

Universidad ORT Uruguay

Facultad de Ingeniería

# **Ingeniería de Software 2**

## **Obligatorio 1**

Paula Areco - 234219

Agustín Hernandorena - 233361

Entregado como requisito de la materia Ingeniería de  
Software 2

11 de mayo de 2020

# Declaraciones de autoría

Nosotros, Paula Areco 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 trabajo realizado es mantener una aplicación realizada en lenguaje Java, siguiendo un conjunto de buenas prácticas, como respetar las reglas de Java Code Conventions, aumentar las pruebas unitarias y mejorar la cobertura del código, así como también implementar nuevas funcionalidades. La aplicación a mantener es Alimentación Saludable, que promueve la comunicación entre los pacientes con los profesionales, almacenando un historial de alimentos consumidos y en base a ellos se hace la realización de planes de alimentación.

# Índice general

<b>1. Estado de calidad inicial del software</b>	<b>3</b>
1.1. Cobertura de las pruebas unitarias . . . . .	3
1.1.1. Caja blanca . . . . .	3
1.1.2. Caja negra . . . . .	5
1.2. Calidad de código y cumplimiento de estándares . . . . .	11
1.2.1. Análisis dinámico . . . . .	11
1.2.2. Análisis estático . . . . .	13
1.3. Calidad de los casos de prueba . . . . .	29
<b>2. Solicitud de cambios</b>	<b>33</b>
2.1. Formulario de solicitud de cambios . . . . .	33
2.1.1. Mala visibilidad y selección del JCalendar en las opciones de registro. . . . .	33
2.1.2. Falta de validación de la fecha de nacimiento o fecha de graduación dentro del JCalendar . . . . .	34
2.1.3. Ingreso de campo de fecha de una determinada ingesta. . . . .	34
2.1.4. Solo es posible subir al sistema imágenes con extensión png . . . . .	35
2.1.5. Menú principal al ingresar al sistema como Profesional o Usuario . . . . .	35
2.1.6. Listas mostradas en pantalla en el inicio de sesión en caso de no haber registrado usuarios o profesionales. . . . .	36
2.1.7. Opción de salir de la aplicación desde todas las ventanas. . . . .	37
2.2. Proceso de gestión de solicitudes de los cambios . . . . .	37
2.3. Análisis de impacto . . . . .	38
2.4. Repositorio . . . . .	45
<b>3. Plan de proyecto</b>	<b>46</b>
3.1. Identificación de interesados . . . . .	46
3.2. Objetivos SMART . . . . .	48
3.2.1. Nivel de cobertura de pruebas . . . . .	48
3.2.2. Nivel de pruebas aprobadas . . . . .	48
3.2.3. Nivel de confiabilidad en el código . . . . .	48
3.2.4. Complejidad ciclomática promedio . . . . .	49
3.2.5. Cantidad de clics al registrar un profesional. . . . .	49
3.2.6. Cantidad de clics al salir de la aplicación . . . . .	49
3.3. Alcance del producto y del proyecto . . . . .	49
3.3.1. Requisitos del proyecto . . . . .	49

3.3.2.	Requisitos del producto . . . . .	51
3.3.3.	Matriz de trazabilidad de requisitos . . . . .	53
3.3.4.	Enunciado del Alcance . . . . .	54
3.3.5.	EDT . . . . .	56
3.3.6.	Diccionarios de la EDT . . . . .	57
3.4.	Estimación del esfuerzo . . . . .	60
3.5.	Especificación del ciclo de vida . . . . .	61
3.6.	Cronograma de trabajo . . . . .	62
3.6.1.	Lista de tareas . . . . .	62
3.6.2.	Gráfico Ganttter . . . . .	65
3.6.3.	Relación entre el cronograma y el ciclo de vida . . . . .	66
3.7.	Estimación de costos . . . . .	66
3.8.	Plan de calidad . . . . .	67
3.9.	Plan de métricas . . . . .	70
	Bibliografía . . . . .	73
<b>4.</b>	<b>Anexo</b>	<b>74</b>
4.0.1.	JaCoCoverage . . . . .	74
4.0.2.	SonarQube . . . . .	74
4.0.3.	Funcionamiento de FindBugs . . . . .	76
4.0.4.	ISO 9126 . . . . .	77
4.0.5.	GitFlow . . . . .	79
4.0.6.	Objetivos Smart . . . . .	80

# 1. Estado de calidad inicial del software

## 1.1. Cobertura de las pruebas unitarias

### 1.1.1. Caja blanca

Las pruebas de unidad son el proceso de probar componentes del programa, como métodos o clases de objetos. Las funciones o los métodos individuales son el tipo más simple de componente. [1]

Cuando diseñamos los casos de prueba debemos brindar cobertura a todas las características del objeto. Esto significa que debe:

- Probar todas las operaciones asociadas con el objeto.
- Establecer y verificar el valor de todos los atributos relacionados con el objeto.
- Poner el objeto en todos los estados posibles. Esto quiere decir que tiene que simular todos los eventos que causen un cambio de estado.

Para realizar el análisis de cobertura de pruebas, utilizamos el *plugin* de *NetBeans TikiOne JaCoCoverage*.

Por información sobre cómo funciona y por qué lo escogimos, dirigirse al anexo **sección 4.0.1- JaCoCoverage**

En la figura 1.1, se muestran los datos arrojados por *TikiOne JaCoCoverage*:

Se observa que el porcentaje total de sentencias ejecutadas en las pruebas es del 61 %, mientras que el de ramas ejecutadas es del 57 %.

Consideramos que son porcentajes relativamente bajos al tratarse de un sistema que no tiene un alcance muy grande. El valor 57 % nos indica que solo se esta ejecutando poco mas de la mitad de los casos posibles cuando tenemos sentencias del tipo: *if*, *switch*, o iteradores.

Algunos ejemplos que reflejan lo mencionado anteriormente:

En la figura 1.2, se observa un diamante color verde, debido a que se han contemplado todos los casos posibles en esa sentencia.

En la figura 1.3, se observa un diamante color amarillo, porque solo se probó uno

## dominio

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">Sistema</a>	<div><div></div></div>	58%	<div><div></div></div>	51%	62	121	108	273	11	42	0	1
<a href="#">Sistema.Paises</a>	<div><div></div></div>	0%	<div><div></div></div>	n/a	4	4	4	4	4	4	1	1
<a href="#">Usuario</a>	<div><div></div></div>	43%	<div><div></div></div>	25%	12	21	20	45	6	15	0	1
<a href="#">Conversacion</a>	<div><div></div></div>	53%	<div><div></div></div>	50%	9	21	14	39	4	13	0	1
<a href="#">Persona</a>	<div><div></div></div>	87%	<div><div></div></div>	78%	4	21	3	30	1	12	0	1
<a href="#">InformacionMensaje</a>	<div><div></div></div>	81%	<div><div></div></div>	n/a	1	9	4	19	1	9	0	1
<a href="#">Sistema.DiasDeLaSemana</a>	<div><div></div></div>	90%	<div><div></div></div>	n/a	2	4	0	2	2	4	0	1
<a href="#">Sistema.Preferencias</a>	<div><div></div></div>	87%	<div><div></div></div>	n/a	2	4	0	2	2	4	0	1
<a href="#">Sistema.Restricciones</a>	<div><div></div></div>	84%	<div><div></div></div>	n/a	2	4	0	2	2	4	0	1
<a href="#">Sistema.IngestasPorDia</a>	<div><div></div></div>	81%	<div><div></div></div>	n/a	2	4	0	2	2	4	0	1
<a href="#">Alimento</a>	<div><div></div></div>	93%	<div><div></div></div>	90%	2	16	2	29	1	11	0	1
<a href="#">Ingesta</a>	<div><div></div></div>	98%	<div><div></div></div>	83%	2	13	1	21	0	7	0	1
<a href="#">PlanAlimentacion</a>	<div><div></div></div>	100%	<div><div></div></div>	75%	2	17	0	29	0	13	0	1
<a href="#">Profesional</a>	<div><div></div></div>	100%	<div><div></div></div>	75%	2	12	0	23	0	8	0	1
<a href="#">ComposicionAlimento</a>	<div><div></div></div>	100%	<div><div></div></div>	100%	0	6	0	12	0	5	0	1
Total	1.071 of 2.720	61%	105 of 244	57%	108	277	156	532	36	155	1	15

Figura 1.1: Resumen de cobertura de las clases del dominio.

de los dos casos posibles.

En la figura, 1.4, se observa un diamante rojo, producto de que en las pruebas realizadas, no se probó ningún caso de la sentencia.

```

♦ if (actual.getProfesional().equals(profesional) && actual.getUsuario().equals(usuario)
♦ && actual.getFueAtendidoElPlan() == false) {

```

Figura 1.2: Sentencia *if* en donde se probaron todos los casos posibles.

```

♦ if (listaIngestasDelUsuario.get(i).getFechaDeIngesta().equals(fechaIngesta)) {
    ArrayList<Alimento> listaAlimentosActual = listaIngestasDelUsuario.get(i).get
    Alimento alimentoAAgregar = devolverAlimentoDadoNombre(nuevoAlimento);

```

Figura 1.3: Sentencia *if* en donde se probaron parte de los casos posibles.

```

♦ if (!nombresIngresados.contains(nombreCompleto)) {
♦ if (nombreUsuarioCompleto.equals(nombreUsuarioConversacion)) {

```

Figura 1.4: Sentencia *if* en donde no se probó ningún caso.

En cuanto a la cobertura de sentencias totales es del 61 %, un valor significativamente bajo, que en cierto modo esta relacionado al 57 % de cobertura mencionado anteriormente.

Estos porcentajes se ven afectados principalmente por la clase *Países*, cuyo porcentaje de cobertura es del 0 %, este valor es totalmente inadmisibles para un sistema que se encuentra en producción, ya que estamos entregando funcionalidades a usuarios que no fueron sometidas ni a un solo caso de prueba.

Es importante destacar también el porcentaje de pruebas superadas, estas son, las que salida coincide con el resultado esperado. Analizando el valor que se observa en la figura 1.5 (76.98 %) consideramos que es significativamente bajo para el alcance

que tiene el sistema.

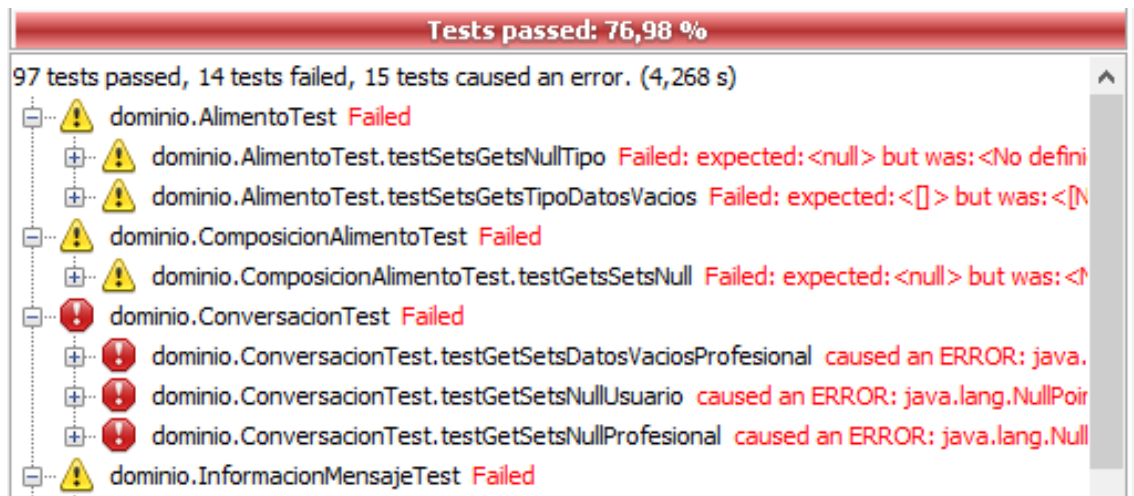


Figura 1.5: Porcentaje de pruebas pasadas.

### 1.1.2. Caja negra

Las pruebas de caja negra se basan en probar los módulos del software en función de su entrada y salida, en ella no se conoce la estructura interna del programa.

El mecanismo que vamos a utilizar nosotros para determinar la cobertura de las pruebas de caja negra es, mediante la técnica de partición equivalente diseñar los casos de prueba para cada funcionalidad, y corroborar si fueron contemplados por los desarrolladores en el documento ESRE.

En esta técnica, se toma cada condición de entrada o salida y se la divide en 1 o más grupos. Se identifican dos tipos de clase de equivalencias: válidas e inválidas.

A continuación, diseñamos casos de prueba un poco menos exhaustivos que si tuviésemos que probar la aplicación desde cero nosotros, con el fin de comparar con los casos de prueba del documento y verificar si tienen cobertura o no. En el caso de prueba, si tiene cobertura lo marcamos con un "Si", y si no la tiene con un "No". Al final vamos a hacer un porcentaje en base a cuantos fueron cubiertos y cuántos no, para tener una idea de la cobertura total con pruebas de caja negra.

#### ■ Prueba 1: Registrar usuario o profesional

##### Entrada:

##### **Clases válidas:**

- Nombre de 1 o más caracteres (letras)
- Apellido de 1 o más caracteres (letras)

##### **Clases inválidas:**

- Algún campo vacío (nombre o apellido)
- Nombre y/o apellido con caracteres numéricos o especiales.



- Elementos no seleccionados ( nacionalidad, preferencias o restricciones)
- Fecha de nacimiento sin seleccionar.

**CP1:**

**Entrada:** Nombre y apellidos correctos y todos los datos seleccionados.

**Resultado:** Registro correcto

**Cobertura:** Sí

**CP2:**

**Entrada:** Todos los campos vacíos

**Resultado:** Aviso de que no pueden haber campos vacíos y elementos sin seleccionar.

**Cobertura:** Si

**CP3:**

**Entrada:** Nombre y apellido numérico

**Resultado:** Se esperaba error pero registra el usuario igual.

**Cobertura:** No

**CP4:**

**Entrada:** Fecha de nacimiento vacía (eliminando el valor predeterminado) y el resto de los campos correctos.

**Resultado:** Registra usuario

**Cobertura:** Si

■ **Prueba 2: Consulta con un profesional**

Pre: ingreso como un usuario de la lista de registrados.

**Entrada:**

**Clases válidas:**

- Elijo un profesional y le envío un mensaje de 1 o más caracteres

**Clases inválidas:**

- Mensaje vacío

**CP1:**

**Entrada:** Mensaje vacío

**Resultado:** El sistema envía un mensaje vacío. Se esperaba que de un aviso de que el mensaje está vacío.

**Cobertura:** Si (se descubre en base a otro caso de prueba)

**CP2:**

**Entrada:** Mensaje con formato correcto

**Resultado:** El sistema envía el mensaje.

**Cobertura:** Si

■ **Prueba 3: Plan de alimentación**

Pre: ingreso como un usuario de la lista de registrados.

**Entrada:**

**Clases válidas:**

-En solicitar plan elijo un profesional de la lista y envío.

-La lista de planes disponibles está vacía y muestra un mensaje de que no hay planes para mostrar.

**Clases inválidas:**

-En solicitar no elijo ningún profesional de la lista

**CP1:**

**Entrada:** Envío eligiendo un profesional de la lista.

**Resultado:** El sistema envía la solicitud del plan.

**Cobertura:** Si

**CP2:**

**Entrada:** Envío sin elegir un profesional de la lista.

**Resultado:** Nos da un mensaje de solicitud incorrecta.

**Cobertura:** No

**CP3:**

**Entrada:** Lista vacía e intento enviar

**Resultado:** No me deja enviar.

**Cobertura:** No (las funcionalidades de la solicitud de planes siempre se prueban desde el lado del profesional.)

■ **Prueba 4: Ingreso de alimento ingerido**

Pre: ingreso como un usuario de la lista de registrados.

**Entrada:**

**Clases válidas:**

-Selecciono un alimento de la lista y una fecha de ingesta y envío.

**Clases inválidas:**

-No selecciono alimento pero si fecha, o viceversa, y envío.

**CP1:**

**Entrada:** Selecciono alimento y fecha correctamente.

**Resultado:** El sistema lo registra correctamente.

**Cobertura:** No (se plantea pero no se prueba)

**CP2:**

**Entrada:** No selecciono alimento pero si fecha

**Resultado:** El sistema muestra un mensaje de que no hay alimentos registra-

dos.

**Cobertura:** No

**CP2:**

**Entrada:** No selecciono fecha pero si alimento

**Resultado:** Igual se registra la ingesta de un alimento.

**Cobertura:** No (ídem de que se plantea pero no se prueba)

## **Rol Profesional**

### ■ **Prueba 4: Consultas pendientes**

Pre: Hay al menos un profesional registrado en el sistema.

**Entrada:**

**Clases válidas:**

-Elijo un profesional e ingreso a consultas pendientes.

-Hay al menos una consulta pendiente.

-No hay consultas pendientes.

**Clases inválidas:**

**CP1:**

**Entrada:** Ingreso a la sección consultas pendientes, y hay al menos una consulta.

**Resultado:** El sistema muestra el mensaje enviado por el usuario.

**Cobertura:** Si

**CP2:**

**Entrada:** Ingreso a la sección consultas pendientes, cuando todavía ningún usuario realizo una consulta al profesional

**Resultado:** El sistema informa que no hay consultas pendientes.

**Cobertura:** Si

### ■ **Prueba 5: Consultas pendientes - Responder consulta**

Pre: Hay al menos una consulta

**Entrada:**

**Clases válidas:**

-Elijo un profesional e ingreso a consultas pendientes.

-Respondo con un mensaje no vacío.

**Clases inválidas:** Respondo con un mensaje vacío.

**CP1:**

**Entrada:** Ingreso a la sección consultas pendientes, y respondo al usuario con un mensaje no vacío.

**Resultado:** El sistema envía el mensaje al usuario.

**Cobertura:** Si

**CP2:**

**Entrada:**

Ingreso a la sección consultas pendientes, y respondo al usuario con un mensaje no vacío.

**Resultado:** El sistema envía el mensaje.

**Cobertura:** No

#### ■ Prueba 6: Ver planes de alimentación solicitados

**Entrada:**

**Clases válidas:**

-Elijo un profesional e ingreso a planes de alimentación solicitados.

-Hay al menos un plan de alimentación solicitado por un usuario.

**Clases inválidas:**

-No hay planes de alimentación solicitados.

-No hay alimentados registrados en el sistema por parte de algún profesional.

**CP1:**

**Entrada:** Ingreso a la sección planes de alimentación solicitados, y no hay planes de alimentación pendientes.

**Resultado:** El sistema informa que no hay planes de alimentación pendientes.

**Cobertura:** Si

**CP2:**

**Entrada:**

Ingreso a la sección planes de alimentación solicitados, y no hay alimentos registrados.

**Resultado:** El sistema informa que no hay alimentados registrados.

**Cobertura:** Si

**CP3:**

**Entrada:**

Ingreso a la sección planes de alimentación solicitados, hay alimentos registrados y hay un plan solicitado. Selecciono al usuario que solicito el plan, y deajo el campo nombre de plan vacío.

**Resultado:** El sistema informa que debe completar el campo nombre de plan.

**Cobertura:** Si

**CP3.1:**

**Entrada:**

Ingreso a la sección planes de alimentación solicitados, hay alimentos registrados y hay un plan solicitado. Selecciono al usuario que solicito el plan, ingreso el nombre del plan, pero no agrego ningún alimento.

**Resultado:** El sistema envía el plan aunque no haya alimentos incluidos. Debería haber informado que se debe agregar al menos un alimento a la dieta.

**Cobertura:** No

**CP3.2:**

**Entrada:**

Ingreso a la sección planes de alimentación solicitados, hay alimentos registrados y hay un plan solicitado. Seleccione al usuario que solicite el plan, ingreso el nombre del plan, y agregue algún alimento a la dieta.

**Resultado:** El sistema envía el plan.

**Cobertura:** Si

■ **Prueba 7: Ingresar alimento al sistema**

**Entrada:**

**Clases válidas:**

- Elijo un profesional e ingreso a la sección ingresar alimento.
- Campo nombre no numérico ni vacío.
- Tipo de alimento no vacío.

**Clases inválidas:**

- Campo nombre numérico.
- Campo nombre vacío o solo compuesto por espacios en blanco.

**CP1:**

**Entrada:** Ingreso a la sección cargar alimentos, coloco en el campo nombre un *String* no numérico ni vacío, selecciono un tipo de alimento y doy en enviar.

**Resultado:** El sistema registra el alimento con éxito.

**Cobertura:** Si

**CP2:**

**Entrada:** Ingreso a la sección cargar alimentos, coloco en el campo nombre un *String* numérico, selecciono un tipo de alimento y doy en enviar.

**Resultado:** El sistema registra el alimento con éxito. Debería indicar que el nombre no puede ser un carácter numérico.

**Cobertura:** No

**CP3:**

**Entrada:** Ingreso a la sección cargar alimentos, coloco en el campo nombre un *String* vacío, selecciono un tipo de alimento y doy en enviar.

**Resultado:** El sistema indica que se debe ingresar un nombre.

**Cobertura:** Si

**CP4:**

**Entrada:** Ingreso a la sección cargar alimentos, coloco en el campo nombre un *String* solo compuesto por espacios en blanco, selecciono un tipo de alimento y doy en enviar.

**Resultado:** El sistema registra el alimento con éxito. Debería informar el

campo es vacío.

**Cobertura:** No

Diseñemos un total de 25 escenarios de pruebas, de los cuales 15 están contemplados por los casos de prueba realizados por los desarrolladores y 10 no. Si lo llevamos a porcentaje, los casos de prueba desarrollados cubren un 60 % del total de escenarios posibles.

## 1.2. Calidad de código y cumplimiento de estándares

Para interpretar la calidad del código nos basamos en *Java Code Conventions*, y en *Java Code Inspection Checklist*[4].

Estas convenciones mejoran la legibilidad del software, lo que ayuda al mantenimiento del código a largo plazo, ya que siguiendo estas convenciones, obtenemos código que es fácilmente entendible para un desarrollador que no está familiarizado con el sistema, como puede ocurrir en la etapa de mantenimiento del software.

