# UD 4
# PRUEBAS, DOCUMENTACIÓN Y DISTRIBUCIÓN DE APLICACIONES

## DOCUMENTACIÓN

### INTRODUCCIÓN

La documentación de los programas es un aspecto sumamente importante, tanto en el desarrollo de la aplicación como en el mantenimiento de la misma. Mucha gente no hace este parte del desarrollo y no se da cuenta de que pierde la posibilidad de la reutilización de parte del programa en otras aplicaciones, sin necesidad de conocerse el código al dedillo.

La documentación de un programa empieza a la vez que la construcción del mismo y finaliza justo antes de la entrega del programa o aplicación al cliente. Así mismo, la documentación que se entrega al cliente tendrá que coincidir con la versión final de los programas que componen la aplicación.

La documentación que se entrega al cliente se divide claramente en dos categorías, interna y externa:

- **Interna**: es aquella que se crea en el mismo código, ya puede ser en forma de comentarios o de archivos de información dentro de la aplicación.
- **Externa**: es totalmente ajena a la aplicación en sí (manuales o guías que no estén incluidos en la propia aplicación).

### DOCUMENTACIÓN INTERNA

Los tres elementos más significativos de la documentación interna son:

- **Elección de nombres significativos**: la elección de nombres significativos para los identificadores (tanto de constantes, como variables, funciones...) es crucial para que un programa sea inteligible. El nombre de los identificadores debe elegirse de forma que no deje lugar a duda sobre su objetivo o el significado del valor que va a contener.
- **Comentarios**: la posibilidad de expresar comentarios en lenguaje natural dentro del código fuente es algo que aparece en todos los lenguajes de programación. Los comentarios permiten al programador comunicarse con otros lectores del código, resultando una clara guía de comprensión durante las etapas de mantenimiento.
  Se pueden distinguir, básicamente, dos tipos de comentarios:
  - Los **comentarios de prólogo** deben aparecer al principio de cada programa o subprograma y deben incluir:

    Propósito de la función del módulo.

    Descripción de la interfaz

    Datos del autor y fecha de modificación
  - Los **comentarios descriptivos** se insertan dentro del código para explicar algún trozo complicado de éste.
- **Indentación**: sangrado del código que supone una importante contribución a la legibilidad del mismo.

### DOCUMENTACIÓN EXTERNA

Dentro de la documentación externa existen diferentes documentos (no siempre son necesarios todos):

- Guía técnica

- Manual de instalación y explotación
- Manual de usuario
- Guía de referencia
- Guía rápida

## GUÍA O MANUAL TÉCNICO

En la guía o manual técnico se reflejan el **diseño** del proyecto, la **codificación** de la aplicación y las **pruebas** realizadas para su correcto funcionamiento. Generalmente este documento está diseñado para personas con conocimientos de informática, generalmente programadores.

El principal objetivo es el de facilitar el desarrollo, corrección y futuro mantenimiento de la aplicación de una forma rápida y fácil.

## MANUAL DE INSTALACIÓN Y EXPLOTACIÓN

El manual de instalación y explotación contiene la información necesaria para instalar, utilizar y explotar la aplicación, es decir, va orientado a la instalación, configuración, puesta en marcha y su posterior mantenimiento.

No hay una norma para su elaboración, pero debe contener al menos los siguientes puntos:

- Requerimientos mínimos del sistema en cuanto a procesador, memoria, espacio libre…
- Proceso de instalación:
    - Necesidad o no de preparación previa, si es necesaria alguna tarea para ello (por ejemplo, habilitar conexiones, activar o desactivar permisos…)
    - Opciones de instalación: a través de USB, red…
    - Procedimiento de instalación paso a paso, detallado y explicando sus diferentes opciones
- Actualizaciones del sistema y copias de seguridad

En función de la complejidad de la aplicación puede dividirse en varios manuales:

- Manual de instalación
- Manual de configuración
- Manual de explotación

## MANUAL DE USUARIO

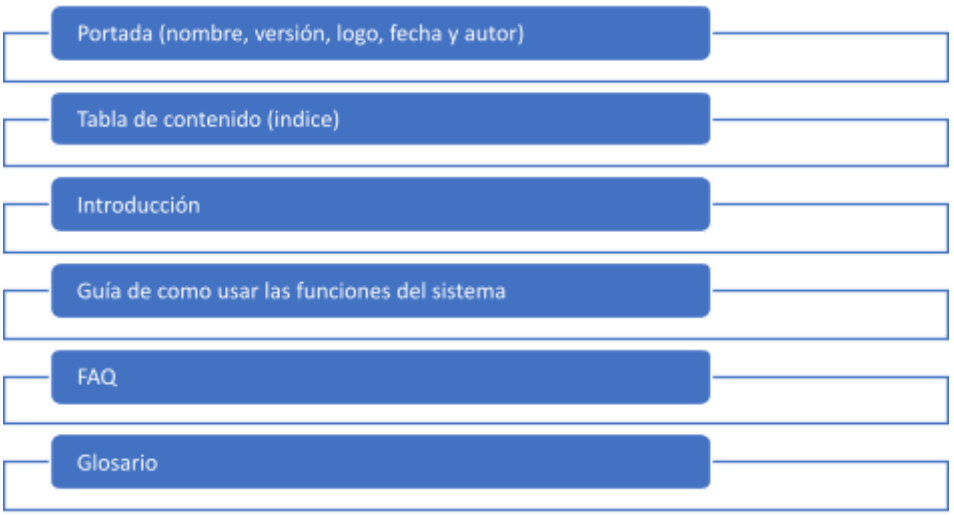
El manual de usuario contiene toda la información necesaria con el fin de hacer más fácil el uso y la comprensión del programa.

Entre las recomendaciones necesarias para la elaboración de un manual de usuario de una aplicación o un proceso se encuentran:

- Debe tener una portada donde deberían aparecer los siguientes elementos:
    - Nombre y versión de la aplicación (mejor si va acompañada del logotipo de la aplicación)
    - Fecha de realización
    - Autor o autores del manual
- Debe contar con un índice de contenido para facilitar su manejo e identificación de los puntos importantes (con número de páginas y links).
- Debe proporcionar una breve descripción general del programa o proceso e indicar a quién se dirige la información (introducción).
- Debe hacer uso de imágenes o capturas de pantalla, para describir el uso y funcionamiento de cada una de las secciones que conforman la aplicación, con el fin de ayudar a entender el texto.

- El lenguaje utilizado debe ser fácil de entender para los usuarios a los que va dirigido.
- Puede incorporar un apartado de preguntas frecuentes.
- Puede incorporar una lista con el significado de los términos, conceptos o tecnicismos usados en el manual y que no sean de dominio público.
- Puede incluir como conseguir soporte técnico (personas de contacto, webs, libros, ayuda de los programas, …).

Una posible estructura del manual de usuario puede ser la siguiente:




### GUÍA RÁPIDA

La guía rápida se orienta a usuarios del sistema y/o encargados de mantenimiento. En función de la complejidad de la aplicación puede ser necesario realizar varias, cada una con distinta temática.

Las guías rápidas aportan información muy concreta y muy diferente, relacionada con diversos procedimientos o campos de una aplicación.

### GUÍA DE REFERENCIA

Al contrario que en el caso de las guías rápidas, las guías de referencias suelen estar pensadas para usuarios experimentados.

Por ello, es habitual que contengan información relacionadas con aspectos más técnicos:

- Mensajes de error y como recuperarse de ellos.
- Guía con comandos para usar la aplicación en modo comando.
- Tipos de datos de entrada que son permitidos en la aplicación.

## ACCESO A LA DOCUMENTACIÓN DESDE LA APLICACIÓN

Se puede tener acceso a parte de la documentación desde la propia aplicación.

Para ello vamos a ver algunas formas de acceso a la documentación:

- Acceder a la documentación en pdf dentro de los archivos de nuestra aplicación.
- Acceder a la documentación alojada en un sitio web.
- Acceder a la documentación en html dentro de los archivos de nuestra aplicación.
- Los tooltips también permiten proporcionar algunas indicaciones.

## ACCEDER A LA DOCUMENTACIÓN EN PDF DENTRO DE LOS ARCHIVOS DE NUESTRA APLICACIÓN

Vamos a ver como se hace con un ejemplo en el que tenemos una ventana principal con un elemento de menú que es Ayuda y al pinchar sobre este elemento se abrirá un visor pdf con la documentación que está dentro de los archivos de nuestra aplicación.

Vamos a necesitar una librería para poder ver los pdf: **jPDFViewerFX.jar**

La librería es Qoppa JavaFX Viewer que es de pago (por lo que nos aparecerá una marca de agua en las páginas de nuestro pdf).

Primero tenemos que crear los archivos pdf con la documentación y guardarlos dentro de nuestro proyecto:


En module-info aparte de las entradas necesarias para JavaFX necesitamos añadir:

```
requires javafx.swing;
requires jPDFViewerFX;
```

En Scene Builder creamos la siguiente ventana:

Principal.fxml
En el OnAction del MenuItem Ayuda hemos añadido el método abrirAyuda

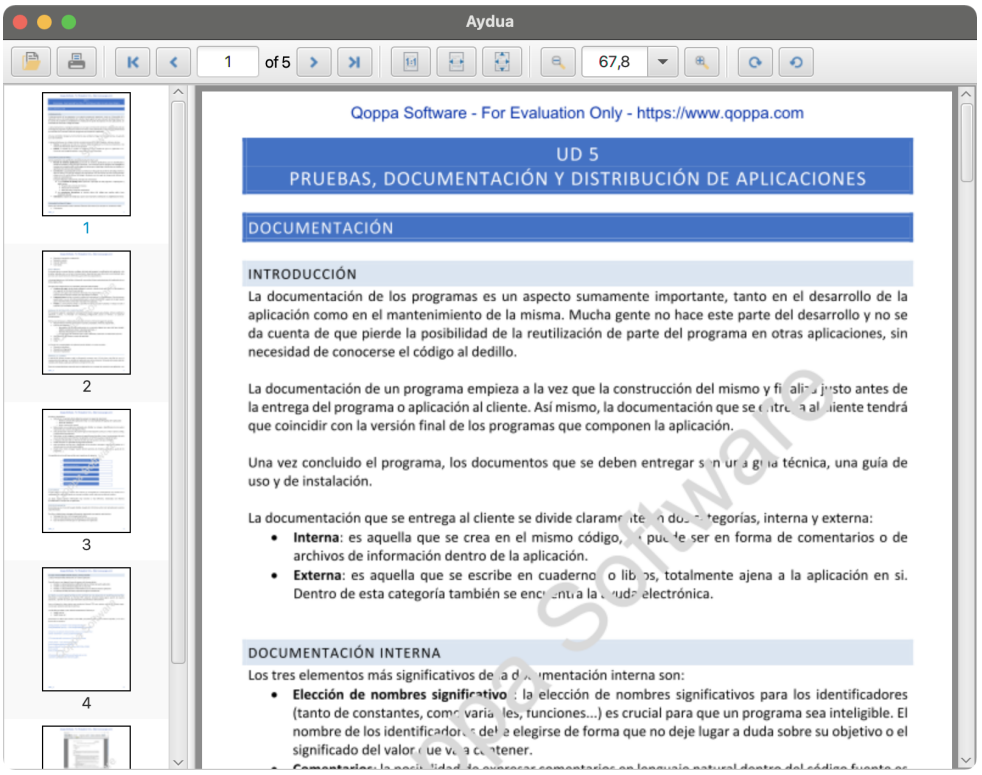

La programación del controlador será la siguiente:

