Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería de Software 1
Letra del Obligatorio 2

Martín Rzeszytkowski - 201239
Gonzalo Strauss - 213188

Entregado como requisito de la materia Ingeniería de Software 1

Version vieja:
https://github.com/gonchistrauss/IngSoft-App
Version final:
https://github.com/gonchistrauss/fooding2.0-strauss_rezy

Declaraciones de autoría

Nosotros, Gonzalo Strauss y Martin Rzeszytkowski , 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

En primer lugar, cabe destacar que para este obligatorio nos propusimos como meta poder implementar todas las técnicas de ingeniería de Software estudiadas en clase sumados a los conocimientos previos de programación. La aplicación realizada se basa en un sistema que tiene como objetivo orientar al usuario a alimentarse mejor basándose en su alimentación diaria. Para eso el usuario se registra y se le da la posibilidad de elegir sus preferencias y restricciones alimenticias como parte del registro. A su vez, los profesionales que asesorarán a los usuarios también son registrados en la aplicación. De esta manera ambos actores del sistema trabajarán en paralelo. El usuario tendrá la posibilidad de hacer una consulta directa cualquiera o bien solicitar un plan de alimentación a un profesional. Ademas de la implementacion de código, este obligatorio da evidencia de todo el proceso de la gestion de configuracion del software sea identificandolo, definiendo sus piezas, controlando sus modificaciones y versiones y registrando el estado y pruebas de las mismas.

Índice general

1. Rúbrica

En esta sección se presenta la rúbrica del obligatorio, en la que se enumeran los criterios de evaluación usados por los docentes. Además, ayuda a los estudiantes a entender lo que se les pide que realicen y a autoevaluar su trabajo.

Los aspectos evaluados se dividen en distintas categorías. En cada una de ellas se indica el puntaje máximo que se puede obtener.

- Redacción y forma: 3 puntos
- Versionado e instalación: 6 puntos
- Calidad de código: 3 puntos
- Pruebas unitarias: 3 puntos
- Implementación: 6 puntos
- Diseño de interfaz de usuario y usabilidad: 3 puntos
- Pruebas funcionales: 3 puntos
- Reporte de defectos: 3 puntos
- Defensa y reflexión (total de los puntos)

-

2. Plantilla

2.1. Versionado

2.1.1. Repositorio utilizado

El sistema de versionados es utilizado con la herramienta de Git, apoyándose en GitHub para la colaboración compartida y remota. El acceso y cambios dentro los repositorios locales y remotos fueron mediantes la consola de comandos (de macOS y Windows)y mediante la interfaz visual de Sourcetree.

Se incluyen los links a los repositorios utilizados para la configuración y distribución del software en este obligatorio. Vale aclarar que son dos los repositorios utilizados dados algunos inconvenientes técnicos, los cuales detallamos en el reporte de defectos mas adelante.

- El link al primer repositorio (repositorio obsoleto) utilizado es:
  https://github.com/gonchistrauss/IngSoft-App.
- El link al segundo repositorio (que contiene la versión estable) es:
  https://github.com/gonchistrauss/fooding2.0-straussrezy

Son varios los elementos que podemos identificar dentro de la configuración del software. Los mismos los podemos dividir en 3 secciones, donde se detallarán los más importantes.

- ***Programas:*** Dentro del repositorio se pueden encontrar elementos de los cuales hace uso el mismo programa. Por ejemplo, se encuentran el código fuente, recursos audiovisuales (las imágenes utilizadas por el sistema) y el ejecutable *".jar"*
- ***Documentos:*** Hace referencia a todos los documentos que puedan hacer uso técnico, administrativo o a nivel de usuario. Por ejemplo, se puede visualizar al documento **README.md**, el cual se usa para dar una síntesis del programa así como información del uso, distribución y desarrollo del mismo. También aquí se encuentran reportes, casos de prueba y otros archivos que evalúan la ingeniería de requerimientos, diseño, construcción y pruebas del proyecto.
- ***Estructura de datos:*** Como dice la palabra, son estructuras que contienen datos. Por ejemplo, se utiliza como forma de base de datos a la serialización de java (fooding.app en nuestro caso), que permite persistir los datos utilizados en cada tiempo de corrida del mismo.

2.1.2. Criterios de versionado

Los criterios de versionado de los ECS utilizados en nuestro proyecto se basan en Gitflow Workflow. El mismo es un diseño de trabajo que define un modelo de ramas sobre el proyecto, ayudando de gran manera a la colaboración y brindando un estándar sencillo y rápido de utilizar. El trabajo se organiza en 2 ramas principales:

- ***Rama máster:*** Aquí se encuentran las versiones completamente estables y funcionales del proyecto. En otras palabras, cualquier commit presente en esta rama debe estar preparado para ser subida a producción y no causar ningún tipo de riesgo al manejo del mismo. Cada vez que se incorpora código a máster, se realiza una nueva versión del proyecto.

- ***Rama develop:*** Tal como dice la palabra, esta es la rama de desarrollo. Es decir, aquí se trabaja en las versiones que potencialmente estén planificadas para ser parte del proyecto actual, pero que están continuamente actualizándose.

Además de las ramas master y develop, Gitflow Workflow propone tres ramas auxiliares que ayudan a la fluidez del proyecto. Las mismas son:

1. ***Feature:*** Esta rama se utiliza para desarrollar nuevas funcionalidades y características del sistema. Pueden haber cuantas ramas features se necesiten, y las mismas tienen su punto de partida desde develop. Una vez terminada de desarrollar, se incorporan nuevamente a develop.

2. ***Release:*** Tienen la función de preparar el código de salida a producción. Por lo tanto, las mismas se originan desde develop y se incorporan a máster (en ciertos casos también a develop.) En otras palabras, en estas ramas se pulen los últimos detalles y se corrigen los bugs.