Estas convenciones comprenden entre otras cosas: gestión de archivos, sangría, comentarios, declaraciones, sentencias y convenciones de nombres.

En particular, nos basamos en las convenciones de Java porque es el lenguaje que se utilizó para el desarrollo de la aplicación a analizar.

Dentro de todas las convenciones que se mencionan en el *Java Code Conventions* nos centramos en algunas para determinar la calidad actual del código, entre ellas se encuentran:

- **Longitud de código:** Ésta es una medida del tamaño de un programa. Por lo general, cuanto más grande sea el tamaño del código de un componente, más probable será que el componente sea complejo y proclive a errores.
- **Complejidad ciclomática:** Ésta es una medida de la complejidad del control de un programa. Tal complejidad del control puede relacionarse con la comprensibilidad del programa.

Realizamos el análisis del código mediante dos métodos:

**Análisis dinámico:** Consiste en analizar las funcionalidades del sistema en ejecución, e ir comparando si dichas funcionalidades se ven reflejadas en el código.

**Análisis estático:** Consiste en realizar un análisis del código fuente sin ejecutar el programa, que nos permiten obtener resultados que nos ayudan a determinar si el código fuente cumple o no los estándares definidos previamente, esto es, si cumple con las convenciones de código para el lenguaje de programación Java.

### 1.2.1. Análisis dinámico

En este punto se analiza la aplicación desde el ejecutable, prestándole atención a las funcionalidades del programa y cuál es la respuesta esperada. En caso que

se encuentre una falla o una respuesta inesperada, procedemos a analizar el código manualmente y plantear por qué puede que se haya producido la misma.

La razón por la que en una primera instancia lo analizamos desde el ejecutable es que desde el IDE la aplicación no compila.

No se nos facilitó un manual de uso de la aplicación, por lo que comenzamos por la suposición de que la misma cumple con las 10 reglas heurísticas de usabilidad de *Jakob Nielsen*. Entre sus reglas se encuentran algunas como que el sistema nos mantiene siempre informados de lo que está ocurriendo, que tiene mensajes de error claros y concisos, entre otras. Todas estas se ven reflejadas en el código.

- **Estado del sistema:** Si se hace clic en cualquier parte de la lista de usuarios, el sistema elige uno de todos los usuarios pero nunca avisa al usuario a cuál de todos eligió. Lo mismo ocurre con la lista de profesionales.

*Posible arreglo:* En alguna parte de la interfaz poner "Usuario logueado + nombre".

- **Error en la información mostrada:** Si un profesional entra a la lista de planes de alimentación solicitados, elige un usuario y mira su información pero luego va a otra pestaña y regresa, la información de las preferencias, ingestas y restricciones se siguen mostrando en pantalla pero no su perfil con el nombre y fecha de nacimiento.

*Posible arreglo:* setear la visibilidad de los campos de preferencias, ingestas y restricciones a no visibles al momento que se elige otra opción del menú.

- **Acceso a las funcionalidades:** Dependiendo el tamaño de la pantalla en la que se estaba usando, algunas funcionalidades no se visualizan de forma correcta. Por ejemplo: cuando vamos a la solicitud de planes de alimentación, no podemos acceder al paso que suponemos es crear el plan porque el tamaño de la ventana es mayor/igual al de la pantalla del ordenador. Este error fue documentado por parte del desarrollador.

*Error en el código:* Se eligió como tamaño de la pantalla, a la resolución y tamaño que tienen las máquinas del laboratorio de la facultad.

*Posible arreglo:* Que el tamaño de la ventana sea en relación al dispositivo del cuál se está usando la aplicación.

*Nota:* no se puede continuar con el análisis de la solicitud de planes porque el botón de continuar queda oculto.

- **Falta de flexibilidad en el intercambio de mensajes:** En esta sección se producen errores como que no hay una petición de confirmar si el profesional desea mandar el mensaje o no cuando aprieta enviar, ya que una vez que lo hace no puede mandar otro o ver el historial de la conversación con el paciente hasta que este último solicite otra consulta.

Esto hace que se creen errores o interferencias innecesarias.

*Possible arreglo:* Agregar una ventana de confirmación al enviar la respuesta, con un mensaje directo como "¿Está seguro que desea enviar el mensaje? Una vez que lo haga no puede volver a responder."

- **Interfaz:** Si bien la observación del diseño de la interfaz se hace un poco subjetivo, hay algunos aspectos que nos parecieron importante destacarlos y que a futuro deberían mejorarse.

- El programa no ayuda mucho al usuario a saber qué hacer, probarlo para nosotros se basó en una gran cantidad de probar "a ver si funciona", muchos de los botones no tienen escrito para qué son y los que tienen iconos son poco descriptivos, no representan la realidad de su uso.

- A lo largo de toda la aplicación se intercambian los iconos a los cuales se refiere a los usuarios y a los profesionales. A estos últimos a veces se los refiere con la silueta de una persona (misma que para el usuario), o el contorno de un gorro de chef. Lo que hace confundir a la persona que la usa de desde qué perfil estamos accediendo.

- Una observación que va por lo estético, es que los mensajes de que una lista está vacía o que no hay peticiones, se encuentran todos dispersos, no están centrados ni en el mismo lugar uno de los otros. Lo hace poco profesional y no agradable a la vista por falta de consistencia.

- Se muestran muchas pantallas vacías, si no seleccionamos una opción o apenas ingresamos en el sistema, se espera que en ese lugar haya un mensaje indicándonos de como proceder, o simplemente pidiéndonos que elijamos una opción del menú.

### 1.2.2. Análisis estático

Para realizar un análisis estático del código, utilizamos tres herramientas analizadoras de código que nos permiten determinar algunas propiedades tales como la conformidad con estándares de codificación, métricas de calidad o anomalías en el flujo de datos.

## SonarQube

SonarQube es una herramienta de revisión automática de código para detectar errores, vulnerabilidades y code smells en el código.

Para ver cuáles son las métricas y cómo funciona SonarQube dirigirse al **anexo sección 4.0.2 - SonarQube**



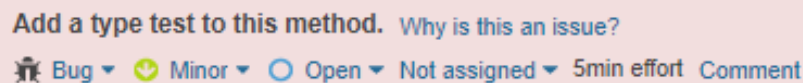
A continuación se detallan bugs, vulnerabilidades y Code Smell encontrados:

## Bugs

Para el siguiente método *equals* :

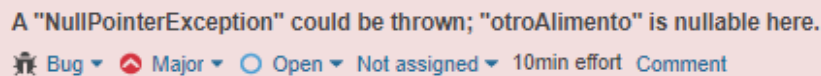
```
Alimento otroAlimento = (Alimento) obj;  
return (this.getNombre().equals(otroAlimento.getNombre()));
```

### ■ Chequear tipo de parámetro



Como el método *equals* recibe como parámetro un objeto de tipo *Object*, se le puede pasar cualquier objeto. En este caso debería verificar el tipo de parámetro, y no suponer que solo se usará para comparar objetos de la clase *Alimento*.

### ■ Posible excepción



Otro bug encontrado y que es de severidad mayor es que el objeto *otroAlimento* puede ser NULL, y eso no está siendo verificado, por lo cual esta función podría lanzar *NullPointerException*.

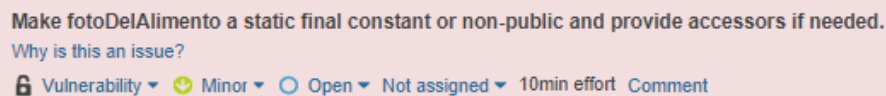
Estos bugs se repiten en 4 métodos *equals* más.

## Vulnerabilidades

Dada la siguiente variable con privacidad pública:

```
public ImageIcon fotoDelAlimento;
```

El programa nos indica:



Debemos convertir la variable *fotoDelAlimento* en una constante o cambiar su visibilidad a privada.

Las variables públicas no respetan el principio de encapsulación, se expone la representación interna del objeto, y su valor puede ser cambiado desde cualquier parte del código lo cual es muy inseguro.

Las restantes vulnerabilidades detectadas también hacen referencia a variables públicas que deben ser convertidas a constantes o que debemos cambiar su visibilidad a privada.

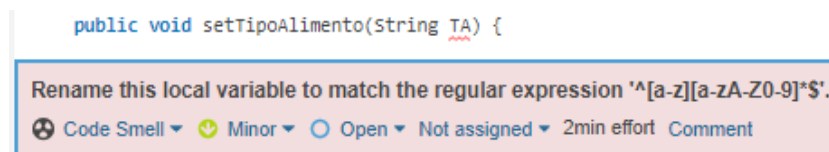
## Code Smells

### Importaciones que no se utilizan (Minor)



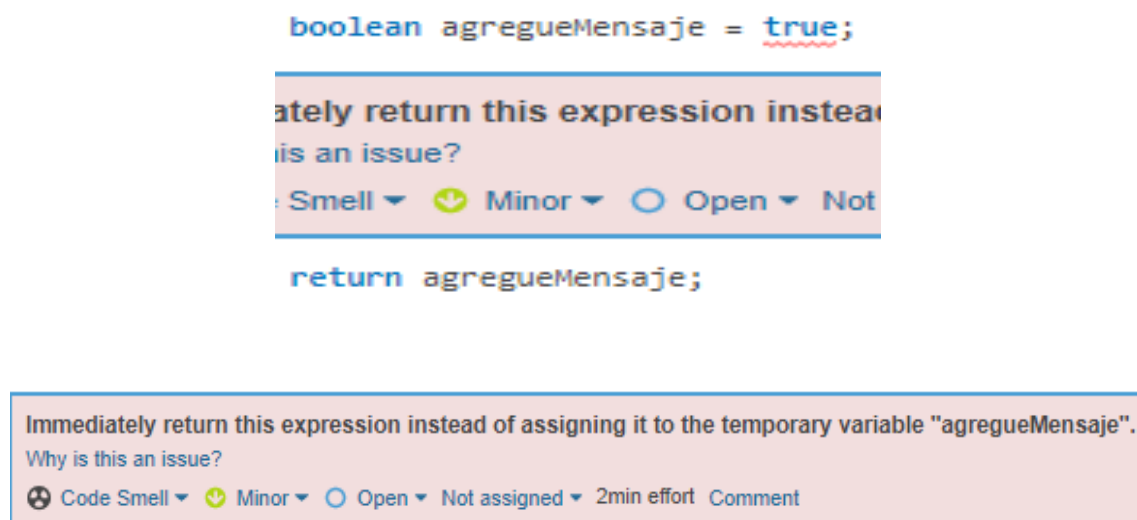
Las importaciones de esos paquetes son innecesarias ya que no se utiliza ninguna clase correspondiente a dichos paquetes.

### Nombre de variables con mayúscula (Minor)



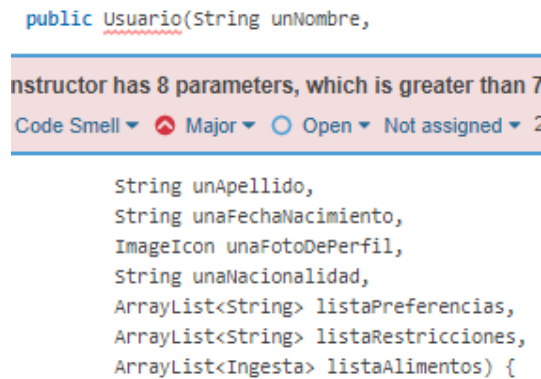
De acuerdo a las convenciones de Java, los nombres de las variables deben comenzar en minúscula (excepto constantes).

### Asignaciones innecesarias (Minor)



Esta creando la variable booleana *agregueMensaje* e inmediatamente la retorna, se pudo haber simplificado con solo una sentencia: *return true;*

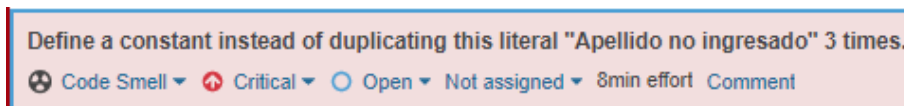
### Constructor con demasiados parámetros (Major)



```
public Usuario(String unNombre,  
               String unApellido,  
               String unaFechaNacimiento,  
               ImageIcon unaFotoDePerfil,  
               String unaNacionalidad,  
               ArrayList<String> listaPreferencias,  
               ArrayList<String> listaRestricciones,  
               ArrayList<Ingesta> listaAlimentos) {
```

El constructor no debería tener mas de 7 parámetros, en este caso tiene 8.

### Necesidad de definir una constante (Critical)



En este caso el String *Apellido no ingresado* se utiliza en 3 lugares diferentes, por lo que sería adecuado definir una constante con dicho valor.

## FindBugs

La segunda herramienta de análisis de código que usamos fue FindBugs y utilizamos la versión de escritorio de la misma, la misma busca errores y *bugs* de código en programas escritos en lenguaje Java.

Para ver una explicación de cómo funciona FindBugs dirigirse a Anexo sección 4.0.3- Funcionamiento de FindBugs

A continuación explicaremos gran parte de los bugs encontrados por FindBugs (en este caso se encontraron 123 bugs). Es importante destacar que muchas veces si se encuentra el mismo bug en varios lugares creemos innecesario mencionarlo repetidamente.

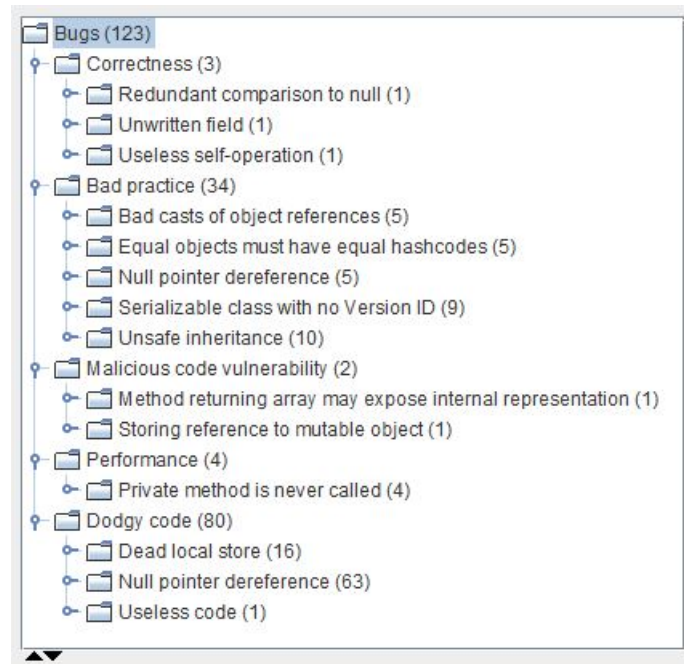


Figura 1.6: Vista general de los bugs encontrados.

## ■ Correctitud

i.

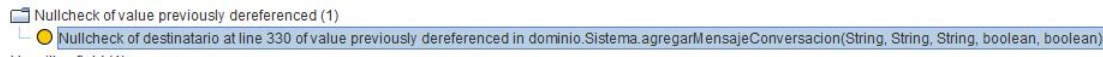


Figura 1.7: Bug encontrado.

Nos indica que se está chequeando si un valor es null que ya se sabe de antemano que no puede ser null, ya que hubiese dado un *null pointer exception* en alguna parte anterior a ésta en el código. Puede haber ocurrido dos cosas: que el chequeo sea redundante o que la llamada previa es errónea.

ii.



Figura 1.8: Bug encontrado.

No se asigna un valor al campo y por ende se devuelve el valor por defecto. FindBugs nos sugiere seguir uno de dos caminos: chequear si hay errores o ver si debería haber sido inicializada correctamente.

iii.

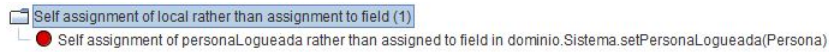


Figura 1.9: Bug encontrado.

Se produce una asignación de una variable a sí misma. Nos pregunta chequear si no quisimos asignarle el valor a otra variable.

Este bug también se encuentra en la sección de código dudoso en donde explicamos qué quiso probablemente poner el desarrollador.

## ■ Malas prácticas

i.

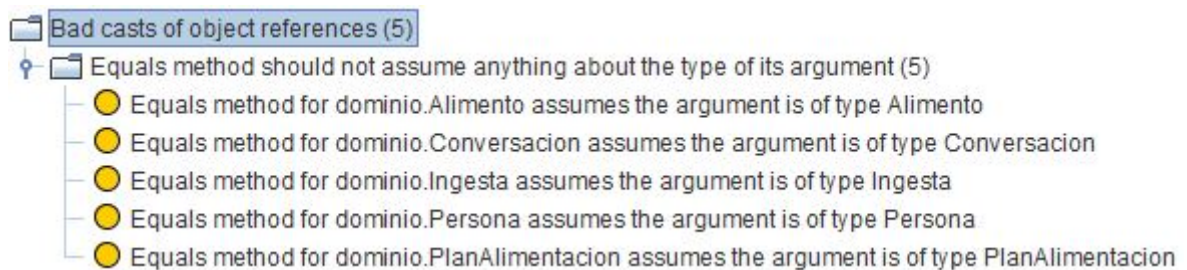


Figura 1.10: Bug encontrado.

Se asume que el parámetro que se le pasa a la función *equals* es del mismo tipo que el esperado, no se debería asumir el tipo del objeto que se le pasa.

Por ejemplo, el primer error de la captura anterior: *"Equals method should not assume anything about the type of its argument"*. Se asume que el método *equals* de la clase *Alimento* es del tipo *Alimento*, se debería retornar *false* si el objeto no es igual a *this*.

ii.

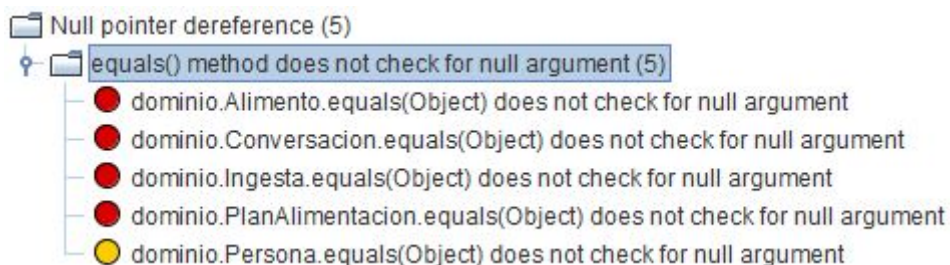


Figura 1.11: Bug encontrado.

Al igual que el anterior, un error en el método *equals* es no chequear que lo que se le pasa por parámetro no sea *null*. Esta implementación de dicha función

rompe con como se define la función por *java.lang.Object.equals()*, todos los métodos *equals* deberían retornar falso si el valor pasado es *null*.

## ■ Vulnerabilidad del código

i.

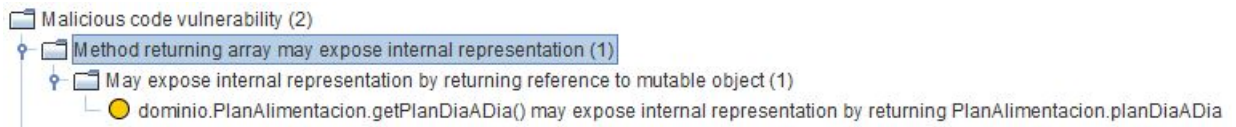


Figura 1.12: *Bug* encontrado.

El bug está en la línea 58 del paquete *dominio*, clase *PlanAlimentación*, método *getPlanDiaADia()*.

Nos indica que se puede exponer la representación interna si retornamos un objeto que puede ser cambiado, puede comprometer la seguridad u otras propiedades. FindBugs nos recomienda que deberíamos retornar una nueva copia del objeto.

## ■ Performance

i.



Figura 1.13: *Bug* encontrado.

Se define un método que nunca es usado, se debería borrar/sacar del código. Esto ocurre en tres instancias de la clase *Sistema* del paquete *dominio*, con los métodos: *agregarConversacionALaLista*, *copiarLista*, *yaExisteIngestaEnEsaFecha* y en el paquete *interfaz*, clase *VentanaMenuPrincipalProfesional* con el método *ocultarCheckBox*.

En total eliminando los cuatro métodos sugeridos del código, evitaríamos 28 líneas de código.

## ■ Código dudoso

i.



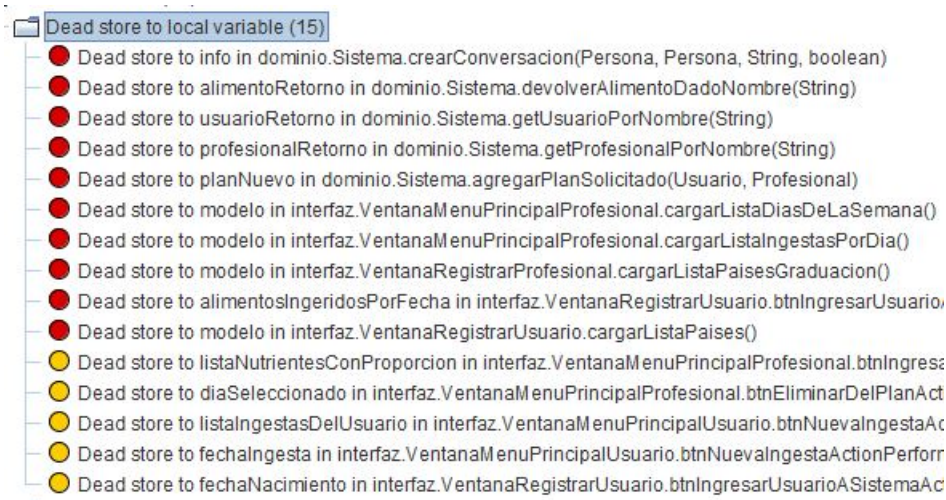


Figura 1.14: Bug encontrado.

Este es un ejemplo de una falsa alarma de FindBugs. El bug nos dice que se le asigna un valor a una variable pero este nunca es usado, usualmente esto indica un error porque el valor computado nunca se usa. Pero al analizar el código manualmente vemos que en todos los casos sí se usa el valor computado, por ende podemos descartar esta advertencia.

ii.



Figura 1.15: Bug encontrado.