PrincipalController.java

```
public class PrincipalController {
        private PDFViewer visorPDF;

        @FXML
```

```
    public void abrirAyuda(ActionEvent event) {
        visorPDF = new PDFViewer();
        try {
            visorPDF.loadPDF(getClass().getResource("/pdf/prueba.pdf"));
            Stage stage = new Stage();
            BorderPane borderPane = new BorderPane(visorPDF);
            Scene scene = new Scene(borderPane);
            stage.setTitle("Aydua");
            stage.setScene(scene);
            stage.centerOnScreen();
            stage.show();
        } catch (PDFException e) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setHeaderText(null);
            alert.setContentText(e.getMessage());
            alert.showAndWait();
        }
    }
}
```

Al pinchar en ayuda se abrirá la siguiente pantalla:




---

ACCEDER A LA DOCUMENTACIÓN ALOJADA EN UN SITIO WEB

Vamos a ver como se hace con un ejemplo en el que tenemos una ventana principal con un menú con un elemento que es ayuda, al pinchar sobre este elemento se abrirá el sitio web donde se encuentra la ayuda de nuestra aplicación:

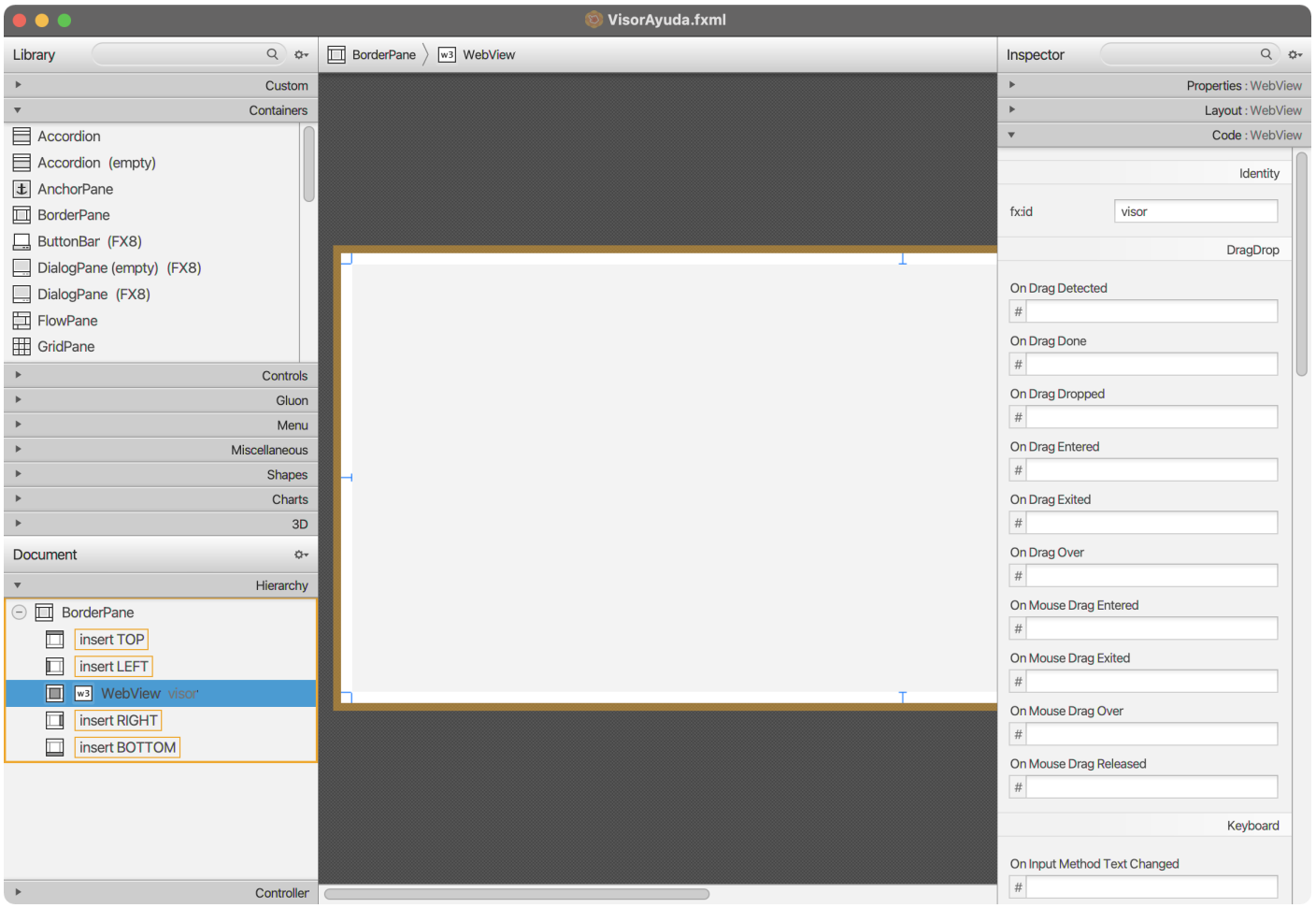
En module-info aparte de las entradas necesarias para JavaFX necesitamos añadir:

```
requires javafx.web;
```

En Scene Builder creamos las siguientes ventanas:

Principal.fxml
En el OnAction del MenuItem Ayuda hemos añadido el método abrirAyuda


VisorAyuda.fxml
En un BordePane que tiene dentro un WebView




La programación de los controladores será la siguiente:

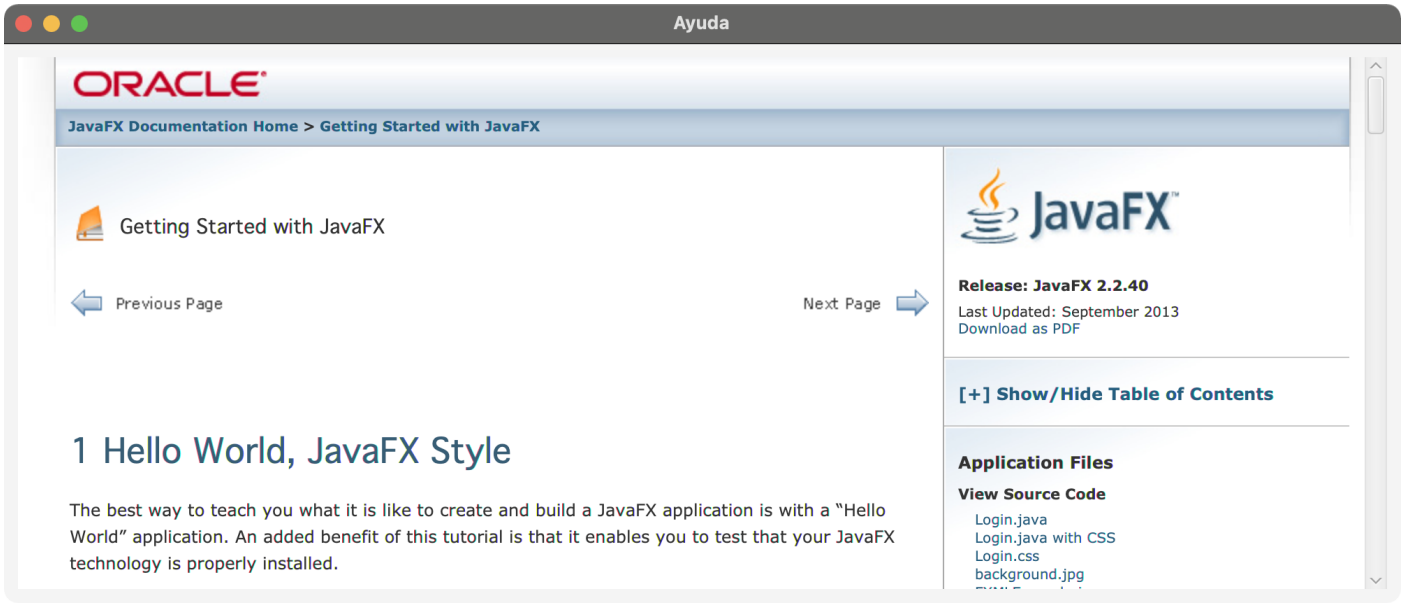
PrincipalController.java

```
public class PrincipalController {
	@FXML
	public void abrirAyuda(ActionEvent event) {
		FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/VisorAyuda.fxml"));
		Parent root;
		try {
			root = loader.load();
			Scene scene = new Scene(root);
			Stage stage = new Stage();
			stage.setScene(scene);
			stage.setTitle("Ayuda");
			stage.show();
		} catch (IOException e) {
			Alert alert = new Alert(Alert.AlertType.ERROR);
			alert.setHeaderText(null);
			alert.setContentText(e.getMessage());
			alert.showAndWait();
		}
	}
}
```

VisorAyudaController.java

```
public class VisorAyudaController implements Initializable {
	@FXML
	private WebView visor;
	private WebEngine webEngine;

	@Override
	public void initialize(URL arg0, ResourceBundle arg1) {
		webEngine = visor.getEngine();
		webEngine.load("https://docs.oracle.com/javafx/2/get_started/hello_world.htm");
	}
}
```

Al pinchar en ayuda se abrirá la siguiente pantalla:



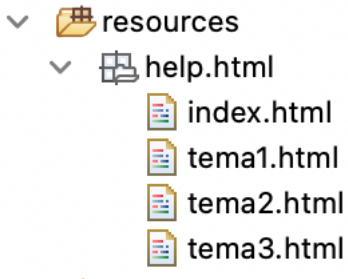

## ACCEDER A LA DOCUMENTACIÓN EN HTML DENTRO DE LOS ARCHIVOS DE NUESTRA APLICACIÓN

Vamos a ver como se hace con un ejemplo en el que tenemos una ventana principal con un menú con un elemento que es ayuda, al pinchar sobre este elemento se abrirán la documentación en html dentro de los archivos de nuestra aplicación:

En **module-info** aparte de las entradas necesarias para JavaFX necesitamos añadir:

```
requires javafx.web;
```

Primero tenemos que crear los archivos html con la documentación y guardarlos dentro de nuestro proyecto:




Vamos a crear una clase help donde cada objeto tendrá tres atributos: Título, Fichero html, booleano que indica si el fichero es local o no.

