Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio 2

Agustín Introini (211064)
Juan Ignacio Balian (211150)

Entregado como requisito de la materia Ingeniería de Software 1

25 de noviembre de 2019

Declaraciones de autoría

Nosotros, Agustín Introini, Juan Balian , 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 Ingeniería de Software 1 ;
- 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 trabajo es el desarrollo de una aplicación, utilizando prácticas tecnológicas y de gestión de la ingeniería de software. Se espera como resultado un software de calidad y la aplicación de un conjunto de prácticas profesionales. En el obligatorio no se evalúa únicamente la implementación, sino el proceso de desarrollo y su impacto en la calidad del software. Los temas centrales de aplicación para este obligatorio son los siguientes:

- Control de versiones
- Código profesional
- Pruebas unitarias automatizadas
- Diseño de interfaz de usuario y usabilidad
- Prueba funcional y reporte de defectos.

Se desea desarrollar una aplicación que soporte las actividad de un **EchoShop** que busca reducir desperdicio en el ciclo de venta. El sistema debe manejar los siguientes conceptos:

1. Gestionar la información básica de artículos. Debe explicar el origen de la materia prima, su respectivo precio, material, código identificador.
2. Empaquetado / envasado: el EcoShop promueve la reutilización de envases, el sistema informa los envases reutilizables que puede aplicarse para cada tipo de producto.
3. Puntos de venta. El sistema deberá mostrar la ubicación de los puntos de venta del EchoShop.
4. Registro de venta. Se deberá registrar la venta, junto con el envase reutilizado y generar ticket electrónico sin imprimir papel. Se podrá utilizar como referencia la normativa de DGI [1].
5. Reportes de estadísticas de relevancia para el negocio: productos más vendidos, envases reutilizados, total de ventas en un mes dado y estimación del beneficio del impacto ambiental generado.
6. Preventa de producto. Se debe disponer de un calendario que permita la preventa de productos. Esto ahorra tiempos de transporte y mejora la conservación de productos perecederos.
7. Se deberá presentar una funcionalidad por parte del equipo que promueva el uso de las 3R y considere aspectos de emprendedurismo del EchoShop.

Índice general

1. Versionado

1.1. Repositorio utilizado

Decidimos utilizar GitHub para alojar nuestro proyecto dado que ya contábamos con cuentas en el sitio y teníamos experiencia previa trabajando con otros proyectos en dicha plataforma.

Tal como se solicita en la letra de este obligatorio, el repositorio fue nombrado *balian-introini* (apellidos de los integrantes). Al mismo se puede acceder mediante el siguiente link: **github.com/juanBalian35/balian-introini**

Cabe destacar que es un repositorio privado, por lo tanto sólo se podrá acceder al mismo con una invitación. La invitación fue mandada a la cuenta (github.com/aadorian) establecida por el docente de la materia.

1.1.1. Elementos de la Configuración de Software (ECS)

El principal elemento incluido en el repositorio es el proyecto entero de NetBeans. Dentro del mismo se pueden encontrar distintos ECS, el principal es el código fuente del sistema. A su vez, dentro de la carpeta se versionan los recursos del sistema, tales como:

- Imagenes de los productos y envases
- Iconos de la interfaz de usuario
- Hojas de estilo en cascada (.css)

Otro ECS que se versiona son las pruebas unitarias (JUnit) ya que se encuentran integradas dentro del proyecto de NetBeans.

Por otro lado, nos parecía interesante hacer un versionado de este documento. Pero tras consultar y averiguar sobre como versionar a través de la herramienta sugerida Overleaf, vimos que no era viable ya que la única forma de generar este versionado es teniendo una cuenta premium (paga) de dicha herramienta. Por lo que decidimos que no sería tenido encuenta este elemento para su versionado.

Por último, otro elemento que se tiene en cuenta, a través de la herramienta GitHub, es el de los issues [1]. Con esta herramienta fuimos versionando, y teniendo un control sobre los problemas con el sistema en su fase de desarrollo.