Otro ejemplo de una falsa alarma de FindBugs. En este método de test, el desarrollador pretende comparar un objeto con los valores null a otro objeto creado que también tiene valores null, como puede verse en la siguiente imagen:

```

@Test
public void testGetSetsNullUsuario() {
    Persona usuario = null;
    Persona profesional = null;
    ArrayList listaMensajes = null;
    Conversacion conversacionATestear = new Conversacion(usuario, profesional, listaMensajes);
    Persona usuarioEsperado = new Usuario(null, null, null, null, null, null, null, null);
    assertEquals(conversacionATestear.getUsuario(), usuarioEsperado);
}

```

Figura 1.16: Método de test.

iii.



Figura 1.17: Bug encontrado.

Encontró un método con retorno del tipo *void*, que no tiene utilidad alguna. Puede haber ocurrido dos cosas: que el desarrollador se haya equivocado al definir el comportamiento o que efectivamente no sea necesario. En este caso si chequeamos de forma manual el código nos damos cuenta que en realidad el desarrollador tuvo un error al definir el método. Está asignando una variable a si misma (lo cual no tiene ninguna utilidad). En vez de poner *personaLogueada = personaLogueada*;, se quería poner ***this.personaLogueada = personaLogueada***;;.

## EasyPmd

PMD es una herramienta que chequea la calidad del código reportando code smells, malos hábitos de codificación, entre otros.

Funciona mediante la elección de filtros a aplicar, eligiendo qué tipo de errores o bugs debe buscar, en este caso elegimos los siguientes: Basic, Imports, Braces, Code Size, Comments, Design, Unnecessary Code, Empty Code.

En esta aplicación se encontraron un total de 124 errores y 76 warnings.

A continuación, se presenta una vista general de los mismos:

- **Nombre de variables:**



	Description	File ▲
●	3 - Avoid excessively long variable names like listaNutrientesConProporcion	Alimento.java
●	3 - Avoid variables with short names like TA	Alimento.java

Figura 1.18: Bug encontrado.

Nos recomienda evitar nombres muy largos o muy cortos de variables. Este chequeo corresponde al punto 1 de Java Inspection Checklist (*Variable, Attribute, and Constant Declaration Defects (VC)*)

Este es un error recurrente a lo largo de todo el código. Otro ejemplo del mismo:

●	3 - Avoid excessively long variable names like fueAtendidaConsulta	Conversacion.java
●	3 - Avoid variables with short names like pr	Conversacion.java
●	3 - Avoid excessively long variable names like intercambioRemitente	Conversacion.java

Figura 1.19: Bug encontrado.

#### ■ Uso de variables

	Description	File ▲
●	3 - Consider simply returning the value vs storing it in local variable 'agregueMensaje'	Conversacion.java

Figura 1.20: Bug encontrado.

Este bug hace referencia a un método en el cual se crea una variable y se le setea el valor true, y en la siguiente línea se hace un *return variable*. La creación y declaración de dicha variable es innecesaria ya que se puede retornar el valor *true* directamente.

#### ■ Nombre de paquetes

	Description	File
●	3 - Package name contains upper case characters	Main.java

Figura 1.21: Bug encontrado.

El nombre del paquete no sigue con los estándares de que deben ser definidos en minúscula.

#### ■ Variables de clase

	Description	File
●	3 - Too many fields	MostrarPerfilUsuario.java

Figura 1.22: Bug encontrado.

Hay muchas variables de clase, esto hace que tienda a tener muchas responsabilidades lo que dificulta entenderla y además dificulta el mantenimiento a largo plazo de la aplicación.

### ■ Complejidad

	Description	File
●	3 - This class has a bunch of public methods and attributes	VentanaMenuPrincipalProfesional.java
●	3 - Avoid really long classes.	VentanaMenuPrincipalProfesional.java
●	3 - This class has too many methods, consider refactoring it.	VentanaMenuPrincipalProfesional.java

Figura 1.23: Bug encontrado.

Se ven dos tipos de bugs, el primero es que una clase tiene muchos métodos y atributos, y el segundo es que la longitud de la clase es excesiva.

Corresponde al punto 1 de *Java Inspection Checklist (Layout and Packaging Defects (LP))*.

Otro ejemplo de alta complejidad dentro de un método:

	Description	File
●	3 - The method <code>nutrientesSeleccionados()</code> has an NPath complexity of 2187	VentanaMenuPrincipalProfesional.java

Figura 1.24: Bug encontrado.

### ■ Comparación de Strings

	Description	File
●	3 - Position literals first in String comparisons	VentanaMenuPrincipalProfesional.java

Figura 1.25: Bug encontrado.

Se refiere al punto de *Java Inspection Checklist* que trata sobre *Comparison/-Relational Defects (CR)*. Sugiere que el orden de los elementos comparados es incorrecto.

Este bug es recurrente a lo largo de todo el código.

### ■ Elección de estructura

	Description	File
	3 - Avoid if (x != y) ..; else ..;	VentanaMenuPrincipalProfesional.java
	3 - Avoid if (x != y) ..; else ..;	VentanaMenuPrincipalProfesional.java
	3 - Avoid if (x != y) ..; else ..;	VentanaMenuPrincipalProfesional.java
	3 - Avoid if (x != y) ..; else ..;	VentanaMenuPrincipalProfesional.java
	3 - Avoid if (x != y) ..; else ..;	VentanaMenuPrincipalProfesional.java
	3 - Avoid if (x != y) ..; else ..;	VentanaMenuPrincipalProfesional.java
	3 - Avoid if (x != y) ..; else ..;	VentanaMenuPrincipalProfesional.java

Figura 1.26: Bug encontrado.

Bug encontrado con el filtro de diseño.

Es una convención de que al tener un método con muchos *if-else* se está buscando problemas ya que la lógica condicional es difícil de manejar.

#### ■ Elección de operador

	Description	File
	3 - Substitute calls to size() == 0 (or size() != 0, size() > 0, size() < 1) with calls to isEmpty()	VentanaMenuPrincipalUsuario.java

Figura 1.27: Bug encontrado.

Para verificar si un String está vacío, lo mejor es utilizar la función *isEmpty()* en vez de consultar por su tamaño y comparar con cero.

Se refiere también al punto de *Java Inspection Checklist* que dice *Comparison/Relational Defects (CR)*, en la pregunta *Are the comparison operators correct?* (¿los operadores de comparación son los correctos?).

#### ■ Ausencia de *import*

	Description	File
	cannot find symbol symbol: class ArrayList location: class dominio.Sistema	Sistema.java
	cannot find symbol symbol: class ArrayList location: class dominio.Sistema	Sistema.java
	cannot find symbol symbol: class ArrayList location: class dominio.Sistema	Sistema.java
	cannot find symbol symbol: class ArrayList location: class dominio.Sistema	Sistema.java
	cannot find symbol symbol: class ArrayList location: class dominio.Sistema	Sistema.java
	cannot find symbol symbol: class ArrayList location: class dominio.Sistema	Sistema.java
	cannot find symbol symbol: class ArrayList location: class dominio.Sistema	Sistema.java
	cannot find symbol symbol: class ArrayList location: class dominio.Sistema	Sistema.java
	cannot find symbol symbol: class ArrayList location: class dominio.Sistema	Sistema.java
	cannot find symbol symbol: class ArrayList location: class dominio.Sistema	Sistema.java

Figura 1.28: Bug encontrado.

Este error no nos permite compilar la aplicación, hace falta importar la clase de *ArrayList* para poder trabajar con listas.

# Análisis manual del código

Si bien en el análisis dinámico vimos manualmente un poco del comportamiento de la aplicación, no nos enfocamos en el código sino en el funcionamiento y la interfaz de la misma.

Analizando el código manualmente encontramos errores o bugs que algunas o todas las herramientas no los detectaron. Los consideramos importantes de mencionar porque corresponden a secciones y preguntas de *Java Inspection Checklist*.

- El que consideramos más importante es que el código no compila. Tiene errores en muchas clases (pertenecientes al dominio, la interfaz y los paquetes de test), desde llamar a funciones con parámetros de tipo equivocado hasta no haber importado las librerías correctas. La aplicación corre desde el ejecutable pero no desde el IDE, es decir, que el ejecutable no corresponde al código fuente entregado.

- Al no corresponderse el ejecutable con el código fuente entregado, hay algunas funcionalidades que funcionan correctamente al correr el sistema desde el ejecutable pero no si lo hacemos desde el IDE.

Al ingresar a la sección consultas con un profesional, si hay al menos uno de ellos registrados, el sistema lanza una excepción del tipo *NullPointerException* y no se permite continuar.

Otro error detectado, es que a la hora de que un usuario quiere visualizar los planes de alimentación, el sistema lanza una excepción del tipo *NullPointerException*, y no se permite continuar con la consulta.

Sucede algo similar en el momento de solicitar un plan de alimentación a un profesional, en el momento que hacemos clic en el icono para enviar la solicitud, el sistema lanza una excepción del tipo *NullPointerException* y no se puede continuar.

Al ingresar a la sección *Ingresar alimento ingerido*, ocurre que al hacer clic en el icono para registrar el consumo del alimento, el sistema lanza una excepción del tipo *NullPointerException*.

Si ingresamos al menú principal del profesional, y nos dirigimos a la sección *Consultas pendientes*, el sistema muestra una lista (vacía) de usuarios, y lanza una excepción del tipo *NullPointerException*.

De forma similar sucede si ingresamos a la sección *Planes de alimentación solicitados* habiendo registrado algún alimento previamante. Al ingresar, se muestra una lista vacía de usuarios, el sistema lanza una excepción del tipo *NullPointerException* y no se permite continuar.

En conclusión, si ejecutamos la aplicación mediante el IDE, las únicas funcionalidades que funcionan correctamente son: **Registro de usuario**, **Registro**

de profesional y Registro de alimento.

- Comentarios:
  - Las clases, métodos o archivos no tienen comentarios de encabezado.
  - Los atributos, declaraciones de variables y declaraciones de constantes no tienen comentarios.
  - El comportamiento de los métodos no está brevemente explicado en lenguaje natural.
  - El código no tiene suficientes comentarios lo que dificulta su comprensión.
- Complejidad:
  - Hay clases de más de 2000 líneas, por ejemplo: en el *package* interfaz.
- Se importan librerías de Java que nunca son usadas, deberían de borrarse esas líneas de código.
- Gran cantidad de líneas sobrepasan los 80 caracteres sugeridos (visualmente en el IDE se ve que se pasan de la línea vertical)

## Análisis general y comparación entre analizadores

Las ventajas que encontramos con el *plugin EasyPMD* es que se muestran los errores/sugerencias en el IDE mismo de *NetBeans*, así como también el grado de severidad de los errores, mostrándolos de diferentes colores según el grado: *high*, *medium*, *medium low*, *low* y *warnings*. A partir de esto el desarrollador puede saber cuales son los errores que debe priorizar, y a su vez cuando posteriormente realice cambios en el proyecto sabrá si el mismo mejoro, comparando la cantidad de errores con grado de severidad alto que tenia con respecto a los que tiene ahora.

En cuanto a FindBugs; no usamos el *plugin* de *NetBeans* sino la aplicación de escritorio, y si bien la misma nos dice el número de línea en que está el *bug*, requiere estar constantemente cambiando de aplicación entre esta y el IDE para ver los errores en su contexto.

Un aspecto positivo de FindBugs sobre PMD es que los errores están mejores explicados, con palabras más cercanas al idioma cotidiano y explicaciones más extensas, generalmente con leerlas nos imaginamos la solución. PMD usa frases más cortas que hace que busquemos en el navegador una mejor explicación.

Con PMD nos fue muy fácil aplicar filtros del tipo de bug que queremos visualizar, ocultando los que no nos interesan, esto hizo que podamos trabajar por

secciones de forma mas organizada.

También utilizamos la herramienta SonarQube, la cual en cuanto a interfaz y detalle de errores es la más atractiva informándonos acerca de: *bugs*, vulnerabilidades, aspectos de seguridad, así como también la cantidad de bloques y líneas duplicadas de código. En cada uno de estos puntos se indica un *rating* compuesto por una letra de la A a la E (siendo A y E el mejor y peor estado respectivamente). Esto nos permite saber si realmente nuestro sistema mejoró en cuanto a calidad de código, considerando que lo ideal seria que todas las métricas tengan categoría A, y que la cantidad de lineas duplicadas tiendan a ser cero.

Esta herramienta también es muy útil, porque al hacer clic en un determinado error nos lleva al código y nos muestra con colores en qué líneas se encuentra el error.

Dentro de cada uno de los tipos de errores mencionados, el programa clasifica a su vez cada uno según su grado de severidad de la siguiente forma: *blocker*, *critical*, *major*, *minor*, *info* (esto nos ayuda aún más a priorizar algunos errores sobre otros).

Por lo tanto, concluimos que las tres herramientas son complementarias y es beneficioso utilizar las tres a la vez.

Las principales convenciones presentes en el *Java Code Conventions*[3], y que no se cumplen son: falta de comentarios en métodos y clases, nombre de paquetes y variables con mayúscula, líneas de mas de 80 caracteres, variables sin inicializar, funciones con muchos parámetros, clases con muchas lineas de código y variables publicas que deberían ser privadas o bien ser definidas como constantes.

En cuanto a la cantidad de lineas totales y la complejidad ciclomática del sistema, ambas métricas fueron calculadas por el programa *TikiOne JaCoCoverage* y los resultados se pueden observar en la figura 1.29.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty
Sistema		58%		51%	62	121
Sistema.Paises		0%		n/a	4	4
Usuario		43%		25%	12	21
Conversacion		53%		50%	9	21
Persona		87%		78%	4	21
InformacionMensaje		81%		n/a	1	9
Sistema.DiasDeLaSemana		90%		n/a	2	4
Sistema.Preferencias		87%		n/a	2	4
Sistema.Restricciones		84%		n/a	2	4
Sistema.IngestasPorDia		81%		n/a	2	4
Alimento		93%		90%	2	16
Ingesta		98%		83%	2	13
PlanAlimentacion		100%		75%	2	17
Profesional		100%		75%	2	12
ComposicionAlimento		100%		100%	0	6
Total	1.071 of 2.720	61%	105 of 244	57%	108	277

Figura 1.29: Complejidad ciclomática y cantidad de lineas totales.

La complejidad ciclomática total del dominio del sistema es de 277, pero para realizar un análisis de este valor consideraremos la complejidad ciclomática promedio por clase, cuyo valor es de 18.5.

Basándonos en la tabla que se observa en la figura 1.30, concluimos que el sistema tiene una complejidad media y un riesgo moderado.

Cyclomatic Complexity	Risk Evaluation	Probability of Bad fix
1-10	Low risk, testable code	5%
11-20	Moderate Risk	10%
21-50	High Risk	30%
>50	Very High Risk, untestable code	40%

Figura 1.30: Evaluación de riesgo a partir de complejidad ciclomatica

En conclusión, mediante el uso de las cuatro herramientas y la revisión manual del código fuente determinamos que la calidad actual del sistema es regular ya que notamos una serie de convenciones que se declaran en el *Java Code Conventions* y que no se cumplen. Además, mediante el análisis de la complejidad ciclomática, determinamos que el sistema tiene una complejidad media y un riesgo moderado.

### 1.3. Calidad de los casos de prueba

Los atributos de calidad son las cualidades que un programa debe satisfacer. Un producto de calidad se traduce a una reducción de costos y en una mejora general de todo el programa, una buena calidad en el producto se transfiere a una buena calidad en el uso del mismo.

Un estándar internacional para la evaluación de la calidad del software es **ISO 9126**. Este modelo de calidad establece una serie de características, que a su vez cada una tiene una serie de atributos (los atributos pueden ser verificado o medidos): **funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad, portabilidad, conformidad**.

Para una explicación de estos atributos y del estándar ISO 9126 dirigirse al anexo sección 4.0.4- ISO 9126.

Aplicaremos este estándar a los casos de prueba de la aplicación a analizar con el fin de determinar la calidad de cada caso.

#### Funciones generales del sistema

##### ■ Ingreso a la aplicación

La **portabilidad** se ve afectada en un aspecto que ya fue mencionado en otras secciones. El tamaño de la ventana está pensado para un tamaño de pantalla específico y para poder usar la aplicación con todas sus funcionalidades tenemos que redimensionarlo en el código.

##### ■ Ingresar al sistema (Profesional):

- Si no hay profesionales cargados en el sistema, la lista se encuentra vacía, al hacer clic sobre ella se indica en el documento que el sistema debe mostrar un mensaje que indique que no hay profesionales creados, pero esto no sucede, ya que en ese caso el sistema no muestra ningún mensaje al usuario, afectando la **usabilidad** y generando un cierto grado de incertidumbre, el cual consideramos que puede ser solucionado mediante un mensaje claro indicando que no hay profesionales registrados.
- Un problema de la **exactitud** en la **funcionalidad** es que habiendo profesionales registrados en el sistema, al hacer clic en la lista, ocurre el problema de que si no hacemos clic precisamente encima de un profesional de todas formas nos accede al perfil de uno de ellos, lo que genera una gran incertidumbre ya que además al ingresar al perfil en ningún momento se muestra el nombre del profesional logueado, es decir, que no sabemos a que perfil hemos ingresado. Como solución, no se debería permitir ingresar al perfil de un usuario en caso de que no se haga clic



exactamente encima de su nombre en la lista, y además, para mejorar la **usabilidad** y fortalecer la visibilidad del estatus del sistema, sería oportuno al momento de ingresar al perfil que muestre en alguna sección de la interfaz el nombre del profesional que está logueado.

#### ■ Registrar profesional/usuario

- Al registrar un profesional o usuario, encontramos un problema que puede afectar la **usabilidad** del sistema, y que hace referencia a la fecha de nacimiento. Al seleccionar la misma, se despliega un calendario que tiene como fecha por *default* la fecha del día en que se está usando la aplicación, y al tratarse de la fecha de nacimiento del usuario, el mismo tiene que realizar una gran cantidad de clics (al menos 18 si consideramos personas mayores de edad) hasta alcanzar su fecha de nacimiento, por lo cual consideramos adecuado poner por defecto una fecha de por lo menos 18 años atrás para que el usuario no tenga que hacer tanto esfuerzo para llegar a su fecha de nacimiento, así como también, que se permita cambiar la fecha directamente desde el teclado.

En esta funcionalidad persiste como en otras, el problema de que se puede registrar un usuario (o profesional) dejando el campo fecha de nacimiento sin completar.

- Si bien no se permite registrar dos usuarios iguales con el mismo nombre y apellido, el sistema en ningún momento nos avisa dicha restricción. Para mejorar la **usabilidad** y visibilidad del sistema lo mejor sería poner un mensaje de error, ya que en este momento el mismo queda sin responder y hasta no poner otra opción o cambiar el nombre/apellido no hace nada.

### Funcionalidades (ROL USUARIO)

#### ■ Consulta con profesional

- En el caso que no hay profesionales en el sistema, se informa un mensaje claro de que no es posible realizar la consulta, consideremos que la calidad de este caso es óptima.
- Cuando aún no hemos iniciado una conversación con un profesional, se hace muy largo el proceso para iniciar la conversación, esto afecta la **usabilidad** del sistema, en este caso, si un usuario quiere iniciar una conversación con un profesional tiene que realizar un total de 6 clics (contando el envío del mensaje).

Mostrando directamente la lista de todos los profesionales registrados a los que se puede enviar consulta esta operación podría ser realizada por el usuario en un total de 4 clics.

Otro aspecto que afecta la usabilidad del sistema es que a la hora de iniciar una conversación con un profesional con el cual aún no hemos entablado una conversación, debemos clickear en el símbolo  $\pm$ , y si no hay ningún profesional pendiente para iniciar una conversación no se informa

ningún mensaje al usuario, esto puede generar confusión porque no sabe si realmente es que no hay profesionales disponibles o bien ocurrió un error en el sistema. Lo adecuado sería informar al usuario en ese momento que es lo que está ocurriendo en el sistema.

#### ■ Plan de alimentación

- Si no hay profesionales creados, el sistema muestra claramente un mensaje indicando que no se puede solicitar el plan porque no hay profesionales registrados. En este caso el funcionamiento es óptimo.
- Habiendo profesionales registrados en el sistema, es posible solicitar un plan de alimentación. En términos generales la calidad de esta funcionalidad es óptima, pero consideramos que un aspecto que podría mejorar la **usabilidad**, sería cambiar el icono de agregar un nuevo plan, el cual tiene un calendario por uno que represente a un alimento y debajo de este un pequeño texto que indique: *Solicitar un nuevo plan*, realizando este cambio pensamos que el tiempo requerido por el usuario para hacer uso de esta funcionalidad puede disminuir sensiblemente.
- Habiendo un plan creado por un profesional para el usuario, se ingresa en el icono de libreta con lupa, y se muestra claramente los datos del profesional y la dieta indicada. Para esta funcionalidad consideramos que la calidad es óptima, aunque como se explica en el punto anterior si se acompañara el icono con un pequeño texto que indique: *Ver planes indicados*, consideramos que disminuiría el tiempo que le lleva al usuario hacer uso de esta funcionalidad.

#### ■ Ingresar alimento ingerido

- Si no hay alimentos registrados en el sistema, se muestra un mensaje claro indicando que no hay alimentos registrados. La calidad de esta funcionalidad es óptima.
- Habiendo alimentos registrados en el sistema, si se completan todos los datos solicitados se registra la ingesta del alimento. Un aspecto a mejorar en cuanto a **usabilidad** es que se pueda cambiar la fecha de ingesta directamente utilizando el teclado, esto implica que el usuario no tenga que realizar tantos clicks en el calendario para llegar a la fecha de ingesta, considerando que la misma pudo haber sido de días o incluso semanas atrás.
- Si se deja el campo fecha sin completar, el sistema realiza de todas formas el registro de la ingesta. En este caso se visualiza un error **funcional** que puede implicar una gran pérdida de tiempo tanto para el usuario que va a tener que volver a registrar la ingesta, así como también para el profesional que deberá informar al usuario que el registro de la ingesta no se realizó de forma correcta. Además, esto afecta la **confiabilidad** del sistema ya que el mismo sigue operando aun bajo la presencia de errores. Una solución concreta a este problema es que todos los campos sean obligatorios.