3. ***Hotfix:*** Son ramas que deberían llegar a ser innecesarias, pero que siempre deben estar por si acaso. Las mismas cumplen la función de corregir bugs u otros errores en el código de producción, sin saber que ocurrían los mismos en su salida a producción (no planificados.)

El uso de los mecanismos de trabajo distribuido se vio acotado dada la cercanía física y coincidencias horarias entre los desarrolladores (nosotros los autores). De todas maneras, ya sean por separado en conjunto, el ciclo de trabajo aplicado siempre siguió una linea en común bajo la fluidez del Gitflow Workflow. Además, si bien se trataron de usar todas las funcionalidades que esta metodología de trabajo propone, algunas no fueron necesarias de ser utilizadas (por ej. las ramas hotfix no tuvieron uso y las rama release tuvo un único uso para la salida a la primera y ultima versión estable del proyecto).

2.1.3. Resumen del log de versiones

En primer lugar, detallaremos nuestra forma de trabajo a lo largo de todo el proceso de construcción, desarrollo y producción del proyecto.

Como ya se explicó de breve manera anteriormente, dada la poca distancia que hay entre nuestras casas y nuestra posibilidad de juntarnos permanentemente en ellas, no fue muy necesario el uso de la distribución para colaborar concurrentemente en el repositorio. Dejamos constancia que el trabajo se hizo en su mayoría desde una sola computadora (Mac) salvando excepciones. De todas formas el uso del versionado se ha utilizado siempre y de manera constante, cumpliendo con las técnicas habituales para con su uso y siguiendo la linea de trabajo de GitFlow Workflow.

Cabe aclarar que la mayoría de las referencias al versionado del proyecto son hacia la rama develop. ya que la rama master solo se utilizo para el commit inicial y para el único release final antes de la entrega.

1. ***Repositorio 1(obsoleto)***: El versionado del primer repositorio da muestra de los primeros pasos que tiene el proyecto hacia el dominio. Además de la identificación de los elementos de configuración como por ejemplo la serialización (estructura de datos de persistencia) y el README.md (documentación/guía del proyecto), se puede visualizar que las clases del dominio son creadas y van siguiendo una evolución constante a lo largo de la rama develop. También es posible distinguir a las primeras ventanas de interfaz, que logran comunicar al dominio con la interacción física/visual del usuario. Por otro lado, el repositorio contiene a las primeras versiones en paralelo visualizadas a través de sus ramas feature. Un caso practico puede ser el desarrollo de la selección de profesional y del registro de profesional como atributos por separado para luego en caso de éxito poder fusionarlas a una misma rama (merge). Acerca de la autoría de los commits en este repositorio, es posible dar muestra de lo explicado anteriormente; los commits raramente son intercalados de manera variada, sino que por el contrario, siguen una línea de autoría fija en serie que varía en pocos momentos. Por último, es posible detectar al final del logueo de este repositorio un caso particular. La última constancia de versionado data del 17/11. Esto se debe a que el proyecto quedó obsoleto dados problemas de detección y eliminación de las clases .form. Las mismas dejaron con problemas irreversibles al proyecto, lo que causó una inmediata creación de otro repositorio para continuar/rehacer el proyecto nuevamente.

2. **Repositorio 2 (actual y versión estable)**: Podemos decir que este repositorio es la inmediata continuación del repositorio 1. Los primeros logueos dejan constancia de la reutilización de código de las versiones anteriores del repositorio, para rearmar lo mas rápido y eficazmente posible al proyecto. Además, se hace uso del *.gitignore*, archivo de configuración de git que permite mantener que archivos ignorar y cuales no a la hora de pushear y commitear las versiones. El uso del mismo se debe como forma de prevención al problema ocurrido en el proyecto anterior, para evitar de cierta manera el uso delicado de las clases .form y otras de aspecto privado. Las versiones de este repositorio continúan evolucionando hasta sobrepasar el status global de proyecto anterior, dejándolo obsoleto de manera definitiva. Otro dato importante para mencionar es el cambio de autoría de commits. Aquí únicamente se hace uso de un único autor, ya que decidimos correcto mantenernos siempre juntos a la hora del desarrollo. Esto se debe a la cercanía con la fecha límite y a la

urgencia y delicadeza que llevaba el proyecto en estas instancias.

2.2. Codificación

2.2.1. Estándar de codificación

Los estándares de codificación son la base de la calidad del código. Facilitan el mantenimiento del mismo, y ayudan a trabajar en equipo.