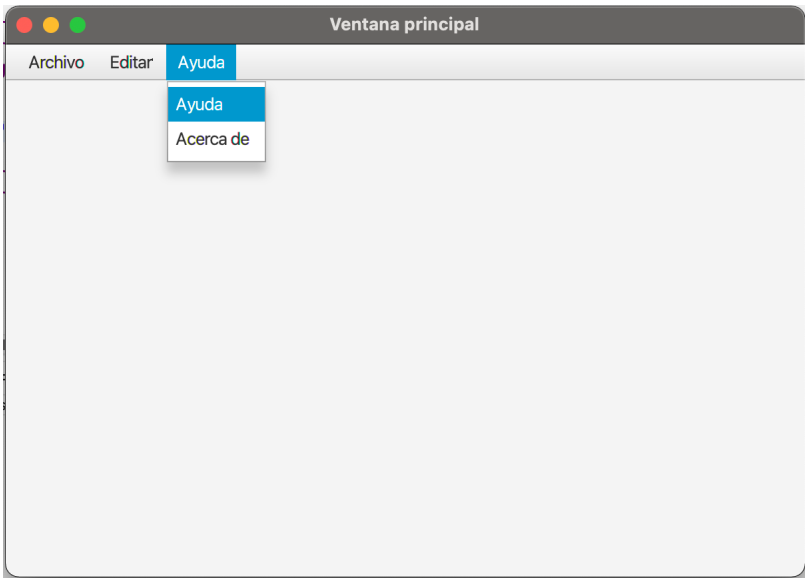
Help.java

```
public class Help {
        private String text;
        private String html;
        private boolean local = true;

        public Help(String text, String html) {
                this.text = text;
                this.html = html;
        }
```

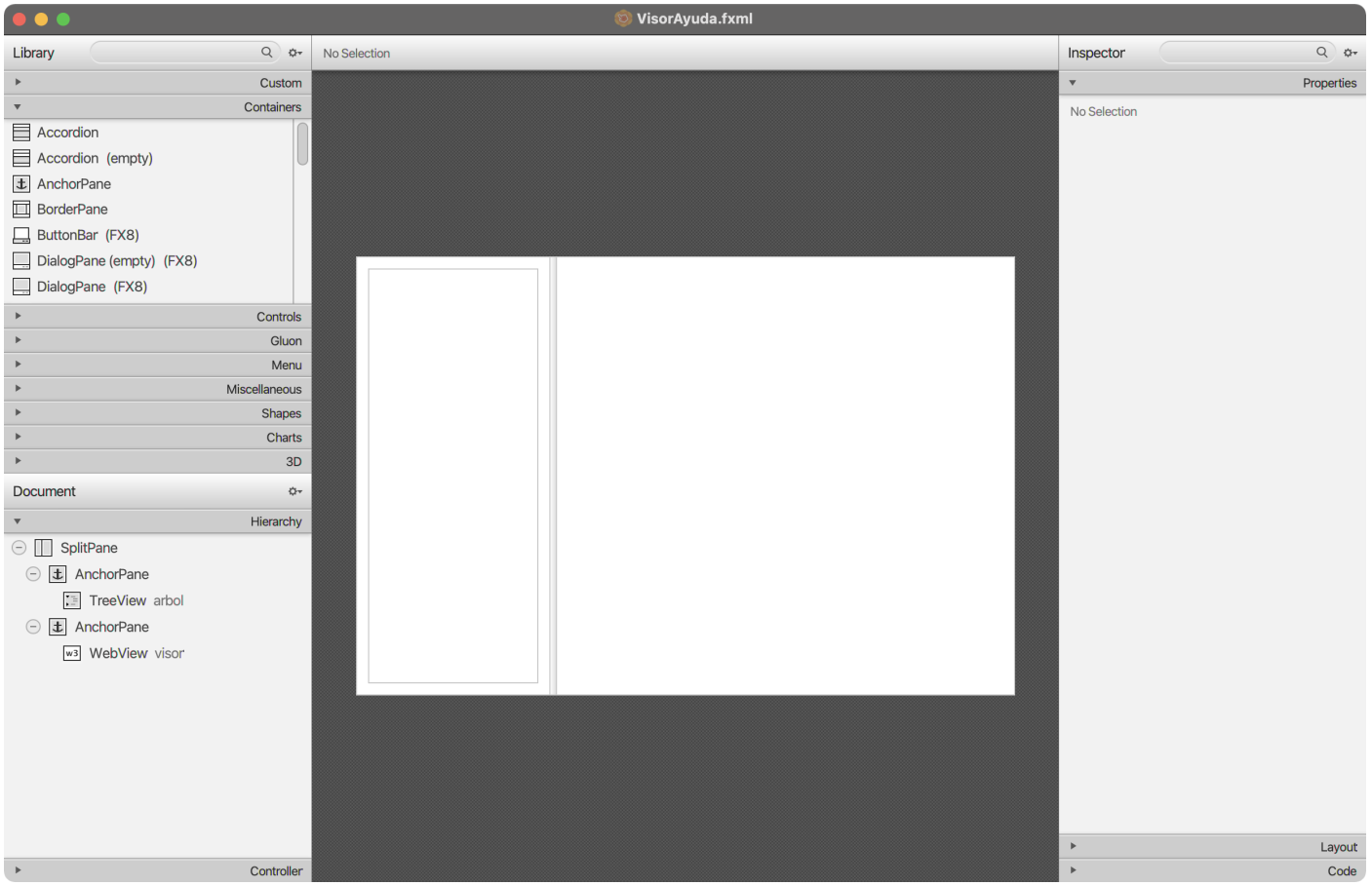
```
    public Help(String text, String html, boolean local) {
        this.text = text;
        this.html = html;
        this.local = local;
    }

    public String getText() {
        return text;
    }

    public String getHtml() {
        return html;
    }

    public boolean isLocal() {
        return local;
    }

    @Override
    public String toString() {
        return text;
    }
}
```

En Scene Builder creamos las siguientes ventanas:

Principal.fxml
En el OnAction del MenuItem Ayuda hemos añadido el método abrirAyuda




VisorAyuda.fxml




La programación de los controladores será la siguiente:

PrincipalController.java

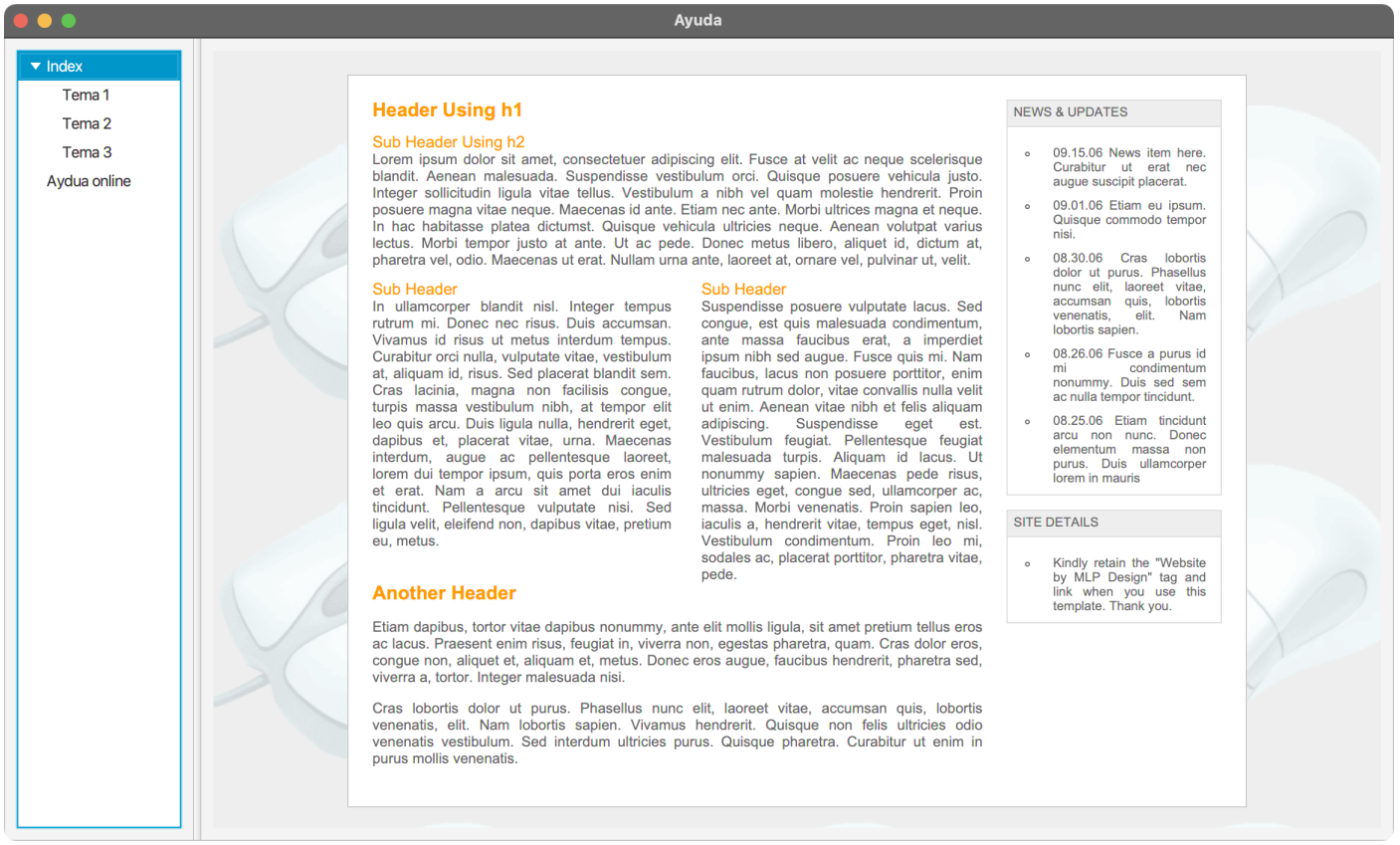
```
public class PrincipalController {
	@FXML
	public void abrirAyuda(ActionEvent event) {
		FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/VisorAyuda.fxml"));
		Parent root;
		try {
			root = loader.load();
			Scene scene = new Scene(root);
			Stage stage = new Stage();
			stage.setScene(scene);
			stage.setTitle("Ayuda");
			stage.show();
		} catch (IOException e) {
			Alert alert = new Alert(Alert.AlertType.ERROR);
			alert.setHeaderText(null);
			alert.setContentText(e.getMessage());
			alert.showAndWait();
		}
	}
}
```