## Funcionalidades (ROL PROFESIONAL)

### ■ Consultas pendientes

- Si no hay consultas pendientes de respuesta, el sistema informa claramente que no hay consultas pendientes. En este caso el funcionamiento es óptimo.
- Si hay alguna consulta pendiente, se puede acceder rápidamente a la misma (3 clics), un aspecto que afecta la **usabilidad** del sistema es que se le permite tanto al profesional como al cliente enviar mensajes sin contenido, es decir vacíos, que pueden dar lugar a confusiones en la conversación. Para solucionar esto, no se debería permitir a usuarios y profesionales enviar mensajes sin contenido.

### ■ Planes de alimentación solicitados

- Si no hay alimentos registrados, se muestra un mensaje claro indicando que no hay alimentos. En este caso el funcionamiento es óptimo.
- Si no hay planes solicitados por usuarios, se muestra un mensaje claro indicando que no hay planes solicitados. En este caso el funcionamiento es óptimo.
- Si hay algún plan de alimentación solicitado y algún alimentado registrado se permite elaborar el plan. Cuando se selecciona el usuario que solicitó el plan, se observa un icono que permite visualizar el perfil del solicitante, con respecto a esto, notamos que hay ocasiones en que al clickear el icono el sistema no responde, y no se visualiza el perfil del usuario. Luego al ingresar en Elaborar Plan, se observa una interfaz un tanto sobrecargada que puede llevar a confundir al profesional a la hora de elaborar un plan. También, notamos que es posible elaborar un plan sin agregar alimentos en ningún día, lo cual es bastante absurdo y va en contra de lo acordado para esta función, no cumple con la **funcionalidad**. Si eso sucede probablemente sea por un error del profesional y no un hecho intencional, por lo cual, con el fin de mejorar la **usabilidad** del sistema y ayudar a la prevención de errores, se debería mostrar un mensaje indicando que debe agregar por lo menos un alimento al plan.

### ■ Ingresar alimento al sistema

- La calidad en aspectos generales de esta funcionalidad es óptima. Un aspecto de **usabilidad** que se debería mejorar es la consistencia, reduciendo un poco el tamaño de los iconos de enviar y adjuntar imagen, ya que si lo comparamos con el resto de las fuentes, y campos, son un tanto grandes.

## 2. Solicitud de cambios

### 2.1. Formulario de solicitud de cambios

Los cambios serán todos propuestos por el equipo docente y aceptados por el equipo de desarrollo (Agustín Hernandorena, Paula Areco), en caso de que un cambio sea propuesto por los desarrolladores será explícitamente escrito.

#### 2.1.1. Mala visibilidad y selección del JCalendar en las opciones de registro.

**Nombre de proyecto:** Alimentación saludable

**Fecha:** 10/04/2020

**Lugar:** Uruguay

**Descripción de la propuesta del cambio:**

El calendario provisto por la librería de Java no resulta muy cómodo para el usuario a la hora de seleccionar una determinada fecha. Se propone cambiar el funcionamiento de este calendario, colocando como fecha por defecto una fecha de al menos 18 años atrás (considerando que tanto los usuarios como profesionales son mayores de edad), así como también permitir cambiar la fecha directamente desde el teclado, disminuyendo así la cantidad de clics necesarios para llegar a la fecha deseada.

**Justificación de la propuesta del cambio**

Este cambio es necesario, porque si consideramos que tanto los profesionales como los usuarios son mayores de edad, necesitarían al menos un total de 18 clics para llegar a su fecha de nacimiento desde el calendario, lo cual puede resultar bastante incómodo y tedioso. Al realizar los cambios de poner una fecha predeterminada de unos 18 años atrás, y permitir modificar la fecha directamente desde el teclado, llevarían a disminuir sensiblemente la cantidad de clics necesarios para llevar a cabo esta funcionalidad.

**Alcance:** Al tratarse un cambio que no requiere agregar una nueva funcionalidad al sistema, no implica una modificación del alcance del proyecto. Simplemente se trata de un cambio sencillo en el calendario, con el fin de que el proceso de encontrar la fecha de nacimiento sea mas rápido para el usuario.

**Prioridad:** Media.

**Esfuerzo estimado:** 11 horas.

### **2.1.2. Falta de validación de la fecha de nacimiento o fecha de graduación dentro del JCalendar**

**Nombre de proyecto:** Alimentación saludable

**Fecha:** 10/04/2020

**Solicitante:** Agustín Hernandorena

**Área del solicitante:** Desarrollador

**Lugar:** Uruguay

#### **Descripción de la propuesta del cambio:**

Actualmente en el registro de un usuario o profesional si no se coloca la fecha de nacimiento o graduación, el sistema toma como predeterminada la fecha del día de hoy y realiza el registro, en este caso, se propone que si se dejan estos campos vacíos, se notifique al usuario y no se realice el registro hasta que los campos sean completados.

#### **Justificación de la propuesta del cambio**

Este cambio es necesario, porque se trata de un error de funcionamiento en el sistema. No se debería permitir registrar a un usuario que no proporcione su fecha de nacimiento o graduación, y menos, tomar por defecto que su fecha de nacimiento o graduación es la del día de hoy sin avisarle en ningún momento.

**Alcance:** Al tratarse un cambio que no requiere agregar una nueva funcionalidad al sistema, no implica una modificación del alcance del proyecto. Simplemente se trata de reparar un error de funcionamiento del sistema.

**Prioridad:** Alta.

**Esfuerzo estimado:** 8 horas.

### **2.1.3. Ingreso de campo de fecha de una determinada ingesta.**

**Nombre de proyecto:** Alimentación saludable

**Fecha:** 10/04/2020

**Solicitante:** Agustín Hernandorena

**Área del solicitante:** Desarrollador

**Lugar:** Uruguay

**Gerente de proyecto:** Paula Areco

#### **Descripción de la propuesta del cambio:**

El cambio consiste en que a la hora de registrar una ingesta de un alimento, en lugar de pedirle al usuario que seleccione manualmente la fecha desde un calendario, se asuma que si está ingresando una ingesta, la misma ocurrió ese día, y se tome la fecha directamente desde la computadora.

#### **Justificación de la propuesta del cambio**

Este cambio es necesario ya que de esta forma, se incentiva a que el usuario utilice periódicamente la aplicación, y que no acumule una gran cantidad de ingestas de días anteriores. De esta forma se fomenta que cada vez que un usuario ingiere un alimento lo registre en la aplicación, logrando así un registro mas preciso de la información.

**Alcance:** Al tratarse un cambio que no requiere demasiado esfuerzo ni costo, no se modifica el alcance del proyecto. No se agrega una nueva funcionalidad, sino que simplemente se modifica una con el fin de que el registro de ingestas por parte del usuario se mas preciso y diario.

**Prioridad:** Baja.

**Esfuerzo estimado:** 10 horas.

### **2.1.4. Solo es posible subir al sistema imágenes con extensión png**

**Nombre de proyecto:** Alimentación saludable

**Fecha:** 10/04/2020

**Lugar:** Uruguay

#### **Descripción de la propuesta del cambio:**

El cambio consiste en permitir que a la hora del registro de un profesional o de un usuario el mismo pueda subir una imagen no solo en formato PNG, sino que se admitan más formatos como: JPEG y JPG.

#### **Justificación de la propuesta del cambio**

Este cambio es necesario ya que la mayor parte de los usuarios utilizan otros formatos que no son PNG, principalmente JPG y JPEG. Estas ultimas tienen un peso menor que PNG, y son altamente recomendables para imágenes de tamaño pequeño, como puede ser una imagen de perfil de un profesional o de un usuario.

**Alcance:** Al tratarse un cambio que no requiere demasiado esfuerzo ni costo, no se modifica el alcance del proyecto.

**Prioridad:** Baja.

**Esfuerzo estimado:** 3 horas.

### **2.1.5. Menú principal al ingresar al sistema como Profesional o Usuario**

**Nombre de proyecto:** Alimentación saludable

**Fecha:** 10/04/2020

**Lugar:** Uruguay

**Descripción de la propuesta del cambio:**

El cambio consiste en reemplazar la ventana vacía inicial y mostrar información relevante e interesante al momento de ingresar al sistema, datos de interés para el usuario o profesional en base a los datos que el sistema ha guardado (como lo pueden ser las preferencias y/o los planes alimenticios). Si ingresamos desde la sección del cliente, los datos relevantes que se podrán visualizar son: un listado con los alimentos mas consumidos por ese usuario y un listado con los profesionales que reciben mas consultas.

Ingresando a la sección de un profesional, se podrán visualizar una serie de listas, en donde cada una de ellas incluye: profesionales que son del mismo país que el profesional logueado, profesionales que tienen el mismo título que el profesional logueado, y por ultimo, profesionales que se graduaron en el mismo año que el logueado.

**Justificación de la propuesta del cambio**

Este cambio es importante ya que la ventana de inicio es lo primero que ve el usuario/profesional, es la primera impresión de la aplicación y si queremos captar la atención del mismo necesitamos mostrar información de forma dinámica y que se actualice en base al uso que se le da.

En particular para el usuario, le sera útil en el sentido en que podrá tener una noción de los alimentos mas consumidos, y a partir de esto, ingerir nuevos alimentos que lo lleven a lograr una alimentación mas balanceada.

En cuanto a los profesionales, esta nueva función sera útil en el sentido en que un profesional puede obtener información acerca de sus pares que de cierto modo tienen alguna relación con el, ya sea por tener el mismo título, ser del mismo país o haberse graduado en el mismo año.

**Alcance:** Se le agrega una nueva funcionalidad al sistema, se debe analizar la información almacenada y proseguir en base a lo que contenga.

**Prioridad:** Alta

**Esfuerzo estimado:** 16 horas

## **2.1.6. Listas mostradas en pantalla en el inicio de sesión en caso de no haber registrado usuarios o profesionales.**

**Nombre de proyecto:** Alimentación saludable

**Fecha:** 10/04/2020

**Solicitante:** Paula Areco

**Área del solicitante:** Desarrollador

**Lugar:** Uruguay

**Descripción de la propuesta del cambio:**

Cuando hay alguna lista vacía, como puede ser la lista de Usuarios o de Profesionales sin nadie registrado, el sistema muestra un espacio para ella pero sin datos. El cambio consiste en que si la lista no contiene datos, entonces en su lugar se muestre un mensaje indicando las opciones de registrar usuario o registrar profesional.

#### **Justificación de la propuesta del cambio**

Consideramos que es un cambio importante porque ayuda al usuario a navegar por la aplicación, se le muestran las posibilidades con lenguaje natural ("Para ingresar al sistema registre un usuario"). El mismo disminuirá el tiempo de aprendizaje que una persona necesita para entender la aplicación.

**Alcance:** No requiere modificar el proyecto sino reordenar los elementos de la interfaz y modificar su visibilidad en relación a los estados de las listas.

**Prioridad:** Media

**Esfuerzo estimado:** 11 horas

### **2.1.7. Opción de salir de la aplicación desde todas las ventanas.**

**Nombre de proyecto:** Alimentación saludable

**Fecha:** 10/04/2020

**Lugar:** Uruguay

#### **Descripción de la propuesta del cambio:**

Actualmente la aplicación solo permite salir de la misma desde el inicio solamente. Proponemos implementar un botón de salida desde todas sus ventanas.

#### **Justificación de la propuesta del cambio**

Es un cambio importante ya que le da más libertad al usuario al momento de usar la aplicación, el mismo no debe recordar la información de que el botón de salida está solo en el inicio y puede optar salir de la aplicación en el momento que desee.

**Alcance:** Requiere implementar un botón en las ventanas. No cambia el alcance del proyecto significativamente.

**Prioridad:** Alta

**Esfuerzo estimado:** 7 horas

## **2.2. Proceso de gestión de solicitudes de los cambios**

Para explicar el proceso de gestión de cambios, utilizamos la figura 2.1.



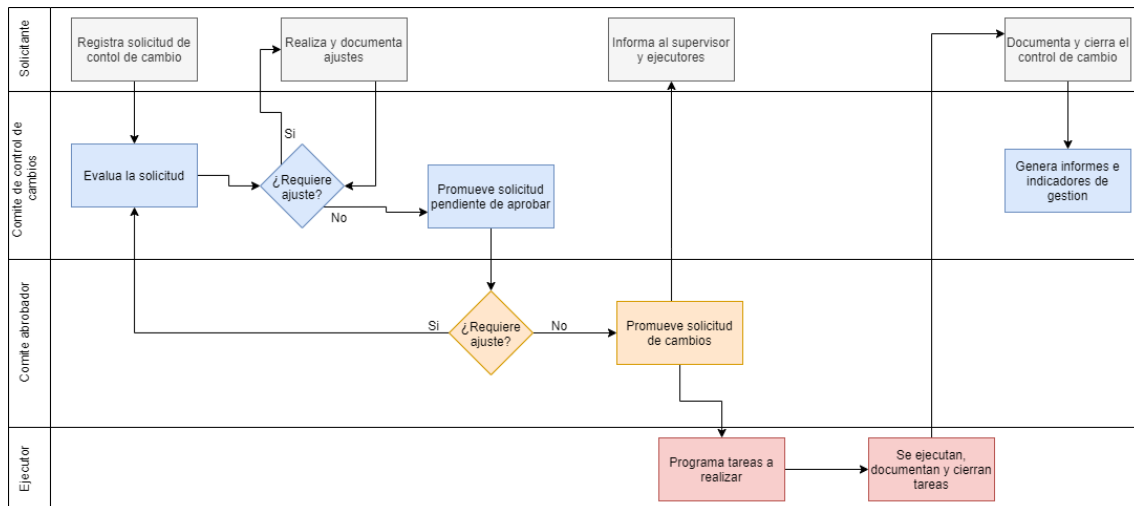


Figura 2.1: Proceso de gestión de cambios

El proceso de gestión de cambios comienza cuando un cualquier interesado en el proyecto envía una solicitud de cambios.

Esta solicitud es evaluada por Comité de control de cambios, el cual determina si es necesario realizar algún ajuste en esa solicitud de cambio. En caso afirmativo, se le indica al solicitante, el cual deberá documentar y realizar los ajustes necesarios, para luego volver a enviar la solicitud. En el caso de que la solicitud no requiera de ajustes, este comité promueve una solicitud pendiente, cuya aprobación estará determinada por el Comité aprobador. Este último, hará una revisión de la solicitud de cambio y determinará si es necesario realizar algún ajuste, en caso de que así sea, le notifica al Comité de control de cambios para que vuelva a hacer una revisión de ese cambio. En caso de que considere que esta todo correcto, se aprueba la solicitud de cambios y se le comunica al solicitante, quien informará al supervisor y ejecutores acerca de la aprobación del cambio.

Luego obtener información acerca del cambio a realizar, el equipo ejecutor planifica, ejecuta, documenta y cierra las tareas.

Una vez ejecutado el cambio, el solicitante documenta y cierra el cambio.

Por último, el comité de control de cambios genera informes e indicadores de gestión, en donde se analizará hasta que punto o en que medida se lograron los objetivos.

## 2.3. Análisis de impacto

El análisis de impacto es una forma de predecir qué elementos del programa van a verse afectados por los cambios propuestos en el sistema.

Es importante que realicemos un análisis de impacto sobre los cambios porque esto influye en decidir cuánto tiempo nos va a tomar realizarlos, la cantidad de recursos que necesitamos, qué pruebas de regresión debemos introducir (estas se basan en probar las partes del código que se pudieron ver afectadas por el cambio) y los recursos que nos va a tomar hacerlos.

En las tablas de impacto hacemos referencia a las principales funcionalidades de la aplicación:

- F1: Registrar paciente.
- F2: Registrar profesional.
- F3: Ingresar ingesta de alimento.
- F4: Consultar a un profesional.
- F5: Solicitar plan de alimentación a un profesional.
- F6: Registrar alimento.
- F7: Ingresar al menú principal de paciente o profesional.
- F8: Ingresar a la Ventana de inicio de la aplicación.

▪ **Cambio 1- Mala visibilidad y selección del JCalendar en las opciones de registro**

Se soluciona intercambiando el calendario elegido (*dateChooser*) por un calendario *JDateChooser* que tiene un campo para ingresar fechas por el teclado además de poder seleccionar desde un *calendar dropdown*.

Funciones que el cambio afecta:

- Función: *VentanaRegistrarProfesional(Sistema unSistema)* (constructor)  
Clase: interfaz.VentanaRegistrarProfesional
- Función: *VentanaRegistrarUsuario(Sistema unSistema)* (Constructor)  
Clase: interfaz.VentanaRegistrarUsuario

Los cambios a realizar en estas funciones son poco profundo, simplemente es necesario cambiar un objeto de tipo *dateChooser* por uno de tipo *JDateChooser*.

	F1	F2	F3	F4	F5	F5	F6	F7	F8
Nivel de documentación									
Especificación de requerimientos									
Especificación de diseño	X	X							
Casos de prueba									
Código fuente	X	X							

Tabla 2.1: Análisis de impacto horizontal y vertical del cambio 1.

Estimación de esfuerzo: 11 horas.

▪ **Cambio 2- Falta de validación de la fecha de nacimiento o fecha de graduación dentro del JCalendar**

Este cambio consiste técnicamente en realizar modificaciones en las funciones: *void btnIngresarUsuarioASistemaActionPerformed(...)* y *void btnIngresarProfesionalASistemaActionPerformed(...)*. En ambos se verifica mediante un condicional *if* que todos los campos a llenar estén completos, excepto la fecha

de nacimiento (en caso de usuario) y la fecha de graduación (en caso del profesional). En caso de que no haya seleccionado la fecha, la misma quedara como un String vacío, es decir, que tenemos: *fechaDeNacimiento* = "" y *fechaDeGraduacion* = "". Luego en el condicional que se verifica si los restantes campos son vacíos, se debería agregar la condición de si la fecha es vacía. Si efectivamente es vacía, el condicional sera *True*, y se invocará a la función *void mostrarErrores (...)*, en ella se debe agregar un condicional que si el campo fecha es vacío, cargue un icono de una cruz roja, y que habilite un campo que muestre: Dato vacío, al costado del lugar donde se ingresa la fecha.

Funciones que el cambio afecta:

- Función: *void btnIngresarUsuarioASistemaActionPerformed(...)*  
Clase: interfaz.VentanaRegistrarUsuario
- Función: *void btnIngresarProfesionalASistemaActionPerformed(...)*  
Clase: interfaz.VentanaRegistrarProfesional
- Función: *void mostrarErrores(...)*  
Clase: interfaz.VentanaRegistrarUsuario
- Función: *void mostrarErrores(...)*  
Clase: interfaz.VentanaRegistrarProfesional

	F1	F2	F3	F4	F5	F5	F6	F7	F8
Nivel de documentación									
Especificación de requerimientos									
Especificación de diseño	X	X							
Casos de prueba	X	X							
Código fuente	X	X							

Tabla 2.2: Análisis de impacto horizontal y vertical del cambio 2.

Estimación de esfuerzo: 8 horas

- **Cambio 3- Ingreso de campo de fecha de una determinada ingesta**  
Para este cambio se requiere la eliminación del objeto *JDateChooser* (calendario), y en la función: *void btnNuevaIngestaActionPerformed (...)* en lugar de tomar la fecha del calendario, se crea un nuevo objeto de la clase *GregorianCalendar*, y se obtiene la fecha del día de hoy.  
Además de esto, decidimos incluir en la lista de ingestas que ve el profesional cuando un usuario solicita un plan de alimentación, no solo el nombre de cada alimento ingerido, sino que también la fecha en que fue consumido. Para esto en la función *void listaPlanesPendientesValueChanged (...)* de la clase interfaz.VentanaMenuProfesional, se debería cargar en la lista de ingestas no solo el nombre de la misma, sino que también la fecha en que fue consumida.  
Una forma de probar esta modificación es que un usuario registre una ingesta y solicite un plan de alimentación. Luego al entrar desde el perfil de un profesional se debería ver el plan solicitado junto a cada alimento ingerido y la fecha en que fue consumido.

Funciones que el cambio afecta:

- Función: *Ingesta(String f, ArrayList -Alimento- LA)* (constructor)  
Clase: dominio.ingesta
- Funcion: *void setFechaDeIngesta(String unaFecha)*  
Clase: dominio.ingesta
- Funcion: *void listaPlanesPendientesValueChanged (...)*  
Clase: intefsz.VentanaMenuProfesional

Casos de prueba que afecta:

- Caso de prueba: *void testGetsSetsDatosValidosListaAlimentos()*  
Clase: IngestaTest
- Caso de prueba: *void testGetsSetsDatosValidosToString()*  
Clase: IngestaTest
- Caso de prueba: *void testEqualsIgualesLista()*  
Clase: IngestaTest
- Caso de prueba: *void testEqualsDistintosFecha()*  
Clase: IngestaTest

	F1	F2	F3	F4	F5	F5	F6	F7	F8
Nivel de documentación									
Especificación de requerimientos							X		
Especificación de diseño							X		
Casos de prueba							X		
Código fuente							X		

Tabla 2.3: Análisis de impacto horizontal y vertical del cambio 3.

Estimación de esfuerzo: 10 horas

#### ■ Cambio 4- Solo es posible subir al sistema imágenes con extensión png

Técnicamente este cambio consiste en modificar las funciones *void btnIngresarFotoPerfilActionPerformed(...)* y *private void btnIngresarFotoPerfilActionPerformed(...)*, en ellas hay solo un objeto del tipo *FileNameExtensionFilter*, creado para admitir imágenes con formato PNG. Para permitir otros tipos de archivo de imagen, debemos crear un objeto de tipo *FileNameExtensionFilter* para cada nueva extensión que deseemos permitir. En este caso se agregaran las extensiones: JPG y JPEG. Finalmente al objeto de tipo *JFileChooser* que es la ventana que nos permite seleccionar los archivos, le setearmos cada una de las extensiones nombradas anteriormente para que las acepte como formatos posibles, para ello utilizamos el método *setFileFilter* que recibe un objeto del tipo *FileNameExtensionFilter*. Realizando estos cambios lograremos que un usuario o un profesional puedan subir una foto de perfil en formatos: JPEG y

JPG, además de PNG.

Para probar este cambio, simplemente se intenta subir una imagen cuyo formato sea uno de los mencionados arriba, y debería permitir cargarla correctamente.