A continuación insertamos una parte del código para poder analizar el mismo:

```
1  import java.util.*;
2  public class Usuario extends Persona implements Serializable
       {
3
4      //Atributos Usuario
5      private ArrayList<Consulta> consultas;
6      private Locale nacionalidad;
7      private ArrayList<Alimento> preferencias;
8      private ArrayList<Alimento> restricciones;
9      private HashMap<String, ArrayList<Alimento>>
           alimentosIngeridos;
10
11         public ArrayList<Alimento> getPreferencias() {
12         return preferencias;
13     }
14
15     public HashMap<String, ArrayList<Alimento>>
           getAlimentosIngs() {
16         return alimentosIngeridos;
17     }
18
19     public void agregarAlimentosIngeridos(ArrayList<Alimento>
           ingeridos, String fecha) {
20         alimentosIngeridos.put(fecha, ingeridos);
21     }
22
23     public ArrayList<Alimento> getRestricciones() {
24         return restricciones;
25     }
26
27     public void agregarRestriccion(ArrayList<Alimento>
           nuevosAlimentos) {
28         for (Alimento alimento : nuevosAlimentos) {
29             restricciones.add(alimento);
30         }
31     }
32
33     public void agregarPreferencia(ArrayList<Alimento>
           nuevosAlimentos) {
```

```
34          for (Alimento alimento : nuevosAlimentos) {
35              preferencias.add(alimento);
36          }
37      }
38
39      public Locale getNacionalidad() {
40          return nacionalidad;
41      }
42
43      public ArrayList<Consulta> getConsultas() {
44          return consultas;
45      }
46
47      public void setNacionalidad(String countryCode) {
48          this.nacionalidad = new Locale("", countryCode);
49      }
50
51      public Usuario(String nombre, String apellidos, String
            pais, Date nacimiento, String pathPerfil) {
52          super(nombre, apellidos, nacimiento, pathPerfil);
53          this.setNacionalidad(pais);
54          restricciones = new ArrayList<Alimento>();
55          preferencias = new ArrayList<Alimento>();
56          consultas = new ArrayList<Consulta>();
57          alimentosIngeridos = new HashMap<String, ArrayList<
                Alimento>>();
58      }
59
60      public Usuario() {
61          super("nombre", "apellido", new Date(), "pathPerfil")
                ;
62          this.setNacionalidad("nacionalidad");
63          restricciones = new ArrayList<Alimento>();
64          preferencias = new ArrayList<Alimento>();
65          consultas = new ArrayList<Consulta>();
66          alimentosIngeridos = new HashMap<String, ArrayList<
                Alimento>>();
67      }
68
69      public void agregarConsulta(Categoria categoria) {
70          Consulta nuevaConsulta = new Consulta(this, categoria
                );
71          nuevaConsulta.setId(this.getConsultas().size() + 1);
72          this.getConsultas().add(nuevaConsulta);
73      }
74
75      @Override
76      public boolean equals(Object obj) {
77          Usuario unU = (Usuario) obj;
```

```
78          return this.getNombre().equals(unU.getNombre())
79                  && this.getApellidos().equals(unU.
                       getApellidos())
80                  && this.getFechaDeNacimiento().equals(unU.
                       getFechaDeNacimiento());
81      }
82
83      //Metodo toString
84      @Override
85      public String toString() {
86          return this.getNombre() + " " + this.getApellidos();
87      }
88
89  }
```

Listing 2.1: Usuario.java

- ***Indentación:*** Este aspecto es muy importante para navegar en estructuras de control y jerarquías del código. Se puede observar que el nuestro está bien indentado lo que genera una organización importante en el formato para poder leerlo y entenderlo con claridad. Sumado a esto se utiliza un mismo número de espacios fijos en el código.
- ***Agrupamiento de líneas de código:*** Se dejan espacios libres cuando se cambia de método para que no quede código amontonado y pueda ser lo más prolijo posible.
- ***Comentarios:*** Son un elemento esencial para documentar el código. En el caso del ejemplo que se puede observar, se comenta antes de los atributos y del método ToString. Esos comentarios son para aclarar lo que viene a continuación que quizás no sea tan obvio como aclarar antes del método .ªgregarConsulta"que se está agregando una consulta. Ese sería un comentario trivial ya que el nombre del método ya está diciendo lo que se va a hacer. Ese tipo de comentarios son los que hay que obviar.
- ***Convenciones de nombres:*** Se utilizan en el código palabras simples. Los nombres de los métodos y clases son nemotécnicos. Por ejemplo, para esta clase, la misma lleva el nombre de Usuario siendo esta la que tiene los datos del mismo. De esta forma al leer el código uno puede asociar lo que está haciendo esa clase.
- ***Mantener el código simple:*** No se percibe código que no se utiliza. Todo lo que está escrito es utilizado. A su vez no hay funcionalidades agregadas que no estén requeridas en la letra.
- ***Estándares de codificación en Java:*** Se puede observar en el código adjuntado que las clases y las constantes literales comienzan en mayúscula mientras que los paquetes y métodos en minúscula. Por ejemplo la clase se llama Usuario, el paquete dominio y uno de los métodos lleva de nombre, por ejemplo, agregarConsulta.

2.2.2. Pruebas unitarias

Ante todo nos parece válido exponer algunos conceptos relacionados con las pruebas unitarias. Según Glen Myers es el proceso de ejecutar el software con el objetivo de encontrar defectos.Una prueba tiene éxito si descubre errores. Un buen caso de prueba es aquel con alta probabilidad de descubrir un error no encontrado hasta el momento. Un programa hace lo que tiene que hacer y a su vez un programa no hace lo que no tiene que hacer. Hay una frase que por más trabalenguas que pueda sonar es puramente cierta y hay que tenerla en cuenta a la hora de realizar las pruebas; "La ausencia de evidencias no es evidencia de ausencia". Si la prueba no descubre errores no se puede afirmar que no existan.

A continuación adjuntamos la imagen de una de las pruebas unitarias hecha con la herrmienta JUnit:

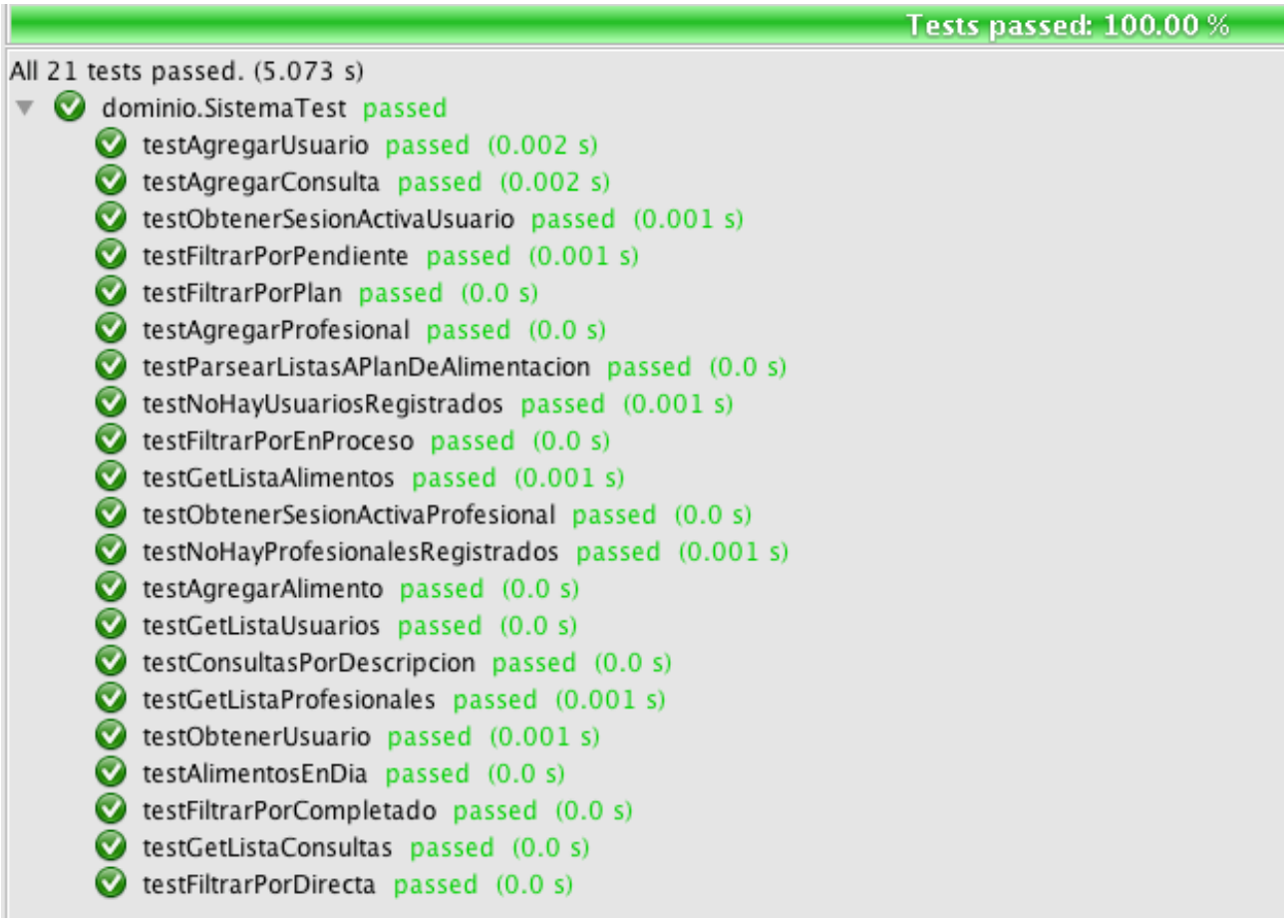

Figura 2.1: Junit.

2.2.3. Análisis de código

Son varias las herramientas utilizadas para la mantener un buen estándar de codificación a lo largo de todo el proyecto. A continuación se hace mención al uso de las mismas junto con una breve explicación del resultado obtenido por parte de ellas:

- ***Java Hints:*** Esta herramienta ya incorporada en la versión 8.1 de Netbeans permite visualizar de cierta manera si se están respetando los estándares de codificación definidos en Oracle(http://www.oracle.com/technetwork/articles/javase/codeconvtoc-136057.html). Aquellos estándares son una guía de lo que por convención se considera código de calidad o no al momento del desarrollo. El uso de la misma nos fue útil para por ejemplo quitar código declarado no utilizado, advertirnos sobre el uso de herramientas de Java en riesgo de quedar obsoletas, sugerencias acerca del uso de operadores para evitar la redundancia de código entre otras.

- ***FindBugs:*** Esta herramienta de código abierto permite el análisis en códigos estáticos para la detección de posibles bugs o problemas en el programa. Los errores son categorizados en cuatro que van variando según la severidad del potencial o actual error. El uso de esta misma herramienta fue un buen complemento con JavaHints, ya que si bien en muchos casos detectaba los mismos sucesos que ella, había otros mas difíciles de detectar que ayudaron al buen mantenimiento del código. Ademas, esta herramienta provee soluciones y ejemplos prácticos a los problemas que van surgiendo. Por ejemplo, el uso de la concatenación de un String dentro de un loop no es una buena práctica (debido al costo cuadrático que implica), y el inspector de FindBugs sugiere la utilización de StringBuilder así como un ejemplo práctico del uso del mismo.

- ***Convenciones de código por Oracle:*** Si bien no es una herramienta que se adhiere al propio ambiente de desarrollo, si es una buena guía para tener en cuenta a la hora de desarrollar. En nuestro caso particular, aprendimos sobre ciertos criterios que solíamos utilizar que no eran precisamente los mejores, no porque fueses a causar un riesgo sino que por simple convención.

Ademas del uso de las herramientas mencionadas arriba, el uso del sentido común y decisiones personales también fueron un factor importante. En tal caso, hay veces que nos vimos obligados a decidir a propia mano qué metodología era mejor, sin en realidad saber el riesgo de su aplicación. En su mayoría de veces esto ocurría para el uso de métodos poco triviales, auxiliares y excepcionales.

2.3. Interfaz de usuario y usabilidad

2.3.1. Criterios de interfaz de usuario

Fueron varios los criterios utilizados para la construcción de la interfaz de usuario basados generalmente por los dados en el curso.

- ***Estilos de interacción:*** Acerca de la implementación del estilo usado para la interacción entre usuarios y sistema, realizamos una mezcla entre varios. Por un lado, decidimos que el uso de **selección de menús** provee al usuario una forma sencilla, rápida y directa de acceder a los contenidos deseados sin pasos intermedios e innecesarios. Si bien se trata de lograr esta metodología, no siempre es posible. Para aquellos datos más flexibles y necesarios del usuario, decidimos necesario utilizar la **entrada de datos en formularios**, que

resultan ser una forma organizada (acota la entrada a lo que se requiere) y simplificada de obtener información necesaria.

