Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería de Software 1
Obligatorio 2

Juan Pablo Barrios
Juan Ignacio Irabedra
github.com/jirabedra/Barrios_Irabedra_Obligatorio_IngSoft

Entregado como requisito de la materia Ingeniería de Software 1

20 de junio de 2019

Declaraciones de autoría

Nosotros, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos la materia;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Esta obra fue basada en la plantilla entregada por la cátedra

Resumen

Realización del segundo obligatorio de la materia Ingeniería de Software 1. En este documento se desarrolla la documentación correspondiente al proyecto. Por otro lado se entrega el proyecto en Java. El proyecto consiste en una aplicación de preguntas y respuestas. Se desarrollará parte del proceso de desarrollo, pruebas, justificación y reflexión sobre el proyecto globalmente, apoyándonos en el proyecto realizado en Java; implementación del proceso de diseño de la aplicación.

Índice general

1. Desarrollo de una aplicación

El objetivo del trabajo es el desarrollo de una aplicación, utilizando prácticas tecnológicas y de gestión de la ingeniería de software. Se espera como resultado un software de calidad y la aplicación de un conjunto de prácticas profesionales. En el obligatorio no se evalúa únicamente la implementación, sino el proceso de desarrollo y su impacto en la calidad del software. Los temas centrales de aplicación para este obligatorio son los siguientes:

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

1.1. Proyecto: Quizzes educativos

Se desea desarrollar un sistema de quizzes para el uso en un contexto educativo. Los casos de uso a implementar son:

1. Cargar preguntas desde archivo.
2. Iniciar quiz.
3. Responder pregunta de un casillero.
4. Finalizar juego y generar reporte.

Se deben cargar las preguntas desde un archivo de texto utilizando el formato GIFT. Los tipos de pregunta soportados son: múltiple opción, verdadero/falso y respuesta corta. Antes de cada pregunta se puede indicar con un comentario el tiempo máximo en segundos (ej: //15). Si no se indica tiempo límite para la pregunta, se asume por defecto 30 segundos.

El sistema es utilizado por un solo usuario o equipo. Se debe mostrar el progreso en forma gráfica con un camino que tenga la misma cantidad de casilleros que preguntas cargadas. Luego de cada pregunta se debe mostrar un GIF y reproducir un sonido correspondiente a respuesta correcta o incorrecta. Los casilleros se van marcando verde o rojo según la respuesta.

Al finalizar se muestra el resultado global, todas las preguntas con sus respuestas correctas y las marcadas durante el juego. También se debe indicar el tiempo utilizado en cada pregunta y el promedio. Por cada sesión de juego se genera un reporte PDF con dicha información.

Todas las interacciones deben realizarse en una única ventana, sin utilizar diálogos o ventanas emergentes.

1.2. Restricciones

- La aplicación se debe ejecutar correctamente en la versión de Java disponible en los laboratorios de la Facultad.
- No se requiere autenticación para utilizar el sistema.

1.3. Entrega

El obligatorio se debe entregar en bedelía con la boleta de inscripción. Adicionalmente se utilizan dos medios de entrega digital.

1. En bedelía se entregará una sola página impresa con la portada del informe. Esta contiene el título, materia, identificación de los estudiantes y un link para acceder al repositorio online utilizado.
2. Se deberá dar acceso al repositorio online por el mecanismo indicado por los docentes. Uno de los integrantes del equipo subirá el PDF del informe a en Aulas. El repositorio y el informe no pueden ser modificados ni subidos luego de la fecha de entrega.

El repositorio debe ser nombrado con el apellido de los integrantes del equipo. Se deben evitar los nombres genéricos para el repositorio, paquete principal o documento. Por ejemplo, los nombres: IS1, Obligatorio, Sistema, Proyecto, Informe, etc. dificultan la identificación del equipo.

Proyecto NetBeans y ejecutable JAR

Se debe incluir todo el proyecto NetBeans con el código fuente y las pruebas unitarias (JUnit). Las referencias a librerías deben ser relativas (el proyecto se puede abrir en cualquier ubicación).

Documentación

Se debe dar evidencia de las actividades realizadas para el desarrollo de la aplicación. En particular, se deben explicar en un documento las actividades de aseguramiento de la calidad.