Funciones que el cambio afecta:

- Función: *void btnIngresarFotoPerfilActionPerformed(...)*  
Clase: *interfaz.VentanaRegistrarUsuario*
- Función: *private void btnIngresarFotoPerfilActionPerformed(...)*  
Clase: *interfaz.VentanaRegistrarUsuario*

Esta modificación, implica agregar un requerimiento no funcional que indique que los únicos formatos permitidos al adjuntar alguna imagen al sistema son: PNG, JPEG, JPG.

	F1	F2	F3	F4	F5	F5	F6	F7	F8
Nivel de documentación									
Especificación de requerimientos									
Especificación de diseño									
Casos de prueba	X	X					X		
Código fuente	X	X					X		

Tabla 2.4: Análisis de impacto horizontal y vertical del cambio 4.

Estimación de esfuerzo: 3 horas

#### ■ Cambio 5- Menú principal al ingresar al sistema como Profesional o Usuario

Para evitar la ventana vacía, tenemos que crear un nuevo panel con elementos como *labels* y *text areas* y *JList* en las clases *VentanaMenuPrincipalUsuario* y *VentanaMenuPrincipalProfesional* del paquete *interfaz*.

En cuanto al usuario, para tener un registro de los alimentos mas consumidos, es necesario llevar un contador en la clase *Alimento* en donde cada vez que el usuario ingiera un alimento, se incremente en una unidad el contador. Luego, simplemente seria necesario ordenar la lista de alimentos por numero de consumiciones decreciente en la clase *Sistema*.

De forma similar para llevar un registro de los profesionales que reciben mas consultas, es necesario llevar un contador de consultas en la clase *Profesional* que se incrementará cada vez que un paciente consulte al profesional. Finalmente en la clase *Sistema* se agrega un método que obtiene la lista de profesionales ordenada por cantidad de consultas decreciente.

Con respecto a las funcionalidades del profesional, se cargan 3 objetos de tipo *JList* a partir de métodos en la clase *Sistema* los cuales obtienen información

del profesional que esta logueado y filtran de la lista total de profesionales aquellos que tengan coincidencias con el mismo, estos son: los nacidos en el mismo país, los que tengan el mismo titulo profesional, y por ultimo, quienes se hayan graduado en el mismo año.

A su vez, a la función de la clase Sistema tenemos que crearle métodos de prueba (test). Funciones que el cambio afecta:

- Función: *VentanaMenuPrincipalProfesional(Sistema unSistema)* (constructor)  
Clase: interfaz.VentanaMenuPrincipalProfesional
- Función: *void btnConsultasPendientesActionPerformed(...)*  
Clase: interfaz.VentanaMenuPrincipalProfesional
- Función: *void btnPlanesSolicitadosActionPerformed(...)*  
Clase: interfaz.VentanaMenuPrincipalProfesional
- Función: *void btnIngresarAlimentoActionPerformed(...)*  
Clase: interfaz.VentanaMenuPrincipalProfesional

Casos de prueba que afecta:

- Introduce nuevos casos de prueba en Test Packages.

Este cambio implica agregar un nuevo requerimiento funcional al sistema, donde se indique que en el menú principal de un usuario o un profesional se debe mostrar información relevante de acuerdo al tipo de usuario logueado.

	F1	F2	F3	F4	F5	F5	F6	F7	F8
Nivel de documentación								X	
Especificación de requerimientos								X	
Especificación de diseño								X	
Casos de prueba								X	
Código fuente								X	

Tabla 2.5: Análisis de impacto horizontal y vertical del cambio 5.

Estimación de esfuerzo: 16 horas

#### ■ Cambio 6- Listas mostradas en pantalla en el inicio de sesión en caso de no haber registrado usuarios o profesionales

Este cambio consiste en que en caso de que no haya usuarios o profesionales registrados no se visualicen las listas vacías. Para lograr esto, debemos incorporar cada lista dentro de un *JPanel*, luego al iniciar la ventana, debemos consultar si la cantidad de usuarios o profesionales registrados es igual a cero, si eso es verdadero, deshabilitamos el/los paneles correspondientes haciendo uso del método: *setVisible (boolean aFlag)*, recibiendo como parámetro *false*. Luego de realizar esto, se habilita el/los textos que indican que no hay usuarios y/o profesionales registrados.

Para probar esta funcionalidad, es necesario abrir la aplicación sin que haya registros, en ese caso no deberían observarse las listas. Luego al registrar algún usuario o profesional, se tendría que visualizar la lista (de profesionales o usuarios, dependiendo de quien se haya registrado).

Funciones que el cambio afecta:

- Función: *VentanaMenuPrincipal(Sistema unSistema)* (constructor)  
Clase: interfaz.VentanaMenuPrincipal
- Función: *void listaUsuariosVentanaValueChanged(...)*  
Clase: interfaz.VentanaMenuPrincipal
- Función: *void listaProfesionalesVentanaValueChanged(...)*  
Clase: interfaz.VentanaMenuPrincipal

	F1	F2	F3	F4	F5	F5	F6	F7	F8
Nivel de documentación									
Especificación de requerimientos									
Especificación de diseño									X
Casos de prueba									X
Código fuente									X

Tabla 2.6: Análisis de impacto horizontal y vertical del cambio 6.

Estimación de esfuerzo: 11 horas

- **Cambio 7- Opción de salir de la aplicación desde todas las ventanas.**  
Para realizar el cambio tenemos que añadir un *JButton* en la parte superior derecha de todos los paneles y una función que llame al método de sistema que serializa los datos en cada ventana del paquete interfaz.

Funciones que el cambio afecta:

- Afecta todos los constructores del paquete interfaz.

Este cambio implica agregar un requerimiento funcional al sistema que indique que se podrá salir del mismo en cualquier ventana mediante el uso de un botón.

	F1	F2	F3	F4	F5	F5	F6	F7	F8
Nivel de documentación									
Especificación de requerimientos									
Especificación de diseño	X	X	X	X	X	X	X	X	X
Casos de prueba	X	X	X	X	X	X	X	X	X
Código fuente	X	X	X	X	X	X	X	X	X

Tabla 2.7: Análisis de impacto horizontal y vertical del cambio 7.

Estimación de esfuerzo: 6 horas

- **Mejorar la calidad del código**

Con mejorar la calidad del código nos referimos tanto a disminuir la cantidad de errores, *code smells* y vulnerabilidades presentes en el código, como a mantener la complejidad ciclomática promedio por clase en el Dominio.

Estimación de esfuerzo: 11 horas

	F1	F2	F3	F4	F5	F5	F6	F7	F8
Nivel de documentación									
Especificación de requerimientos									
Especificación de diseño									
Casos de prueba									
Código fuente	X	X	X	X	X	X	X	X	X

Tabla 2.8: Análisis de impacto horizontal y vertical de la mejora en la calidad del código.

- **Casos de prueba**

En cuanto a los casos de prueba, nos enfocaremos en aumentar tanto la cobertura como el porcentaje de aprobación de pruebas unitarias.

En referencia a pruebas de caja negra, realizaremos nuevos casos de prueba para cada una de las nuevas funcionalidades de la aplicación.

Estimación de esfuerzo: 35 horas.

	F1	F2	F3	F4	F5	F5	F6	F7	F8
Nivel de documentación	X	X	X	X	X	X	X	X	X
Especificación de requerimientos									
Especificación de diseño									
Casos de prueba	X	X	X	X	X	X	X	X	X
Código fuente	X	X	X	X	X	X	X	X	X

Tabla 2.9: Análisis de impacto horizontal y vertical del aumento y modificación de pruebas

## 2.4. Repositorio

Utilizaremos GitFlow, principalmente las ramas *Master*, *Develop* y *Feature*. Pudiendo ocasionalmente realizar algún *HotFix*, y últimamente un *Release*.

Manejaremos el repositorio desde *Sourcetree* (herramienta para manejar repositorios de manera gráfica) para tener una buena representación visual del mismo. Al inicio del proyecto, inicializamos *GitFlow* desde la opción en el menú de dicha herramienta.

Para una explicación sobre el funcionamiento de *GitFlow* dirigirse al anexo sección 4.0.5 - GitFlow



## 3. Plan de proyecto

### 3.1. Identificación de interesados

La identificación de interesados es uno de los procesos con más importancia dentro del plan de proyecto. Es muy importante tener a todos los interesados que se destacan por su nivel de poder, influencia o impacto dentro de la organización identificados y clasificados correctamente.

En el caso de nuestra aplicación, podrá ser usada por cualquier profesional, residente en algún país de América Latina y graduado en alguna de las áreas que se mencionan en la especificación de requerimientos del sistema, pudiendo trabajar en forma independiente o en una clínica junto a otros profesionales.

En el caso de los usuarios (pacientes), podrá ser cualquier persona residente en América Latina, y que tenga necesidad de consultar a un médico, o solicitar un plan de alimentación al mismo.

La gestión eficaz de los interesados del proyecto parte de la oportuna identificación y mantenimiento de un registro de los mismos, para lo cual el Gerente de proyectos cuenta con un instrumento que se denomina registro de los interesados.

En él se documenta información sobre los datos de contacto de cada uno de los interesados, sus requerimientos, expectativas, evaluación de su grado de influencia y clasificación.

En la figura 3.1, se observa el registro de interesados para este proyecto. En ella se observan datos acerca de los interesados como: su nombre, información relevante, su posición en la organización, el rol que cumplen en el proyecto, los requerimientos, es decir, lo que se espera de ellos, sus expectativas, y el nivel de influencia que tienen el proyecto (bajo, medio o alto).

Nombre	Posición	Rol	Información de contacto	Requerimientos	Expectativas	Influencia	Clasificación
Agustín Hermendorena	Programador	Desarrollador	<a href="mailto:agustinhernandorena@gmail.com">agustinhernandorena@gmail.com</a>	Desarrollar un sistema de calidad y que cumpla con los requerimientos especificados.	Que el software desarrollado sea de calidad y mantenible en el tiempo.	Media	Interno, flexible a cambios.
Luis Díaz	Ingeniero en Sistemas	Gerente de proyectos	<a href="mailto:luis.diaz@gmail.com">luis.diaz@gmail.com</a>	Cumplir con el Plan de Proyecto.	Que el proyecto sea culminado exitosamente.	Alta	Líder
Paula Areco	Programador	Desarrollador	<a href="mailto:pauareco@gmail.com">pauareco@gmail.com</a>	Desarrollar un sistema de calidad y que cumpla con los requerimientos especificados.	Que el software desarrollado sea de calidad y mantenible en el tiempo.	Media	Interno, flexible a cambios.
Arturo Pérez	Director ejecutivo en Sport Nutrition Laboratory	Sponsor	<a href="mailto:arturo.perez@snl.com">arturo.perez@snl.com</a>	Asegurar que los objetivos del proyecto están alineados con los objetivos de negocio.	Obtener clientes mediante anuncios en la aplicación.	Alta	Apoyo externo
Martín Rodríguez	Doctor en Medicina	Usuario (profesional)	<a href="mailto:martin.rodriguez@gmail.com">martin.rodriguez@gmail.com</a>	Aportar dinero y recursos que permitan el financiamiento del proyecto.	Que el sistema permite establecer una buena comunicación con el paciente.	Alta	Externo, con un conocimiento tecnológico nivel medio
Jorge Martínez	Comerciante	Usuario (paciente)	<a href="mailto:jorge.martinez78@hotmail.com">jorge.martinez78@hotmail.com</a>	Realizar un registro diario de las ingestas con el fin de obtener un plan de alimentación específico y adecuado.	Por medio del uso del sistema poder evacuar sus dudas con los profesionales, y solicitar planes de alimentación personalizados.	Alta	Externo, poco manejo de la tecnología.
Martínez González	Tester	Tester	<a href="mailto:martin.gonzalez@gmail.com">martin.gonzalez@gmail.com</a>	Probar eficientemente el software, con el fin de detectar errores y comunicarlos a tiempo.	Que el software desarrollado sea de calidad y con una baja cantidad de bugs.	Baja	Interno, flexible a cambios.
Raúl Giménez	Ingeniero en Sistemas	Encargado QA	<a href="mailto:raulgimenez93@gmail.com">raulgimenez93@gmail.com</a>	Realizar un conjunto de actividades con el objetivo asegurar la calidad de un software durante todas sus fases.	Que el producto obtenido sea de calidad, y se haya realizado siguiendo el estándar ISO 9001.	Media	Neutral, Interno

Figura 3.1: Planilla de registro de interesados.

En la figura 3.2 se observa una matriz compuesta por los interesados del proyecto, en donde se describe su compromiso actual y el deseado, el poder o grado de influencia que tiene y el interés que presenta. A partir de estos dos últimos parámetros, se determina una estrategia y nivel de comunicación a desarrollar con ese interesado.

MATRIZ DE INTERESADOS

Nombre del Proyecto: Alimentación	Director del Proyecto Luis Díaz					Fecha última actualización 15/4/2020		Versión 1.0
Interesado	Compromiso					Poder / Influencia	Interés	Estrategia
	D e s c o n o c e	S e r e s i s t e	N e u t r a l	A p o y a	L í d e r			
Agustín Hernandorena			X	D		B	B	Monitorear
Luis Díaz				X	D	A	A	Gestionar de cerca
Paula Areco			X	D		B	A	Informar
Arturo Pérez			X	D		A	B	Informar
Martín Rodríguez		X		D		A	A	Gestionar de cerca
Jorge Martínez	X			D		A	A	Gestionar de cerca
Ana Gomez		X	D			B	B	Monitorear
Martin Gonzalez			X	D				Informar
Raul Gimenez			X	D				Informar

Notas:  
X: Actual ; D: deseado  
A: Alto ; B: Bajo  
Estrategias: Gestionar de cerca (A-A); Mantener satisfecho (A-B); Informar (B-A); Monitorear (B-B)

Figura 3.2: Matriz de interesados.

En la figura 3.3 se observan las distintas acciones de comunicación que se llevan a cabo durante el desarrollo del proyecto. Para cada una de las acciones de comunicación, se especifica: el interesado, información (descripción y objetivos), canal de comunicación, frecuencia y destinatario.

Interesado	Información	Método	Frecuencia	Quién envía
Martin Gonzalez(tester)	Se le informa el tiempo necesario que va a tener que dedicar para realizar las pruebas.	Reunión	Al inicio del proyecto, y luego, cada vez que ocurra algun cambio en la planificación.	Encargado QA
Agustín Hernandorena (desarrollador 1)	Se le informa detalladamente el estado actual del proyecto en un archivo de texto. Se le enviará un nuevo reporte de avance al final de cada semana.	Tecnología de la comunicación - Google calendar	Semanalmente	Gerente de proyectos
Arturo Perez (sponsor)	Se le envía un informe acerca del alcance que estan teniendo los anuncios en la aplicacion.	Tecnología de la comunicación - correo electrónico	Semanalmente	Gerente de proyectos
Paula Areco (desarrollador 2)	Se le informa si se cumplieron los cambios propuestos la semana anterior y se le informa cuáles serán los cambios a implementar en el proyecto la siguiente semana.	Tecnología de la comunicación - correo electrónico	Semanalmente	Gerente de proyectos
Jorge Martínez (usuario - paciente)	Se le envía un comunicado en caso de que haya retrasos en la fecha de lanzamiento de la nueva versión del proyecto.	Tecnología de la comunicación - correo electrónico	Cuando sea necesario	Gerente de proyectos
Martín Rodríguez (usuario- profesional)	Se le informa la base teórica de las nuevas funcionalidades implementadas y se le pide un posible feedback.	Reunión	Semanalmente	Desarrollador

Figura 3.3: Planilla de comunicación.

## 3.2. Objetivos SMART

Para una explicación de sus siglas, dirigirse al anexo sección 4.0.6 - Objetivos Smart

### 3.2.1. Nivel de cobertura de pruebas

Aumentar la cobertura de las pruebas en un 10 % detectado por *JaCoCo* (desde su 62 % de cobertura actual) antes del 16/6/2020.

### 3.2.2. Nivel de pruebas aprobadas

Aumentar el porcentaje de pruebas unitarias aprobadas, es decir, las que el resultado esperado coincide con el resultado obtenido del 76 % actual a un 98 % antes del 16/6/2020. Se medirá ejecutando las pruebas unitarias al final del proyecto y comparando con el valor inicial.

### 3.2.3. Nivel de confiabilidad en el código

Disminuir la cantidad de *major bugs*, que permita pasar de un nivel de confiabilidad C (nivel actual) a uno mayor o igual que B, considerando la escala de *SonarQube* antes del 8/6/2020.

### 3.2.4. Complejidad ciclomática promedio

Mantener la complejidad ciclomática promedio por clase en el 18.5 detectado inicialmente por *JaCoCo*, para ello, se ejecutará el software *JaCoCo* cada vez que se introduzca un cambio en el código.

### 3.2.5. Cantidad de clics al registrar un profesional.

Disminuir la cantidad de clics promedio al hacer uso del calendario en la ventana registro de profesional en al menos 18 clics (cantidad de clics actual es mayor a 18) antes del 16/6/2020.

### 3.2.6. Cantidad de clics al salir de la aplicación

Disminuir la cantidad de clics necesarios para salir de la aplicación a 1 clic (valor actual es 2 clics) antes del 16/6/2020.

## 3.3. Alcance del producto y del proyecto

En esta sección se documentan las necesidades de los interesados para convertirlas en requisitos del proyecto.

Para todos los requisitos el nombre del proyecto es "mantenimiento de la aplicación Alimentación Saludable" y su localización es Uruguay.

Todos los requisitos de proyecto tienen el mismo objetivo: Realizar un mantenimiento general de la aplicación Alimentación Saludable, garantizando un nivel de cobertura mayor al actual y una mejora en el nivel de confiabilidad del código manteniendo la complejidad ciclomática promedio por clase en el momento de introducir los cambios establecidos en la solicitud de cambios.

### 3.3.1. Requisitos del proyecto

#### RP1: Cobertura de pruebas

- **Descripción:** Se deberá aumentar el porcentaje de cobertura de las pruebas unitarias.
- **Justificación:** Garantizar el correcto funcionamiento del código.
- **Solicitante/Interesado:** Encargado QA.
- **Criterio de aceptación:** Será aceptable cuando la cobertura de las pruebas unitarias llegue a un 72 %.

#### RP2: Aumento de pruebas unitarias exitosas

- **Descripción:** Se deberá aumentar la cantidad de pruebas exitosas.

- **Justificación:** Verificar el correcto funcionamiento de cada uno de los módulos del sistema de forma independiente.
- **Solicitante/Interesado:** Encargado QA.
- **Criterio de aceptación:** Se considerará aceptable cuando el porcentaje de pruebas unitarias exitosas sea mayor o igual al 98 %.

### **RP3: Mantenimiento de la complejidad ciclomática promedio**

- **Descripción:** Se deberá mantener la complejidad ciclomática promedio de las clases del dominio.
- **Justificación:** No aumentar la complejidad lógica del sistema con el objetivo de no incrementar el riesgo de cambio en la etapa de mantenimiento.
- **Solicitante/Interesado:** Desarrollador.
- **Criterio de aceptación:** Será aceptable cuando la complejidad ciclomática promedio de las clases del dominio, sea menor o igual a 18.5.

### **RP4: Disminución de defectos encontrados**

- **Descripción:** Se deberá reducir la cantidad de bugs encontrados mediante el uso de herramientas analizadoras de código.
- **Justificación:** Es importante porque lleva a reducir la cantidad de instancias en las que utilizando el sistema se desencadena un resultado indeseado.
- **Solicitante/Interesado:** Encargado QA.
- **Criterio de aceptación:** Será aceptable cuando el nivel de confiabilidad sea B, considerando la escala de *SonarQube*.

### **RP5: Aumento en la cantidad de casos de prueba**

- **Descripción:** Aumentar la cantidad de casos de prueba, introduciendo casos tanto para las nuevas funcionalidades indicadas en la solicitud de cambios, como para las funcionalidades ya existentes en el sistema.
- **Justificación:** Garantizar que las funcionalidades que provee el sistema funcionan en forma adecuada, esto es, que cumplen lo que se describe en la especificación de requerimiento en la mayoría de los escenarios posibles.
- **Solicitante/Interesado:** Gerente de proyecto.
- **Criterio de aceptación:** Debe haber al menos 1 caso de prueba para cada funcionalidad nueva del dominio del sistema. Se considera aceptable cuando la misma esta correctamente documentada con los elementos: entrada y salida esperada como también una sesión de prueba asociada a la misma, que cuenta con los elementos: fecha, versión, duración, entorno específico y resultado.

### 3.3.2. Requisitos del producto

#### RF1: Reemplazar objeto Calendario

- **Descripción:** Se reemplazará el objeto calendario de tipo *dateChooser* por uno de tipo *JDateChooser*.
- **Justificación:** Con este cambio se mejorará la usabilidad en el sistema reduciendo la cantidad de clics que el usuario debe realizar a la hora de seleccionar la fecha de graduación o la fecha de nacimiento, ya que podrá introducir el dato directamente desde el teclado.
- **Solicitante/Interesado:** Cliente.
- **Criterio de aceptación:** Se considerará aceptable cuando la cantidad de clics que tenga que hacer un usuario al momento de elegir una fecha del calendario sea al menos 18 clics menos que en la actualidad.

#### RF2: Corrección al no ingresar fecha de nacimiento/graduación en el registro.

- **Descripción:** En caso de que el usuario deje alguno de los campos fecha de graduación o fecha de nacimiento vacíos, se debe notificar al usuario, y no permitir el registro.
- **Justificación:** Este cambio es necesario, porque se trata de un error de funcionamiento en el sistema. No se debería permitir registrar a un usuario que no proporcione su fecha de nacimiento o graduación, y menos, tomar por defecto que su fecha de nacimiento o graduación es la del día del momento del registro sin avisarle en ningún momento.
- **Solicitante/Interesado:** Cliente.
- **Criterio de aceptación:** Será aceptable cuando el sistema no permita registrar un paciente o profesional que no haya seleccionado una fecha de nacimiento o graduación respectivamente.

#### RF3: Cambio en la fecha de ingesta de un usuario.

- **Descripción:** Consiste en que a la hora de registrar una ingesta de un alimento, en lugar de pedirle al usuario que seleccione manualmente la fecha desde un calendario, se asuma que si está ingresando una ingesta, la misma ocurrió ese día, y se tome la fecha directamente desde la computadora.
- **Justificación:** Este cambio es necesario, porque se incentiva a que el usuario utilice la aplicación de forma diaria, y que no acumule una gran cantidad de ingestas de días anteriores. Consideramos que esta modificación implicará que el registro de ingestas sea preciso y acorde a la realidad del paciente.
- **Solicitante/Interesado:** Cliente.

