Memoria

Hotel Habana

Datos de contacto:

josecarlos.lucientes@gmail.com
Teléfono: +34 605 239 368

cpifp Bajo Aragón
centro público integrado
de formación profesional

CONTROL DEL DOCUMENTO

Actualizaciones

Fecha	Autor	Versión	Ref. del cambio
29/05/2024	Jose Carlos Lucientes	1.0	Creación

Revisores

Fecha	Nombre	Posición
29/05/2024	Jose Carlos Lucientes	Alumno

ÍNDICE

1. INTRODUCCIÓN

El presente documento recoge el **ANÁLISIS FUNCIONAL Y TÉCNICO** para el proyecto **FORMULARIO RESERVAS** que consiste en el desarrollo de un formulario para la reserva del Salón Habana.

El objetivo de este **ANÁLISIS FUNCIONAL Y TÉCNICO** es doble. Por un lado analizar la funcionalidad del proyecto y por otro, especificar la arquitectura del sistema, el entorno tecnológico que le dará soporte, y los componentes del sistema de información.

Así pues, este documento es una fusión del análisis del sistema de información y su diseño técnico.

2. ANÁLISIS

2.1 Requisitos Funcionales

1. **Gestión de Reservas**

    - **Creación de Reservas**

        - El usuario debe poder abrir una ventana para crear una nueva reserva.
        - El usuario debe poder ingresar el nombre, teléfono, tipo de evento, número de jornadas, si se requieren habitaciones, tipo de cocina, número de personas, y la fecha del evento.
        - El sistema debe verificar que todos los campos obligatorios están completos antes de guardar una reserva.
        - El sistema debe mostrar un mensaje de error si los datos ingresados son incorrectos o incompletos.
        - El sistema debe almacenar la información de la nueva reserva.

2. **Listado de Reservas**

    - El usuario debe poder abrir una ventana para ver la lista de todas las reservas.
    - El sistema debe mostrar un listado con las reservas existentes, incluyendo detalles como nombre, teléfono, tipo de evento, número de jornadas, habitaciones, tipo de cocina, número de personas, y fecha del evento.

3. **Interacción con la Interfaz de Usuario**

    - La aplicación debe tener un menú con opciones para crear una nueva reserva y para ver el listado de reservas.
    - El sistema debe permitir al usuario interactuar con los botones y elementos del menú para acceder a las diferentes funcionalidades (crear reserva, ver listado).

Derivación de Requisitos Funcionales desde las Clases

Principal

- Debe coordinar la apertura de las ventanas de creación y listado de reservas.
- Debe contener elementos de la interfaz de usuario (botones, menú) que permitan al usuario acceder a las funcionalidades de creación y visualización de reservas.

Reservas

- Debe gestionar el formulario de creación de nuevas reservas.
- Debe validar los datos de entrada del usuario antes de guardar una reserva.
- Debe almacenar la información de la reserva en una estructura de datos adecuada (en este caso, una matriz).

Listado

- Debe mostrar una lista de todas las reservas existentes.
- Debe actualizar la lista de reservas cuando se agregue una nueva reserva.

ID	Título	Descripción	Estado	Prioridad
RF_001	Home	El sistema debe iniciar la aplicación con una interfaz de usuario en la que aparezca una imagen del hotel.	Aprobado	Alta
RF_002	Nueva reserva	El sistema debe permitir al usuario abrir una ventana para crear una nueva reserva.	Aprobado	Alta

RF_003	Información reserva	El sistema debe permitir al usuario ingresar detalles de la reserva, incluyendo nombre, teléfono, tipo de evento, número de jornadas, habitaciones, tipo de cocina, número de personas, y fecha.	Aprobado	Alta
RF_004	Selección tipo de evento	El sistema debe permitir al usuario seleccionar el número de jornadas si se selecciona el tipo de evento "CONGRESO" en cualquier otro tipo de evento ha de estar deshabilitada esta opción.	Aprobado	Alta
RF_005	Selección tipo de evento	El sistema debe permitir al usuario indicar si requiere habitaciones si selecciona el tipo de evento "CONGRESO" en cualquier otro tipo de evento ha de estar deshabilitada esta opción.	Aprobado	Alta
RF_006	Validación información reserva	El sistema debe validar que todos los campos obligatorios están completos antes de permitir que la reserva sea guardada.	Aprobado	Alta

RF_007	Control de errores	El sistema debe mostrar un mensaje de error si los datos ingresados son incorrectos o incompletos.	Aprobado	Alta
RF_008	Almacenar reserva	El sistema debe guardar la información de la nueva reserva.	Aprobado	Alta
RF_009	Listar reservas	El sistema debe permitir al usuario abrir una ventana para ver el listado de reservas existentes.	Aprobado	Alta
RF_010	Ver reservas	El sistema debe mostrar una lista de todas las reservas, incluyendo detalles como nombre, teléfono, tipo de evento, número de jornadas, habitaciones, tipo de cocina, número de personas, y fecha.	Aprobado	Alta
RF_011	Menú gestión reservas	La aplicación debe tener un menú con opciones para crear una nueva reserva y ver el listado de reservas.	Aprobado	Alta
RF_012	Gestión reservas	El sistema debe permitir al usuario interactuar con los botones y elementos del menú para acceder a las	Aprobado	Alta

		diferentes funcionalidades.		
RF_013	Actualizar reservas	El sistema debe actualizar la lista de reservas cuando se agregue una nueva reserva.	Aprobado	Alta

2.1.1 Casos de Uso

- **CU001: Iniciar Aplicación**
  - **Actor**: Usuario
  - **Descripción**: El usuario inicia la aplicación del hotel Habana.
  - **Precondiciones**: El sistema está instalado y accesible.
  - **Flujo Principal**:
    1. El usuario abre la aplicación.
    2. La ventana principal de la aplicación se muestra.
  - **Postcondiciones**: La ventana principal está lista para recibir interacciones del usuario.
- **CU002: Crear Nueva Reserva**
  - **Actor**: Usuario
  - **Descripción**: El usuario crea una nueva reserva.
  - **Precondiciones**: La aplicación está abierta y la ventana principal está visible.
  - **Flujo Principal**:
    1. El usuario hace clic en el botón "NUEVA RESERVA" o selecciona "Nueva Reserva" en el menú.
    2. Se abre la ventana de creación de nueva reserva.
    3. El usuario introduce los datos de la reserva.

4. El usuario guarda la reserva.

- **Postcondiciones**: La nueva reserva se guarda en el sistema.

- **CU004: Ver Reservas Existentes**

  - **Actor**: Usuario
  - **Descripción**: El usuario consulta las reservas existentes.
  - **Precondiciones**: La aplicación está abierta y la ventana principal está visible.
  - **Flujo Principal**:
    1. El usuario hace clic en el botón "VER RESERVAS" o selecciona "Ver Reservas" en el menú.
    2. Se abre la ventana de listado de reservas.
    3. El usuario revisa las reservas mostradas.
  - **Postcondiciones**: Las reservas existentes se muestran al usuario.