El documento entregado debe cumplir con los estándares de la facultad para proyectos de fin de carrera [1]. La preparación del documento se debe realizar en

LaTeX[1]. El código fuente del presente documento se encuentra disponible[2] para ser usado como plantilla del obligatorio.

[1] LaTeX es un sistema de software para la elaboración de documentos que tiene funciones específicas para creación de documentos técnicos y científicos. Este sistema permite separar la especificación de formato, permitiendo enfocarse en el contenido del documento.

[2] https://www.overleaf.com/read/mcnwxtffddbz

2. Desarrollo

2.1. Versionado

2.1.1. Repositorio utilizado

El repositorio utilizado fue respaldado en GitHub. El enlace al mismo es: github.com/jirabedra/Barrios_Irabedra_Obligatorio_IngSoft

Los Elementos de la Configuración de Software (ECS) incluidos en el repositorio son:

- Código fuente: en el repositorio se incuye, como último commit previo a la entrega del documento en la rama master, el código definitivo del proyecto. El código fuente es un elemento configurable ya que varía cada vez que se agrega una funcionalidad o se soluciona un problema.
- Versionado: en el repositorio se encuentra las diferentes versiones subidas. Más adelante se especificarán los criterios de versionado.
- Dependencias: también están todas las bibliotecas o aplicaciones necesarias para que el proyecto funcione a la hora de la entrega. Estas pueden cambiar con el tiempo e interactuar con el entorno y versión en la que se desarrolla, así que son elementos definitivamente configurables.
- Interfaz y bocetos: se incluyen algunos bocetos realizados previos a crear la GUI. La interfaz no fue estática en el ciclo de desarrollo de este sistema. De hecho, comenzó siendo muy básica y poco usable. A medida que se trabajó en ella, se convirtió en el resultado que es hoy día.
- Elementos visuales y sonoros utilizados: tales como GIFs y sonidos.
- Propuesta y requerimientos (brindada por la cátedra)
- Documentos y recursos auxiliares, tales como ejemplo de formato GIFT
- Pruebas
- Breve manual de usuario
- Datos de pruebas

- Este mismo documento

Además, utilizamos Maven, que es una herramienta para la gestión y construcción de proyectos en Java. Maven, a través de un POM (Project Object Model) que describe detalladamente el proyecto a ser construído y sus dependencias. El ciclo de construcción de Maven sigue un orden riguroso que garantiza solidez en la construcción del proyecto. Las etapas del ciclo son las siguientes:

- Validación: valida que el proyecto sea corecto y toda la información necesaria esté disponible.
- Compilación: compila el código del proyecto
- Prueba: prueba que el código compilado use una interfaz de pruebas unitarias adecuada. Estas pruebas no requieren que el código sea despachado.
- Package: lleva el código compilado en su formato destino, como JAR.
- Verificación: Verifica que el proyecto esté integrado.
- Instalación: instala el package en el repositorio local para ser usado como dependencia local
- Deploy: En el ambiente de construcción copia el package definitiivo a un repositorio remoto.

2.1.2. Criterios de versionado

La etapa más inicial del proyecto fue completamente presencial. Entonces no hubo repositorio en GitHub. Ya que constaba de cuestiones de arquitectura y diseño que consideramos valía más la pena discutir en persona antes de poner manos en el código; por ejemplo, el UML. El versionado en la nube comenzó cuando se empezó a compartimentar las diferentes tareas. Los commits se realizaron cada vez que:

- Un integrante buscaba que el otro evaluara el código escrito recientemente
- Un problema surgió y se le pidió al otro integrante que evalúe la situación
- Cada vez que se implementaba una feature o sección de código reelevante.
- Cada vez que uno de los desarrolladores quería enviarle recursos al otro, tales como elementos gráficos o sonidos o archivos de formato GIFT.

El commit inicial se realizó en la rama master. La rama develop, como su nombre lo indica, se encargó de ir agregando funcionalidades (o a veces simplemente bloques de código) y de ir arreglando errores.

2.1.3. Resumen del log de versiones

El resumen debe dar evidencia de la evolución del proyecto en el tiempo y del trabajo realizado por distintos integrantes del equipo.

2.2. Codificación

El código se realizó en Java 8. Nuestros estándares se remiten a Java Code Convention (1997). A continuación enunciaremos algunas de las prácticas que creemos son correctas y ameritan énfasis y evidencia de calidad en nuestra implementación, seguido de una reflexión de por qué consideramos particularmente destacable la buena práctica.