[1] `https://github.com/juanBalian35/balian-introini/issues`

1.2. Criterios de versionado

Los criterios que tuvimos para versionar fueron:

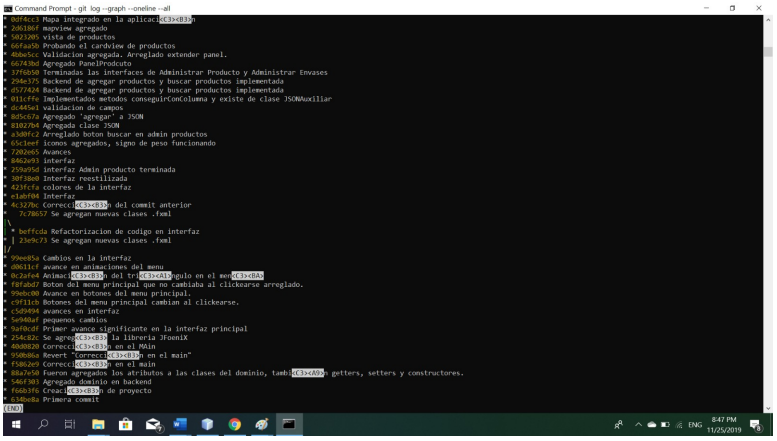
- Utilizar dos ramas master y develop, (en los siguientes párrafos se explica como fueron utilizadas)
- Como la modalidad de trabajo fue siempre estando los dos miembros de éste obligatorio juntos, o bajo una comunicación directa, no se realizó ningún merge, ya que siempre trabajábamos en una sola computadora o no lo hacíamos en simultáneo. Esto se debe en parte, a que es una modalidad un poco más compleja de utilizar y además, siempre se dió de trabajar al mismo tiempo pero en la misma computadora.
- El criterio para realizar commits era cada vez que se avanzaba con algo importante, o que hubiese requerido un tiempo significante de desarrollo.
- Siempre se realizó versionado de todos los elementos de la configuración en conjunto, es decir, incluyendo todo el proyecto de NetBeans.

En todo momento tratamos de hacer un uso profesional de los comentarios en los commits, indicando con precisión que es lo que se estaba avanzando. De todos modos en escasas ocasiones, se incurrió en una falta, ya que se quería realizar el commit de manera rápida y se realizó con comentarios vagos o inconclusos.

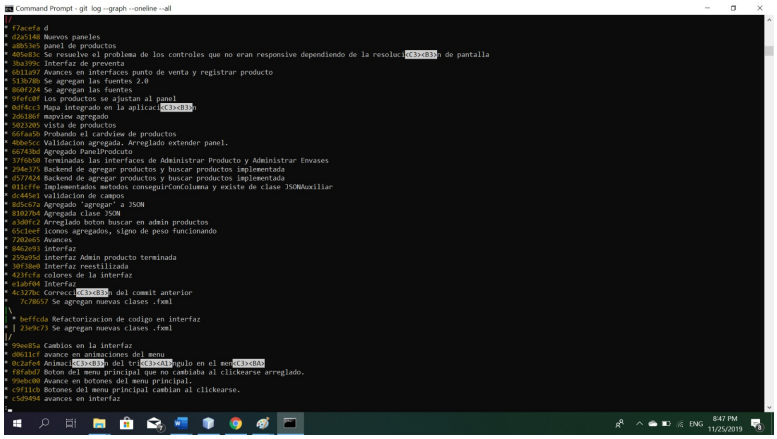
En cuanto al uso de las ramas, abrimos dos de ellas, una llamada master y otra develop. En la master decidimos que la utilizaríamos haciendo merge cuando tuvieramos una versión estable. Esto no sucedió hasta el último día, por lo que se puede ver que hay un único merge hacia esta rama. Tras finalizar el obligatorio, llegamos a la conclusión de que hubiese sido útil abrir otra rama para el diseño de la interfaz, ya que hubiese sido bueno ver el progreso de versiones solo por parte de la interfaz y no de todo en conjunto.

1.3. Resumen del log de versiones

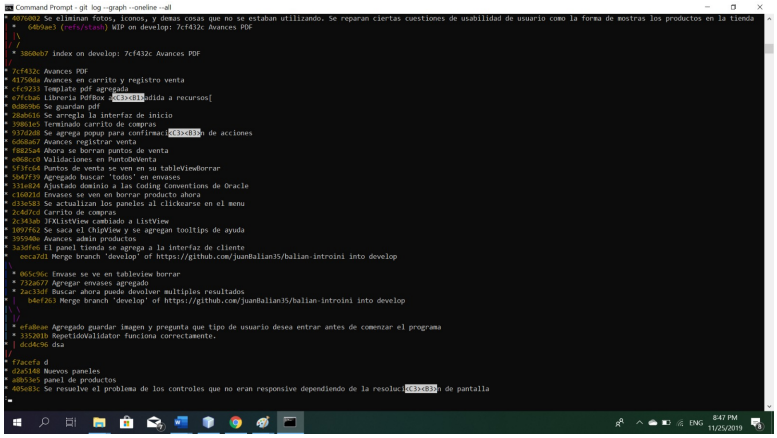
El comando ‘git log –graph –oneline –all‘ nos dispone esta información de la siguiente manera:



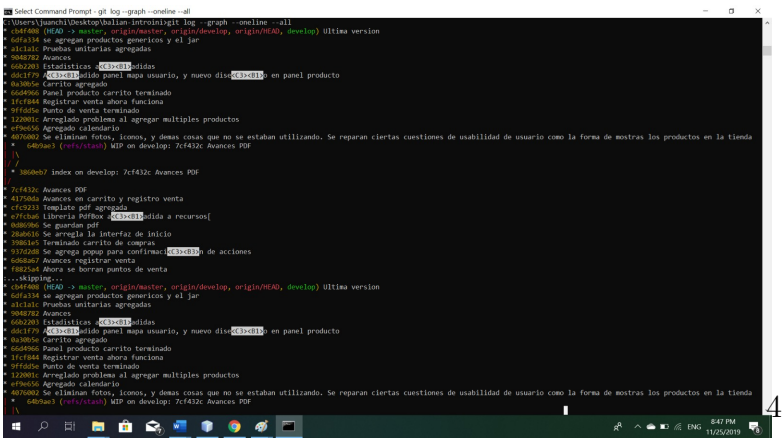
(a) 1



(b) 2



(c) 3



(d) 4

Figura 1.1

2. Codificación

2.1. Estándar de codificación

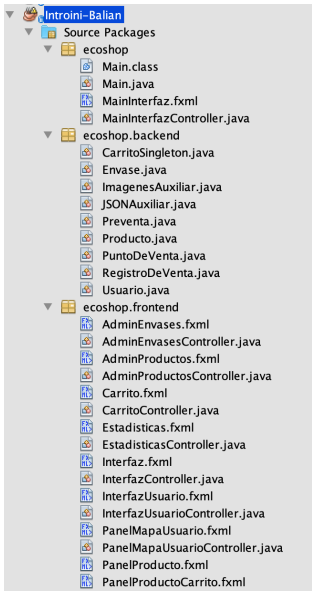
Hemos seguido todas las normas establecidas por los estándares de codificación establecidos. De las más comunes como lo son la indentación, los nombres de los componentes, que son sencillas de llevar a cabo ya que al estar utilizando un IDE es automático, más aún teniendo activada la función Java Hints en Netbeans.

Además, hemos seguido el estándar de Java Code Convention [2], esto se puede apreciar a continuación:

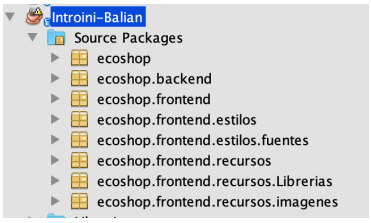
- Clases comienzan en mayúscula (2.1a).
- Los paquetes comienzan con minúscula (2.1b).
- Se utiliza camelCase[1] para nombrar las variables.
- Las variables constantes están expresadas en mayúscula (2.1c).

A su vez, en las funciones más complejas utilizamos comentarios que indican su comportamiento, como los estándares especifican. También tratamos de tener la menor cantidad posible de atributos y funciones públicas.

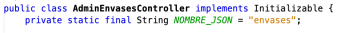
[1] `https://es.wikipedia.org/wiki/Camel_case`



(a) Clases en mayúscula.



(b) Paquetes comienzan en minúscula.

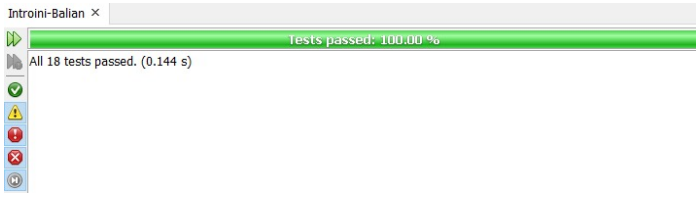


(c) Variables estáticas todas en mayúsculas.

Figura 2.1

2.2. Pruebas unitarias

Para la codificación de pruebas unitarias, el objetivo era probar el correcto funcionamiento de los modulo de código por separado. Los criterios fueron ejecutar estas pruebas sobre las clases del dominio mas influyentes en el sistema y, utilizando la librería JACOCOVERAGE, cubrir al 100 % las pruebas (2.2a).



(a) Test con JACOCOVERAGE de la clase Producto del dominio.

Figura 2.2

Exponer los criterios para la codificación de pruebas unitarias. Incluir aquí el análisis de cobertura de pruebas.