- ***Tolerancia a errores:*** Dentro del programa se puede visualizar una exploración segura donde el usuario puede explorar sin miedo a hacer daños. Las funcionalidades son escasas y eso hace que no haya muchas opciones para equivocarse. En caso que se haya accedido a una ventana por error, siempre estará la opción de volver al menú principal para realizar la operación correcta. En este sentido el programa permite revertir las acciones de los usuarios. Por otro lado, para minimizar la cantidad de errores que el usuario podría cometer, bloqueamos los botones cuando no se pueden utilizar y se desbloquean solos cuando están habilitados para ser usados. Por ejemplo, cuando se va a escribir una respuesta directa si no se escribe nada el botón de envíar se encuentra bloqueado. O bien si un profesional quiere asignarse una consulta que ya está asignada a otro profesional, antes que cometa el error de hacerlo, directamente el botón de .ªsignar.ᵉˢtará bloqueado y no le dará la posibilidad de realizar la acción.

- ***Consistencia:*** Dentro de la aplicación podemos notar una terminología consistente. Por ejemplo, existe un botón para retornar al menú principal que no varía según la ventana que te encuentres. Otro ejemplo lo podemos notar en el registro de usuario y profesional, donde el botón para registrar al cliente tampoco varía. A lo largo del sistema se utilizan 3 formatos y 3 tamaños de letras diferentes lo que lo hace consistente también. De esta forma se genera un sentimiento de pertenencia de aquellos que hacen uso del sistema.

- ***Llamar la atención del usuario:*** Existen en el sistema varios aspectos que hacen que el usuario se sienta atrapado por la aplicación. Podemos afirmar que no es una interfaz monótona. Desde que entras se pueden ver recuadros resaltando las diferentes funcionalidades e imágenes acordes a la temática de la aplicación. Las ventanas tienen un formato prolijo y delicado que son placenteras a la vista. A su vez, el programa contiene muchas imágenes de la vida real que hacen más intuitivo el sistema siendo estos algunos de los aspectos que hacen llamar la atención del usuario.

2.3.2. Evaluación de usabilidad

La usabilidad es un atributo de calidad que establece cuán fácil de usar es un sistema. Para la evaluación de usabilidad de nuestro proyecto, tuvimos que enfrentarnos a la evaluación de tres aspectos clave:

1. ***Eficacia:*** La eficacia es sumamente importante a la hora de entender qué tanto impacto causa el sistema sobre el actor. Podemos afirmar que en nuestro sistema, el usuario logra el objetivo deseado. Existe la posibilidad de registrarse tanto como usuario y como profesional. Los usuarios usando el programa tienen la posibilidad de registrar los alimentos ingeridos durante el día y a su vez consultar directamente a los profesionales sobre las ventajas y desventajas

de lo ingerido. Además, para facilitar la dieta de los usuarios, mediante el programa también existe la elaboración de un plan de alimentación realizado por un profesional. En conclusión, el sistema cumple con todos los requisitos pedidos por lo tanto podemos decir que es eficaz.

2. ***Eficiencia:*** El esfuerzo que el usuario hace para llegar a los objetivos está sumamente al alcance. Esto se sustenta debido a varios ejemplos que podemos notar. Cuando se abre el sistema se puede ver como primer opción el registro. Una vez que se registró un usuario, se vuelve al menú principal y al clickear la ventana "Soy Usuarioüno puede acceder a todas las funcionalidades que el usuario puede hacer en la aplicación luego de seleccionar el usuario con el que desea trabajar. Lo mismo sucede con el profesional. A su vez, a la hora de elegir las consultas, podemos notar filtros para que los profesionales encuentren de forma rápida las consultas posibles para atender ya que algunas, por ejemplo, ya fueron atendidas o están en proceso de serlo.

3. ***Satisfacción:*** Si bien es subjetiva al usuario, podemos afirmar que el mismo queda muy conforme luego de usar la aplicación ya que cumple con todas las funcionalidades pedidas. Gracias a la aplicación el usuario puede mejorar notoriamente su calidad alimenticia. Al tener la posibilidad de registrar preferencias, restricciones y alimentos ingeridos durante el día, los planes se basan partiendo de la base de los gustos de cada usuario, lo que hace que al mismo le resulte más confortante y personal la devolución de los profesionales.

Además de estos tres aspectos podemos tener en cuenta a la hora de evaluar la usabilidad del programa los siguientes dos conceptos:

- Tasa de errores: Si bien el programa no presenta grandes .ºportunidades"de cometer errores, no deja de haber posibilidad de que pasen. Alguno de ellos pueden ser a la hora de querer responder una consulta, eliminarla sin querer, o bien cuando se ingresa una ingesta olvidarse de guardarla y cerrar la ventana perdiendo así la información ingresada.

- Retención o memorabilidad: Si los usuarios dejan de entrar a la aplicación por un determinado tiempo, cuando vuelvan a hacerlo no tendrán mayores problemas para reusarla ya que desde el momento que entras los botones y carteles te dicen paso a paso lo que hacer. Ya sea desde registrar el usuario o profesional, hasta responder las consultas.

2.4. Pruebas funcionales

2.4.1. Técnicas de prueba aplicadas

Para realizar las siguientes pruebas utilizamos una estrategia basada en escenarios para encontrar todos los escenarios del caso de uso. Luego generamos los casos de prueba para los escenarios encontrados y por último terminamos de generar los datos de prueba utilizando la técnica de particiones de equivalencia.Además usamos las pruebas exploratorias con algunos de los casos a modo de ejemplo.