- Orden en la estrctura de una clase:
  De acuerdo con Java Code Conventions, la estrutura de una clase debería seguir la siguiente estructura:

  1. Comentarios de documentación de la clase
  2. Declaración de la clase
  3. Variables de clase
  4. Variables de instancia
  5. Constructores
  6. Métodos

  Creemos que es muy importante apegarnos a este estándar para promoverla legibilidad del código y para compartimentar y ordenar la información que tienen las clases. Especialmente en programación orientada a objetos (OOP) consideramos que hay que ser especialmente rigurosos a la hora de definir clases.

- Convenciones de nombres:
  Creemos que la denominación de variables constituye uno de los mayores pilares que hacen a la comprensibilidad del código. Promovemos la nomenclatura mnemotécnica y autoexplicativa. Además, de acuerdo a lo enunciado en Java Code Conventions, la gran cantidad de comentarios suele reflejar una baja claridad de código. Los nombres elegidos para variables y métodos, así como clases y demás elementos de la OOP, nutren al código de calidad. Mencionaremos algunos ejemplos de variables y métodos cuyo nombre hace real referencia a lo que representan:

  ```
  private String pregunta;
  private int tiempo;
  private HashMap<String, Boolean> respuestas;
  private SimpleIntegerProperty segundosRestantes;
  ```

  Sin tener mucho conocimiento del dominio, cualquier persona podría intuír acertadamente el significado de cada una de estas variables de instancia.

- Indentación
  Creemos que es uno de los factores principales para promover la legibilidad del código. Un código mal indentado sería mucho más difícil de comprender y reconocer estructuras en él.

De acuerdo con Java Code Conventions, la unidad de indentación debe ser 4 (cuatro) espacios. El largo de las tabulaciones también debe ser constante, seteado a 8 espacios.
En nuestro proyecto apostamos a la consistencia en el indentado. A continuación acusaremos evidencia de esto:

```
/**
 * valida que la linea sea o no vacia, comentada o inesperada
 * @param linea es la linea leida
 * @return true si y solo si la linea no esta en formato.
 */
public Boolean esLineaConContenido(String linea) {
    if (linea.trim().length() == 0) {
        return false;
    } else {
        linea = linea.trim();
        char primerCaracter = linea.charAt(0);
        switch (primerCaracter) {
            case '/':
                return false;
            case ':':
                return true;
            case '=':
                return true;
            case '~':
                return true;
            case '#':
                return true;
            case '':
                return true;
            case '}':
                return true;
            default:
                return false;
        }
    }
}
```

El estilo de indentación no solo permite leer más ágilmente el código, sino que también, para el programador incita una estrctura de control dividida en casos, una suerte de pattern matching. La indentación bien utilizada no solo permite leer ordenada; también nos puede brindar información sobre qué se está haciendo.

2.2.1. Estándar de codificación

Como acusado previamente, elegimos Java Code Conventions como principal referencia de estándares de código. Usamos también el plugin CheckStyle para NetBeans 8.1.

2.2.2. Pruebas unitarias

Las pruebas unitarias fueron realizadas con la herramienta JUnit 4.12. Como visto en clase, se realizó una clase de prueba para cada clase del dominio y por cada método de clase existe al menos un método de prueba que lo pruebe.
Intentamos probar el código de manera agresiva, exhaustiva y excluyente. De acuerdo con el concepto de prueba exitosa dado como la que más probabilidad tiene de encontrar errores, intentamos bajar a tierra esta idea en nuestras pruebas.
Utilizamos la herramienta JaCoCoverage 0.8.2. Esta nos lanzó el coverage de pruebas en el dominio. Claramente este coverage es significativamente menor que el aceptable 90 % o el idílico 100 % . En la versión final entregada, JaCoCo nos indicó que el 59 % del dominio fue probado.

2.2.3. Análisis de código

La herramienta principal en la que nos apoyamos para analizar el código fue checkStyle para NetBeans 8.1.