- **CU005: Cambiar Estilo de la Tabla de Reservas**

  - **Actor**: Usuario
  - **Descripción**: El usuario cambia el estilo de la tabla de reservas.
  - **Precondiciones**: El usuario ha accedido a la ventana de listado de reservas.
  - **Flujo Principal**:
    1. El usuario hace clic en el botón para cambiar el estilo.
    2. El sistema aplica el nuevo estilo a la tabla.
  - **Postcondiciones**: La tabla de reservas se muestra con el nuevo estilo.

- **CU006: Cerrar la Aplicación**

  - **Actor**: Usuario
  - **Descripción**: El usuario cierra la aplicación.
  - **Precondiciones**: La aplicación está abierta.
  - **Flujo Principal**:
    1. El usuario cierra la ventana principal.

2. El sistema realiza las operaciones necesarias para cerrar la aplicación.

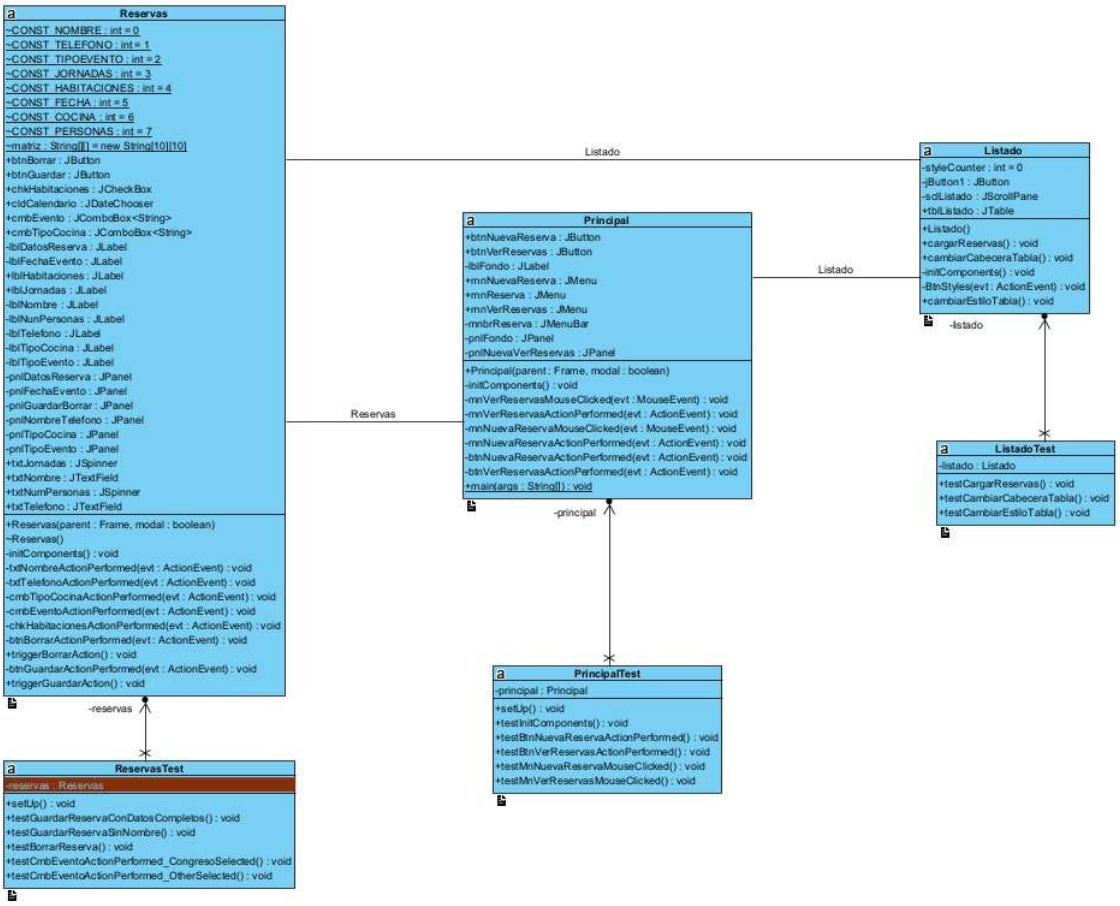
- **Postcondiciones**: La aplicación se cierra correctamente.

2.2 Requisitos No Funcionales

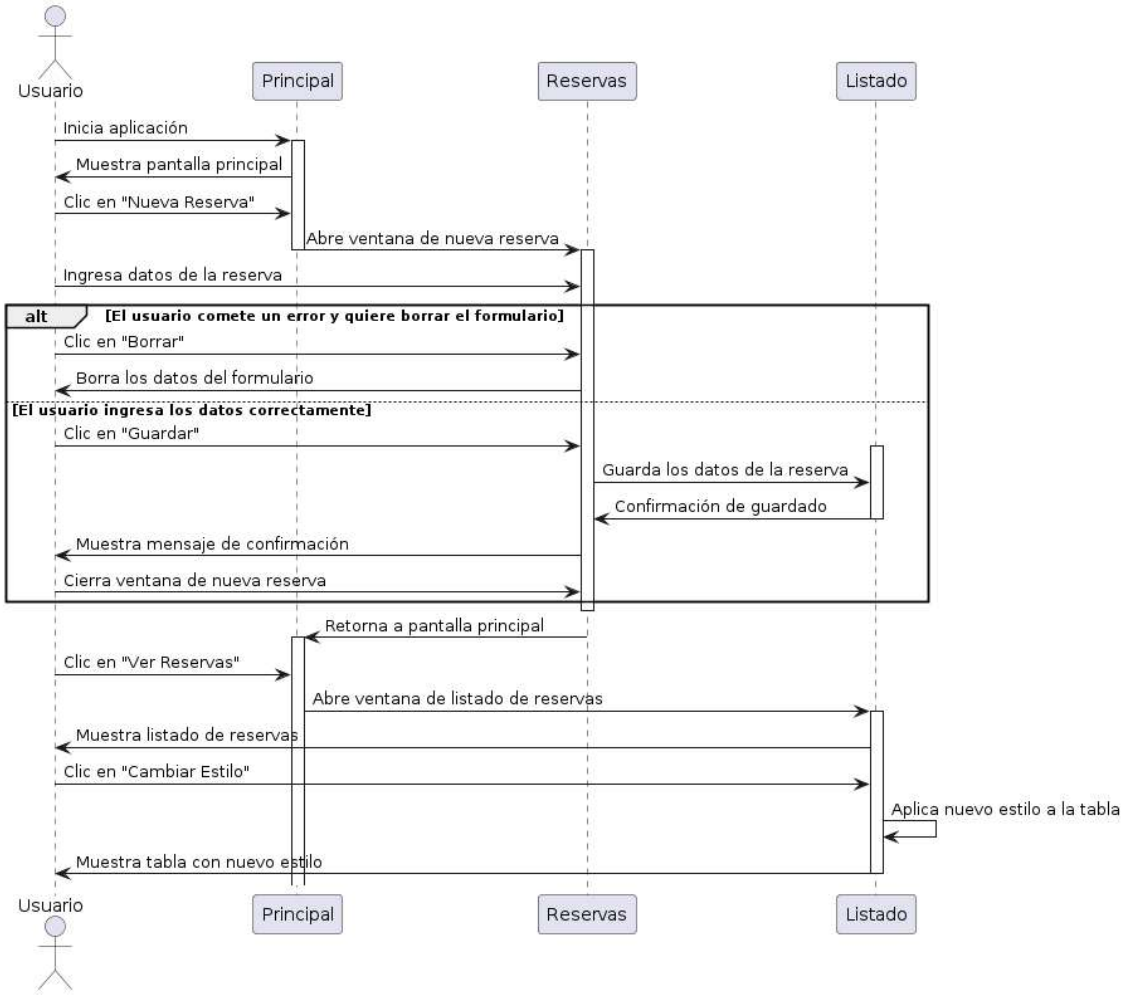
ID	Título	Descripción	Estado	Prioridad
RNF_001	De Plataforma Tecnológica	La aplicación debe ser desarrollada en la versión de Java 19	Aprobado	Alta
RNF_002	De Plataforma Tecnológica	La aplicación debe ser desarrollada con Java Swing	Aprobado	Alta