2.4.2. Casos de prueba

Caso de uso: Respuesta del profesional
* [P] - Acción que realiza el profesional
* [S] - Acción que realiza el sistema

- ***1.Se selecciona una consulta Pendiente***-[P]
- ***1.1 Se selecciona asignar***-[P]
- ***1.1.2 Aparece el cartel .Está seguro que desea asignar esta tarea?***-[S]
- ***1.1.2.3 La consulta pasa a estar en proceso***-[S]
- ***1.2 Se selecciona eliminar***-[P]
- ***1.2.1 La consulta desaparece del inbox de consultas de ese profesional***-[P]
- ***2. Se selecciona la consulta en proceso***-[P]
- ***2.1 Se selecciona la opción de responder consulta directa***-[P]
- ***2.1.1 Aparece la pantalla para responder directamente***-[S]
- ***2.1.2 Se escribe un mensaje y luego se selecciona enviar***-[P]
- ***2.1.3 Aparece el mensaje en el historial de chats***-[S]
- ***2.2 Se selecciona la opción de responder un plan de alimentación***-[P]
- ***2.2.1 Aparece en pantalla la ventana para crear el plan de alimentación con los alimentos ya registrados***-[S]
- ***2.2.2 Se guarda el plan***-[P]
- ***2.3 Se selecciona la opción de finalizar***-[P]
- ***2.3.1 Aparece el cartel .Está seguro que desea finalizar esta tarea?Esta acción es irreversible***-[S]
- ***2.3.1.2 La consulta pasa a ser completada***-[S]
- ***3. Se selecciona la consulta completada***-[P]
- ***3.1 Se selecciona la opción de eliminar***-[P]
- ***3.1.2 La consulta desaparece del inbox de consultas de ese profesional***-[S]

  Casos alternativos que tendremos en cuenta a la hora de hacer los casos de pruebas:

- ***Acción de rechazar una consulta***

  1.1.2 si se selecciona "no"la consulta sigue estando pendiente

  2.3.1 Si se selecciona "no"la consulta sigue en proceso.

  3. Si la consulta esta completada no se puede asignar a ningun profesional.

- ***Consulta ya asignada***

  2. Si esta asignada por otro profesional no se le puede asignar al actual.

- ***No se escribe respuesta directa***

  2.1.2 Si no se escribe ningún mensaje no se permite enviar la consulta.

- ***Alimentos ingresados***

  2.2.1 Si no se guarda no se hace el plan.

  2.2.1 No se puede agregar mas de 8 comidas en un día.

  2.2.1 Si no se agregan alimentos no se puede obtener un plan.

Escenario	Estado consulta/botón	Curso de comienzo	Curso alternativo
Escenario 1	Respuesta normal	curso básico	
Escenario 2	Acción de consulta rechazada	curso básico	CA 1
Escenario 3	consulta ya asignada	curso básico	CA 2
Escenario 4	No se escribe respuesta directa	curso básico	CA 3
Escenario 5	Alimentos ingresados	curso básico	CA 4

Ahora generaremos los casos de prueba para los escenarios encontrados:

Caso de prueba	Escenario	Consulta rechazada	Consulta ya asignada
CP 1.1	Escenario 1	V	V
CP 1.2	Escenario 2	NV	V
CP 1.3	Escenario 3	NV	NV
CP 1.4	Escenario 4	NV	V
CP 1.5	Escenario 5	NV	V

Caso de prueba	Escenario	No se escribe respuesta directa	Alimentos ingresados
CP 1.1	Escenario 1	V	V
CP 1.2	Escenario 2	ND	ND
CP 1.3	Escenario 3	V	V
CP 1.4	Escenario 4	NV	V
CP 1.5	Escenario 5	V	NV

Ahora generaremos los datos de prueba utilizando la técnica de particiones de equivalencia:

Condicion	Clases validas	Clases invalidas
Consulta rechazada	C. pendiente asignada(1)	Vuelve a estado pendiente (7)
	C. en proceso finalizada(2)	Vuelve a estado en proceso(8)
	C, completada asignada(3)	No se asignar consulta(9)
Consulta ya asignada	Consulta eliminada(4)	No se permite asignar(10)
No hay respuesta directa	Enviar respuesta (5)	No se envia respuesta (11)
Alimentos ingresados	Menos de 8 alimentos(6)	No ingresa mas alimentos (12)
	Entre 1 y 8 alimentos (7)	Hay que registrar al menos 1 (13)
	Se guarda (14)	Se cierra sin guardar(15)

Caso de prueba	Escenario	Consulta rechazada	Consulta ya asignada
CP 1.1.1	Escenario 1	Se asigna la consulta	Si
CP 1.1.1	Escenario 1	Se asigna la consulta	Si
CP 1.2.1	Escenario 2	No se asigna la consulta	-
CP 1.2.2	Escenario 2	No se finaliza la consulta	Si
CP 1.2.3	Escenario 2	Asignar la consulta completada	Terminada
CP 1.3.1	Escenario 3	Asignar consulta en proceso por otro	Si
CP 1.4.1	Escenario 4	No	Si
CP 1.5.1	Escenario 5	No	Si
CP 1.5.2	Escenario 5	No	Si
CP 1.5.3	Escenario 5	No	Si

Caso de prueba	Escenario	No se escribe respuesta directa	Alimentos ingresados
CP 1.1.1	Escenario 1	Se escribe mensaje	-
CP 1.1.2	Escenario 1	-	10
CP 1.2.2	Escenario 2	-	-
CP 1.2.2	Escenario 2	-	-
CP 1.2.3	Escenario 2	-	-
CP 1.3.1	Escenario 3	-	-
CP 1.4.1	Escenario 4	No se escribe mensaje	-
CP 1.5.1	Escenario 5	12	
CP 1.5.2	Escenario 5	-	
CP 1.5.3	Escenario 5	-	

Caso de prueba	Escenario	Resultados esperados	Clases
CP 1.1.1	Escenario 1	Consulta directa respondida	1,5
CP 1.1.2	Escenario 1	Plan de alimentación dado	1,6,7
CP 1.2.1	Escenario 2	La consulta vuelve a estar pendiente	7
CP 1.2.2	Escenario 2	La consulta vuelve a estar en proceso	8
CP 1.2.3	Escenario 2	No se habilita la asignación de la consulta	10
CP 1.3.1	Escenario 3	Solo se habilita eliminar la consulta	4,10
CP 1.4.1	Escenario 4	No se habilita el botón para enviar respuesta	11
CP 1.5.1	Escenario 5	No se permite guardar el plan	12
CP 1.5.2	Escenario 5	No se permite guardar el plan	13
CP 1.5.2	Escenario 5	No se permite guardar el plan	15

2.4.3. Sesiones de ejecución de pruebas

A continuación haremos las pruebas exploratorias del registro de usuario y alimento.