2.3. Interfaz de usuario y usabilidad

2.3.1. Criterios de interfaz de usuario

La interfaz de usuario fue diseñada de manera minimalista. A través de sesiones de pruebas con diferentes usuarios piloto, se fueron agregando más elementos para favorecer la accesibilidad y usabilidad del sistema.
El primer defecto encontrado en las versiones preliminares fue la falta de responsividad por parte del sistema ante dudas del usuario. El sistema no ofrecía suficientes instrucciones in-built para guiarlo.
Las sesiones de prueba de usabilidad fueron arrojando la necesidad de determinadas señales o textos para que el usuario se sienta más apoyado al usar la aplicación.
Desde el punto de vista estético, se eligió una paleta de cianes y azules. Luego se extendió a turquesas y neutros. También predominan las formas cuadradas.
Decidimos acceder a las preguntas a través de un mapa en espiral. Creemos que genera mayor satisfacción al usuario que una aplicación en la que se lanzan todas las preguntas en un mismo hilo sin parar. Por un lado, los usuarios piloto se vieron impresionados ya que les genera la sensación de estar lidiando con un sistema mucho más libre y extenso que el típico quiz en línea.
Esto último se desprende de una de las heurísticas de Nielsen, que indica se debe brindar libertad y control al usuario de lo que sucede. A diferencia de otras aplicaciones de preguntas y respuestas, el usuario decide cuándo ingresar a una pregunta.

Set	Archivos	Archivos válidos	Preguntas	Resultado esperado
Set 1	0	0	0	No se pueda iniciar el juego
Set 2	1	0	0	No se pueda iniciar el juego
Set 3	1	1	0	No se pueda iniciar el juego
Set 4	1	1	$\geq 1 \leq 39$	Se inicie el juego con preguntas cargadas
Set 5	1	1	>39	Se inicie el juego con las 39 primeras
Set 6	≥ 1	0	0	No se pueda iniciar el juego
Set 7	≥ 1	≥ 1	$\geq 1 \leq 39$	Se inicie el juego con preguntas cargadas
Set 8	≥ 1	≥ 1	> 39	Se inicie el juego con las 39 primeras

2.3.2. Evaluación de usabilidad

Luego de realizar algunas sesiones de evaluación de usabilidad, constatamos que el sistema no era lo suficientemente responsivo. En las últimas instancias de implementación de interfaz decidimos enfocarnos en que la interacción usuarios-sistema sea muy dialógica. Intantamos que casi todas las interacciones por parte del usuario invoquen una respuesta por parte del sistema a través de la interfaz. También, tuvimos que agregar algunos elementos visuales como imágenes o diálogos para que el usuario se vea un poco más guiado y no requiera de un manual de usuario.
Aún así, optamos por realizar un breve manual de usuario que, en nuestros usuarios piloto, garantizó el éxito a la hora de aprender a usar la GUI. Se realizaron sesiones de evaluación de usabilidad. Están documentadas en el repositorio con videos donde se realizaron las dichas.

2.4. Pruebas funcionales

Previo a detallar las pruebas funcionales, desarrollaremos los casos de uso correspondientes a los requerimientos funcionales brindados por cátedra. Ellos se encuentran más adelante en este documento.

Elegimos el caso de uso iniciar quiz. Para este caso de uso no dependemos del set de datos ingresados ya que solo existe un curso para este caso de uso.

2.4.1. Técnicas de prueba aplicadas

Se realizaron pruebas unitarias y una prueba funcional para la funcionalidad iniciar quiz.

2.4.2. Casos de prueba

Para la funcionalidad cargar preguntas, generamos 8 juegos de datos de prueba de acuerdo con los casos posibles. Las variables involucradas son: cuántos archivos se intentan cargar, cuántos de ellos son válidos, y de esos válidos, cuántas preguntas se cargan. Lo modelamos de la siguiente manera:

El set de pruebas satisface el resultado esperado de acuerdo con ese caso de uso. Esos sets de prueba están en el repositorio en GitHub.

2.4.3. Sesiones de ejecución de pruebas

Las sesiones de evaluación de pruebas se realizaron constantemente los días 19 y 20 de junio, cuando la aplicación ya estaba completamente integrada. Fueron realizadas por los programadores mismos.