- **Criterio de aceptación:** Se considerara aceptable cuando el usuario no deba ingresar desde un calendario la fecha de ingesta de un alimento y que el sistema la tome correctamente como la fecha del día.

#### **RF4: Modificación en el menú principal al ingresar como Profesional o Usuario.**

- **Descripción:** Consiste en realizar cambios al ingresar al menú principal tanto de un paciente como de un profesional. El paciente podrá visualizar un listado con los alimentos que mas ha consumido, como también, un listado con los profesionales que reciben mas consultas.  
En cuanto al profesional, podrá visualizar una serie de listas, donde cada una de ellas contiene profesionales que tienen alguna coincidencia con el profesional logueado, estas coincidencias son: haber nacido en el mismo país, poseer el mismo titulo, y por ultimo, haberse graduado en el mismo año.
- **Justificación:** La importancia de este cambio radica en que la ventana de inicio es lo primero que observa el paciente/profesional, y si queremos captar la atención del mismo necesitamos mostrar información de forma dinámica y que se actualice en base al uso que se le da.
- **Solicitante/Interesado:** Cliente.
- **Criterio de aceptación:** Será aceptable cuando la funcionalidad sea aceptada por el encargado de QA del proyecto.

#### **RF5: Poder salir de la aplicación desde cualquier ventana.**

- **Descripción:** Consiste en colocar un botón en cada una de las ventanas que permita salir de la aplicación por intermedio de cualquiera de ellas.
- **Justificación:** La importancia de este cambio radica en que actualmente el usuario para salir de la aplicación tiene que ir desde la ventana que se encuentra en ese momento hasta la ventana principal que es el único lugar donde esta el botón para salir, lo que en promedio, le lleva tres clics. A partir de esta modificación el usuario podrá salir del sistema sin importar la ventana en la que se encuentre, con tan solo un clic.
- **Solicitante/Interesado:** Gerente de proyecto.
- **Criterio de aceptación:** Se considerará aceptable cuando el sistema permita cerrar la aplicación desde todas las ventanas.

#### **RNF1: Ampliación de formatos de imágenes permitidos**

- **Descripción:** Consiste en permitir que a la hora del registro de un profesional, usuario o alimento, se pueda adjuntar una imagen no solo en formato *PNG*, sino que se admitan mas formatos como: *JPEG* y *JPG*.
- **Justificación:** Este cambio es sustancial ya que gran parte de los usuarios utilizan otros formatos que no son *PNG*, principalmente *JPG* y *JPEG*.

- **Solicitante/Interesado:** Gerente de proyecto.
- **Criterio de aceptación:** Se considerará aceptable cuando el sistema permita al registrar un usuario o alimento, adjuntar una imagen en formato *JPG* o *JPEG*.

#### RNF2: Ocultar listas vacías.

- **Descripción:** Actualmente cuando hay una lista vacía, el sistema muestra un espacio para ella pero sin datos. El cambio consiste en que si la lista no contiene datos, el sistema la oculte, y muestre un mensaje en su lugar.
- **Justificación:** Este cambio es importante porque mejora la navegabilidad del usuario en la aplicación, en el sentido que es más claro mostrar un mensaje que indique que no hay elementos en la lista, que desplegar una lista vacía, en donde el usuario puede confundirse realizando suposiciones tales como que hay un error interno en el sistema o que los datos nos fueron cargados correctamente.
- **Solicitante/Interesado:** Gerente de proyecto.
- **Criterio de aceptación:** Se considerara aceptable cuando el sistema oculte todas las listas vacías al usuario

### 3.3.3. Matriz de trazabilidad de requisitos

#	Descripción	Fecha	Solicitado por	Prioridad	Estado	Entregable	Criterio de aceptación	Responsable
RP1	Aumentar porcentaje de cobertura de pruebas	22/4/2020	Raul Gimenez	Alta	Aprobado	1.1.1	Cobertura de pruebas mayor o igual a 72 %.	Raul Gimenez
RP2	Aumentar cantidad de pruebas exitosas	22/4/2020	Raul Gimenez	Alta	Aprobado	1.2.1	Porcentajes de pruebas unitarias exitosas mayor o igual a 98 %	Raul Gimenez
RP3	Mantener complejidad ciclomatica promedio por clase	22/4/2020	Paula Areco	Alta	Aprobado	2.1.1	Complejidad ciclomatica promedio por clase menor o igual a 18.5	Paula Areco
RP4	Reducir cantidad de bugs	22/4/2020	Raul Gimenez	Media	Aprobado	2.2.1	Nivel de confiabilidad = B	Raul Gimenez
RP5	Aumentar cantidad casos de prueba	22/4/2020	Luis Diaz	Media	Aprobado	3.1.2	Al menos un caso de prueba para cada funcionalidad agregada	Paula Areco
RF1	Reemplazar objeto calendario dateChooser por un JDateChooser	22/4/2020	Martin Rodriguez	Media	Aprobado	4.1.1	Disminucion de al menos 18 clics al hacer uso de esta funcion.	Agustin H.
RF2	Impedir registro al dejar campo fecha vacio	22/4/2020	Martin Rodriguez	Alta	Aprobado	4.2.1	Cuando el sistema no permita el registro, cuando se deja el campo fecha vacio.	Paula Areco

Figura 3.4: Matriz trazabilidad requisitos (1)



RF3	Asumir fecha de ingesta como la del día de hoy	22/4/2020	Martin Rodriguez	Baja	Aprobado	6.4.1.1	Cuando el sistema tome como fecha de ingesta la del día en que se esta registrando la misma.	Agustin H.
RNF1	Agregar como formatos de imágenes permitidos a: JPG y JPEG	22/4/2020	Luis Diaz	Media	Aprobado	5.1.1	Cuando el sistema permita subir una imagen cuya extension es JPG o JPEG	Agustin H.
RF4	Mostrar datos relevantes al ingresar al menu principal de paciente o profesional	22/4/2020	Martin Rodriguez	Media	Aprobado	6.3.1.1	Cuando al ingresar al menu principal de paciente o profesional el sistema muestre datos relevantes	Paula Areco
RNF2	Ocultar listas vacias	22/4/2020	Luis Diaz	Baja	Aprobado	6.2.1.1	Cuando el sistema no muestre listas que no contienen elementos	Agustin H.
RF5	Poder salir de la aplicación desde cualquier ventana	22/4/2020	Luis Diaz	Alta	Aprobado	6.1.1.1	Cuando el sistema permita salir y terminar la ejecucion desde cualquiera de sus ventanas	Paula Areco

Figura 3.5: Matriz trazabilidad requisitos (2)

### 3.3.4. Enunciado del Alcance

Nombre del proyecto: Mantenimiento de aplicación Alimentación Saludable

Fecha última actualización: Mayo 2020

Preparado por: Luis Pérez (gerente de proyecto)

**i. Breve descripción del proyecto:** Alimentación Saludable es una aplicación orientada a la comunicación entre usuarios (pacientes) que solicitan planes de alimentación a profesionales y estos últimos los otorgan. El proyecto de mantenimiento consiste en mejorar el estado actual de la aplicación en cuanto a usabilidad, cobertura y aceptación de pruebas unitarias, mejoras en cuanto a corrección de errores en el código y cumplimiento de estándares de codificación, como también introducir nuevas funcionalidades.

**ii. Alcance del producto:** 72 % de cobertura de pruebas detectado por *JaCoCo*, 98 % de aceptación de pruebas unitarias, mantener la complejidad ciclomática por clase, disminuir la cantidad de *major bugs* desde C a B detectada por *SonarQube*, introducir el botón para cerrar la aplicación desde todas las ventanas, incluir la extensiones .JPG y .JPEG al momento de subir imágenes al sistema, mostrar a los pacientes estadísticas de los alimentos que más ha ingerido y los profesionales más consultados, mostrar a los profesionales los otros profesionales que hayan nacido en el mismo país y que se hayan graduado en el mismo año como también los que compartan su mismo título, ocultar la visibilidad de listas cuando se encuentran vacías, mejorar el uso del calendario y la elección de fechas.

**iii. Entregables:** Imágenes, documentos PDF, presentaciones multimedia. Estructura de Desglose del Trabajo (EDT), cronograma, plan de gestión de calidad, matriz de roles y responsabilidades. Aplicación con nuevas funcionalidades y bugs

corregidos.

**iv. Criterios de aceptación:** Aplicación funcionando sin errores cumpliendo con los criterios mencionados en el alcance del producto.

**v. Exclusiones:** diseño *responsive* que se adapte al dispositivo.

**vi. Supuestos:** La fecha de la entrega no puede ser aplazada bajo ninguna circunstancia.

**vii. Restricciones:** La aplicación es desarrollada en lenguaje Java.

**viii. Riesgos preliminares identificados:** Surgimiento de nuevas versiones de *Java* que no corran en nuestras máquinas, obtener *feedbacks* tardíos por parte del cliente, falta de tiempo, falta de presupuesto.

**ix. Requisitos de aprobación:** El Gerente de proyecto será quien apruebe todas las actividades relacionadas al mantenimiento de la aplicación. Todo cambio debe tener la firma del mismo para poder llevarse a cabo.

### 3.3.5. EDT

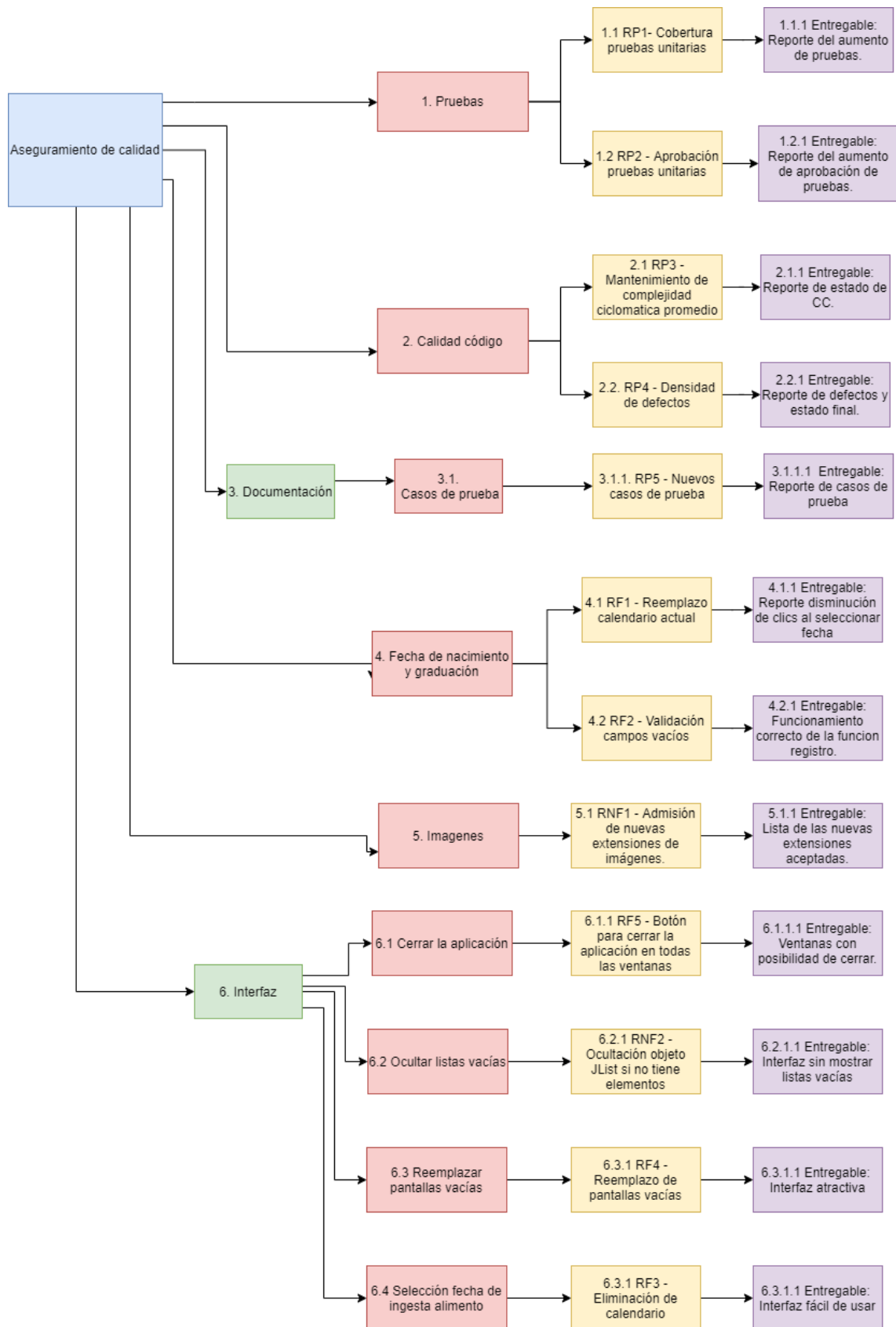


Figura 3.6: Esquema EDT definido en base a los Entregables Principales

### 3.3.6. Diccionarios de la EDT

ID #	Cuenta control #	Ultima Actualización	Responsable
1.1.1	1	22-4-2020	Raúl Gimenez
Descripción: Aumento de la cobertura de pruebas unitarias.			
Criterio de aceptación: ver 3.3.1 - RP1			
Entregables: Presentación dando evidencia del resultado obtenido mediante el uso de la herramienta JacoCo.			
Supuestos: -			
Recursos asignados: 1 tester.			
Duración: 10 horas.			
Hitos: 1. 21 mayo: Informe preliminar aprobado por el Project Manager.			
<b>Costo:</b> \$U 1548			

ID #	Cuenta control #	Ultima Actualización	Responsable
1.2.1	1	22-4-2020	Raúl Gimenez
Descripción: Aumentar la cantidad de pruebas unitarias que pasan correctamente de un 62 % actual a un 98 %.			
Criterio de aceptación: ver 3.3.1 - RP2.			
Entregables: Informe PDF con capturas de pantalla del reporte dado por el IDE al momento de correr las pruebas unitarias.			
Supuestos: -			
Recursos asignados: 1 tester.			
Duración: 10 horas.			
Hitos: 16 de mayo: Presentación preliminar frente al encargado de QA del proyecto.			
<b>Costo:</b> \$U 1891			

ID #	Cuenta control #	Ultima Actualización	Responsable
2.1.1	2	22-4-2020	Paula Areco
Descripción: Mantener la complejidad ciclomática por clase de la aplicación al introducir las nuevas funcionalidades establecidas en la solicitud de cambios y al mantener el código.			
Criterio de aceptación: ver 3.3.1 - RP3.			
Entregables: Imagen en formato JPG que muestre la CC detectada por la herramienta JaCoCo al iniciar el mantenimiento del código y al finalizar el mismo.			
Supuestos: -			
Recursos asignados: 2 desarrolladores.			
Duración: 3 horas.			
Hitos: 7 de junio: Presentación preliminar frente al encargado de QA del proyecto. .			
<b>Costo:</b> \$U 1306			

ID #	Cuenta control #	Ultima Actualización	Responsable
<b>2.2.1</b>	<b>2</b>	<b>22-4-2020</b>	<b>Raúl Gimenez</b>
Descripción: Disminuir la cantidad de major bugs detectados por SonarQube en el código, pasando de un estado actual C a un estado B.			
Criterio de aceptación: ver 3.3.1 - RP4.			
Entregables: Imagen en formato JPG (captura de SonarQube) que muestre el estado inicial de la aplicación y el estado final.			
Supuestos: -			
Recursos asignados: 2 desarrolladores.			
Duración: 8 horas.			
Hitos: 16 de mayo: Presentación del estado inicial en cuanto a bugs frente al encargado de QA en cuanto a bugs.			
18 de mayo: Presentar estado actual y comparación con estado inicial al Project Manager.			
<b>Costo:</b> \$U 3000			

ID #	Cuenta control #	Ultima Actualización	Responsable
<b>3.1.1.1</b>	<b>3</b>	<b>22-4-2020</b>	<b>Paula Areco</b>
Descripción: Agregar nuevos casos de prueba.			
Criterio de aceptación: ver 3.3.1 - RP5.			
Entregables: Documento PDF con todas las nuevas pruebas introducidas.			
Supuestos: Todas las funcionalidades que fueron entregadas inicialmente junto con el proyecto tienen un caso de prueba asociado.			
Recursos asignados: 1 tester.			
Duración: 15 horas.			
Hitos: 12 de junio- Presentación de Documento PDF que contiene las pruebas, al encargado de QA.			
<b>Costo:</b> \$U 1755			

ID #	Cuenta control #	Ultima Actualización	Responsable
<b>4.1.1</b>	<b>4</b>	<b>22-4-2020</b>	<b>Agustín Hernandorena</b>
Descripción: Disminuir la cantidad de clics al seleccionar una fecha de nacimiento en el registro de un paciente y en el registro de un profesional.			
Criterio de aceptación: ver 3.3.2 - RF1			
Entregables: presentación con multimedia e informe en formato PDF			
Supuestos: -			
Recursos asignados: 2 desarrolladores, 1 tester.			
Duración: 11 horas.			
Hitos: 1 de junio: Presentación en forma de vídeo al cliente que muestra la mejora en la cantidad de clics.			
<b>Costo:</b> \$U 4859			

ID #	Cuenta control #	Ultima Actualización	Responsable
<b>4.2.1</b>	<b>4</b>	<b>22-4-2020</b>	<b>Paula Areco</b>
Descripción: Validar que el campo de la fecha no esté vacío al momento del registro de un paciente o profesional.			
Criterio de aceptación: ver 3.3.2 - RF2.			
Entregables: presentación con capturas de pantalla e informe en formato PDF			
Supuestos: -			
Recursos asignados: 1 desarrollador, 1 tester.			
Duración: 8 horas.			
Hitos: 4 de junio: Presentación de la funcionalidad al cliente, mostrando el ejecutable validando correctamente.			
<b>Costo:</b> \$U 2109			

ID #	Cuenta control #	Ultima Actualización	Responsable
<b>5.1.1</b>	<b>5</b>	<b>22-4-2020</b>	<b>Agustín Hernandorena</b>
Descripción: Permitir las extensiones .JPG y .JPEG en el momento de subir una imagen al sistema, cuando se registra un usuario o alimento.			
Criterio de aceptación: ver 3.3.2 - RNF1.			
Entregables: Documento PDF con 3 pruebas exploratorias que muestran la aceptación de los nuevos formatos.			
Supuestos: -			
Recursos asignados: 1 desarrollador, 1 tester.			
Duración: 3 horas.			
Hitos: 2 de junio - Presentación del entregable al cliente con una demo de la aceptación de nuevas extensiones.			
<b>Costo:</b> \$U 1375			

ID #	Cuenta control #	Ultima Actualización	Responsable
<b>6.1.1.1</b>	<b>6</b>	<b>22-4-2020</b>	<b>Paula Areco</b>
Descripción: Permitir cerrar la aplicación desde todas las ventanas.			
Criterio de aceptación: ver 3.3.2 - RF5.			
Entregables: Documento PDF con capturas de todas las ventanas del sistema con el botón de cerrar.			
Supuestos: -			
Recursos asignados: 2 desarrolladores.			
Duración: 7 horas.			
Hitos: 6 de junio: Presentación al cliente con capturas que muestren que es posible salir de la aplicación desde cualquier ventana.			
<b>Costo:</b> \$U 3375			

ID #	Cuenta control #	Ultima Actualización	Responsable
6.2.1.1	6	22-4-2020	Agustín Hernandorena
Descripción: Ocultar todas las listas que no contengan ningún elemento (todas las listas vacías) de la interfaz de usuario.			
Criterio de aceptación: ver 3.3.2 - RNF2.			
Entregables: Documento PDF con capturas de todas las ventanas del sistema que tengan listas.			
Supuestos: -			
Recursos asignados: 2 desarrolladores, 1 encargado de QA.			
Duración: 11 horas.			
Hitos: 24 de mayo- Presentación del entregable al cliente.			
<b>Costo:</b> \$U 3181			

ID #	Cuenta control #	Ultima Actualización	Responsable
6.3.1.1	6	22-4-2020	Paula Areco
Descripción: Reemplazar la pantalla del inicio que actualmente está vacía, por una que muestre estadísticas dependiendo de quién ingrese al sistema (paciente o profesional).			
Criterio de aceptación: ver 3.3.2 - RF4.			
Entregables: Archivo PDF que contenga dos imágenes <i>JPG</i> que muestren la pantalla principal.			
Supuestos: -			
Recursos asignados: 1 desarrollador, 1 <i>tester</i> , 1 encargado de QA.			
Duración: 14 horas.			
Hitos: 26 de mayo - Presentación del entregable al cliente.			
<b>Costo:</b> \$U 4165			

ID #	Cuenta control #	Ultima Actualización	Responsable
6.4.1.1	6	22-4-2020	Agustín Hernandorena
Descripción: Eliminar la opción de que el usuario deba ingresar la fecha del alimento consumido, tomar la fecha del día como la fecha de ingesta.			
Criterio de aceptación: ver 3.3.2 - RF3.			
Entregables: Presentación multimedia que muestre la funcionalidad de ingreso de ingesta de un alimento.			
Supuestos: -			
Recursos asignados: 2 desarrolladores, 1 tester.			
Duración: 10 horas.			
Hitos: 29 de mayo - Presentación del entregable al cliente.			
<b>Costo:</b> \$U 3915			

### 3.4. Estimación del esfuerzo

Para estimar el esfuerzo total del proyecto, nos basamos en el concepto *hora-hombre*, es decir, en el cálculo de estimación de cantidad de esfuerzo humano que

puede realizar un trabajador promedio en una actividad durante una hora. Para realizar estas estimaciones en cada una de las actividades del proyecto, tuvimos en cuenta aspectos diferentes, en aquellas que refieren a la mejora de la calidad del código en cuanto a bugs, nos basamos en el tiempo que estima la herramienta *Sonar Qube*.