Registro de usuario: (probado en la última versión, tester: Martín Rzeszytkowski, 19.40 pm, 23/11/2017)

Tabla 2.1: Registro de usuario

Nombre de la prueba	**Dato ingresado**	**Resultado esperado**	**Resultado obtenido**
Registar usuario sin datos	Click en botón registrar	Cartel: "No se han ingresado datos"	OK
Registrar usuario con numeros en los campos de letras	Nombre: 1234 Apellido: 5	Cartel en rojo: Ïngrese únicamente letras"	OK
Registrar usuario con letras en la fecha de nacimiento	Nacimiento: hola Click en botón registrar	Se guardan los datos sin fecha de nacimiento	OK
Registrar usuario con todos los datos	Nombre: Martín Apellido: Rzeszytkowski Nacimiento: 18/01/1997 País: Uruguay Click en botón registrar	Cartel: Üsuario registrado exitosamente"	OK

Registro de alimento(probado en la última versión, tester: Martín Rzeszytkowski, 19.45 pm, 23/11/2017)

Tabla 2.2: Registro de alimento

Nombre de la prueba	**Dato ingresado**	**Resultado esperado**	**Resultado obtenido**
No se registran alimentos ni categoría	-	No se habilita el botón de registrar alimento	OK
Registrar proporción inválida	Proporción: hola Se clickea el botón de suma	Cartel: "Proporción inválida"	OK
Registrar proporción inválida	Proporción: 200 Se clickea el botón de suma	Cartel:"Proporción inválida"	OK
Registrar alimento con todos los datos normales	Alimento: Hamburguesa Categoría:Comida rápida Nutriente: Carne picada Proporción: 50 Click en el símbolo de suma Click en registrar alimento	Cartel: .Alimento registrado exitosamente"	OK

2.5. Reporte de defectos

2.5.1. Definición de categorías de defectos

El reporte de defectos del proyecto puede dividirse en varias secciones de categorías, de las que se separan por severidad y por contexto:

- ***Severidad:***

    1. **Baja:** Son aquellos defectos que traen como resultado un inconveniente menor, sencillo de revertir y con mínimo impacto a la inter-operabilidad dentro de la plataforma. Es decir, son aquellos errores espontáneos causados generalmente por errores de sencillos de sintaxis y/o semántica dentro del código
    2. ***Media:*** Defectos que afectan de manera parcial ciertas acciones a realizar. Son corregibles aunque requieren de algo de tiempo y pienso para su solución.
    3. ***Alta:*** Defectos con baja probabilidad de ser solucionados. El incumplimiento trae consecuencias graves y generalmente afectan a toda la usabilidad y consistencia del proyecto. En caso de ser solucionados, el proyecto corre riesgo de quedar con secuelas del mal estado anterior.
    4. ***Extrema:*** Defectos insuperables. Son de carácter global y no existe solución al mismo. El proyecto queda totalmente comprometido y su estado es irreversible.

- ***Contexto:***

    1. ***Distribución/versionado:*** Se dan en el contexto del uso de Git, Github, SourceTree o alguna otra herramienta para la distribución y versionado del proyecto.
    2. ***Dominio/lógico:*** Son defectos que se dan en en el ámbito del dominio del proyecto. Tienen fallas en la lógica del programa.
    3. ***Interfaz visual:*** Aquellos defectos que afectan en la visibilidad del sistema. En su mayoría no afectan al dominio del mismo aunque puede dar a confusión, dependiendo de que tan bien se haga la separación dominio/interfaz.
    4. ***Global:*** Esta categoría comprende a aquellos defectos que bien podrían estar categorizados en varias o todas las categorías mencionadas anteriormente, debido a su grado global de impacto.

2.5.2. Defectos encontrados por iteración

A continuación se detalla el reporte de defectos del proyecto.

1.
   - ***Descripcion:*** Falla en la configuración del arhivo .gitignore que no permitia ignorar archivos al momento de hacer commits en el repositorio. El mismo causó problemas en la alteración y distribución de las propiedades del proyecto.
   - ***Categoria:*** Media
   - ***Contexto:*** Distribucion/versionado
   - ***Solucion:*** Se descarga template con un archivo .gitignore para proyectos Java. Se sobrescribe al archivo anterior mediante una serie de comandos git y se realiza commit para habilitar al mismo en el proyecto.

2.
   - ***Descripcion:*** Alteración y eliminación de los archivos .form de las clases JFrame del proyecto. Como consecuencia, no se reconoce al codigo autogenerado por Netbeans como mismo, dejando inhabilitada al diseño visual del JFrame.
   - ***Categoria:*** Extrema
   - ***Contexto:*** Global
   - ***Solucion:*** No existe solución encontrada. La unica manera de eliminar el defecto es borrando también al proyecto y empezar de nuevo. El dominio no se ve afectado por lo que la solución es la realización de una nueva interfaz desde cero.

3.
   - ***Descripcion:*** Inconsistencias en la distribución del proyecto. A veces Netbeans no lo reconoce como proyecto sin ningun motivo aparente. La carpeta build no siempre contiene a los archivos .class.
   - ***Categoria:*** Media
   - ***Contexto:*** Distribución/versionado
   - ***Solucion:*** Tener como back-up al proyecto guardado en la nube e Google Drive y Dropbox, para que en casos de necesaria distribución sea posible su alcance sin problemas.

