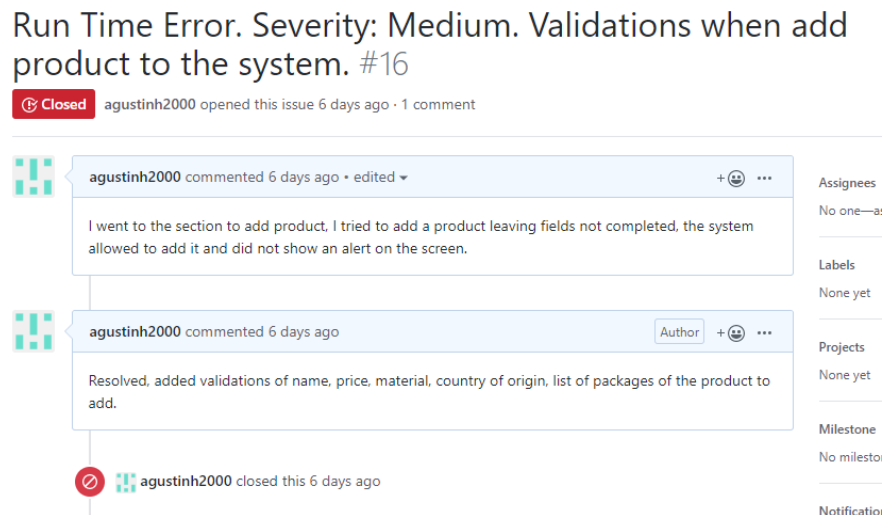


Figura 5.1: Error: ausencia de validaciones al agregar producto.

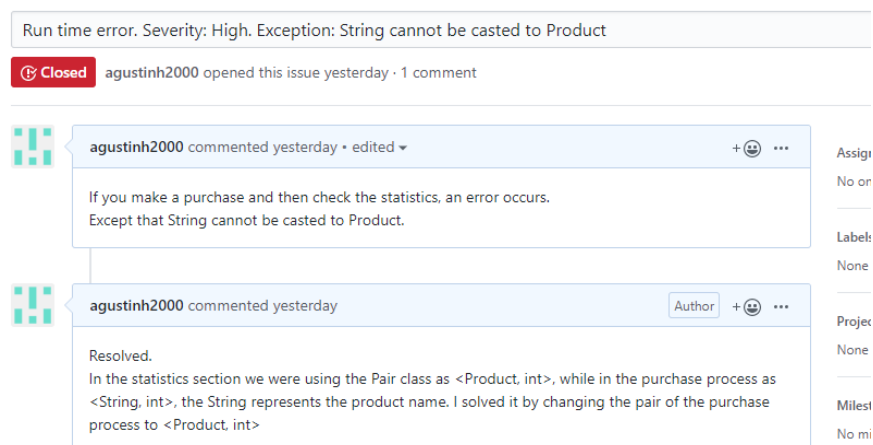




Figura 5.2: Excepción: String no puede ser casteado a Product.


Run time error. Severity: Medium. Purchase price equal to 0 #31

 Closed agustinh2000 opened this issue yesterday · 1 comment





agustinh2000 commented yesterday · edited ▾

When making a purchase, the price is not set, so it remains at 0.



agustinh2000 commented yesterday · edited ▾

Resolved.
In the purchase process it was necessary to invoke the function obtainPrice(), to calculate the sale price.

 agustinh2000 closed this yesterday

Assignees

No one—

Labels

None yet

Projects


None yet


Milestone

No milestone

Figura 5.3: Error: Precio de compra igual a cero.


Run time error. Severity: Medium. Purchase Date =Null

 Closed agustinh2000 opened this issue yesterday · 1 comment



agustinh2000 commented yesterday · edited ▾

If you make a pre-purchase, the date is set correctly, but if it is not pre-purchased, the date is null.



agustinh2000 commented yesterday

Resolved. The purchase date was not included in the default constructor.



 agustinh2000 closed this yesterday

Figura 5.4: Excepción: Fecha de compra = “null”.