2.5. Reporte de defectos

2.5.1. Definición de categorías de defectos

Siendo 1 severo y 5 poco determinante

- Defectos de interfaz estéticos: 4. Estos son defectos que afectan la estética del programa. Impactan en usabilidad.
- Defectos de interfaz funcionales: 2. Estos defectos generan bugs en funcionalidades.
- Defectos de entrada/salida: 3. Son defectos que pueden hacer que la interacción entre el usuario y el sistema sea menos ordenada o prolija.
- Defectos puramente funcionales: 1. Son defectos de funcionalidades del juego mismas. Afectan la jugabilidad.

2.5.2. Defectos encontrados por iteración

Existen defectos encontrados preliminarmente por las sesiones de pruebas exploratorias. Algunos otros fueron encontrados en las sesiones de prueba por terceros. A continuación de enuncian y se clasifican los defectos encontrados.

- Al haber respondido una pregunta corta respuesta, si se intenta responder otra, el campo para ingresar texto de la nueva tendrá el último texto ingresado como texto por defecto. (Entrada salida)
- Cuando se presiona el botón ayuda mientras se responde una pregunta, el timer sigue descontando tiempo. (Funcional)
- Si se usa el deslizador en la parte derecha de la ventana del mapa de botones, no sucede nada, pues esa funcionalidad no fue totalmente implementada. (Funcional)
- El formato del reporte generado no es el más deseable. (Entrada Salida)
- El sonido solo se ejecuta desde el IDE. El JAR no tiene sonido. (Funcional)

2.5.3. Estado de calidad global

El proyecto cumple con los requerimientos funcionales brindados por cátedra de manera correcta. Hay pocos defectos realmente severos, consideramos que el timer cuando se presiona ayuda es el único defecto que afecta la jugabilidad. Hay un deslizador que no funciona, eso disminuye la calidad del sistema un poco. Aún así, es un sistema responsivo, amigable para el usuario tanto funcional como estéticamente.
El estado de las pruebas unitarias también es un poco deficiente. Alrededor del 60 % de las pruebas fue realizado. Siendo un 90 el mínimo deseable.
Apostamos a calificar el sistema como muy bueno. Hay omisiones en algunos aspectos como una funcionalidad sin implementar completamente y pruebas unitarias faltantes. Eso no quita que el software sea de calidad y realizado con un interesante grado de profesionalismo.

2.6. Reflexión

El proyecto englobó muchas cosas. Creemos que puede ser buena idea compartimentar estos aspectos para ser más claros:

- Código: El obligatorio nos hizo replantearnos varios elementos de diseño en orientación a objetos que no nos habían quedado totalmente claras en cursos anteriores. Por ejemplo, si implementar una clase abstracta o una interfaz para la clase Pregunta. También nos hizo pensar en el ciclo de vida de software desde la perspectiva del desarrollador: pensamos que nuestro código podría ser reutilizado, así que tuvimos otra motivación más para hacer un código con cierto profesionalismo. O al menos, que si viene otro programador luego de nosotros a meter mano en el código, este sea fácilmente maneible y comprensible, no solo para nosotros mismos. Personalmente nos habría gustado dedicar más tiempo a la teoría del código profesional y pulirlo aún más, pero el obligatorio engloba muchos más aspectos en lo que es producir software, así que los tiempos no ayudaron.

- Gestion: El obligatorio también nos hizo considerar el factor tiempo y esfuerzo. Nos hizo setear prioridades en las funcionalidades del código y en los demás aspectos: pruebas y docuentación.
  Los tiempos nos jugaron muy a favor, pero nos dio lástima no poder implementar el deslizador de sonido para controlar el volúmen de la música. Hubo un domingo con un corte de suministro de electricidad a nivel nacional y particularmente en nuestro barrio no hubo electricidad por más de 24 horas. Ese día fue un día perdido y nos hizo atrasarnos. Esto repercutió en no poder implementar tal funcionalidad.

- Usabilidad: Por primera vez, intentamos dar un salto en la calidad de la interfaz. Ya que en clase vimos un montón de conceptos orientados a la GUI, quisimos ponerlas en práctica, principalmente las heurísticas de Nielsen. Además, probamos con diferentes usuarios piloto el progreso de la GUI. Estas instancias de pruebas de GUI nos brindaron cuantiosos insumos para mejorarla, y

creemos que el resultado de la GUI es muy bueno. Por un lado, porque la idea que se nos ocurrió de la espiral fue muy flexible; y por otro, el feedback que nos dieron nuestros usuarios piloto.