VisorAyudaController.java

```
public class VisorAyudaController implements Initializable {
```

```
	@FXML
	private TreeView<Help> arbol;
	@FXML
	private WebView visor;
	private WebEngine webEngine;

	@Override
	public void initialize(URL arg0, ResourceBundle arg1) {
		//Creamos el árbol del panel de la izquierda, el índice
		TreeItem<Help> rItem = new TreeItem<Help>(new Help("Raiz", ""));
		TreeItem<Help> r1Item = new TreeItem<Help>(new Help("Index", "index.html"));
		r1Item.getChildren().add(new TreeItem<Help>(new Help("Tema 1", "tema1.html")));
		r1Item.getChildren().add(new TreeItem<Help>(new Help("Tema 2", "tema2.html")));
		r1Item.getChildren().add(new TreeItem<Help>(new Help("Tema 3", "tema3.html")));
		TreeItem<Help> r2Item = new TreeItem<Help>(new Help("Aydua online", "http://www.google.es", false));
		rItem.getChildren().add(r1Item);
		rItem.getChildren().add(r2Item);
		arbol.setRoot(rItem);
		arbol.setShowRoot(false);
		//Mostramos el contenido inicial en el visor de la derecha
		webEngine = visor.getEngine();
		webEngine.load(getClass().getResource("/help/html/index.html").toExternalForm());
		//Añadimos un evento para cambiar de html al pinchar en el árbol
		arbol.setOnMouseClicked(e -> {
			if (arbol.getSelectionModel().getSelectedItem() != null) {
				Help elemento = (Help) arbol.getSelectionModel().getSelectedItem().getValue();
				if (elemento.getHtml() != null) {
					loadHelp(elemento.getHtml(), elemento.isLocal());
				}
			}
		});
	}

	private void loadHelp(String file, boolean local) {
		if (visor != null) {
			if (local) {
				webEngine.load(getClass().getResource("/help/html/" + file).toExternalForm());
			} else {
				webEngine.load(file);
			}
		}
	}
}
```

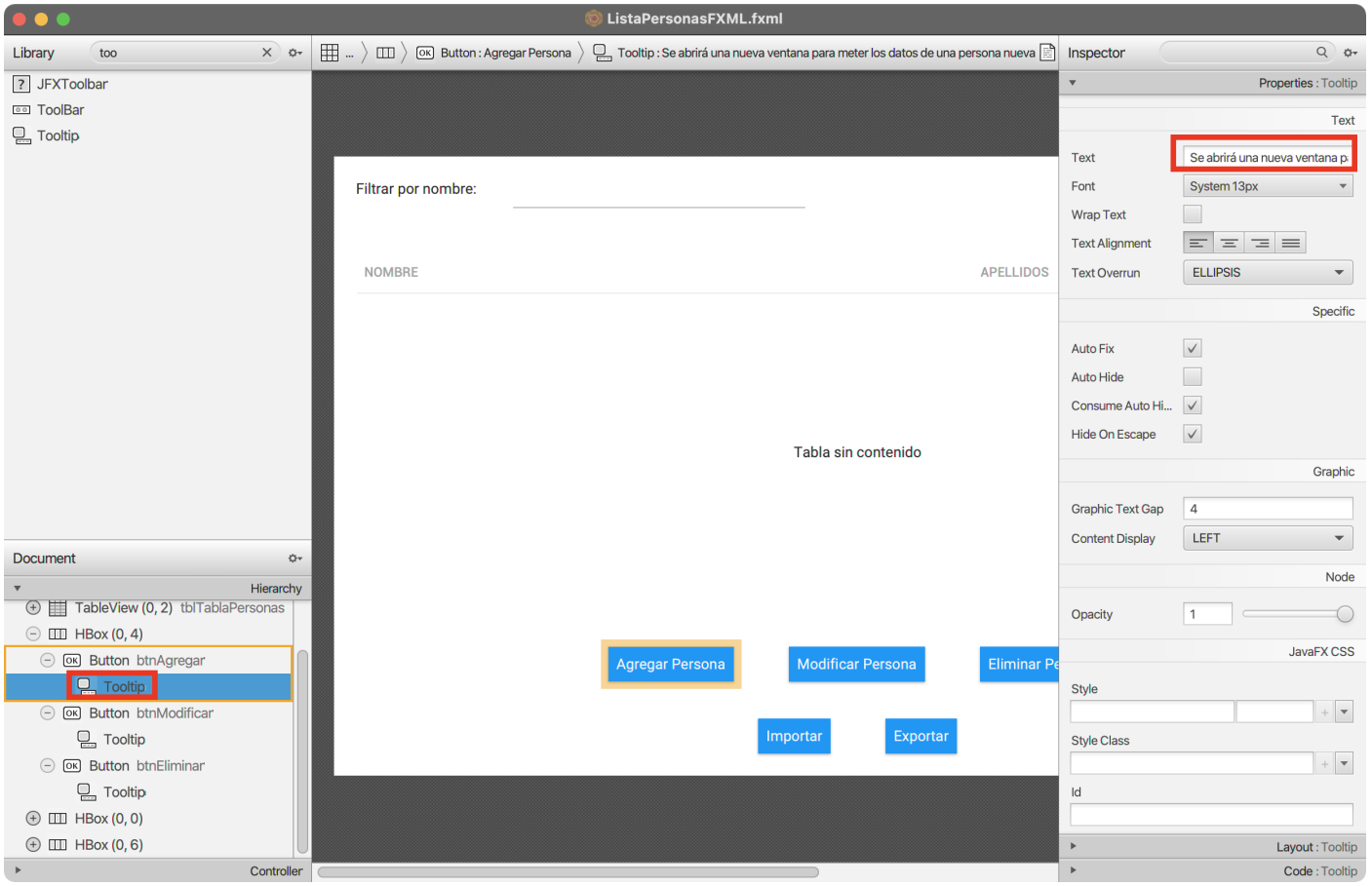
Al pinchar en ayuda se abrirá la siguiente pantalla:



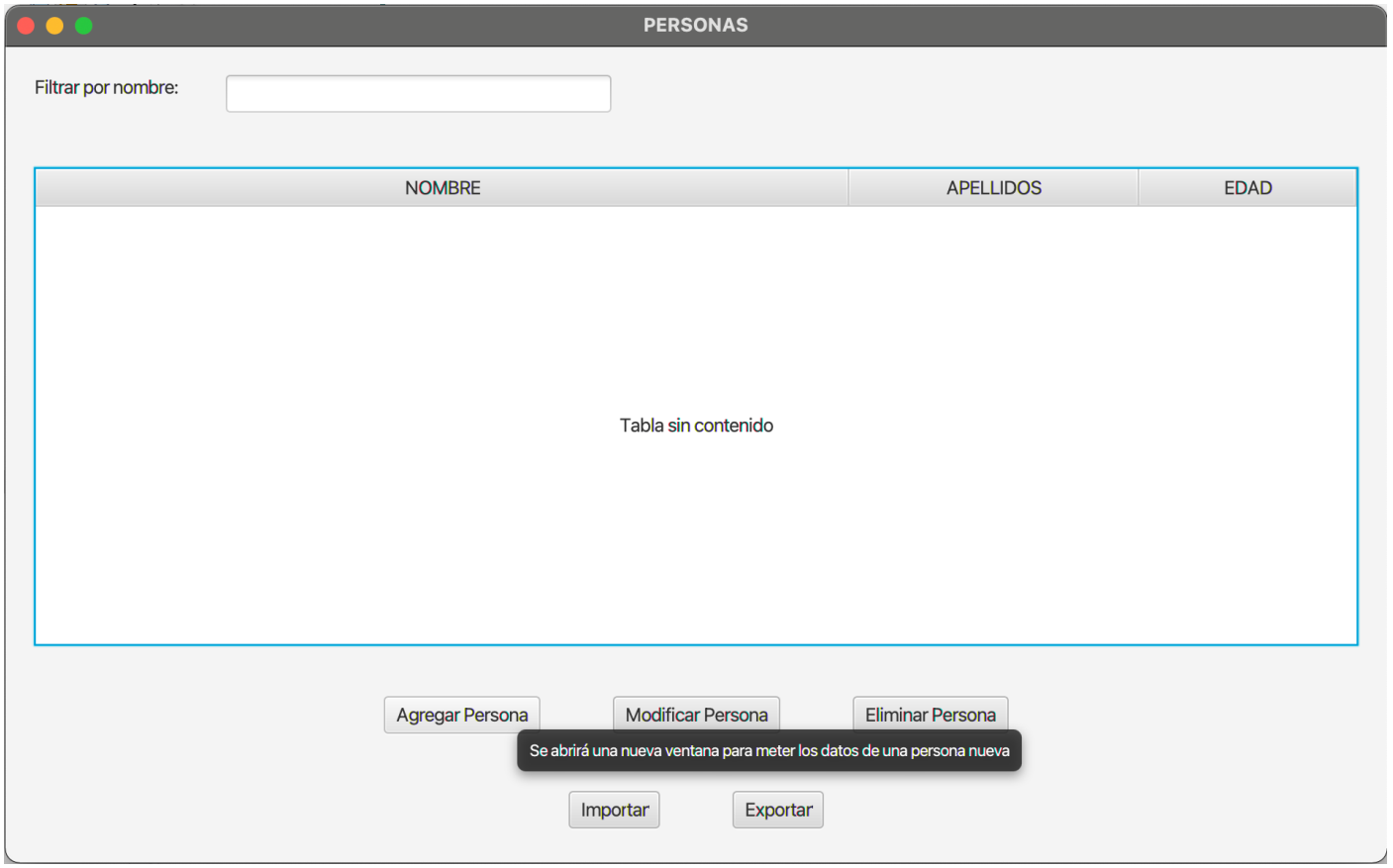

## TOOLTIPS

Un tooltip o globo (también llamado descripción emergente o texto alternativo) es una herramienta de ayuda visual, que funciona al situar el cursor sobre algún elemento gráfico, mostrando una ayuda adicional para informar al usuario de la finalidad del elemento sobre el que se encuentra.

En SceneBuilder tenemos un componente ToolTip que tenemos que añadir a los componentes para los cuales queremos añadir esta ayuda adicional:




Al ejecutar la aplicación al situar el cursor en el botón nos saldrá el siguiente mensaje:



# PRUEBAS

## INTRODUCCIÓN

La **calidad del software** que produzcamos es un requisito indispensable y calidad no es sólo que el software funcione bien y que vaya rápido, sino que además implica que satisfaga las expectativas de los clientes, **usabilidad**, **confiabilidad**, coste, **eficiencia** y cumplimiento de los estándares.

Para asegurar la calidad, debemos hacer uso de las **pruebas de software**.

Las pruebas deben entenderse como:

- **Verificación**: el producto que se está construyendo funciona correctamente; es decir, es capaz de realizar la tarea para la cual ha sido diseñado.
- **Validación**: el producto terminado, además de ser correcto, es conforme con lo que el cliente había esperado.

La realización de pruebas de software consume tiempo y recursos, pero es fundamental para que los fallos de la aplicación sean mínimos.

Las pruebas deben formar parte del presupuesto, la planificación y la asignación de recursos del proyecto. El coste económico relativo a un error se multiplica por un factor de 1 si se comete en la fase de análisis de requisitos, por 3-6 si es en la fase de diseño, por 10 si es en la codificación y por 40-50 si es en la fase de pruebas.