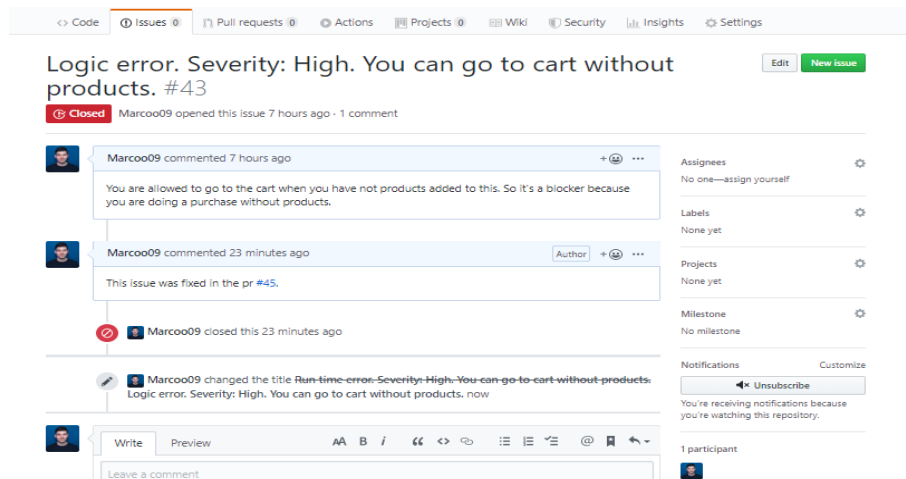


Figura 5.5: Logic error. Severity: High. You can go to cart without products.

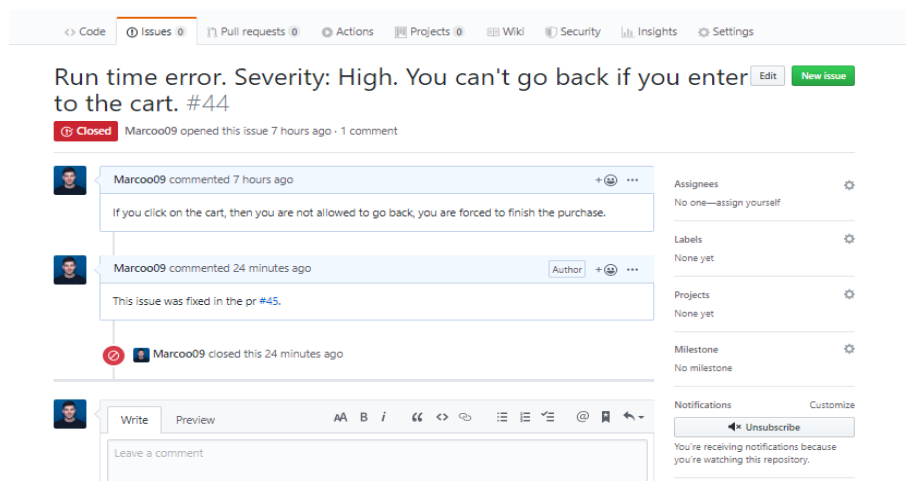


Figura 5.6: Run time error. Severity: High. You can't go back if you enter to the cart.

5.3. Estado de calidad global

Consideramos que el estado de calidad del software es bastante bueno, utilizamos nombres de variables, funciones, clases e interfaces claros en la mayoría de los casos. Utilizamos prácticas conocidas para escribir código limpio.

Mantuvimos un flujo de trabajo el cual nos limita en la cantidad de cosas distintas que se agregan al proyecto, reduciendo cantidad de bugs por omisión al agregar muchas cosas (sin relación entre sí directamente) al proyecto.

Luego realizamos unit test de las clases del dominio llegando a un 100 % de coverage complementado con test manual de los flujos principales de la aplicación (dado que no hicimos unit test de los controladores de las interfaces).

A su vez utilizamos herramientas como FindBugs y JaCoCo para encontrar bugs y saber el coverage de los unit tests.