- Pruebas: Las pruebas fueron algo completamente nuevo. Tanto las funcionales como las unitarias. Priorizamos la docuentación y el código antes que las pruebas. Eso se vio reflejado en un bajo porcentaje de cobertura por parte del JaCoCoverage, que como dicho anteriormente, nos devolvió una cifra no superior al 61 %. Creemos que tenemos el debe más grande el obligatorio en esta parte de pruebas. Aún así, creemos que la correcta implementación y la documentación compensan un poco esta carencia.

- Diseño: Creemos que la idea global del proyecto es genial. Es un juego de preguntas y respuestas con un twist muy interesante. Da una libertad al usuario que muchas otras aplicaciones no dan, sin sacrificar el control que tiene el sistema sobre los potenciales problemas que el usuario pueda generar. Tampoco sacrifica el control que tiene el usuario.

- Nos dió una lástima enorme que el sonido no se ejecute correctamente en el JAR.

Caso de uso: cargar preguntas desde archivo
Actor(es): usuario
Descripción: el usuario puede arrastrar archivos .txt en formato GIFT para cargar preguntas al sistema
Precondiciones: el usuario debe haber iniciado el sistema.

Acción de los actores	Respuesta del Sistema
1. El usuario arrastra un archivo .txt en formato GIFT	**2. El indica cuántas preguntas se cargaron**
3. El usuario arrastra un archivo que no es .txt o no tiene formato GIFT	**4. El sistema no actualiza la cantidad de preguntas**
5. El usuario carga un .txt cuando ya hay 39 preguntas	**6. El sistema no actualiza la cantidad de preguntas**
7. El usuario hace click en comenzar	**8. El sistema ingresa al tablero de juego**
1.1 El usuario hace click en comenzar	**1.2 El sistema espera a que se cargue alguna pregunta antes de avanzar**

Caso de uso: iniciar quiz
Actor(es): usuario
Descripción: el usuario puede elegir casilleros de pregunta para responder a alguna y utilizar algunos botones de funcionalidad (acabar, ayuda, generar reporte)
Precondiciones: haber cargado preguntas exitosamente

Acción de los actores	Respuesta del Sistema
1. El usuario pulsa el botón ayuda	**2. El sistema despliega la ventana de ayuda**
3. El usuario pulsa el botón volver	**4. El sistema regresa al tablero de juego**
5. El usuario hace click en el botón reporte	**6. No sucede nada, ya que no se respondió ninguna pregunta aún**
7. El usuario toca el deslizador de sonido en la parte derecha de la pantalla	**8. No sucede nada: no está implementado aún**

Caso de uso: responder preguntas
Actor(es): usuario
Descripción: el usuario puede elegir casilleros de pregunta para responder preguntas
Precondiciones: el usuario debe haber cargado, sin perder generalidad, 38 preguntas.

Caso de uso: generar reporte
Actor(es): usuario
Descripción: el usuario hace click en botón reporte y el sistema genera un reprote
Precondiciones: el usuario debe haber respondido, sin perder generalidad, al menos una de las preguntas que cargó.

Acción de los actores	Respuesta del Sistema
1. El usuario presiona un botón en gris oscuro	2. El sistema no hace nada: los botones gris oscuro no contienen preguntas.
3. El usuario presiona un botón gris claro correspondiente a una pregunta verdadero falso, cuya respuesta correcta es verdadero	4. El sistema despliega un panel donde se muestra la pregunta, el tiempo restante para responderla y botones verdadero y falso
5. El usuario hace click en verdadero antes que elapse el tiempo límite	6. El sistema indica con un sonido y un GIF que la respuesta fue correcta y redirige al mapa de botones, donde indica con un tick el casillero elegido y lo pinta de celeste
7. El usuario presiona un botón gris claro correspondiente a una pregunta verdadero falso, cuya respuesta correcta es falso	8. El sistema despliega un panel donde se muestra la pregunta, el tiempo restante para responderla y botones verdadero y falso.
9. El usuario hace click en verdadero antes que elapse el tiempo límite	10. El sistema indica con un sonido y un GIF que la respuesta fue incorrecta y redirige al mapa de botones, donde indica con una X el casillero elegido y lo pinta de azul oscuro
11. El usuario hace click en el botón reporte	12. El sistema genera un reporte pdf llamado reporte.pdf en la carpeta del proyecto en la carpeta PDF contenida en Resources contenida en Main contenida en src con las preguntas que ya fueron solucionadas, su información y la respuesta elegida por el usuario
13.El usuario presiona un botón gris claro correspondiente a una pregunta verdadero falso, cuya respuesta correcta es falso	14. El sistema despliega un panel donde se muestra la pregunta, el tiempo restante para responderla y botones verdadero y falso.
15. El usuario hace click en verdadero o falso luego de que elapse el tiempo límite	16. El sistema indica con un sonido y un GIF que la respuesta fue incorrecta y redirige al mapa de botones, donde indica con una X el casillero elegido y lo pinta de azul oscuro