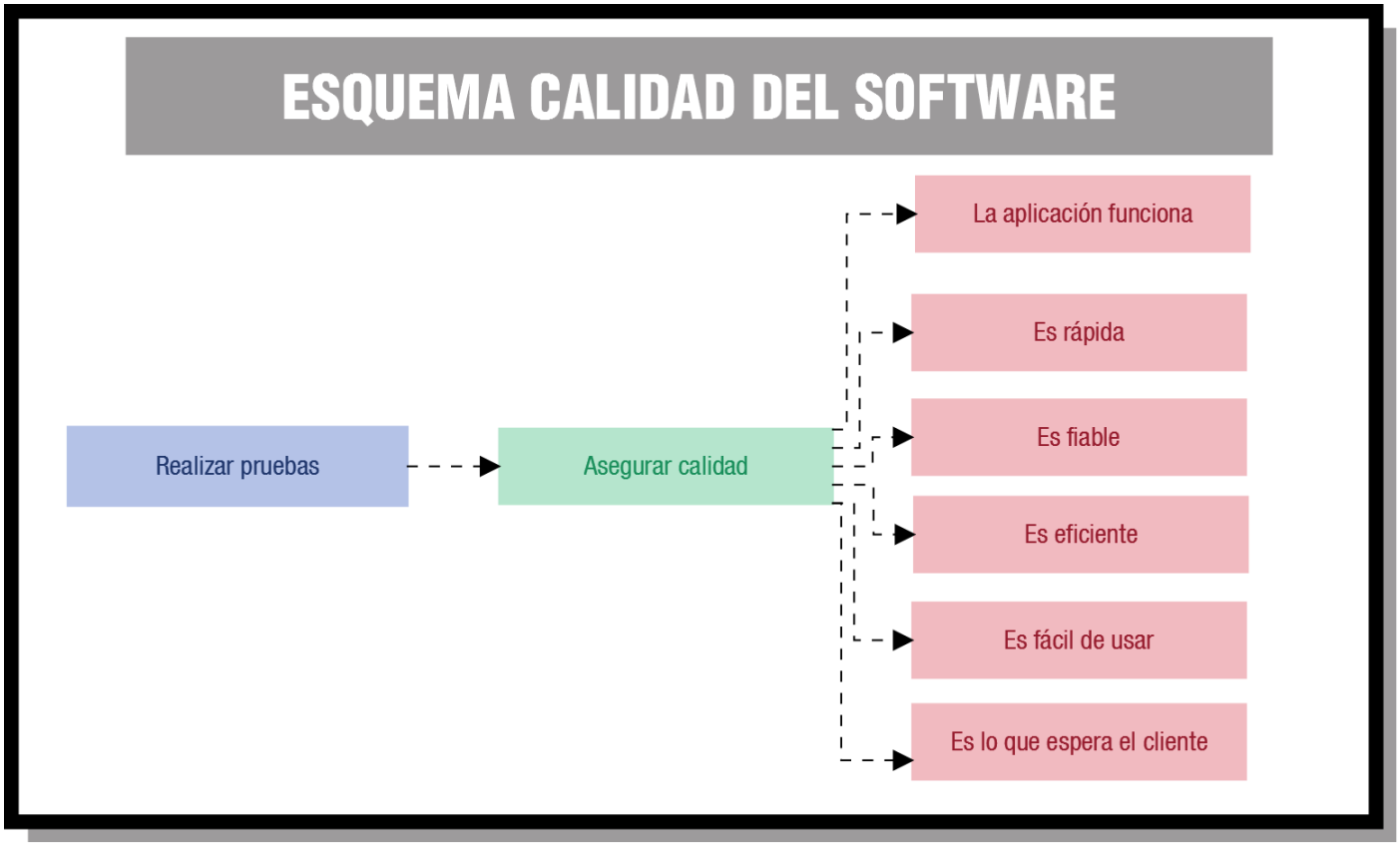
El objetivo de la estrategia de pruebas es determinar los tipos de pruebas a realizar, el calendario de estas, los responsables, los recursos asignados y el alcance de las pruebas.

La ausencia o la mala gestión de una estrategia de pruebas en un proyecto software pueden suponer:

- Falta de satisfacción de los clientes.
- Fallos en la explotación por defectos no detectados y eliminados a tiempo.
- Altos costes finales, por el tiempo empleado en corregir defectos.

Las pruebas de los sistemas de información no pueden realizarse de forma improvisada y sin planificar, por eso, es recomendable tener en cuenta una serie de principios a la hora de llevarlas a cabo:

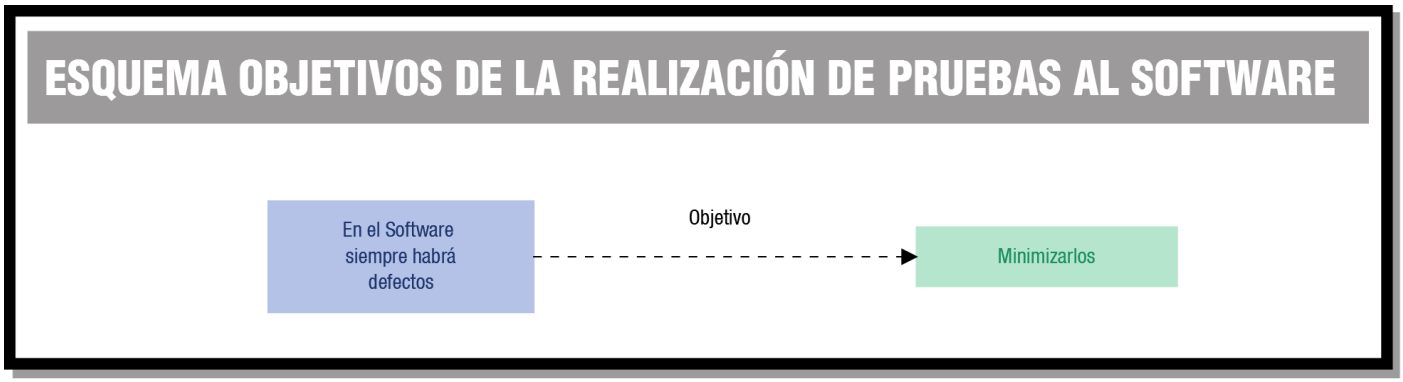
- A todas las pruebas se les debería poder hacer un seguimiento.
- Las pruebas deberían planificarse mucho antes de que empiecen: la planificación de las pruebas puede empezar tan pronto como esté completo el modelo de requisitos.
- Las pruebas deberían empezar por «lo pequeño» y progresar hacia «lo grande».
- No son posibles las pruebas exhaustivas. El número de permutaciones de caminos para incluso un programa de tamaño moderado es excepcionalmente grande. Por este motivo, es imposible ejecutar todas las combinaciones de caminos durante las pruebas.
- Para ser más eficaces, las pruebas deberían ser realizadas por un equipo independiente.




## CASOS DE PRUEBA

Un **caso de prueba** es "un conjunto de entradas, condiciones de ejecución y resultados esperados".

Las pruebas están totalmente condicionadas por un hecho fundamental: es imposible probar exhaustivamente el software, es decir, probar todas las posibles combinaciones que se pueden dar. Este factor se agudiza hasta el extremo en los entornos gráficos GUI, donde el usuario tiene un papel impredecible.




Ya que no se pueden probar todas las posibilidades de funcionamiento del software, la idea fundamental para el diseño de los casos de prueba consiste en elegir algunas de ellas que por sus características se consideren representativas del resto. De esta manera, aunque no se prueben todas las posibilidades, se puede tener un cierto nivel de confianza en la fiabilidad del programa.

Un buen caso de prueba es aquel que:

- Tiene una alta probabilidad de detectar algún error no encontrado hasta ahora.
- No es redundante.
- Es representativo.

- No es, ni muy simple ni muy complejo.

Para cada uno de estos casos de prueba se debe especificar:

- El componente a probar y la información de contexto (descripción)
- Los datos de entrada.
- El resultado esperado.

| CÓDIGO | DESCRIPCIÓN | DATOS DE ENTRADA | RESULTADO ESPERADO | FECHA | ESTADO |
|---|---|---|---|---|---|
| PRUEBA 1 | Dar de alta un usuario con datos válidos | Nombre de usuario: Ainara<br>Apellido: Montoya<br>Edad: 20 | Da de alta el usuario en la base de dato.<br>Muestra al usuario un mensaje indicando que se ha dado de alta.<br>Se refresca la tabla de usuarios en la aplicación. | 26/12/2021 | Correcto |
| PRUEBA 2 | Dar de alta un usuario con datos no válidos | Nombre de usuario: Ainara<br>Apellido: Montoya<br>Edad: c | Informa al usuario de que los datos no son correctos. | 26/12/2021 | Correcto |
| PRUEBA 3 | Dar de alta un usuario que ya existe | Nombre de usuario: Ainara<br>Apellido: Montoya<br>Edad: 20 | Informa al usuario de que el usuario ya existe. | 26/12/2021 | Correcto |
| PRUEBA 4 | Dar de alta un usuario con datos en blanco | Nombre de usuario: Ainara<br>Apellido: Montoya<br>Edad: c | Informa al usuario de que los datos que hay que rellenar obligatoriamente. | 26/12/2021 | Error, no se ha conseguido el resultado esperado |

Existen diferentes técnicas para el diseño de estos casos:

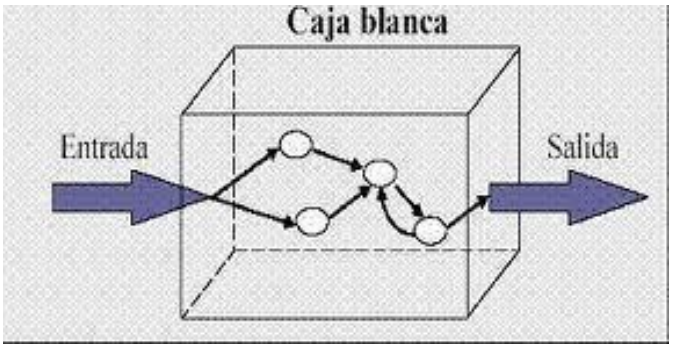
- Enfoque de caja negra
- Enfoque de caja blanca

## PRUEBAS DE CAJA BLANCA

Las pruebas de caja blanca se aplican, normalmente, a pruebas unitarias y a pruebas de integración.

Conociendo el código se diseñan casos de pruebas destinadas a comprobar que el código hace correctamente lo esperado en cada uno de sus caminos, son pruebas estructurales.

El ingeniero de pruebas se interesa por conocer las regiones del código que han sido recorridas. Así, si se encuentra con que una zona del código no ha sido recorrida por los casos de prueba, se añadirán nuevos casos de prueba que fuercen a que se pase por ahí.




Las pruebas de caja blanca intentan garantizar que:

- Se ejecutan al menos una vez todos los caminos de cada módulo.
- Se utilizan todas las estructuras de datos internas.

## PRUEBAS DE CAJA NEGRA

Las pruebas de caja negra se aplican a pruebas unitarias, de integración, de sistema e, incluso, de aceptación.

En las pruebas de caja negra se comprueba el correcto funcionamiento de los componentes del sistema de información, analizando las entradas y salidas y verificando que el resultado es el esperado.


En este tipo de pruebas el elemento que se va a probar se entiende como una caja negra de la que sólo se conocen sus entradas y sus salidas. Así, al elemento bajo prueba se lo somete a una serie de datos de entrada, se observan las salidas que produce y se determina si éstas son conformes a las entradas introducidas.