3. DISEÑO

3.1 Diagrama de Clases



3.1.1 Mapa de navegación



4. PRUEBAS

En este documento de pruebas se presentan las pruebas unitarias desarrolladas para verificar el correcto funcionamiento de las clases **Principal**, **Reservas** y **Listado** de la aplicación de gestión de reservas del hotel Habana. Este documento está organizado en secciones que detallan las pruebas para cada método específico de las clases mencionadas.

Cada prueba incluye la siguiente información:

1. **Código de la prueba**: El fragmento de código utilizado para realizar la prueba.
2. **Método a probar**: El método específico de la clase que se está evaluando.
3. **Datos de entrada**: Los datos que se ingresan al método para realizar la prueba.
4. **Datos de salida esperados**: Los resultados esperados tras la ejecución del método.
5. **Resultado de la prueba**: Una indicación de si la prueba fue exitosa ("Ok") o no ("No Ok").

4.1 PrincipalTest

4.1.1 testInitComponents()

Código de la prueba:

```
@Test
public void testInitComponents() {
    boolean result = principal != null
            && principal.btnNuevaReserva != null
            && principal.btnVerReservas != null
            && principal.mnNuevaReserva != null
            && principal.mnVerReservas != null;

    if (result) {
        System.out.println("Resultado de la prueba de testInitComponents: Ok");
    } else {
        System.out.println("Resultado de la prueba de testInitComponents: No Ok");
    }
    // Asegurando que el test falle si alguno de los componentes es null
    assertNotNull(principal);
    assertNotNull(principal.btnNuevaReserva);
    assertNotNull(principal.btnVerReservas);
    assertNotNull(principal.mnNuevaReserva);
    assertNotNull(principal.mnVerReservas);
}
```

Datos de entrada: Ninguno (Inicialización de la clase "**Principal**")

Método a probar: "initComponents()"

Datos de salida esperados:
"principal" no debe ser "null"
"principal.btnNuevaReserva" no debe de ser "null"
"principal.btnVerReservas" no debe de ser "null"
"principal.mnNuevaReserva" no debe de ser "null"
"principal.mnVerReservas" no debe de ser "null"

Resultado de la prueba: OK

4.1.2 testBtnNuevaReservaActionPerformed

Código de la prueba:

```
@Test
public void testBtnNuevaReservaActionPerformed() {
    principal.btnNuevaReserva.doClick();

    // Verifica si la ventana de nueva reserva está visible
    boolean isVisible = false;
    for (Window window : Window.getWindows()) {
        if (window instanceof Reservas) {
            isVisible = window.isVisible();
            window.dispose();  // Cierra la ventana después de verificar
        }
    }
    if (isVisible) {
        System.out.println("Resultado de la prueba de btnNuevaReservaActionPerformed: Ok");
    } else {
        System.out.println("Resultado de la prueba de btnNuevaReservaActionPerformed: No Ok");
    }
    assertTrue(isVisible, "La ventana de nueva reserva debería estar visible.");
}
```

Datos de entrada: Clic en el botón "btnNuevaReserva"

Método a probar: "btnNuevaReservaActionPerformed()"

Datos de salida esperados: La ventana de nueva reserva ("**Reservas**") de ser visible.

Resultado de la prueba: OK

4.1.3 testBtnVerReservasActionPerformed

Código de la prueba:

```
@Test
public void testBtnVerReservasActionPerformed() {
    principal.btnVerReservas.doClick();

    // Verifica si la ventana de lista de reservas está visible
    boolean isVisible = false;
    for (Window window : Window.getWindows()) {
        if (window instanceof Listado) {
            isVisible = window.isVisible();
            isVisible = true;
            window.dispose();  // Cierra la ventana después de verificar
        }
    }
    if (isVisible) {
        System.out.println("Resultado de la prueba de btnVerReservasActionPerformed: Ok");
    } else {
        System.out.println("Resultado de la prueba de btnVerReservasActionPerformed: No Ok");
    }
    assertTrue(isVisible, "La prueba ha fallado.");
}
```

Datos de entrada: Clic en el boton “btnVerReservasActionPerfomed()”

Método a probar: “btnVerReservasActionPerformed()”

Datos de salida esperados: La ventana de listado de reservas (“**Listado**”) debe de ser visible.

Resultado de la prueba: OK

4.1.4 testMnNuevaReservasMouseClicked()

Código de la prueba:

```
@Test
public void testMnNuevaReservaMouseClicked() {
    // Simula un clic de ratón en el elemento de menú
    for (MouseListener listener : principal.mnNuevaReserva.getMouseListeners()) {
        listener.mouseClicked(null);
    }

    // Verifica si la ventana de nueva reserva está visible
    boolean isVisible = false;
    for (Window window : Window.getWindows()) {
        if (window instanceof Reservas) {
            isVisible = window.isVisible();
            window.dispose();  // Cierra la ventana después de verificar
        }
    }
    if (isVisible) {
        System.out.println("Resultado de la prueba de mnNuevaReservaMouseClicked: Ok");
    } else {
        System.out.println("Resultado de la prueba de mnNuevaReservaMouseClicked: No Ok");
    }
    assertTrue(isVisible, "La prueba ha fallado.");
}
```

Datos de entrada: Clic en el JMenu “Nueva Reserva”

Método a probar: “btnVerReservasMouseClicked”

Datos de salida esperados: La ventana de nueva reserva (“**Reservas**”) ha de ser visible.