Acción de los actores	Respuesta del Sistema
17. El usuario presiona un botón gris claro correspondiente a una pregunta corta respuesta	**18. El sistema despliega un panel donde se muestra la pregunta, el tiempo restante para responderla, un casillero para escribir y un botón ok**
19. El usuario tipea una respuesta incorrecta en el casillero para escribir y presiona ok dentro del tiempo límite	**20. El sistema indica con un sonido y un GIF que la respuesta fue incorrecta y redirige al mapa de botones, donde indica con una X el casillero elegido y lo pinta de azul oscuro**
21. El usuario presiona un botón gris claro correspondiente a una pregunta corta respuesta	**22. El sistema despliega un panel donde se muestra la pregunta, el tiempo restante para responderla, un casillero para escribir y un botón ok**
23. El usuario tipea una respuesta correcta en el casillero para escribir y presiona ok dentro del tiempo límite	**24. El sistema indica con un sonido y un GIF que la respuesta fue correcta y redirige al mapa de botones, donde indica con un tick el casillero elegido y lo pinta de celeste**
25. El usuario presiona un botón gris claro correspondiente a una pregunta corta respuesta	**26. El sistema despliega un panel donde se muestra la pregunta, el tiempo restante para responderla, un casillero para escribir y un botón ok**
27. El usuario toca el botón ok dentro del tiempo límite	**28. El sistema no responde y espera a que se ingrese un texto previo a tocar ok**
29. El usuario ingresa un texto correcto y presiona el botón ok dentro del tiempo límite	**30. El sistema indica con un sonido y un GIF que la respuesta fue correcta y redirige al mapa de botones, donde indica con un tick el casillero elegido y lo pinta de celeste**
31.El usuario presiona un botón gris claro correspondiente a una pregunta corta respuesta	**32. El sistema despliega un panel donde se muestra la pregunta, el tiempo restante para responderla, un casillero para escribir y un botón ok**
33. El usuario escribe una respuesta y toca el botón ok elapsado el tiempo límite	**34. El sistema indica con un sonido y un GIF que la respuesta fue incorrecta y redirige al mapa de botones, donde indica con una X el casillero elegido y lo pinta de azul oscuro**

Acción de los actores	Respuesta del Sistema
35. El usuario presiona un botón gris claro correspondiente a una pregunta múltiple opción	**36. El sistema despliega un panel donde se muestra la pregunta, el tiempo restante para responderla, y las respuestas posibles**
37. El usuario presiona un botón de opción correcta dentro del tiempo límite	**38. El sistema indica con un sonido y un GIF que la respuesta fue correcta y redirige al mapa de botones, donde indica con un tick el casillero elegido y lo pinta de celeste**
39. El usuario presiona un botón gris claro correspondiente a una pregunta múltiple opció	**40. El sistema despliega un panel donde se muestra la pregunta, el tiempo restante para responderla, y las respuestas posibles**
41. El usuario presiona un botón de opción correcta elapsado el tiempo límite	**42. El sistema indica con un sonido y un GIF que la respuesta fue incorrecta y redirige al mapa de botones, donde indica con una X el casillero elegido y lo pinta de azul oscuro**

Acción de los actores	Respuesta del Sistema
1. El usuario hace click en botón reporte	**2. El sistema genera un reporte pdf llamado reporte.pdf en la carpeta del proyecto en la carpeta PDF contenida en Resources contenida en Main contenida en src con las preguntas que ya fueron solucionadas, su información y la respuesta elegida por el usuario**

Bibliografía

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

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

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