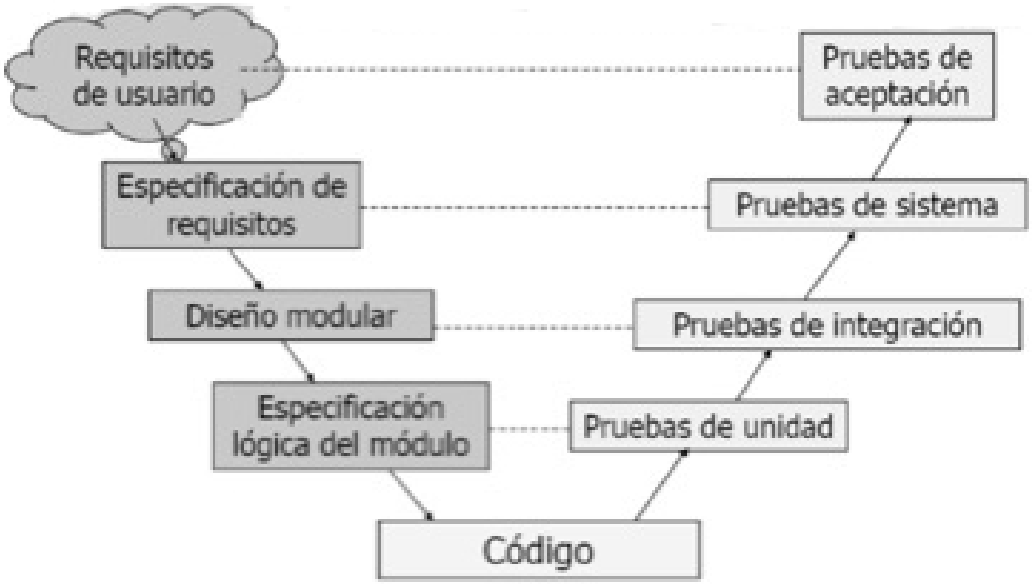
Los datos de prueba se escogerán atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa se ejecute bien.

A diferencia de la prueba de caja blanca, la prueba de caja negra tiende a aplicarse durante fases posteriores de la fase de prueba.

## TIPOS DE PRUEBA DE SOFTWARE

Existen diferentes tipos de pruebas, todas ellas necesarias en un desarrollo software:

- **Pruebas unitarias**: es la prueba de cada módulo, que normalmente realiza el propio personal de desarrollo en su entorno.
- **Pruebas de integración**: con el esquema del diseño del software, los módulos probados se integran para comprobar sus interfaces en el trabajo conjunto, es decir, se prueba el correcto ensamblaje entre los distintos componentes
- **Pruebas del sistema**: el software ya validado se integra con el resto del sistema (rendimiento, seguridad, recuperación y resistencia).
- **Pruebas de aceptación**: el usuario comprueba en su propio entorno de explotación si lo acepta como está o no.




También existen otros tipos de pruebas:

- El funcionamiento correcto del sistema integrado con el resto del hardware y software en el entorno de operación (**pruebas de implantación**).
- Que los cambios sobre un componente de un sistema de información no introduzcan un comportamiento no deseado o errores adicionales en otros componentes no modificados (**pruebas de regresión**).

---

## PRUEBAS UNITARIAS

Las pruebas unitarias tienen como objetivo verificar la funcionalidad y estructura de cada componente individualmente una vez que ha sido codificado.

Las pruebas unitarias constituyen la prueba inicial de un sistema y las demás pruebas deben apoyarse sobre ellas.

El enfoque que suele adoptarse para una prueba unitaria está claramente orientado al diseño de casos de caja blanca, aunque se complemente con los de caja negra. El hecho de incorporar casos de caja blanca se debe, por una parte, a que el tamaño del componente es apropiado para poder examinar toda la lógica y por otra, a que el tipo de defectos que se busca coincide con los propios de la lógica detallada en los componentes.

En este tipo de pruebas:

- Se prueba la "**interfaz**" del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad del programa que está siendo probada.
- Se examinan las "**estructuras de datos**" locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo.
- Se prueban las "**condiciones límite**" para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento.
- Se recorren todos los "**caminos independientes**" (caminos básicos) de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez.
- Se prueban todos los "**caminos de manejo de errores**".

Los pasos necesarios para llevar a cabo las pruebas unitarias son los siguientes:

- **Ejecutar todos los casos de prueba** asociados a cada verificación establecida en el plan de pruebas, registrando su resultado.
  Los casos de prueba deben contemplar tanto las condiciones válidas como las inválidas e inesperadas.
- **Corregir los errores o defectos encontrados y repetir las pruebas que los detectaron**. Si se considera necesario, debido a su implicación o importancia, se repetirán otros casos de prueba ya realizados con anterioridad.

La prueba unitaria se da por finalizada cuando se hayan realizado todas las verificaciones establecidas y no se encuentre ningún defecto, o bien se determine su suspensión.

## PRUEBAS DE INTEGRACIÓN

El objetivo de las pruebas de integración es verificar el correcto ensamblaje entre los distintos componentes, una vez que han sido probados unitariamente, con el fin de comprobar que interactúan correctamente.

Se distinguen dos tipos fundamentales de pruebas integración: integración incremental e integración no incremental.

### Integración incremental

Se combina el siguiente componente que se debe probar con el conjunto de componentes que ya están probados y se va incrementando progresivamente el número de componentes a probar. Con el tipo de prueba incremental lo más probable es que los problemas que surjan al incorporar un nuevo componente o un grupo de componentes previamente probado sean debidos a este último o a las interfaces entre éste y los otros componentes.

Se distinguen las siguientes estrategias de integración incremental:

- **De arriba abajo (top-down)**: el primer componente que se desarrolla y prueba es el primero de la jerarquía. Los componentes de nivel más bajo se sustituyen por componentes auxiliares para simular a los componentes invocados.
  Una de las ventajas de aplicar esta estrategia es que las interfaces entre los distintos componentes se prueban en una fase temprana y con frecuencia.
  Como inconvenientes principales: necesidad de crear y utilizar módulos auxiliares, las condiciones de prueba pueden ser difíciles de crear, la observación de salida de las pruebas es difícil, ...
- **De abajo a arriba (bottom-up)**: en este caso se crean primero los componentes de más bajo nivel y se crean componentes conductores para simular a los componentes que los llaman.
  Las condiciones de prueba son más fáciles de crear y es más sencillo la observación de los resultados de las pruebas.
  Se necesitan módulos conductores y se tarda más en ver el programa final como un todo.
- **Estrategias combinadas**: a menudo es útil aplicar las estrategias anteriores conjuntamente. De este modo, se desarrollan partes del sistema con un enfoque "top-down", mientras que los componentes más críticos en el nivel más bajo se desarrollan siguiendo un enfoque "bottom-up".

### Integración no incremental

Se prueba cada componente por separado y posteriormente se integran todos de una vez realizando las pruebas pertinentes. Este tipo de integración se denomina también Big-Bang.

## PRUEBAS DE SISTEMA

Las pruebas de sistema tienen por objetivo comprobar que el sistema, que ha superado las pruebas de integración, se comporta correctamente con su entorno (con otras máquinas, otro hardware, redes,

fuentes reales de información, etc.).

Estas pruebas permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen. Dan una visión muy similar a su comportamiento en el entorno de producción

En esta etapa pueden distinguirse los siguientes tipos de pruebas, cada uno con un objetivo claramente diferenciado:

- **Pruebas funcionales**: dirigidas a asegurar que el sistema de información realiza correctamente todas las funciones que se han detallado en las especificaciones dadas por el usuario del sistema.
  Normalmente, los responsables de realizar estas pruebas son los **analistas** de la aplicación (las personas responsables de la especificación de los requerimientos) con apoyo de los usuarios finales. Su aprobación dará paso a la fase de producción propiamente dicha.
  Son **pruebas de caja negra**.
  Dentro de las pruebas funcionales, encontramos tres tipos:
  - **Análisis de valores límite**: como entradas, seleccionamos aquellos valores que están en el límite donde la aplicación es válida. (Si estos valores no dan problemas, el resto tampoco los dará).
  - **Particiones equivalentes**: como entradas, se seleccionará una muestra representativa de todas las posibles. El resultado será extrapolable al resto.
  - **Pruebas aleatorias**: como entradas, se selecciona una muestra de casos de prueba al azar. Se utiliza sólo en aplicaciones no interactivas.
- **Pruebas de rendimiento**: consisten en determinar que los tiempos de respuesta están dentro de los intervalos establecidos en las especificaciones del sistema.
- **Pruebas de volumen**: consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos, simulando las cargas de trabajo esperadas.
- **Pruebas de sobrecarga (o resistencia)**: consisten en comprobar el funcionamiento del sistema en el umbral límite de los recursos, sometiéndole a cargas masivas. El objetivo es establecer los puntos extremos en los cuales el sistema empieza a operar por debajo de los requisitos establecidos.
- **Pruebas de disponibilidad de datos (o de recuperación)**: consisten en demostrar que el sistema puede recuperarse ante fallos, tanto de equipo físico como lógico, sin comprometer la integridad de los datos.
- **Pruebas de facilidad de uso**: consisten en comprobar la adaptabilidad del sistema a las necesidades de los usuarios.
- **Pruebas de entorno**: consisten en verificar las interacciones del sistema con otros sistemas dentro del mismo entorno.
- **Pruebas de seguridad**: intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán de accesos no autorizados. Mide la habilidad de un sistema para soportar ataques (tanto accidentales como intencionados) contra su seguridad.
  Se intenta verificar que el sistema es robusto frente a problemas de seguridad, tales como:
  - Intentar conseguir las claves de acceso de cualquier forma.
  - Atacar con software a medida.
  - Bloquear el sistema.
  - Provocar errores del sistema, entrando durante su recuperación.

  Hay que tener especial cuidado con las entradas no autorizadas cuando en el sistema de interés se maneja información de carácter privado y personal.
  Durante la prueba, lo usual es que el encargado desempeña el papel de intruso y trata de violar los mecanismos de seguridad de acceso al sistema. Intenta obtener la clave de acceso de cualquier forma.
  Además, también debe intentar bloquear el sistema, curiosear los datos públicos en busca de claves e introducir errores de forma consciente al sistema para ver cómo responde éste.

## PRUEBAS DE ACEPTACIÓN

Estas pruebas las realiza el cliente y son básicamente pruebas funcionales sobre el sistema completo.

La experiencia muestra que aún después del más cuidadoso proceso de pruebas por parte del desarrollador, quedan una serie de errores que sólo aparecen cuando el cliente comienza a usarlo.

Una prueba de aceptación puede ir desde un informal caso de prueba hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho, las pruebas de aceptación pueden tener lugar a lo largo de semanas o meses, descubriendo así errores latentes o escondidos que pueden ir degradando el funcionamiento del sistema. Estas pruebas son muy importantes, ya que definen el paso a nuevas fases del proyecto como el despliegue y mantenimiento.