Resultado de la prueba: OK

4.1.5 testMnVerReservasMouseCicked()

Código de la prueba:

```
@Test
public void testMnVerReservasMouseClicked() {
    // Simula un clic de ratón en el elemento de menú
    for (MouseListener listener : principal.mnVerReservas.getMouseListeners()) {
        listener.mouseClicked(null);
    }

    // Verifica si la ventana de lista de reservas está visible
    boolean isVisible = false;
    for (Window window : Window.getWindows()) {
        if (window instanceof Listado) {
            isVisible = window.isVisible();
            isVisible = true;
            window.dispose();  // Cierra la ventana después de verificar
        }
    }

    if (isVisible) {
        System.out.println("Resultado de la prueba de mnVerReservasMouseClicked: Ok");
    } else {
        System.out.println("Resultado de la prueba de mnVerReservasMouseClicked: No Ok");
    }
    assertTrue(isVisible, "La prueba ha fallado.");
}
```

Datos de entrada: Clic en el JMenu “Ver Reservas”

Método a probar: “mnVerReservasMouseClicked()”

Datos de salida esperados: La ventana de nueva reserva (“**Listado**”) ha de ser visible.

Resultado de la prueba: OK

4.2 ReservasTest

4.2.1 testGuardarReservaConDatosCompletos()

Código de la prueba:

```
@Test //Guardado datos completos
public void testGuardarReservaConDatosCompletos() {
    // Preparación de datos
    reservas.txtNombre.setText("John Doe");
    reservas.txtTelefono.setText("123456789");
    reservas.cmbEvento.setSelectedItem("CONGRESO");
    reservas.txtJornadas.setValue(2);
    reservas.chkHabitaciones.setSelected(true);
    reservas.cmbTipoCocina.setSelectedItem("BUFE");
    reservas.txtNumPersonas.setValue(100);
    reservas.cldCalendario.setDate(new java.util.Date());
    // Ejecución del método
    // reservas.btnGuardarActionPerformed(null);
    reservas.triggerGuardarAction();
    // Verificación
    boolean result = Reservas.matriz[0][CONST_NOMBRE].equals("John Doe")
            && Reservas.matriz[0][CONST_TELEFONO].equals("123456789")
            && Reservas.matriz[0][CONST_TIPOEVENTO].equals("CONGRESO")
            && Reservas.matriz[0][CONST_JORNADAS].equals("2")
            && Reservas.matriz[0][CONST_HABITACIONES].equals("SI")
            && Reservas.matriz[0][CONST_COCINA].equals("BUFE")
            && Reservas.matriz[0][CONST_PERSONAS].equals("100")
            && Reservas.matriz[0][CONST_FECHA] != null;
    if (result) {
        System.out.println("Resultado de la prueba testGuardarReservaConDatosCompletos: Ok");
    } else {
        System.out.println("Resultado de la prueba testGuardarReservaConDatosCompletos: No Ok");
    }
}
```

Datos de entrada:

txtNombre: "John Doe"

txtTelefono: "123456789"

cmbEvento: "CONGRESO"

txtJornadas: 2

chkHabitaciones: true

cmbTipoCocina: "BUFE"

txtNumPersonas: 100

cldCalendario: new java.util.Date()

Método a probar: “guadarReserva()”

Datos de salida esperados:
Reservas.matriz[0][CONST_NOMBRE]: "John Doe"

Reservas.matriz[0][CONST_TELEFONO]: "123456789"

Reservas.matriz[0][CONST_TIPOEVENTO]: "CONGRESO"

Reservas.matriz[0][CONST_JORNADAS]: "2"

Reservas.matriz[0][CONST_HABITACIONES]: "SI"

Reservas.matriz[0][CONST_COCINA]: "BUFE"

Reservas.matriz[0][CONST_PERSONAS]: "100"

Reservas.matriz[0][CONST_FECHA]: No null

Resultado de la prueba: OK

4.2.2 testGuardarReservaSinNombre()

Código de la prueba:

```
@Test
public void testGuardarReservaSinNombre() {
    // Preparación de datos
    reservas.txtNombre.setText("");
    reservas.txtTelefono.setText("123456789");
    reservas.cmbEvento.setSelectedItem("BANQUETE");
    reservas.txtJornadas.setValue(0);
    reservas.chkHabitaciones.setSelected(false);
    reservas.cmbTipoCocina.setSelectedItem("BUFE");
    reservas.txtNumPersonas.setValue(50);
    reservas.cldCalendario.setDate(new java.util.Date());
    // Capturar el mensaje de error
    //reservas.btnGuardarActionPerformed(null);
    reservas.triggerGuardarAction();
    // Verificación
    boolean result = !JOptionPane.getRootFrame().isVisible();
    if (result) {
        System.out.println("Resultado de la prueba testGuardarReservaSinNombre: Ok");
    } else {
        System.out.println("Resultado de la prueba testGuardarReservaSinNombre: No Ok");
    }
}
```

Datos de entrada:
txtNombre: ""

txtTelefono: "123456789"

cmbEvento: "BANQUETE"

txtJornadas: 0

chkHabitaciones: false

cmbTipoCocina: "BUFE"

txtNumPersonas: 50

cldCalendario: new java.util.Date()

Método a probar: “guardarReserva()”

Datos de salida esperados: Ventana de error “Has de indicar un nombre”

Resultado de la prueba: OK

4.2.3 testBorrarRserva()

Código de la prueba:

```
@Test
public void testBorrarReserva() {
    // Preparación de datos
    reservas.txtNombre.setText("John Doe");
    reservas.txtTelefono.setText("123456789");
    reservas.cmbEvento.setSelectedItem("BANQUETE");
    reservas.txtJornadas.setValue(2);
    reservas.chkHabitaciones.setSelected(true);
    reservas.cmbTipoCocina.setSelectedItem("BUFE");
    reservas.txtNumPersonas.setValue(100);
    reservas.cldCalendario.setDate(new java.util.Date());
    // Ejecución del método
    reservas.triggerBorrarAction();
    // Verificación
    boolean result = reservas.txtNombre.getText().isEmpty()
            && reservas.txtTelefono.getText().isEmpty()
            && reservas.cmbEvento.getSelectedItem().equals("Selecciona")
            && ((int) reservas.txtJornadas.getValue()) == 0
            && !reservas.chkHabitaciones.isSelected()
            && reservas.cmbTipoCocina.getSelectedItem().equals("Selecciona")
            && ((int) reservas.txtNumPersonas.getValue()) == 0
            && reservas.cldCalendario.getDate() == null;

    if (result) {
        System.out.println("Resultado de la prueba testBorrarReserva: Ok");
    } else {
        System.out.println("Resultado de la prueba testBorrarReserva: No Ok");
    }
}
```

Datos de entrada:

txtNombre: "John Doe"

txtTelefono: "123456789"

cmbEvento: "CONGRESO"

txtJornadas: 2

chkHabitaciones: true

cmbTipoCocina: "BUFE"

txtNumPersonas: 100

cldCalendario: new java.util.Date()