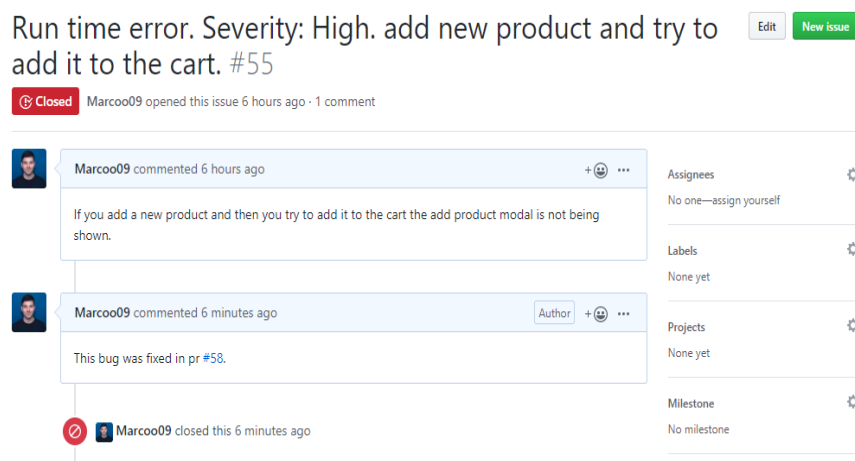


Figura 5.7: Run time error. Severity: High. add new product and try to add it to the cart.

Por último nos parece que en terminos de diseño la aplicación resultante es buena, dado que usa principios de Material Design y una paleta de colores bien definida.

6. Reflexión

Cómo reflexión nos parece que aprendimos mucho en este obligatorio, complementando lo aprendido en el obligatorio 1 vinculado a la organización, distribución de tareas, practicas de desarrollo de software y uso de Git para el versionado, Trello para la organización, test unitarios y herramientas como FindBugs.

Nos pareció muy útil utilizar Trello para crear las tareas que debíamos hacer, así como las que estaban en progreso, realizadas o estaría bueno realizarlas.

Con respecto a Git, tratamos de tener practicas lo más reales posible para estar preparados para el mercado laboral y entender porque es bueno agregar valor lo más atómico posible.

Luego JaCoCo nos ayudó a testear flujos que no habíamos pensado dado que muchas veces sucede de que testeas manualmente los casos ideales y no los erróneos.

Por último FindBugs lo utilizamos al final del proyecto, pero, nos hubiera gustado utilizarlo en desarrollo antes de subir cada pull request chequear que los estándares de calidad siempre son los más altos.

Nos parece muy bueno este approach de hacer obligatorios más prácticos dado que nos hace pensar como utilizaría un usuario real la aplicación, que le sería intuitivo y que no, así como funcionalidades útiles para los usuarios como en nuestro caso fue un Dashboard con varias gráficas para el vendedor. Además este approach permite auto exigirse y no atarse al camino más fácil e investigar cosas nuevas.

Además al ser abierto por así decirlo, te da libertad de probar metodologías como en nuestro caso Git Flow o intentar integrar herramientas, librerías útiles.

Bibliografía

- [1] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.
- [2] Git-scm. (2019) git. [Online]. Available: <https://git-scm.com/>
- [3] Trello. (2019) Trello. [Online]. Available: <https://trello.com/>
- [4] NetBeans. (2013) TikiOne JaCoCoverage. [Online]. Available: <http://plugins.netbeans.org/plugin/48570/tikione-jacocoverage>
- [5] Paradigma digital. (2019) TDD como metodología de diseño de software. [Online]. Available: <https://www.paradigmadigital.com/dev/tdd-como-metodologia-de-diseno-de-software/>
- [6] NetBeans. (2007) FindBugs(TM). [Online]. Available: <http://plugins.netbeans.org/plugin/912/findbugs-tm-plugin>
- [7] ProgramaEnLinea. (2015) ¿Que es JavaFX? [Online]. Available: <http://programaenlinea.net/que-es-javafx/>
- [8] Material Design. (2019) Material Design. [Online]. Available: <https://material.io/>
- [9] JFoenix. (2019) JavaFX Material Design Library. [Online]. Available: <http://www.jfoenix.com/>
- [10] Enrique Velasco Silva. (2016) Usabilidad: 10 heurísticas de Nielsen. [Online]. Available: <https://www.ramonramon.org/blog/2016/06/21/jakob-nielsen-modelo-usabilidad/>
- [11] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, eight ed. McGraw-Hill, 2014.
- [12] Universidad ORT Uruguay. (2013) Documento 302 - Facultad de Ingeniería. [Online]. Available: <http://www.ort.edu.uy/fi/pdf/documento302facultaddeingenieria.pdf>