Se emplean dos técnicas para las pruebas de aceptación:

- **Pruebas alfa**: se llevan a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario. Las pruebas alfa se llevan a cabo en un entorno controlado.
  Para que tengan validez, se debe primero crear un ambiente con las mismas condiciones que se encontrarán en las instalaciones del cliente. Una vez logrado esto, se procede a realizar las pruebas y a documentar los resultados.
- **Pruebas beta**: son realizadas por los usuarios finales del software en sus lugares de trabajo. A diferencia de la prueba alfa, el desarrollador no suele estar presente. Así, la prueba beta es una aplicación "en vivo" del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas (reales o imaginarios) que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador.
  Como resultado de los problemas informados durante la prueba beta, el desarrollador del software lleva a cabo modificaciones y prepara una nueva versión del producto.

## PRUEBAS DE REGRESIÓN

Se denominan pruebas de regresión a cualquier tipo de pruebas de software que intentan descubrir las causas de nuevos errores, carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software, inducidos por cambios recientemente realizados en partes de la aplicación que con anterioridad al citado cambio no eran propensas a este tipo de error.

Resulta que cada vez que se añade un nuevo módulo a la aplicación, o cuando se modifica, el software cambia y como resultado de esa modificación (o adición) de módulos, podemos introducir errores en el programa que antes no teníamos. Por ello, se hace necesario volver a probar la aplicación. La prueba de regresión es volver a ejecutar un subconjunto de pruebas que se han llevado a cabo anteriormente para asegurarse de que los cambios no han propagado efectos colaterales no deseados.

Así, el objetivo de las pruebas de regresión es comprobar que los cambios sobre un componente de una aplicación no introducen errores adicionales en otros componentes no modificados.

El conjunto de pruebas de regresión contiene, a su vez, tres clases de prueba diferentes:

- Planificar una muestra de pruebas sobre todas las funciones del software.
- Plantear pruebas adicionales que se centren en las funciones del software que se van a ver afectadas por el cambio.
- Planificar pruebas que se centren únicamente en los componentes que han cambiado.

# DESPLIEGUE DE APLICACIONES

## INTRODUCCIÓN

Tras implementar una aplicación llega el momento de su distribución, para ello será necesario realizar un proceso de instalación que requiere de un paso previo: el empaquetado.

Para realizar un correcto empaquetado se deben agrupar todos los componentes necesarios para el despliegue de una aplicación.

Un paquete contiene no sólo el código que implementa una aplicación, sino que constará de todo lo necesario para desplegar una aplicación y realizar una correcta distribución.

Lo componentes principales son:

- Las librerías y bibliotecas
- Los ficheros ejecutables
- Los elementos multimedia

## INSTALADORES Y PAQUETES DE INSTALACIÓN

Una vez que un programador ha finalizado el desarrollo de una aplicación, debe decidir el mecanismo que va a utilizar para su distribución.

Si el producto desarrollado se va a instalar en sistemas Windows deberá crear instaladores para Windows (fichero ejecutable .exe). Si lo va a distribuir en Linux lo normal es que cree un paquete de instalación (paquete .deb en Ubuntu)…

En la actualidad existe un amplio abanico de herramientas de creación de instaladores y/o paquetes de instalación y permiten personalizar las instalaciones de nuestro software. Las más extendidas para crear instaladores son las siguientes: InstallAnywhere, InstallBuilder, Windows Installer, InstallShield, InstallAware, Wise Installation Studio, NSIS, IzPack, Launch4j...

La instalación de una aplicación es el primer contacto que va a tener un usuario con la aplicación por lo que un instalador rápido y amigable debe ser parte esencial en cualquier producto software

Los pasos en una instalación son los siguientes:

1. Verificación de la compatibilidad: se debe comprobar que se cumplen los requisitos para la instalación, tanto hardware como software.
2. Verificación de la integridad: se verifica que el paquete de software es el original.
3. Creación de los directorios requeridos.
4. Creación de los usuarios requeridos: cada grupo de usuarios puede usar un determinado software.
5. Copia, desempaque y descompresión de los archivos desde el paquete de software.
6. Compilación y enlace con las bibliotecas requeridas.
7. Configuración.
8. Definición de las variables de entorno requeridas.
9. Registro de la aplicación ante el autor o autora de la aplicación.

En el proceso de instalación suele ser necesario la interacción con el usuario para que configure la aplicación según necesidades, para lo cual se siguen una serie de pasos aceptados y estandarizados, que son los siguientes:

1. Ventana de selección de idioma.

Lo primero que decide el usuario es el idioma en el que se le mostrarán los mensajes del proceso de instalación del programa.

2. Ventana de bienvenida.
   Información de la versión de la aplicación, recomendaciones, etc.
3. Acuerdo de licencia: aceptación de los términos de uso.
   Se tienen que aceptar los términos de licencia del software que se quiere instalar. Si no es así, la instalación será abortada en este momento.
4. Aceptación o no aceptación de herramientas opcionales a instalar.
   Podemos seleccionarlos o no, y esa elección no tendrá efecto sobre nuestra instalación.
5. Selección de la ubicación donde se guardan los archivos.
   Normalmente, el programa nos propone una ruta, pero el usuario debe poder cambiar esta ubicación y elegir otra distinta.
6. Selección de accesos directos.
   Se puede seleccionar los accesos directos a crear.
7. Proceso de instalación.
   En este punto, comienzan a copiarse los archivos en el disco duro del ordenador del usuario, en la ubicación que él haya elegido.
8. Finalización.
   El proceso de instalación termina en este punto y debe avisar al usuario de su terminación.

A la hora de crear nuestro instalador es conveniente personalizarlo:

- Logotipo: se debe personalizar el instalador con el logotipo de la aplicación.
- Diseño gráfico: el instalador debe ser consistente con la aplicación. Mantener la consistencia implica que el formato de las ventanas, los colores, iconos, imágenes, botones y demás componentes gráficos guarden el mismo formato, tanto en colores, fuentes y tamaños.
- Idioma: el idioma del instalador debería ser el mismo que el de nuestra aplicación y en caso de que se trate de una aplicación multidioma el programa instalador tiene que permitir hacer la instalación en los idiomas de la aplicación. Al empezar la instalación el usuario elegirá el idioma para la instalación.

## GENERAR PAQUETES DE INSTALACIÓN

Para generar el paquete de instalación de una aplicación, disponemos de varias alternativas:

- Utilizar entornos de desarrollo.
- Hacer uso de herramientas externas: Inno Setup, Launch4j, NSIS, IzPack…

La primera alternativa es la utilización de las herramientas de generación de paquetes incorporadas en el entorno de desarrollo. Los entornos de desarrollo suelen disponer de herramientas de generación de paquetes, pero no suelen incorporar herramientas que creen instaladores "amigables". En el caso de Eclipse, podemos crear un paquete de instalación JAR de nuestra aplicación Java, sin embargo, la instalación y la ejecución del paquete, deberá realizarla el usuario.

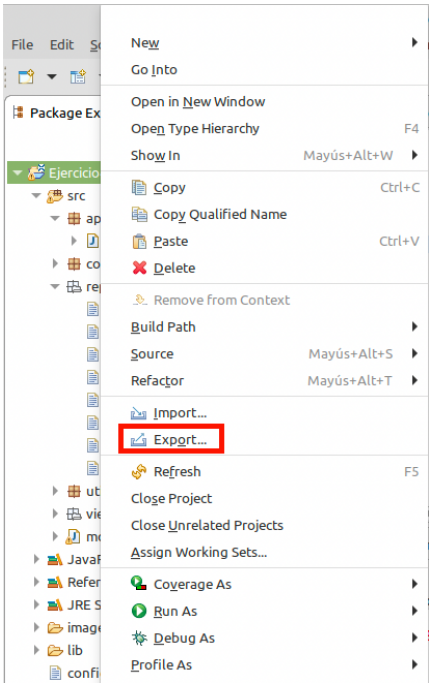
Existen en el mercado herramientas que complementan a los entornos de desarrollo para la creación de paquetes de instalación. Estas herramientas, en el caso de aplicaciones Java, pueden crear el paquete JAR, o directamente partiendo del paquete JAR generado a partir del entorno de desarrollo, crear un nuevo proyecto de instalación, donde se cree un instalable o un autoinstalable, para la aplicación desarrollada.

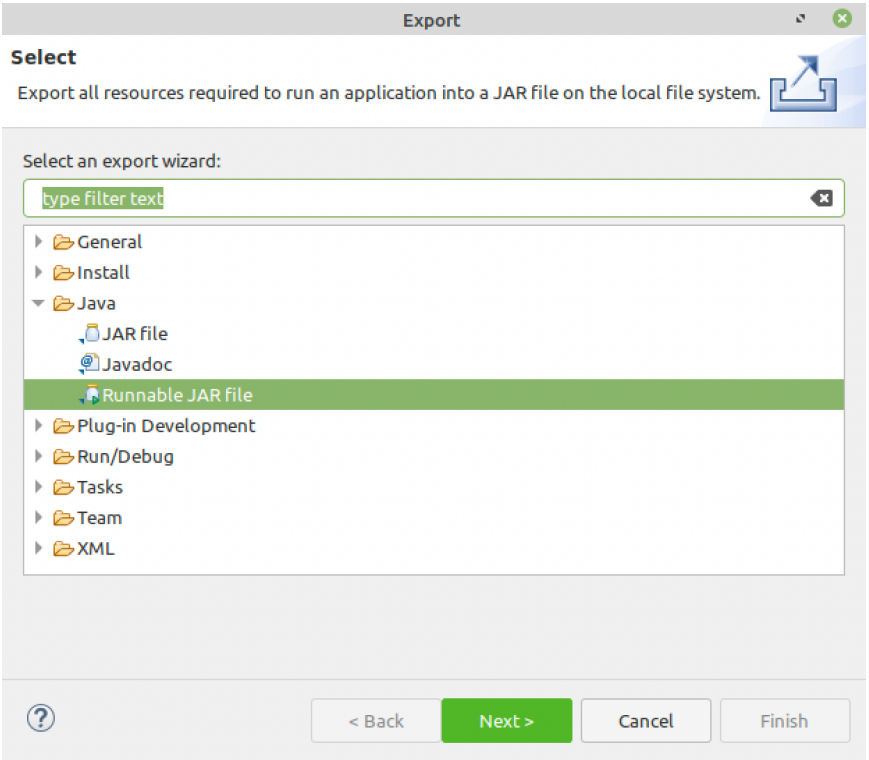
Por último, en las instalaciones desatendidas el usuario no interviene en la instalación de un paquete, ni elige la ubicación, ni la configuración de la instalación…