Método a probar: “bnorrarReservas()”

Datos de salida esperados:
txtNombre: ""

txtTelefono: ""

cmbEvento: Sin selección

txtJornadas: 0

chkHabitaciones: false

cmbTipoCocina: Sin colección

txtNumPersonas: 1

cldCalendario: “null”

Resultado de la prueba: Ok

4.2.4 testCmbEventoActionPerformed_CongresoSelected()

Código de la prueba:

```
@Test
public void testCmbEventoActionPerformed_CongresoSelected() {
    // Preparación de datos
    reservas.cmbEvento.setSelectedItem("CONGRESO");

    // Simulación de evento de cambio de selección
    ActionEvent event = new ActionEvent(reservas.cmbEvento, ActionEvent.ACTION_PERFORMED, "");
    reservas.cmbEvento.getActionListeners()[0].actionPerformed(event);

    // Verificación
    boolean result = reservas.lblJornadas.isEnabled()
            && reservas.txtJornadas.isEnabled()
            && reservas.lblHabitaciones.isEnabled()
            && reservas.chkHabitaciones.isEnabled();
    if (result) {
        System.out.println("Resultado de la prueba testCmbEventoActionPerformed_CongresoSelected: Ok");
    } else {
        System.out.println("Resultado de la prueba testCmbEventoActionPerformed_CongresoSelected: No Ok");
    }
}
```

Datos de entrada: Selección de “CONGRESO” en “cmbEvento”

Método a probar: “cmbEventoActionPerformed_CongresoSelected()”

Datos de salida esperados:

“txtJornadas”: Habilitado

“txtNumPersonas”: Habilitado

Resultado de la prueba: Ok

4.2.5 testCmbEventoActionPerformed_OtherSelected()

Código de la prueba:

```
@Test
public void testCmbEventoActionPerformed_OtherSelected() {
    // Preparación de datos
    reservas.cmbEvento.setSelectedItem("BANQUETE");

    // Simulación de evento de cambio de selección
    ActionEvent event = new ActionEvent(reservas.cmbEvento, ActionEvent.ACTION_PERFORMED, "");
    reservas.cmbEvento.getActionListeners()[0].actionPerformed(event);

    // Verificación
    boolean result = !reservas.lblJornadas.isEnabled()
            && !reservas.txtJornadas.isEnabled()
            && !reservas.lblHabitaciones.isEnabled()
            && !reservas.chkHabitaciones.isEnabled();
    if (result) {
        System.out.println("Resultado de la prueba testCmbEventoActionPerformed_OtherSelected: Ok");
    } else {
        System.out.println("Resultado de la prueba testCmbEventoActionPerformed_OtherSelected: No Ok");
    }
}
```

Datos de entrada: Selección de “BANQUETE” en “cmbEvento”

Método a probar: “cmbEventoActionPerformed()”

Datos de salida esperados:

“txtJornadas”: Deshabilitado

“txtNumPersonas”: Deshabilitado

Resultado de la prueba: Ok

4.3 ListadoTest

4.3.1 testCargarReservas()

Código de la prueba:

```
@Test
public void testCargarReservas() {
    // Preparación de datos de prueba
    Reservas.matriz[0][Reservas.CONST_NOMBRE] = "John Doe";
    Reservas.matriz[0][Reservas.CONST_TELEFONO] = "123456789";
    Reservas.matriz[0][Reservas.CONST_TIPOEVENTO] = "CONGRESO";
    Reservas.matriz[0][Reservas.CONST_JORNADAS] = "2";
    Reservas.matriz[0][Reservas.CONST_HABITACIONES] = "true";
    Reservas.matriz[0][Reservas.CONST_FECHA] = "2024-05-20";
    Reservas.matriz[0][Reservas.CONST_COCINA] = "BUFE";
    Reservas.matriz[0][Reservas.CONST_PERSONAS] = "100";

    // Método a probar
    listado = new Listado();
    // Ejecución del método
    listado.cargarReservas();
    // Verificación
    boolean result
            = listado.tblListado.getValueAt(0, Reservas.CONST_NOMBRE).equals("John Doe")
            && listado.tblListado.getValueAt(0, Reservas.CONST_TELEFONO).equals("123456789")
            && listado.tblListado.getValueAt(0, Reservas.CONST_TIPOEVENTO).equals("CONGRESO")
            && listado.tblListado.getValueAt(0, Reservas.CONST_JORNADAS).equals("2")
            && listado.tblListado.getValueAt(0, Reservas.CONST_HABITACIONES).equals("true")
            && listado.tblListado.getValueAt(0, Reservas.CONST_FECHA).equals("2024-05-20")
            && listado.tblListado.getValueAt(0, Reservas.CONST_COCINA).equals("BUFE")
            && listado.tblListado.getValueAt(0, Reservas.CONST_PERSONAS).equals("100");

    if (result) {
        System.out.println("Resultado de la prueba testCargarReservas: Ok");
    } else {
        System.out.println("Resultado de la prueba testCargarReservas: No Ok");
    }
}
```

Datos de entrada: Datos en “Reservas.matriz”
Reservas.matriz[0][CONST_NOMBRE]: "John Doe"

Reservas.matriz[0][CONST_TELEFONO]: "123456789"

Reservas.matriz[0][CONST_TIPOEVENTO]: "CONGRESO"

Reservas.matriz[0][CONST_JORNADAS]: "2"

Reservas.matriz[0][CONST_HABITACIONES]: "true"

Reservas.matriz[0][CONST_FECHA]: "2024-05-20"

Reservas.matriz[0][CONST_COCINA]: "BUFE"

Reservas.matriz[0][CONST_PERSONAS]: "100"

Método a probar: “cargarReservas()”

Datos de salida esperados: “tblListado” debe contener los valores cargados desde “Reservas.matriz”

Resultado de la prueba: Ok

4.3.2 testCambiarCabeceraTabla()

Código de la prueba:

```
@Test
public void testCambiarCabeceraTabla() {
    // Método a probar
    Listado listado = new Listado();
    listado.cambiarCabeceraTabla();

    // Verificación
    boolean result = listado.tblListado.getTableHeader().getFont().getFontName().equals("Arial Rounded MT Bold")
            && listado.tblListado.getTableHeader().getFont().getStyle() == Font.BOLD
            && listado.tblListado.getTableHeader().getFont().getSize() == 14;

    if (result) {
        System.out.println("Resultado de la prueba testCambiarCabeceraTabla: Ok");
    } else {
        System.out.println("Resultado de la prueba testCambiarCabeceraTabla: No Ok");
    }
}
```

Datos de entrada: Ninguno (llama al método)

Método a probar: "cambiarCabeceraTabla()"

Datos de salida esperados: La cabecera de la tabla debe tener la fuente "**Arial Rounded MT Bold**" en estilo "**BOLD**" y tamaño "**14**"

Resultado de la prueba: Ok

4.3.3 testCambiarEstiloTabla()