4.
   - ***Descripción:*** La liberería externa JCalendar es reconocida por Netbeans pero la interfaz presenta problemas al reconocer sus posibles herramientas. Si bien JCalendar está contenido dentro del proyecto, la interfaz no lo reconoce como tal al agregarse un elemento.
   - ***Categoría:*** Media
   - ***Contexto:*** Interfaz
   - ***Solución:*** Se elimina del proyecto la librería, y se baja el mismo desde otra fuente (versión 1.3.1). Luego de un reinicio del IDE se reconoce a la librería para su uso.

5.
   - ***Descripción:*** Al correr el ejecutable del programa ciertos datos no presentan persistencia ya que algunas clases del dominio no extendían de la interfaz de serialización.
   - ***Categoría:*** Media

- ***Contexto:*** Dominio/lógico
- ***Solución:*** Se serializa a todas las clases del dominio y los datos persisten en su totalidad.

6.
- ***Descripción:*** Al eliminar una consulta del inbox de un profesional, se eliminaba la misma en la de todos los demás. Esto se debe a que la consulta era exactamente la misma en la de todos los demás (mismo objeto,misma dirección de memoria asignada).
- ***Categoría:*** Baja
- ***Contexto:*** Dominio/lógico
- ***Solución:*** Se usa la función clone() de java para la shadow copy de la lista.

2.5.3. Estado de calidad global

Se puede decir que el estado de calidad global de la entrega es el esperado. Creemos que el sistema responde a todo lo solicitado. Estamos conformes con la interfaz ya que la vemos muy profesional y muy accesible para cualquier usuario. Sin embargo el nivel de cobertura de pruebas está muy cerca de llegar al 100 porciento pero no llega en su totalidad. Hay algunos detalles pequeños que fallaron. De todos modos el 92 porciento refleja un muy buen trabajo en el cual creemos que pudimos aplicar de manera correcta los conceptos y recursos adquiridos en cada clase.

Figura 2.2: Estado de calidad global.

2.6. Reflexión

Luego de un exhausto y largo proceso, sobre el cual pudimos aplicar todos los conceptos aprendidos del curso y otros sobre la marcha, entendemos la verdadera importancia que tiene la ingeniería del software a la hora de del comienzo, desarrollo y post-lanzamiento de un proyecto de este carácter.

A continuación, se hace un análisis de las principales conclusiones que sacamos, nosotros los autores, luego de este arduo proceso.

- En primer lugar, entendemos la importancia que tiene la investigación y análisis previo al desarrollo, ya que es una pieza fundamental a la hora de lograr un sistema consistente y mantenible. Calcular bien los plazos de tiempo es una tarea muy difícil para un desarrollador, pero se hace imposible si no se aplica antes la ingeniería de software. Nosotros lo vivimos en propia carne, debido a que por diferentes percances nos vimos en apuros para poder presentar en

tiempo y forma un proyecto de calidad, que creemos que, de todas formas, pudimos lograr.

- En segundo lugar, aprendimos que no solo un buen sistema pasa por su comportamiento lógico, sino que también por su interacción con el usuario. De nada sirve tener un sistema completo por detrás cuando el usuario no puede aprovechar al máximo su contenido. La usabilidad del usuario y la interfaz no pueden ni deben pasar por desapercibido, ya que a fin de cuentas, es la muestra visual de todo lo realizado y lo que verdaderamente impacta al usuario.

- Muy pocos proyectos de esta índole pueden ser llevados a cabo de manera individual y son pocos los que se animan a hacerlo. Aprender a trabajar en equipo tiene sus complicaciones, pero hoy en día es casi obligatorio. Por eso mismo, el uso de sistemas de versionado y distribución como Git y Github nos hizo darnos cuenta lo sencillo que puede ser trabajar de manera conjunta sin dejar secuelas en el código, teniendo el control de cambios en tiempo real. Ademas, el uso de ramificaciones nos da la posibilidad de mantener en paralelo diferentes versiones y posibilidades que luego pueden ser puestas en conjunto, o si no tienen éxito, ser descartadas.

- Llevar un código de calidad puede parecer fácil e intuitivo, pero la realidad es que no siempre ocurre así, y más cuando son miles de líneas escritas por cientos de desarrolladores de manera concurrente. Por eso mismo, aprender el uso de herramientas de análisis de calidad como lo son JavaHints y FindBugs hacen que el código presente un nivel de reusabilidad y presentación acorde para con las convenciones que existen hoy en día.

- El uso de testing es fundamental para detectar no solo presentes errores, sino a aquellos que tienen el potencial para convertirse en uno. Sin profundizar al código por diferentes pruebas unitarias, exploratorias y funcionales, no hay seguridad de que el código no vaya a fallar en un futuro. Herramientas como JUnit y Jacocoverage nos permitieron visualizar el alcance de cobertura de las pruebas y el éxito de las mismas.

- No se espera que el código sea perfecto de principio a fin, ni tampoco creemos que exista tal premisa. Pero llevar a cabo un constante reporte de defectos si permite prevenir o mismo solucionar otros errores propios o ajenos. También, la realización de casos de prueba permite formalizar a los casos de uso y facilita un posterior uso de testing en los mismos.

Por último, y de manera personal, creemos que el proceso por el cual pasamos nosotros los autores fue arduo y extenso, presentando una infinidad de inconvenientes y problemas. Pero intentar superar cada uno de esos desafíos creemos que nos hizo crecer como profesionales y estudiantes. Tal como dice Doug Linder, ün buen programador es aquel que mira a ambos lados antes de cruzar en una calle de una sola vía". La necesidad de ser cauteloso y organizado ya no son mas una necesidad, sino una obligación para poder ser competencia en el mundo actual.

Bibliografía

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

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

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