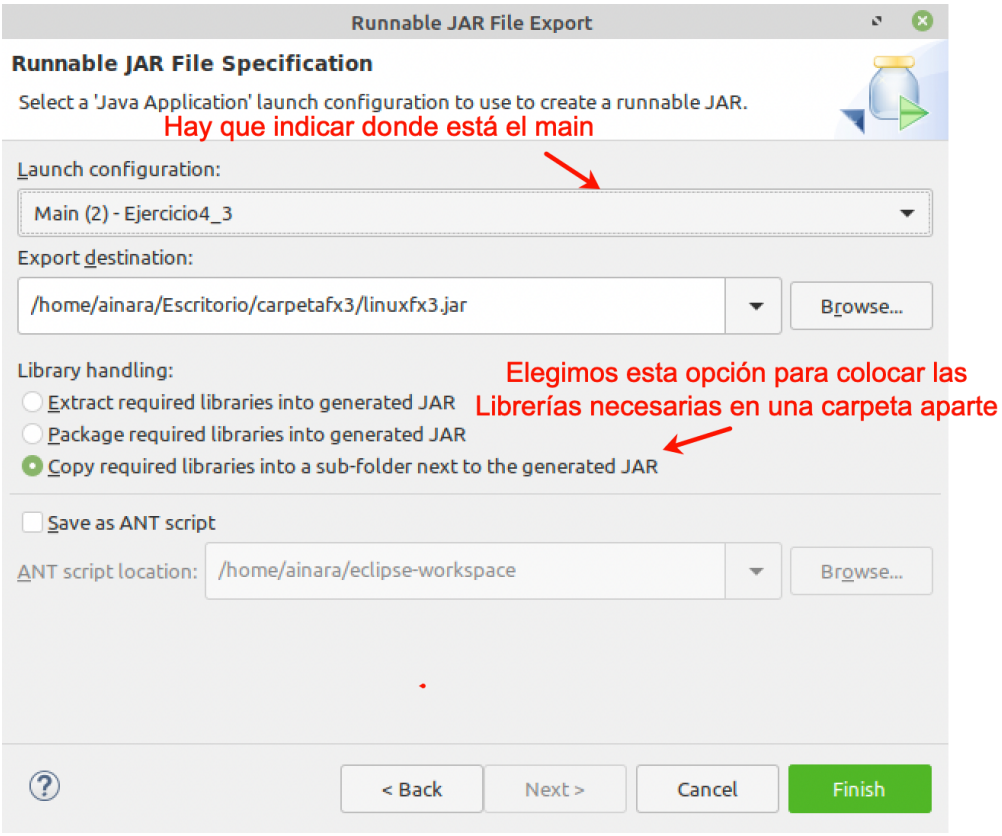
En el caso del desarrollo de aplicaciones Java, el objetivo para distribuir una aplicación, es generar el fichero **jar**.

Vamos a ver como hacerlo para aplicaciones javaFX.

Crear un jar para aplicaciones javaFX es un poco más complejo que para otro tipo de aplicaciones, como aplicaciones swing, porque javaFX necesita su propio SDK.



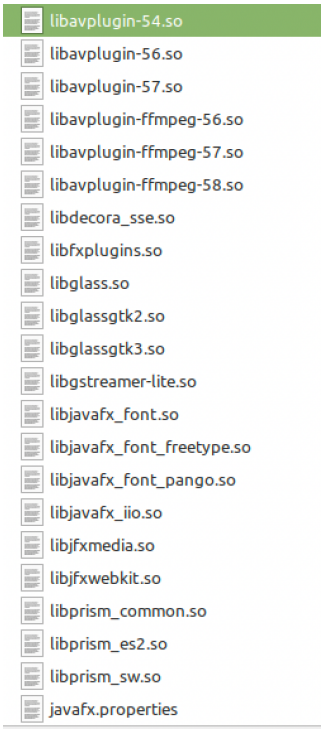


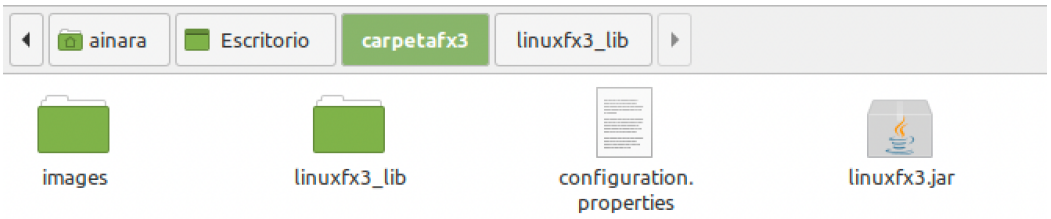

Se generará la carpeta indicada (en caso de que no exista) con los siguientes elementos:

- .jar (con el contenido de los resource folders)
- Carpeta con las librerías necesarias

Hay que añadir algunos elementos más a la carpeta:

- Hay que añadir los elementos que no estén en ninguna resource folder (por ejemplo: configuration.properties, imágenes a las que no se acceda con getResource…)
- En la carpeta de las librerías hay que añadir algunos elementos para javafx (los elementos que tenemos en la carpeta de librerías de nuestro proyecto pero que no son .jar), de hecho estos elementos cambian según el sistema operativo.






Para ejecutar la aplicación (en un equipo que tenga ya instalada la máquina virtual de java) y comprobar que funciona correctamente fuera del IDE tenemos que ejecutar el siguiente comando:

*java --module-path carpeta_librerias --add-modules javafx.controls,javafx.fxml,javafx.web -jar fichero_jar.jar*

Escribir este comando puede ser un poco complicado para un usuario normal, por lo que si se quiere se puede crear un script que ejecute este comando:

Ejemplo para Linux:
*sudo nano inicio.sh*

El contenido del fichero será:
*java --module-path linuxfx3_lib --add-modules javafx.controls,javafx.fxml,javafx.web -jar linuxfx3.jar*

Le damos permisos de ejecución
*chmod a+x inicio.sh*

Ahora para ejecutar la aplicación se puede hacer doble click sobre inicio.sh o ejecutarlo en la terminal

Ejemplo para Windows:
Creamos en PowerShell el archivo inicio.ps1 cuyo contenido será.
*java --module-path linuxfx3_lib --add-modules javafx.controls,javafx.fxml,javafx.web -jar linuxfx3.jar*

Ahora para ejecutar la aplicación se puede hacer doble click sobre inicio.ps1 o ejecutarlo desde PowerShell (recordar que hay cambiar la política de ejecución de scripts *Set-ExecutionPolicy Unrestricted*).

Si la aplicación funciona correctamente fuera del IDE creamos un fichero zip con todos los ficheros

---

### INNO SETUP

Inno Setup es una herramienta específica para entornos Windows que nos proporciona un completo entorno para crear instaladores, tanto para aplicaciones Java, como aplicaciones desarrolladas en otros lenguajes.

Inno Setup proporciona un amplio abanico de scripts de ejemplo, que combinándolos y adaptándolos a nuestras necesidades, nos permiten crear instaladores completos, incluidos los desinstaladores.

Más información: https://jrsoftware.org/ishelp/

## INSTALACIONES DESATENDIDAS

Una instalación en modo desatendido es aquélla en la que el usuario no tiene que decidir el lugar de instalación de la aplicación, ni tampoco características tales como el idioma, variables de entorno, etc.

El uso de instalaciones desatendidas es útil cuando se deben instalar programas en gran cantidad de ordenadores.

Para automatizar el proceso de instalación, podemos crear un fichero de respuestas que nos guarde los datos que introducimos, de forma que el instalador lea los datos que necesita de dicho fichero.

## FICHEROS FIRMADOS DIGITALMENTE

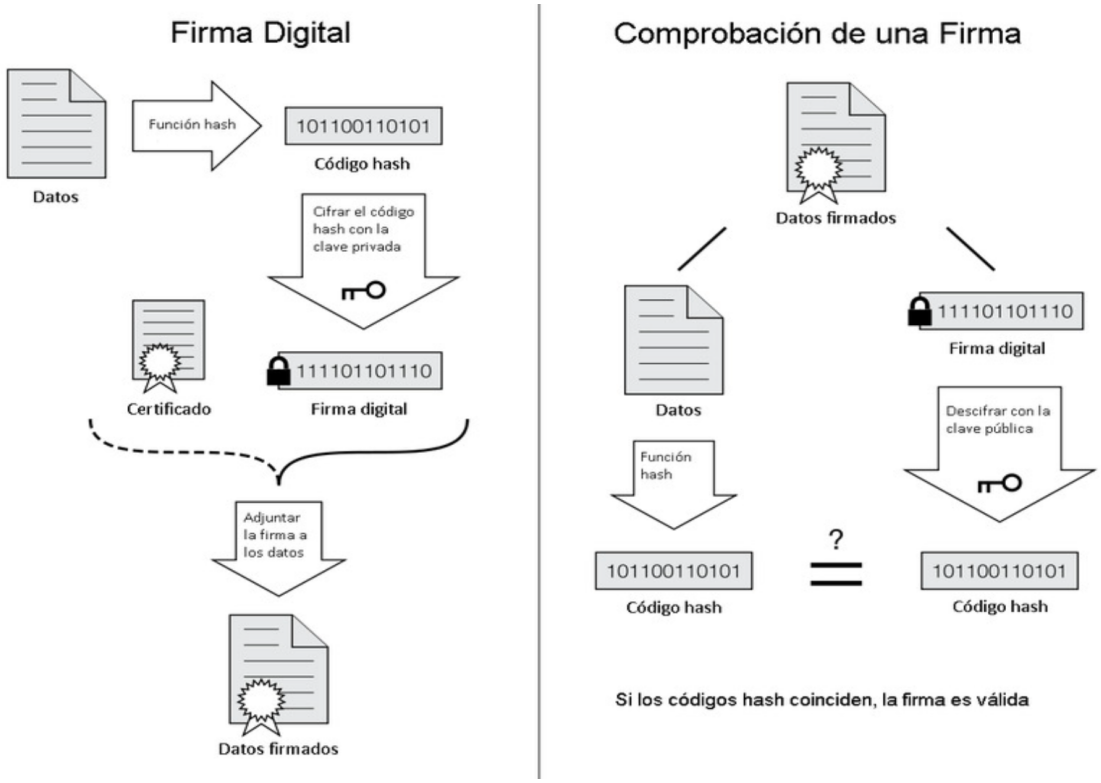
Una firma electrónica es un conjunto de datos electrónicos que acompañan a un documento electrónico y nos garantiza que el fichero ha sido creado por una empresa o autor conocido y que no ha sido modificado durante su transferencia.

Por ello, hoy en día es necesaria la firma en la distribución de software, dado que en muchas ocasiones la descarga de nuevas aplicaciones se realiza a través de Internet, por lo que será necesario utilizar mecanismos que garanticen la autenticidad del software. Se debe conocer que las firmas digitales están formadas por una clave pública y una privada.

La clave privada está pensada para que este siempre bajo el control del firmante, la clave pública asociada a la clave privada representa la identidad de una entidad y se distribuye a las entidades con las que se vaya a interactuar.

En el caso de la distribución de paquetes de software, en concreto, de aquellos desarrollados utilizando Java, es posible realizar la firma digital sobre los ficheros jar, lo que permite verificar la autenticidad del software descargado.

Existe un elemento adicional necesario para la firma y verificación: certificado que el firmador incluye en un fichero JAR firmado. El certificado es una declaración digital firmada reconocida por una autoridad de certificación que indica quién es el dueño de una clave pública.



Para realizar la firma de ficheros jar se puede utilizar la herramienta **JarSigner**.