Código de la prueba:

```
@Test
public void testCambiarEstiloTabla() {
    // Método a probar
    Listado listado = new Listado();
    // Simulación de presionar el botón varias veces
    listado.cambiarEstiloTabla();
    Font font1 = listado.tblListado.getTableHeader().getFont();
    Color headerColor1 = listado.tblListado.getTableHeader().getBackground();
    Color rowColor1 = listado.tblListado.getBackground();

    listado.cambiarEstiloTabla();
    Font font2 = listado.tblListado.getTableHeader().getFont();
    Color headerColor2 = listado.tblListado.getTableHeader().getBackground();
    Color rowColor2 = listado.tblListado.getBackground();

    listado.cambiarEstiloTabla();
    Font font3 = listado.tblListado.getTableHeader().getFont();
    Color headerColor3 = listado.tblListado.getTableHeader().getBackground();
    Color rowColor3 = listado.tblListado.getBackground();
    // Verificación
    boolean result1 = font1.equals(new Font("Arial Rounded MT Bold", Font.BOLD, 14))
            && headerColor1.equals(Color.LIGHT_GRAY)
            && rowColor1.equals(Color.WHITE);

    boolean result2 = font2.equals(new Font("Verdana", Font.ITALIC, 12))
            && headerColor2.equals(Color.DARK_GRAY)
            && rowColor2.equals(Color.LIGHT_GRAY);

    boolean result3 = font3.equals(new Font("SansSerif", Font.PLAIN, 16))
            && headerColor3.equals(Color.BLUE)
            && rowColor3.equals(Color.CYAN);

    if (result1 && result2 && result3) {
        System.out.println("Resultado de la prueba testCambiarEstiloTabla: Ok");
    } else {
        System.out.println("Resultado de la prueba testCambiarEstiloTabla: No Ok");
    }
}
```

Datos de entrada: Ninguno (llama al método en tres ocasiones)

Método a probar: “cambiarEstiloTabla()”

Datos de salida esperados:
La primera llamada debe establecer los siguientes valores:

Fuente: **Arial Rounded MT Bold**, estilo **BOLD**, tamaño **14**.

Color de fondo de la cabecera: **LIGHT_GRAY**.

Color de fondo de las filas: **WHITE**.

La segunda llamada debe establecer los siguientes valores:

Fuente: **Verdana**, estilo **ITALIC**, tamaño **12**.

Color de fondo de la cabecera: **DARK_GRAY**.

Color de fondo de las filas: **LIGHT_GRAY**.

La tercera llamada debe establecer los siguientes valores:

Fuente: **SansSerif**, estilo **PLAIN**, tamaño **16**.

Color de fondo de la cabecera: **BLUE**.

Color de fondo de las filas: **CYAN**.

Resultado de la prueba: Ok

5. METODOLOGIA

5.1 Guía SCRUM

5.1.1 ¿Qué es SCRUM?

Scrum es un proceso de gestión que reduce la complejidad en el desarrollo de productos para satisfacer las necesidades de los clientes. La gerencia y los equipos de Scrum trabajan juntos alrededor de requisitos y tecnologías para entregar productos funcionando de manera incremental usando el empirismo.

Scrum es un marco de trabajo simple que promueve la colaboración en los equipos para lograr desarrollar productos complejos. Ken Schwaber y Jeff Sutherland han escrito La Guía Scrum para explicar Scrum de manera clara y simple.

5.1.2 Características de SCRUM

Scrum es simple, no es una gran colección de partes y componentes obligatorios definidos de manera prescriptiva. Scrum no es una metodología, Scrum está basado en un modelo de proceso empírico. con respeto a las personas y basado en la autoorganización de los equipos para lidiar con lo imprevisible y resolver problemas complejos inspeccionando y adaptando continuamente.

5.1.3 Ciclo SCRUM

¿Qué es un Sprint? ¿Cuánto dura?
El primer paso para alcanzar este objetivo -o hito del proyecto- es la reunión de planificación, una sesión en la que debe participar todo el equipo ‘scrum’ y que supone el pistoletazo de salida del ‘sprint’. Esta reunión se divide en dos partes que tratan de dar respuesta a dos preguntas fundamentales: ¿Qué se va a entregar? y ¿cómo se va a realizar el trabajo? Tiene una duración máxima de 4 semanas.

5.1.4 ARTEFACTOS

¿Qué es el Product Backlog?

El product backlog (o pila de producto) es un listado de todas las tareas que se pretenden hacer durante el desarrollo de un proyecto.
Algunos product backlog pueden asociarse con proyectos de varios años, incluso.

Todas las tareas deben listarse en el product backlog, para que estén visibles ante todo el equipo y se pueda tener una visión panorámica de todo lo que se espera realizar.

¿Qué es el Sprint backlog?

El Sprint Backlog es la suma de el Objetivo del Sprint, los elementos del Product Backlog elegidos para el Sprint, más un plan de acción de cómo crear el Incremento de Producto.

Es uno de los 3 artefactos de Scrum y se construye durante el evento del Sprint Planning. Es un plan realizado por y para los desarrolladores.
El equipo generalmente divide el trabajo en elementos llamados Sprint Backlog Ítems (SBI). Estos elementos pueden representar tareas que el equipo debe completar, bloques de construcción intermedios que se combinan en una entrega, o cualquier otra unidad de trabajo que ayude al equipo a comprender cómo lograr el Sprint Goal dentro del Sprint.

¿Qué es un Daily scrum meeting? ¿Cuánto Dura? ¿Quién lo organiza?

El Daily Scrum (o Scrum diario) es uno de los 5 eventos de Scrum con un bloque de tiempo de 15 minutos para que los Developers se sincronicen.
Esta reunión diaria se realiza a la misma hora y en el mismo lugar para reducir la complejidad. Aquí se busca la transparencia y la inspección de lo realizado para tener una oportunidad de adaptación para el día siguiente.

El propósito de la Daily Scrum es inspeccionar el progreso hacia el Objetivo del Sprint y adaptar el Sprint Backlog según sea necesario. Se busca que los

Developers logren sincronizarse. Para ello se planea el trabajo de las siguientes 24 horas.
La Daily Scrum es un evento interno de los Develoeprs. Si el Product Owner o el Scrum Master están trabajando activamente en elementos del Sprint Backlog, participan como Developers.

5.1.5 EQUIPO

¿Cuántas personas componen un equipo?

Un equipo Scrum suele estar compuesto por grupos de trabajo de entre 3 a 9 miembros del equipo de desarrollo, más el Scrum Master y el Product Owner. Cada uno de estos roles tiene diferentes responsabilidades y debe de rendir cuentas de distinta manera, tanto entre ellos como para el resto de la organización. La suma de todos los roles es lo que llamamos Equipo Scrum.

¿Quién es el Product Owner? ¿Cuáles son sus funciones y responsabilidades?