En cuanto a las actividades asociadas a los cambios/funcionalidades solicitadas, nos basamos en la experiencia que tenemos de otros proyectos similares, en particular, en aquellos cuyo objetivo era desarrollar una aplicación de escritorio utilizando el lenguaje *Java*.

En la tabla 3.1, se puede observar la estimación de esfuerzo en cantidad de horas-hombre para cada actividad del proyecto.

Si sumamos el esfuerzo de cada actividad, obtenemos que el total del mismo será de **113** horas-hombre.

Actividad	Hora-hombre
Aumento cantidad de pruebas pasadas	10
Disminución de bugs	8
Aumento cobertura de pruebas	10
Ocultar listas vacías	11
Reemplazar pantallas vacías	16
Eliminar calendario (ingreso ingesta)	10
Reemplazar calendario actual	11
Validar campos vacíos	8
Admitir nuevas extensiones imágenes	4
Cerrar aplicación en todas las ventanas	7
Complejidad ciclomatica	3
Nuevos casos de prueba	15

Tabla 3.1: Cantidad de horas-hombre estimadas para cada actividad.

### 3.5. Especificación del ciclo de vida

El ciclo de vida que decidimos utilizar será el iterativo incremental. Al inicio hacemos el relevamiento y análisis e iteramos sobre el diseño, construcción y pruebas.

Es una unión del incremental con el iterativo, por una parte tenemos al incremental que se encarga de desarrollar por partes el producto de software para luego integrarlas con el resto del sistema, y por otra parte en cada ciclo o iteración se hace una revisión para mejorar el producto, en cada ciclo se mejora la calidad de la aplicación. La unión de ambas nos dan como resultado un software con mayor calidad y nuevas funcionalidades, que es esencialmente el objetivo del proyecto de mantenimiento de la aplicación "Alimentación Saludable".

De esta forma podemos a lo largo del mantenimiento tener un *feedback* temprano del cliente, con entregables que muestren un avance del proyecto y el estado en el que está el mismo. Como sabemos que en cualquier momento el cliente puede pedir



nuevos cambios, si implementáramos un ciclo de vida en cascada por ejemplo, se nos dificultaría volver a etapas anteriores en caso de que el cliente solicite una nueva funcionalidad y nosotros ya hayamos terminado la etapa de la especificación de los requerimientos.

## 3.6. Cronograma de trabajo

Para realizar el cronograma de trabajo utilizamos la herramienta *Ganttter*. En ella, definimos en primer lugar un calendario personalizado que se adapte a la disponibilidad horaria del equipo. De lunes a viernes el equipo trabajara un total de 3 horas diarias en el proyecto (excepto los días jueves que se trabajara 2 horas). Considerando que el equipo tendrá mas disponibilidad horaria durante el fin de semana, se trabajara 5 horas los días sábados, y 4 horas los domingos.

Para las actividades del cronograma, definimos un orden de precedencia basándonos en aquellas tareas cuya finalización es imprescindible para iniciar otras. Para ello, decidimos priorizar las tareas de: aumentar la cobertura de pruebas unitarias, aumentar la cantidad de pruebas exitosas y disminuir la cantidad de *bugs* frente a las tareas relacionadas a desarrollar las nuevas funcionalidades o cambios solicitados.

Con respecto a estas ultimas, también establecimos un orden de precedencia entre ellas, decidimos priorizar aquellas tareas de mayor riesgo, porque consideramos mas adecuado realizar al inicio del proyecto, en donde, si es necesario se puede llegar a ajustar el cronograma, aquellas tareas de mayor complejidad y que pueden tener riesgos asociados, que dejarlas para realizarlas al final del proyecto, en donde los tiempos y el margen de error son muy acotados.

Otro aspecto que tuvimos en cuenta a la hora de diseñar el calendario, es identificar aquellas tareas que pueden realizarse en paralelo con otras, y las que no, centrándonos principalmente en evaluar la dificultad de cada tarea, determinando si es necesaria o no la intervención de ambos miembros del equipo en una misma tarea.

### 3.6.1. Lista de tareas

Para ver en conjunto la lista de tareas y el gráfico Ganttter adjuntamos otro archivo en la carpeta de la entrega.

	1	Nombre	Duración	Coste	Inicio	Fin	Predecesoras	Recursos
1		1.2.1 - Aumento cantidad pruebas pasadas.	2días?	\$1891	05/13/2020	05/16/2020		
2		Realizar un inventario de las pruebas que no pasan.	1hora?	\$117	05/13/2020	05/13/2020		Martin Gonzalez
3		Identificar el origen de los errores de las pruebas.	4horas?	\$468	05/13/2020	05/14/2020	2	Martin Gonzalez
4		Implementar los cambios necesarios para que las pruebas pasen correctamente.	4horas?	\$1000	05/15/2020	05/16/2020	3	Paula Areco
5		Documentar	1hora?	\$306	05/16/2020	05/16/2020	4	Raul Gimenez
6		Presentar informe preliminar frente a encargado QA del proyecto	0hora?	\$0	05/16/2020	05/16/2020	5	Martin Gonzalez,F
7		2.2.1 - Disminución de bugs	1día?	\$3000	05/16/2020	05/18/2020		
8		Realizar un inventario de los bugs detectados por SonarQube que se van a reparar.	2horas?	\$500	05/16/2020	05/16/2020	1	Agustin Hernando
9		Presentar estado inicial del producto en cuanto a bugs al encargado QA	0hora?	\$0	05/16/2020	05/16/2020	8	Agustin Hernando
10		Reparar los bugs seleccionados.	4horas?	\$2000	05/16/2020	05/17/2020	8	Agustin Hernando
11		Correr el analizador SonarQube, y verificar si se alcanzó el grado esperado.	1hora?	\$250	05/17/2020	05/17/2020	10	Agustin Hernando
12		Documentar	1hora?	\$250	05/18/2020	05/18/2020	11	Agustin Hernando
13		Presentar estado actual y comparacion con estado inicial al Project Manager	0día?	\$0	05/18/2020	05/18/2020	12	Agustin Hernando
14		1.1.1 Aumento de pruebas	3días?	\$1548	05/18/2020	05/21/2020		
15		Realizar un inventario de las pruebas unitarias existentes y las faltantes.	2horas?	\$612	05/18/2020	05/18/2020	7	Raul Gimenez
16		Codificar pruebas faltantes.	8horas?	\$936	05/19/2020	05/21/2020	15	Martin Gonzalez
17		Presentar informe preliminar a Project Manager	0día?	\$0	05/21/2020	05/21/2020	16	Luis Diaz,Martin C
18		6.2.1.1 - Ocultar listas vacías si no hay elementos.	0día?	\$3181	05/22/2020	05/24/2020		
19		Realizar un inventario de las listas que pueden llegar a estar vacías.	2horas?	\$500	05/22/2020	05/22/2020	16	Agustin Hernando
20		Diseñar la solución.	2horas?	\$500	05/22/2020	05/23/2020	19	Agustin Hernando
21		Implementar los cambios.	4horas?	\$1000	05/23/2020	05/23/2020	20	Agustin Hernando
22		Revisar el cumplimiento de estándares de codificación.	1hora?	\$306	05/24/2020	05/24/2020	21	Raul Gimenez
23		Documentar.	1hora?	\$250	05/24/2020	05/24/2020	22	Agustin Hernando
24		Preparar informe de avance para presentar al cliente	1hora?	\$625	05/24/2020	05/24/2020	23	Luis Diaz
25		Presentar entregable al cliente	0día?	\$0	05/24/2020	05/24/2020	24	Luis Diaz
26		6.3.1.1 - Reemplazo de pantallas vacías.	2días?	\$4165	05/22/2020	05/26/2020		

Figura 3.7: Lista de actividades parte 1

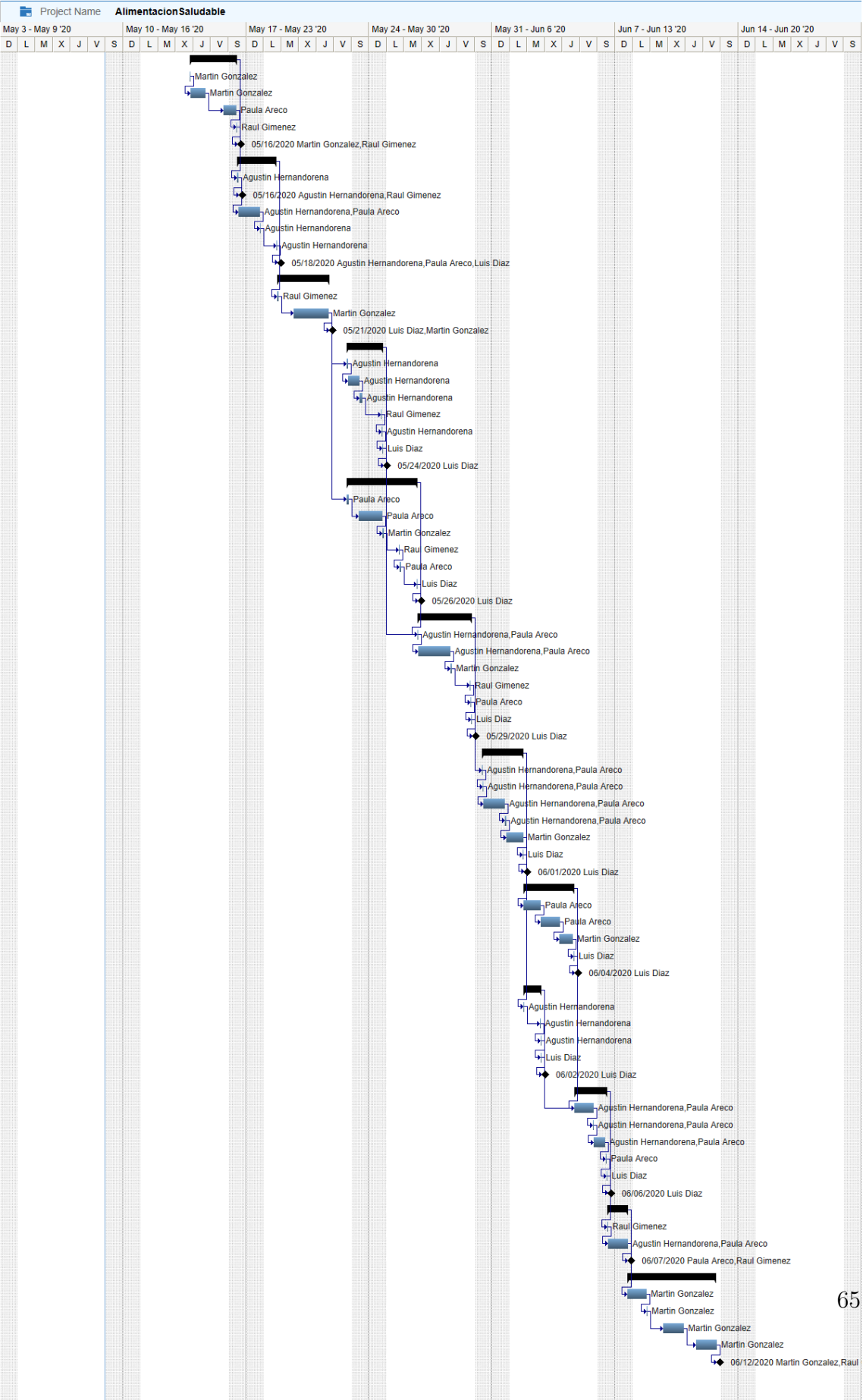
26		6.3.1.1 - Reemplazo de pantallas vacías.	2días?	\$4165	05/22/2020	05/26/2020		
27		Diseñar la solución.	3horas?	\$750	05/22/2020	05/22/2020	16	Paula Areco
28		Implementar cambios en dominio e interfaz.	7horas?	\$1750	05/23/2020	05/24/2020	27	Paula Areco
29		Codificar pruebas unitarias.	2horas?	\$234	05/24/2020	05/24/2020	28	Martin Gonzalez
30		Revisar el cumplimiento de estándares de codificación.	1hora?	\$306	05/25/2020	05/25/2020	29	Raul Gimenez
31		Documentar.	2horas?	\$500	05/25/2020	05/25/2020	30	Paula Areco
32		Preparar informe de avance para el cliente	1hora?	\$625	05/26/2020	05/26/2020	31	Luis Diaz
33		Presentar entregable al cliente	0día?	\$0	05/26/2020	05/26/2020	32	Luis Diaz
34		6.4.1.1 - Eliminación de calendario (ingreso ingesta)	3días?	\$3915	05/26/2020	05/29/2020		
35		Investigar la clase Local Date de Java.	1hora?	\$500	05/26/2020	05/26/2020	18,26	Agustin Hernando
36		Implementar los cambios.	4horas?	\$2000	05/26/2020	05/28/2020	35	Agustin Hernando
37		Codificar pruebas unitarias.	2horas?	\$234	05/28/2020	05/28/2020	36	Martin Gonzalez
38		Revisar el cumplimiento de estándares de codificación.	1hora?	\$306	05/29/2020	05/29/2020	37	Raul Gimenez
39		Documentar cambios.	1hora?	\$250	05/29/2020	05/29/2020	38	Paula Areco
40		Preparar informe de avance para el cliente	1hora?	\$625	05/29/2020	05/29/2020	39	Luis Diaz
41		Presentar entregable al cliente	0día?	\$0	05/29/2020	05/29/2020	40	Luis Diaz
42		4.1.1 - Reemplazo calendario actual.	1día?	\$4859	05/30/2020	06/01/2020		
43		Investigar librerías de calendarios en Java.	1hora?	\$500	05/30/2020	05/30/2020	34	Agustin Hernando
44		Diseñar el cambio	1hora?	\$500	05/30/2020	05/30/2020	43	Agustin Hernando
45		Implementar el cambio de calendario.	4horas?	\$2000	05/30/2020	05/31/2020	44	Agustin Hernando
46		Revisar otras funcionalidades que se ven afectadas por este cambio.	2horas?	\$1000	05/31/2020	05/31/2020	45	Agustin Hernando
47		Codificar pruebas unitarias.	2horas?	\$234	05/31/2020	06/01/2020	46	Martin Gonzalez
48		Preparar informe de avance para el cliente	1hora?	\$625	06/01/2020	06/01/2020	47	Luis Diaz
49		Presentar entregable al cliente	0día?	\$0	06/01/2020	06/01/2020	48	Luis Diaz
50		4.2.1 - Validación de campos vacíos	2.88días?	\$2109	06/01/2020	06/04/2020		
51		Investigar los métodos que no realizan esta validación.	2horas?	\$500	06/01/2020	06/02/2020	42	Paula Areco
52		Implementar cambios en el método e interfaz para que realice la validación.	3horas?	\$750	06/02/2020	06/03/2020	51	Paula Areco
53		Codificar pruebas unitarias	2horas?	\$234	06/03/2020	06/04/2020	52	Martin Gonzalez
54		Preparar informe de avance para el cliente	1hora?	\$625	06/04/2020	06/04/2020	53	Luis Diaz
55		Presentar entregable al cliente	0día?	\$0	06/04/2020	06/04/2020	54	Luis Diaz

Figura 3.8: Lista de actividades parte 2

56	□5.1.1 - Admisión de nuevas extensiones de imagenes	1día?	\$1375	06/01/2020	06/02/2020		
57	Investigar librerías para el manejo de imagenes en Java.	1hora?	\$250	06/01/2020	06/01/2020	42	Agustin Hernand
58	Implementar el cambio.	1hora?	\$250	06/02/2020	06/02/2020	57	Agustin Hernand
59	Documentar.	1hora?	\$250	06/02/2020	06/02/2020	58	Agustin Hernand
60	Preparar informe de avance para el cliente	1hora?	\$625	06/02/2020	06/02/2020	59	Luis Diaz
61	Presentar entregable al cliente	0día?	\$0	06/02/2020	06/02/2020	60	Luis Diaz
62	□6.1.1.1 - Boton para cerrar la aplicacion en todas las ventanas	1.13días?	\$3375	06/04/2020	06/06/2020		
63	Investigar el/los métodos que serializan los datos del sistema.	2horas?	\$1000	06/04/2020	06/05/2020	56,50	Agustin Hernand
64	Diseñar el cambio	1hora?	\$500	06/05/2020	06/05/2020	63	Agustin Hernand
65	Implementar el cambio.	2horas?	\$1000	06/05/2020	06/06/2020	64	Agustin Hernand
66	Documentar.	1hora?	\$250	06/06/2020	06/06/2020	65	Paula Areco
67	Preparar informe de avance para el cliente	1hora?	\$625	06/06/2020	06/06/2020	66	Luis Diaz
68	Presentar entregable al cliente	0día?	\$0	06/06/2020	06/06/2020	67	Luis Diaz
69	□2.1.1 - Complejidad ciclométrica	0día?	\$1306	06/06/2020	06/07/2020		
70	Correr herramienta JacoCo y verificar complejidad ciclomatica	1hora?	\$306	06/06/2020	06/06/2020	62	Raul Gimenez
71	Realizar cambios para disminuir la complejidad ciclomatica si fuera necesario.	2horas?	\$1000	06/06/2020	06/07/2020	70	Agustin Hernand
72	Presentar informe preliminar frente a encargado QA	0día?	\$0	06/07/2020	06/07/2020	71	Paula Areco,Raul
73	□3.1.1.1 - Agregar nuevos casos de prueba	5días?	\$1755	06/07/2020	06/12/2020		
74	Determinar las clases de equivalencia para cada nueva funcionalidad.	5horas?	\$585	06/07/2020	06/08/2020	69	Martin Gonzalez
75	Realizar un template para documentar el caso de prueba.	1hora?	\$117	06/08/2020	06/08/2020	74	Martin Gonzalez
76	Probar casos de prueba	5horas?	\$585	06/09/2020	06/10/2020	75	Martin Gonzalez
77	Documentar casos de prueba.	4horas?	\$468	06/11/2020	06/12/2020	76	Martin Gonzalez
78	Presentar informe frente a encargado QA	0día?	\$0	06/12/2020	06/12/2020	77	Martin Gonzalez,F

Figura 3.9: Lista de actividades parte 3

### 3.6.2. Gráfico Gantt



### 3.6.3. Relación entre el cronograma y el ciclo de vida

La forma en que diseñamos el cronograma de trabajo, nos permite llevar adelante un ciclo de vida iterativo e incremental. Al finalizar todas las tareas relacionadas con una funcionalidad, se le presenta al cliente el avance del proyecto, y en esa etapa de intercambio podrá darnos un *feedback* temprano, así como proponer nuevos cambios, que serán analizados por el equipo.

Al tener revisiones tanto dentro del equipo de trabajo como por parte del cliente de forma periódica, nos aseguramos que el producto sea de calidad al final de cada etapa y no avanzar a la siguiente iteración hasta que la calidad del software cumpla con los estándares propuestos al inicio del proyecto.

Nos evitamos el posible escenario en el que se termina el proyecto y recién al final nos daríamos cuenta de que los estándares no fueron cumplidos en etapas anteriores.

De esta forma podemos a lo largo del mantenimiento tener un *feedback* temprano del cliente, con entregables que muestren un avance del proyecto y el estado en el que está el mismo. Como sabemos que en cualquier momento el cliente puede pedir nuevos cambios, si implementáramos un ciclo de vida en cascada por ejemplo, se nos dificultaría volver a etapas anteriores en caso de que el cliente solicite una nueva funcionalidad y nosotros ya hayamos terminado la etapa de la especificación de los requerimientos

## 3.7. Estimación de costos

El costo total del proyecto se compone de la suma de los costos de tres factores: los salarios de los miembros del equipo cuya remuneración se puede observar en la tabla 3.2, el costo fijo de algunos servicios esenciales que la empresa deberá asumir para realizar el proyecto, y por ultimo, la ganancia de la empresa.

El costo total en cuanto a salarios es igual a: \$ 32479 (pesos uruguayos). El desglose del costo de cada actividad del cronograma, y de los recursos asignados a cada actividad, se pueden observar en el *gantt*.

Los costos fijos están compuestos por: gastos administrativos, tributos y transporte, gastos de los equipos (mantenimiento, internet, impresora...)y ascienden al importe de \$ 6000 (pesos uruguayos).

La ganancia de la empresa, se calcula como el 30 % de la suma de los costos fijos y los salarios, asciende a \$ 11544 (pesos uruguayos).

En suma, el presupuesto inicial estimado es de: \$ 50023 (pesos uruguayos).

Recurso	Rol	Costo por hora
Agustín Hernandorena	Desarrollador	\$U250
Paula Areco	Desarrollador	\$U250
Raúl Giménez	Encargado de QA	\$U306
Luis Díaz	Gerente de Proyecto	\$U625
Martín González	Tester	\$U117

Tabla 3.2: Valores precio por hora expresado en pesos uruguayos

### 3.8. Plan de calidad

Dentro del plan del proyecto tenemos el plan de calidad, en él vamos a establecer cómo evaluar la calidad del producto actual y qué métricas o actividades realizar para mantener o incrementar la misma.

#### Métricas

Las métricas nos establecen si estamos cumpliendo con un objetivo, las utilizadas en este proyecto serán: complejidad ciclomática, líneas de código, horas trabajadas, cantidad de defectos, cobertura de las pruebas, aprobación de las pruebas y calidad de diseño.

En la figura 3.10, se muestran los aspectos de calidad presentes en el plan, cual es el objetivo para cada uno de ellos, que métrica se utiliza, así como la frecuencia y momento de medición del atributo de calidad.