2.3. Análisis de código

Para analizar el código utilizamos FindBugs[2], un plugin para el IDE NetBeans que busca e identifica los bugs que puedan existir en el código de Java. Esta he-

[2] `http://plugins.netbeans.org/plugin/912/findbugs-tm-plugin`

rramienta fue sumamente útil para algunas cosas que se nos habían pasado por alto, como los nombres de variables, o algunos métodos que nunca eran llamados. De todos modos, muchos de los bugs que identificó, no teníamos otra solución que mantenerlos en nuestro código. (2.3a)

(a) Análisis con FindBugs.

Figura 2.3

3. Interfaz de usuario y usabilidad

3.1. Criterios de interfaz de usuario

Lo primero a destacar es que utilizamos en gran parte el lenguaje de diseño de Google, Material Design [1], con esta guía pudimos ver muchas formas de mejorar la usabilidad de la interfaz, como por ejemplo, la herramienta que brinda para seleccionar colores de una interfaz, en la cual nos indica, entre otras cosas, que color de fuente se puede utilizar sobre un fondo con otro color. Así incrementando la legibilidad de la interfaz. Para poder implementar éste lenguaje de diseño a nuestro proyecto, dado que los componentes visuales que trae por defecto JavaFX no se adecúan, decidimos incorporar la librería llamada JFoenix[2], la cual cuenta con una gran cantidad de controles y demás componentes que cuentan con el estilo que los estándares de Material Design plantean. La implementación de ésta librería no fue del todo sencilla, por lo que además tuvimos que crear nuestra propia hoja de estilos en cascada, o css que nos permitió crear nuevos y estilizados componentes visuales.

Tratamos de hacer una interfaz que se adecuara tanto al cliente como al empleado del EcoShop, para esto, dividimos en dos interfaces principales, una para cada tipo de usuario, con distintas formas de mostrar las cosas. Por ejemplo, para un empleado, utilizar la aplicación será algo diario, por lo que es necesario que sean controles sencillos y no tan estilizados, ya que prioriza la eficiencia y no la estética. En cambio para el usuario cliente, se creo una interfaz con más estética en cuanto a los controles.

Para incrementar la precisión en la interfaz, incorporamos tooltips y prompt texts que le indiquen al usuario información de un comando cuando éste no era del todo claro.
Para gestionar los errores cometidos por los usuarios, utilizamos las sugerencias de Material Design, en poner un label de error debajo del control donde se cometió la falta. Todo esto se puede visualizar en la Figura 3.1a

Por último, se creó un control que imita a los snackbars de Google, utilizado para transmitirle al usuario información de confirmación u error ante los eventos del sistema.

[1] `https://material.io/`
[2] `https://github.com/jfoenixadmin/JFoenix/`

(a) Labels de error y tooltips de la interfaz

Figura 3.1

3.2. Evaluación de usabilidad

Para evaluar la usabilidad, lo primero que fuimos haciendo fue en todo momento ir cargando datos y cosas que nos ayudaran visualmente con el sistema. Es decir, si es un sistema de ecoshop, poco ayudaría que se cargaran productos con nombres aleatorios o con fotos de otra área.

Luego de esto, lo primero que hicimos para medir la usabilidad, fue la prueba con usuarios. Para comenzar con ésta técnica, en las etapas más prematuras del desarrollo, fuimos nosotros quienes simulamos ser el usuario que se encargaba de manipular la información. Ya más avanzados y con un programa mas funcional, conseguimos usuarios a los cuales observar mientras utilizaban el software, estos usuarios fueron mayoritariamente compañeros y familiares. A los mismos se les sugirió que se pusieran en un rol. Tanto de empleado como de cliente, y que navegaran como si fueran ellos quienes administran la tienda o quieren comprar un producto. La práctica fue satisfactoria, ya que de allí pudimos darnos cuenta por ejemplo, que algunos controles no eran del todo informativos, ya que veíamos como la persona tardaba tiempo e incluso no sabía que hacer en dicha situación. También tratamos de medir tiempos en las tareas, pero estos no podían ser del todo precisos ya que, por ejemplo, al no trabajar con una base de datos grande no podemos saber los tiempo de demora en el manejo de datos. Eso si, a la hora de cargar algunos controles de las librerías, podía tardar un tiempo indeseado.

4. Pruebas funcionales

4.1. Técnicas de prueba aplicadas

La técnica de prueba que usamos fue la prueba de caja blanca, en la cual se analiza la estructura lógica interna de la aplicación. Mediante esta prueba, se pretende demostrar que los componentes internos de la aplicación se comportan adecuadamente.

Utilizamos este método de prueba tanto nosotros al usar el programa mientras lo desarrollábamos, como con las pruebas unitarias.

4.2. Casos de prueba

Partimos del siguiente caso de uso:

Descripción: Administrador desea agregar un punto de venta al sistema.

Actor: Administrador.