El Product Owner es el encargado de optimizar y maximizar el valor del producto, siendo la persona encargada de gestionar el flujo de valor del producto a través del Product Backlog. Adicionalmente, es fundamental su labor como interlocutor con los stakeholders y sponsors del proyecto, así como su faceta de altavoz de las peticiones y requerimientos de los clientes. Si el Product Owner también juega el rol de representante de negocio, su trabajo también aportará valor al producto.

Tradicionalmente, se ha entendido la labor del Product Owner como un gestor de requisitos o un cliente que se encarga de gestionar el Product Backlog, pero es mucho más que eso. No solo tiene la responsabilidad de mantener el Product Backlog bien estructurado, detallado y priorizado, sino que además tiene que entender perfectamente cuál es la deriva que se desea para el producto en todo momento, debiendo poder explicar y trasmitir a los stakeholders cuál es el valor del producto en el que están invirtiendo.

Con cada Sprint, el Product Owner debe hacer una inversión en desarrollo que tiene que producir valor. Marcar el Sprint Goal de manera clara y acordada con el equipo de desarrollo, hace que el producto vaya incrementando constantemente su valor.

Es fundamental otorgar el poder necesario al Product Owner para que este sea capaz de tomar cualquier decisión que afecte al producto. En el caso de que el

Product Owner no pueda tomar estas decisiones sin consultarlas previamente con otra persona, deberá ser investido para tomarlas él mismo, o ser sustituido por esa persona. A su vez, el Product Owner debe convertirse en el altavoz del cliente, en el transmisor de las demandas y del feeback otorgado por los mismos.

¿Quién es el Scrum Master? ¿Cuáles son sus funciones y responsabilidades?

El Scrum Master tiene dos funciones principales dentro del marco de trabajo: gestionar el proceso Scrum y ayudar a eliminar impedimentos que puedan afectar a la entrega del producto. Además, se encarga de las labores de mentoring y formación, coaching y de facilitar reuniones y eventos si es necesario.

1. **Gestionar el proceso Scrum:** el Scrum Master se encarga de gestionar y asegurar que el proceso Scrum se lleva a cabo correctamente, así como de facilitar la ejecución del proceso y sus mecánicas. Siempre atendiendo a los tres pilares del control empírico de procesos y haciendo que la metodología sea una fuente de generación de valor.

2. **Eliminar impedimentos:** esta función del Scrum Master indica la necesidad de ayudar a eliminar progresiva y constantemente impedimentos que van surgiendo en la organización y que afectan a su capacidad para entregar valor, así como a la integridad de esta metodología. El Scrum Master debe ser el responsable de velar porque Scrum se lleve adelante, transmitiendo sus beneficios a la organización facilitando su implementación.

Puede que el Scrum Master esté compartido entre varios equipos, pero su disponibilidad afectará al resultado final del proceso Scrum.

5.1.6 MEETINGS

¿Qué es el Sprint planning meeting?

El Sprint Planning es una reunión que se realiza al comienzo de cada Sprint donde participa el equipo Scrum al completo; sirve para inspeccionar el Backlog del Producto (Product Backlog) y que el equipo de desarrollo seleccione los Product Backlog Items en los que va a trabajar durante el

siguiente Sprint. Estos Product Backlog Items son los que compondrán el Sprint Backlog.

Durante esta reunión, el product owner presenta el Product *Backlog* actualizado que el equipo de desarrollo se encarga de estimar, además de intentar clarificar aquellos ítems que crea necesarios.

Si bien en el Sprint Planning participa el equipo Scrum al completo, no participan los stakeholders. En el Sprint Planning se inspeccionan el Product Backlog, los acuerdos de la Retrospectiva, la capacidad y la Definition of Done y se adaptan el Sprint Backlog, Sprint Goal y el plan para poder alcanzar ese Sprint Goal.

El Sprint Planning se divide en dos partes. En la primera parte de la reunión se trata Qué se va a hacer en el siguiente Sprint y, en la segunda parte, se discute el Cómo. La primera parte está organizada y liderada por el product owner, mientras que de la segunda parte se encarga el Development Team. La única labor del Scrum Master es asegurarse de que la reunión existe como parte de Scrum y que se mantiene dentro de las duraciones estimadas.

El Sprint Planning puede durar hasta 8 horas para Sprints de 4 semanas. En la práctica esta ceremonia suele llevar una mañana completa (alrededor de 5 horas) y, sólo si el producto o el Sprint definido por el Product Owner son complejos o están poco claros, se llegan a alcanzar las 8 horas definidas en la metodología. La razón del Sprint Planning es conseguir alineamiento entre negocio y desarrollo de producto en relación a las prioridades.

¿Qué es el Sprint review?

El Sprint Review es la reunión que ocurre al final del Sprint, generalmente el último viernes del Sprint, donde el product owner y el Develpment Team presentan a los stakeholders el incremento terminado para su inspección y adaptación correspondientes. En esta reunión organizada por el product owner se estudia cuál es la situación y se actualiza el Product Backlog con las nuevas condiciones que puedan afectar al negocio.

El equipo ha pasado hasta cuatro semanas desarrollando un incremento terminado de software que ahora mostrará a los stakeholders. No se trata de una demostración, sino de una reunión de trabajo. El software ya ha sido mostrado y validado junto con el product owner previamente a esta reunión.

Por un lado, se revisará el incremento terminado. Se mostrará el software funcionando en producción y los stakeholders tendrán la oportunidad de hacer cuantas preguntas estimen oportunas sobre el mismo. El software

funcionando ha sido validado previamente por el product owner, que se ha encargado de trabajar con el equipo durante el Sprint para asegurarse que cumple con la Definition of Done y, efectivamente, hace que el Sprint Goal sea válido. Si no existe software funcionando, el Sprint Review carece de sentido, aunque en ciertas ocasiones y oportunidades se sigue manteniendo.

El Development Team tiene que tener un papel importante en esta reunión. Muchas veces no es el product owner quien demuestra el incremento producido, sino que son los propios miembros del Development Team quienes lo hacen. Es una buena práctica no sólo el que lo lleven a cabo, sino también el que lo hagan de forma rotatoria y, tras varios Sprints, hayan participado todos.

El equipo de desarrollo comenta posteriormente qué ha ocurrido durante el Sprint, los impedimentos que se han encontrado, así como soluciones tomadas y actualizan a los stakeholders con la situación del equipo. Por último, el product owner actualiza -con la información de negocio recibida en esta reunión- el Product Backlog para el siguiente Sprint.

En contra de lo que comúnmente se cree, el Sprint Review no se trata de una demo para un cliente o para los stakeholders o incluso para el product owner, ni es tampoco una reunión para felicitar al Equipo de Desarrollo. Es una reunión de trabajo, una de las más importantes porque sirve para marcar la estrategia de negocio. La duración estimada en el estándar para un Sprint Review es de 8 horas para un Sprint de 4 semanas, aunque habitualmente estas reuniones se ejecutan en un entorno de entre 2 y 3 horas.

¿Qué es el Sprint retrospective?