ID	FACTOR DE CALIDAD RELEVANTE	OBJETIVO DE CALIDAD	MÉTRICA A UTILIZAR	FRECUENCIA Y MOMENTO DE MEDICIÓN
QC1	Complejidad ciclomática promedio por clase (CC)	CC <= 18.5	Mediante JacoCo obtener la CC de cada clase del dominio, luego hacer el promedio.	Se medirá al final de la etapa de mantenimiento y se comparará con el valor de CC al inicio.
QC2	Cobertura de las pruebas (TC)	TC >= 72%	Número de pruebas realizadas / número de pruebas requeridas para tener una cobertura total.	Al final de la etapa de mantenimiento y al integrar cada funcionalidad o cambio nuevo al sistema.
QC3	Aprobación de las pruebas (TA)	TA >= 98%	Número de pruebas aprobadas / número de pruebas realizadas	Se medirá al final de la etapa de mantenimiento y se comparará con el valor TA al inicio.
QC4	Líneas de código	CL == 100 %	cantidad de clases y métodos que cumplen las reglas / cantidad de métodos y clases presentes	En toda la etapa de mantenimiento, cada vez que se desarrolla un nuevo cambio o funcionalidad.
QC5	Manejo de cambios y uso del repositorio	Fomentar la revision del codigo de una rama antes de integrarla al resto para garantizar el cumplimiento de estándares de codificación.	Cantidad de pull request solicitados en relación a las ramas feature creadas.	En toda la etapa de mantenimiento, cada vez que se desarrolla un nuevo cambio o funcionalidad.
QC6	Calidad de diseño	Mejorar el diseño y usabilidad del sistema en general.	Número de heurísticas de Nielsen que se cumplen en el sistema.	Se mide al final de la etapa de mantenimiento y se compara con lo obtenido en un comienzo.

Figura 3.10: Plan de calidad.

### QC1 - Complejidad ciclomática promedio por clase

La complejidad ciclomática del programa la calcularemos con el *plugin* de *NetBeans JaCoCo* tomando el promedio de la complejidad ciclomatica para las clases del dominio. Durante la etapa de codificación de las nuevas funcionalidades y los cambios solicitados vamos a mantenerla en el entorno de 18.5 (promedio por clase), es decir, la misma complejidad que tenia el programa en un inicio. Con respecto a esto, si bien lo ideal seria bajar la complejidad calculada en un principio, consideramos que esto implicaría un gran costo en cuanto a tiempo.

Para llevar un registro continuo de la complejidad ciclomatica del sistema, se ejecutará el software *JaCoCo* cada vez que se integre un nuevo cambio o funcionalidad desde una rama *feature* a la rama *develop*.

## QC2 - Cobertura de las pruebas

Para asegurarnos de que se cumpla el porcentaje de cobertura de pruebas que establecimos en los objetivos (por lo menos 72%) vamos a utilizar el *plugin* de *NetBeans JaCoCo*. El mismo se realizará en la etapa de mantenimiento para aumentar el estado actual de la cobertura pero también a lo largo de la duración del proyecto, ya que nuestra estrategia para lograrlo será codificar las funcionalidades y las pruebas de las mismas en paralelo.

## QC3 - Aprobación de las pruebas

Como en la cobertura de pruebas, la herramienta utilizada será *JaCoCo* y previo a implementar cualquier nueva funcionalidad vamos a modificar las pruebas para aumentar la cobertura, el valor aceptable será 98%.

## QC4 - Líneas de código

Para cumplir con las convenciones del código en Java, en el mantenimiento del software y en la introducción de nuevas funcionalidades no se introducirán líneas con más de 80 caracteres, como así tampoco se admitirán clases que sobrepasen las 2000 líneas. Los mismos serán tomados en cuenta en el momento de codificar las nuevas funcionalidades, y para asegurarnos de que se cumplan dichas reglas correremos las herramientas de análisis de código como lo son *SonarQube*, *EasyPMD* y *FindBugs* que detectarán en caso de que no se cumplan, si esto ocurre no se permitirá integrar la nueva funcionalidad al sistema hasta que no se solucionen (ver sección de manejo de cambios y uso de repositorio).

## QC 5- Manejo de cambios y uso de repositorio

Para organizar el trabajo con ramas usamos *GitFlow*. En este flujo de trabajo se utilizan dos ramas principales:

- Rama *master*: cualquier commit que pongamos en esta rama debe estar preparado para subir a producción
- Rama *develop*: rama en la que está el código que conformará la siguiente versión planificada del proyecto

Luego, contamos con ramas auxiliares tales como: *feature* y *release*.

En nuestro caso, cada vez que tengamos que implementar un cambio en el sistema, abriremos una nueva rama *feature* a partir de *develop*. Luego de haber implementado todos los cambios en esa rama, es necesario fusionarla con *develop*. Antes de realizar la integración con *develop*, el desarrollador deberá solicitar un *pull request* con el objetivo de que el resto del equipo de desarrollo: proporcione comentarios, retroalimentación y verifique el cumplimiento de estándares de codificación de acuerdo a lo



estipulado en *Java Code Conventions*. Este punto lo consideramos realmente importante porque es una instancia donde el equipo tiene la oportunidad de detectar *bugs*, o código que no siguen lineamientos, convenciones y/o buenas practicas, antes de que el código sea integrado al resto de las funcionalidades que ya fueron chequeadas previamente.

Luego de que el equipo de desarrollo (en nuestro caso el desarrollador que no codifico la funcionalidad) haya hecho la revisión del código, podrá aceptar o rechazar el *pull request*. En caso de que sea aceptado, se fusiona esa rama *feature* a *develop*. En caso contrario, se le indica al desarrollador los motivos por los cuales no fue aprobado el *pull request*.

Consideramos que este flujo de trabajo es importante, porque ayuda a disminuir costos. Notoriamente no tiene el mismo costo encontrar y corregir fallas en producción que en etapas tempranas del desarrollo.

## QC6 - Calidad del diseño

Para medir la calidad del diseño, tomamos en cuenta al inicio proyecto cuales son las heurísticas de *Nielsen* que se cumplen y las que no (definidas en el análisis dinámico), con el objetivo de que al final del proyecto podamos hacer el mismo relevamiento, y al comparar con lo obtenido en un principio, determinar en que medida mejoró la usabilidad general del sistema.

## 3.9. Plan de métricas

### GQM 1 - Mejorar la calidad del código

**Meta:** Mejorar la calidad del código

**Preguntas:**

- ¿Cuál es el numero de *bugs*?
- ¿Cuál es la complejidad ciclomática promedio?
- ¿Se siguen los estándares dispuestos en el *Java Code Conventions*?
- ¿Se hacen revisiones de código en equipo cada vez que se integra una nueva funcionalidad al sistema?

**Métricas:**

1. Número de *bugs* encontrados.
2. Complejidad ciclomática promedio por clase.
3. Número de estándares presentes en el *Java Code Conventions* que se cumplen.
4. Relación entre cantidad de *pull request* y ramas *feature* al hacer *merge* con *develop*.

**Implementación:**

1. Medir mediante el uso de la herramienta SonarQube.
2. Medir mediante el uso de la herramienta JacoCo, luego realizando el promedio manualmente.
3. Medir la cantidad de puntos que se cumplen del *Java Check List*.
4. Visualizar mediante *GitHub* que antes de mergear una rama *Feature* con *Develop* se solicitó un *pull request*.

**Referencia a objetivos del proyecto:**

Objetivos: 3.2.4, 3.2.3.

**GQM 2 - Disminuir los errores en producción**

**Meta:** Disminuir los errores en producción

**Preguntas:**

- ¿Cuál es la cobertura de pruebas unitarias?
- ¿Cuál es el porcentaje de pruebas exitosas y fallidas?

**Métricas:**

1. Porcentaje de cobertura de pruebas unitarias.
2. Porcentaje de pruebas exitosas y fallidas.

**Implementación:**

1. Medir mediante herramienta JacoCo.
2. Medir corriendo las pruebas unitarias (*JUnit*).

**Referencia a objetivos del proyecto:**

Objetivos: 3.2.1, 3.2.2.

**GQM 3- Esfuerzo al usar la aplicación**

**Meta:** Disminuir el esfuerzo del usuario al momento de usar la aplicación

**Preguntas:**

- ¿Cuál es la cantidad promedio de clics que debe realizar un usuario al registrarse?
- ¿Cuántos clics tiene que hacer un paciente para poder hablar con un profesional?
- ¿Cuánto esfuerzo lleva salir de la aplicación?
- ¿Cuánta información debe memorizar el usuario?

**Métricas:**

- Medir mediante un análisis dinámico la cantidad de clics necesarios para llevar a cabo dicha acción.
- Realizando un *checklist* con las Heurísticas de Nielsen.

**Referencia a objetivos del proyecto:**

Objetivos: 3.2.5, 3.2.6.

# Bibliografía

- [1] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.
- [2] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, eight ed. McGraw-Hill, 2014.
- [3] I. Sun Microsystems, *Java Code Conventions*, 1997.
- [4] C. Fox., *Java Inspection Checklist*, 1999.

## 4. Anexo

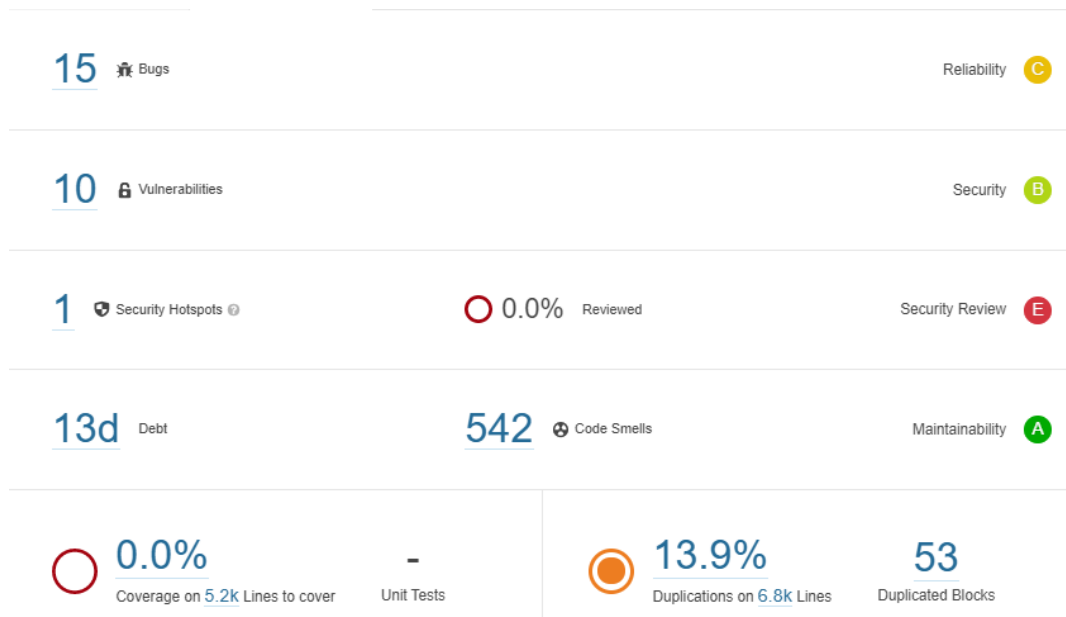
### 4.0.1. JaCoCoverage

Decidimos utilizar este programa porque consideramos que realiza un profundo análisis de la cobertura, tomando en cuenta:

- **Instructions:** Proporciona información acerca sobre la cantidad de código que se ha ejecutado en las pruebas.
- **Branches:** Calcula la cobertura para todas las sentencias *if* y *switch* en cada función. Esta métrica cuenta el número total de tales ramas en un método y determina el número de ramas ejecutadas (y las que no se ejecutaron). Para cada sentencia *if*, realiza la siguiente clasificación:
  - Sin cobertura (Rojo): No se probó ningún caso de la sentencia.
  - Cobertura parcial (Amarillo): Solo se ejecuto una parte de la sentencia, esto es por ejemplo, solo se probó el caso en que el condicional es verdadero.
  - Cobertura total (Verde): Se probaron todos los casos posibles de la sentencia.
- **Cyclomatic Complexity:** El programa calcula la complejidad ciclomatica para cada método, esto es, según *McCabe*, el numero mínimo de caminos independientes que pueden ocurrir en la ejecución, por lo tanto, consideramos que este valor es un buen indicador para determinar el numero de casos de prueba que hay que considerar en cada función.

### 4.0.2. SonarQube

Métricas:



El programa nos muestra en resumen el resultado del análisis realizado, clasificando los defectos en distintas categorías:

- **Bugs:** Un error de codificación que romperá el código y debe corregirse de inmediato. (dominio de fiabilidad). El programa categoriza con una letra comprendida de la A a la E el nivel de fiabilidad que tiene nuestro sistema, con las siguientes reglas:
  - **A:** 0 errores.
  - **B:** Al menos 1 error menor. Estos son defectos de calidad que pueden afectar ligeramente la productividad del desarrollador: Por ej.: líneas de código demasiado largas.
  - **C:** Al menos 1 error mayor. Estos son defectos de calidad que pueden afectar la productividad del desarrollador: código descubierto, bloques duplicados, parámetros no utilizados.
  - **D:** Al menos 1 error crítico. Estos errores son aquellos que tienen una baja probabilidad de afectar el comportamiento de la aplicación en producción. Ej.: un bloque *catch* vacío, inyección de SQL.
  - **E:** Al menos 1 error bloqueador. Estos errores son aquellos que tienen una alta probabilidad de afectar el comportamiento de la aplicación en producción. En este caso el código debe corregirse de inmediato.

El programa analizando contabiliza un total de 15 *bugs*, con un nivel de fiabilidad C, esto quiere decir, que el sistema cuenta con al menos 1 error mayor.

- **Vulnerabilities:** Una vulnerabilidad es un punto en el código que está abierto a ataques. El programa categoriza con una letra comprendida de la A a la E el nivel de seguridad que tiene nuestro sistema (de forma similar a como categoriza el nivel de fiabilidad).

La aplicación analizada contiene un total de 10 vulnerabilidades, todas de severidad menor, por lo cual el analizador indica que a la aplicación tiene un nivel de seguridad B.

- **Security Hotposts:** Resalta un fragmento de código sensible a la seguridad que el desarrollador debe revisar. Tras la revisión, descubrirá si no hay amenaza o que necesita aplicar una solución para proteger el código.
- **Maintainability:** El analizador divide esta sección en Code Smells y Technical Debt. El primero hace referencia a síntomas en el código fuente de un programa que posiblemente indican un problema más profundo, mientras que el segundo refiere al esfuerzo que debemos realizar para corregir todos los Code Smells en días, suponiendo un día de 8 horas. La aplicación analizada contabiliza un total de 542 Code Smells y un Technical Debt de 13 días.  
El grado de mantenibilidad de la aplicación es A, para asignar este valor el sistema toma en cuenta el cociente de deuda técnica, el cual se obtiene a partir del costo de desarrollar el software y el costo de arreglarlo (en este caso 3.3%).
- **Duplications:** El programa nos indica la cantidad de bloques de código duplicados (en este caso 53 bloques), y el porcentaje de líneas duplicadas con respecto al total de líneas (en este caso 13.9%).

#### 4.0.3. Funcionamiento de FindBugs

El programa nos muestra la cantidad de errores, y nos deja elegir cómo queremos que se ordenen. En este caso están ordenados por categoría, tipo de *bug*, *bug pattern* y *ranking* de severidad. Se puede elegir otras opciones como que se ordenen por clase, paquete o por orden en el que fueron encontrados.

Detecta errores de correctitud, malas prácticas de codificación, vulnerabilidad del código, de rendimiento y código dudoso. El análisis aportado es impreciso, puede que haya falsos *warnings* que no indiquen verdaderos errores, aunque se ha encontrado que la cantidad de *warnings* falsos que son reportados por esta herramienta es generalmente menor al cincuenta por ciento.

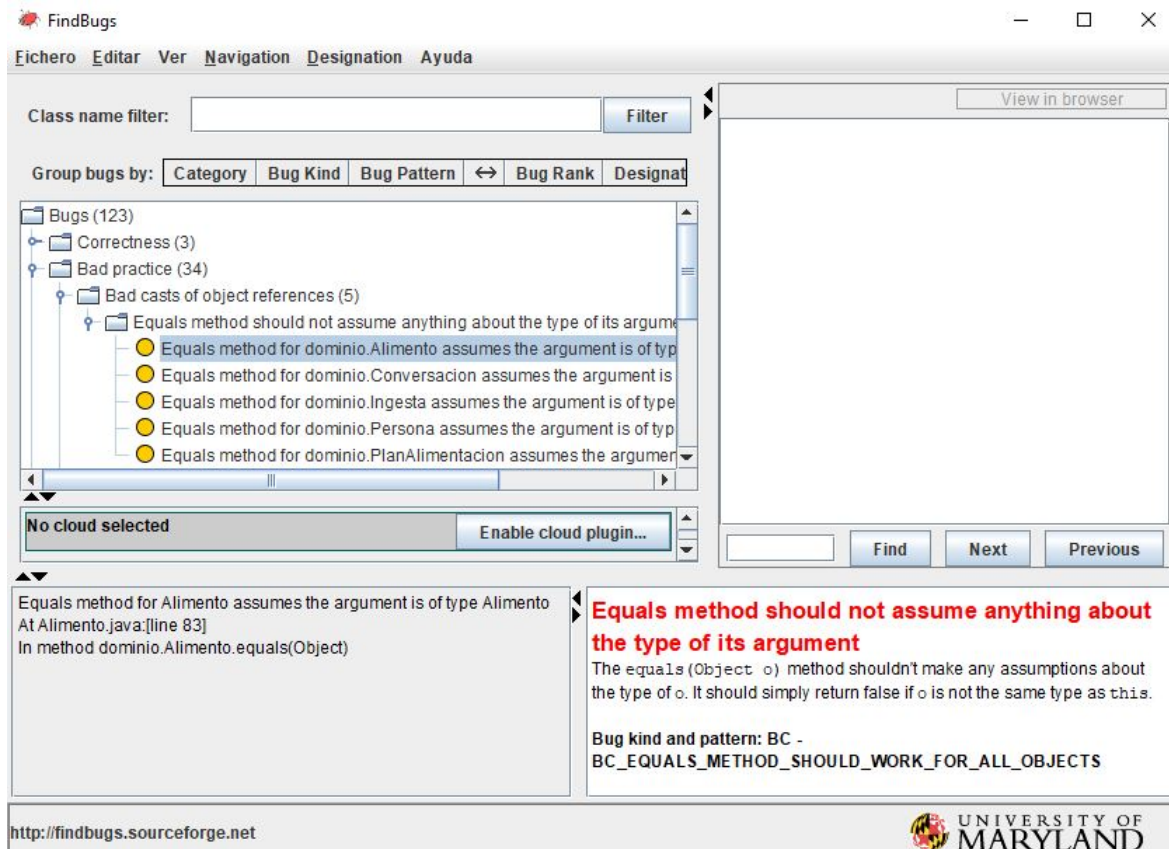


Figura 4.1: Vista general de la herramienta.

Como vemos en la imagen, FindBugs se divide en cuatro partes principales. En el rectángulo de arriba a la izquierda se encuentran los errores ordenados por el criterio que elegimos, abajo a la izquierda se determina la ubicación del *bug* en el código, y abajo a la derecha se encuentra una explicación más detallada del *bug* con una posible solución al mismo (en esta captura en específico no se muestra una posible solución).

La severidad de los errores se visualiza por un círculo a la izquierda con un color que puede ser rojo, amarillo y violeta.

#### 4.0.4. ISO 9126

Estándar internacional para la evaluación de la calidad del software. Establece una serie de características, que a su vez cada una tiene una serie de atributos (los atributos pueden ser verificado o medidos):

- **Funcionalidad:** se fija si un software adecuadamente maneja las funciones que satisfacen las necesidades especificadas.  
Atributos: adecuación, exactitud, interoperabilidad, conformidad, seguridad.



- **Confiabilidad:** es que el programa pueda seguir operando aún bajo la presencia de errores.  
Atributos: madurez, tolerancia a fallas, recuperación.
- **Usabilidad:** mide el tiempo en el que un usuario invertirá en comprender, aprender y usar el software.  
Atributos: tiempo de aprendizaje, comprensión y operatibilidad.
- **Eficiencia:** la relación entre el desempeño del software y la cantidad de recursos utilizados.  
Atributos: comportamiento con respecto al tiempo y comportamiento con respecto a recursos.
- **Mantenibilidad:** mide el esfuerzo necesario para introducir modificaciones al software, añadir funcionalidades o corregir el software.  
Atributos: facilidad de cambios, facilidad de pruebas, capacidad de análisis, estabilidad.
- **Portabilidad:** capacidad de un sistema de ser transferido a diferentes plataformas, de un ambiente a otro.  
Atributos: adaptabilidad, capacidad de instalación, capacidad de reemplazo.

Dentro de todos los items mencionados se encuentra también la **conformidad**, por ejemplo la conformidad de la portabilidad es la capacidad que tiene el software de cumplir con todos los estándares referentes a la portabilidad.



Figura 4.2: Representación esquemática de ISO 9126.

#### 4.0.5. GitFlow

Git-flow es un modelo de ramificación, ayuda en la gestión de repositorios git mediante la gestión de ramas (*branches*) y flujos de trabajo.

El proyecto se divide en dos ramas principales:

- **Master:** en ella se encuentran las versiones que están prontas para subir a producción.
- **Develop:** es donde se encuentra el código sobre el que se trabajará para la próxima versión.

En la figura 4.3 se puede ver una esquematización de su funcionamiento.

La idea es que ni la rama master, ni la rama develop reciban algún commit de forma directa, siempre reciben código desde las ramas auxiliares (**Feature**, **Release**, **Hotfix**).

En la rama *feature* vamos a implementar las nuevas funcionalidades del sistema. La rama *release* siempre saldrá desde la rama *develop* y se unirá luego a ella también, esta rama es usada y creada cuando se finalizan los features y está lista para pasar

el código a producción y unir con la rama *master*.

En este caso no vamos a tener ninguna rama *Hotfix* porque no se nos presentarán nuevos errores que deban ser corregidos con urgencia hasta la entrega del proyecto obligatorio.

Las principales ventajas de utilizar este modelo son, entre otras, disminuir la posibilidad de mezclar ramas, independizar partes del código para el momento que entregamos a producción, el equipo de pruebas recibe más rápido código para probar y así evitamos que muchos bugs se "pasen por alto" o que lleguen a producción por accidente.



Figura 4.3: Esquema de git-flow

#### 4.0.6. Objetivos Smart

Es un modelo para definir los cambios, de su nombre salen 5 pautas:

- **Specific:** deben ser lo más específicos posible, entre más detallado mejor.
- **Measurable:** debe ser medible de forma cuantitativa, para mostrar que fue logrado.
- **Achievable:** los objetivos deben ser posibles de realizar.
- **Realistic:** deben poder ser alcanzados con los recursos humanos y materiales disponibles.
- **Time-related:** se establece el marco de tiempo para lograr dichos objetivos.