Precondiciones:

- Administrador ingresado en el sistema.

Curso normal:

Usuario	Sistema
1. Administrador selecciona 'Puntos de venta' en el menú.	
	2. Cambia el panel activo al panel 'Puntos de venta'
3. El usuario ingresa los datos ID, nombre, calle, numero, ciudad y departamento.	
4. El usuario presiona el boton Agregar punto de venta	
	5. El sistema busca a la dirección mediante el API de Google Maps y devuelve un marcador en la latitud y longitud encontrada.

Cursos alternativos:

- 3.1 En el caso que el administrador se olvide de ingresar cualquiera de esos campos, el sistema le va a informar que todos los campos son obligatorios.
- 3.2 Si la ID que ingreso el administrador ya esta en uso, saldrá un mensaje notificando al mismo de la situación, esperando por una ID que sea única.
- 5.1 Si no se encuentra la dirección, se muestra un mensaje en la pantalla informando que la dirección es errónea y dejando la dirección que agrego en pantalla para que el administrador la examine.

Poscondiciones:

- El punto de venta es guardado en los archivos JSON de serialización.

Escenario	Nombre	Curso de comienzo	Curso alternativo
1	Se crea el punto de venta.	Curso básico.	
2	Dirección no especificada.	Curso básico	CA 3.1
3	ID ya existente.	Curso básico	CA 3.2
4	Dirección no valida.	Curso básico	CA 5.1

Con estos escenarios podemos generar sus casos de prueba:

Caso de Prueba	Escenario	Dirección	ID	Resultado esperado
CP 1.1	Escenario 1	V	V	Se agrega el punto de venta
CP 1.2.1	Escenario 2.	ND	V	Dirección no especificada.
CP 1.2.2	Escenario 3	V	NV	ID no valida
CP 1.2.3	Escenario 4	NV	V	Direccion no valida

4.3. Sesiones de ejecución de pruebas

Realizamos una sesión de ejecución de prueba el lunes 25 de Noviembre a las 15:00 horas, probando la ultima versión del programa, utilizando la JDK $1.8.0_251. Lostesters fuimos Juan B$

5. Reporte de defectos

5.1. Definición de categorías de defectos

Las categorias que utilizamos para los defectos fueron principalemente las definidas en los labels de los issues en GitHub. Los labels que aparecen en inglés es porque tomamos la decisión de dejarlos de esta forma en la que github los provee (menos la de diseño que fue creada por nosotros), ya que es como mejor se adecuan al ambiente de desarrollo. Categorias:

- **bug:** algo que no está funcionando.
- **enhancement:** Una funcionalidad nueva que necesita el sistema
- **wontfix:** Un error que se llegó a la conclusión de no ser arreglado
- **diseño:** Problema en la visibilidad de la interfaz

5.2. Defectos encontrados por iteración

La principal herramienta que utilizamos para controlar los defectos encontrados por iteración fueron los issues en GitHub. Esto nos permitió abrir un defecto cuando era encontrado, y cerrar el mismo con un comentario indicando por que sucedía o como se resolvió. También nos permitió etiquetar a estos defectos con labels de forma de categorizarlos. Algunos de los errores cuentan con captura de pantalla de lo que sucedía. La solución y la fecha están dadas en cada issue y se referencia al commit que las arreglo.

5.3. Estado de calidad global

Para una implementación completa quedo sin estar completo: Realizar y consultar preventas, guardar el PDF con la información dispuesta de manera correcta y realizar descuentos a los productos por la interfaz.

6. Reflexión

Este obligatorio nos amplió fuertemente nuestros conocimientos en la herramienta git. Nuestra experiencia trabajando con la misma se remonta a proyectos que hicimos en el liceo, y nunca fue satisfactoria. No hasta ahora que podemos decir que hicimos un buen uso de git, quizás no completo, pero un buen uso en fin.
Por otro lado, nos llevó a investigar por completo a nivel de programación, ya que trabajamos con una tecnología que nunca habíamos trabajado, además de las librerías, que ninguna fue fácil de implementar. De todos modos, aprender a usarlas nos llevó a trabajar de una manera muy fluida en el diseño de interfaces.

Bibliografía

[1] DGI. (2019) Dgi. [Online]. Available: https://bit.ly/35mBDn3

[2] Oracle. (1997) Java Code Convention. [Online]. Available: https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

[3] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, eight ed. McGraw-Hill, 2014.

[4] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.

[5] Universidad ORT Uruguay. (2013) Documento 302 - Facultad de Ingeniería. [Online]. Available: http://www.ort.edu.uy/fi/pdf/documento302facultaddeingenieria.pdf