La retrospectiva ocurre al final del Sprint, justo después del Sprint Review. En algunos casos y por comodidad de los equipos, se realiza conjuntamente con el Sprint Planning, siendo la retrospectiva la parte inicial de la reunión.

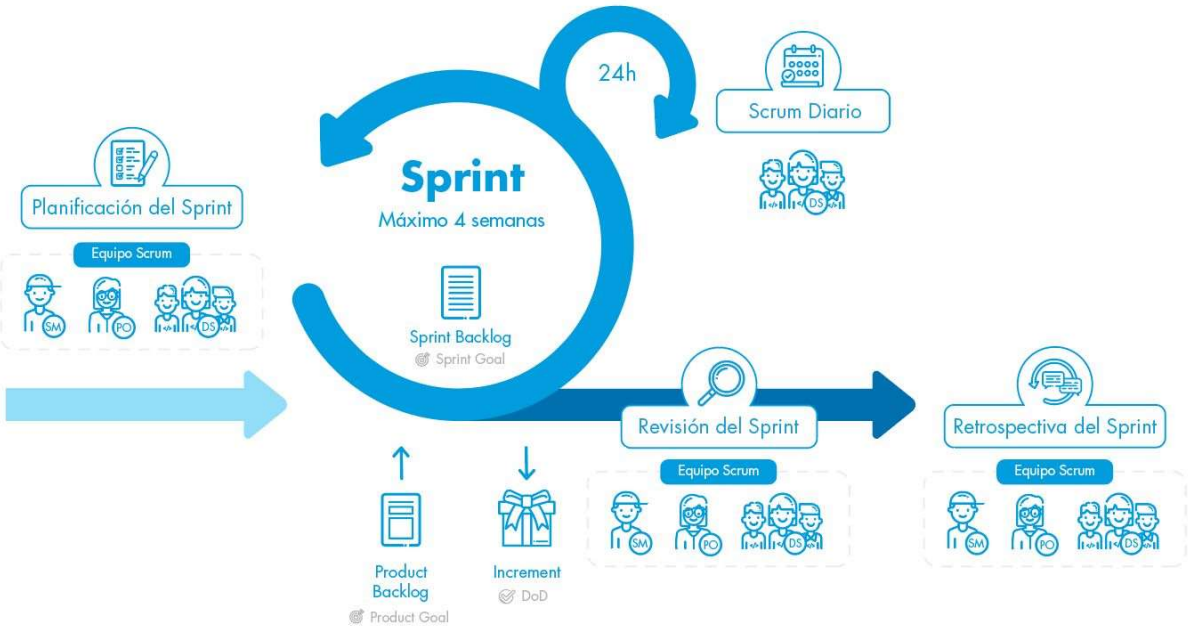
El objetivo de la retrospectiva es hacer de reflexión sobre el último Sprint e identificar posibles mejoras para el próximo. Aunque lo habitual es que el Scrum Master sea el facilitador, es normal que distintos miembros del equipo Scrum vayan rotando el rol de facilitador durante la retrospectiva.

Un formato común es analizar qué ha ido bien durante el Sprint, qué ha fallado y qué se puede mejorar. Este formato se puede facilitar pidiendo a los miembros del equipo Scrum que escriban notas en post-its para luego agruparlas y votar aquellos ítems más relevantes, dando la oportunidad a todos de hablar y expresar sus inquietudes.

También se utiliza el formato de retrospectiva basado en cinco fases:

1. **Preparar el ambiente:** un pequeño ejercicio para romper el hielo.
2. **Recolectar información:** durante esta fase, se utilizan actividades para intentar construir una imagen de lo que ha sido el último Sprint, resultando una imagen conjunta de equipo.
3. **Generación de ideas:** el equipo intenta generar ideas para identificar acciones que ayuden a mejorar el rendimiento del equipo durante el siguiente Sprint.
4. **Decidir qué hacer:** de las ideas generadas, se proponen acciones que el equipo pueda implementar en el próximo Sprint.
5. **Cierre:** Una pequeña actividad de cierre, normalmente unida a una evaluación de la propia retrospectiva, ayuda al equipo a decidir hacia dónde dirigirse en próximas ocasiones. Un recordatorio de la mejora continua.

La duración recomendada por Scrum para un Sprint de 4 semanas es de un máximo de 3 horas, aunque habitualmente se destina entre 1 y 2 horas a este evento



.

5.2 Esquema Sprints

Sprint 1

- Analisis y diseño: Crear plantilla documendo
- Analisis: Requisitos funcionales
- Analisis: Requisitos no funcionales
- Analisis: Casos de uso
- Diseño: Diagrama de secuencias
- Diseño: Diagrama de clases
- Plan de pruebas: Crear plantilla document
- Plan de pruebas: Actualizar document resultado tests

Sprint 2

- Memoria: Crear plantilla documento
- Memoria: Metodologia, crear guía Scrum
- Memoria: Metodologia, crear esquemas Sprints

5.2.1 Sprint 1 - Planificado

ISSUE ID	TYPE	DESCRIPTION	ESTADO
Iss#1	Análisis y diseño	Crear plantilla documento	TO DO
Iss#2	Análisis	Requisitos funcionales	TO DO
Iss#3	Análisis	Requisitos no funcionales	TO DO
Iss#4	Análisis	Casos de uso	TO DO
Iss#5	Diseño	Diagrama de secuencias	TO DO
Iss#6	Diseño	Diagrama de clases	TO DO
Iss#7	Plan de pruebas	Crear plantilla documento	TO DO
Iss#8	Plan de pruebas	Actualizar documento resultado tests	TO DO

5.2.2 Sprint 1 – Resultado

ISSUE ID	TYPE	DESCRIPTION	ESTADO
Iss#1	Análisis y diseño	Crear plantilla documento	DONE
Iss#2	Análisis	Requisitos funcionales	DONE
Iss#3	Análisis	Requisitos no funcionales	DONE
Iss#4	Análisis	Casos de uso	DONE
Iss#5	Diseño	Diagrama de secuencias	DONE
Iss#6	Diseño	Diagrama de clases	DONE

Iss#7	Plan de pruebas	Crear plantilla documento	DONE
Iss#8	Plan de pruebas	Actualizar documento resultado tests	DONE

5.2.3 Sprint 2 – Planificado

ISSUE ID	TYPE	DESCRIPTION	ESTADO
Iss#9	Memoria	Metodología, crear plantilla documento	TO DO
Iss#10	Memoria	Metodología, crear guía Scrum	TO DO
Iss#11	Memoria	Metodología, crear esquemas Sprints	TO DO

5.2.4 Sprint 1 – Resultado

ISSUE ID	TYPE	DESCRIPTION	ESTADO
Iss#9	Memoria	Metodología, crear plantilla documento	DONE
Iss#10	Memoria	Metodología, crear guía Scrum	DONE
Iss#11	Memoria	Metodología, crear esquemas Sprints	DONE