

# Escuela Colombiana de Ingeniería Julio Garavito

## CSDT - Maestría en Informática

### Aplicación de prácticas de gestión de la deuda técnica a proyectos de Software - Un caso de estudio "Recuperando el reproductor de audio"

Ing. Sebastián Camilo Martínez Reyes  
Centro de estudios en ingeniería de software  
Email: sebastian.martinez@escuelaing.edu.co  
01 de Mayo 2022

**Resumen**—Un debate común en los proyectos de desarrollo de software se da entre dedicar tiempo a mejorar la calidad del software versus concentrarse en construir características más valiosas. Por lo general, la presión para ofrecer funcionalidad domina la discusión, lo que lleva a muchos desarrolladores a quejarse de que no tienen tiempo para trabajar en la arquitectura y la calidad del código. [1] La deuda técnica en las soluciones de software puede jugar un papel en ventaja de los equipos de trabajo. Saber identificarla, minimizarla en lo posible, interactuar con ella en el ciclo de desarrollo, verla de forma integral, conducirá a soluciones mas estables, donde la generación de valor siempre esté a la vista.

## I. INTRODUCCIÓN

Este documento pretende ilustrar este proceso de identificación, con un plan y flujo de trabajos detallados que permitirán servir como guía a equipos que comienzan la ardua tarea de lidiar con los intereses de una deuda que crece cada día. [2], así encontrarán compilado, el resultado de la aplicación de técnicas de gestión de la deuda técnica a un proyecto publico de código legado como caso de estudio. Donde se cubrirá el análisis, gestión y resolución de la deuda técnica a una solución de reproducción de audio construida por un estudiante de ciencias de la computación.

Se identificarán las oportunidades de mejora y se implementarán los cambios necesarios en el repositorio [3] relacionado como Fork [4].

## II. METODOLOGÍA

### II-A. Bitácora

Detalle del cronograma de actividades :

1. 22- 24 de febrero de 2022 : Instalación y documentación inicial de la solución, fundación de este repositorio.
2. 25-26 de febrero de 2022 : Identificación de Code-Smells
3. 05-06 de marzo de 2022 : Valoración de características de Clean Code y prácticas XP
4. 09-13 de marzo de 2022 : Creación de pruebas de unidad, análisis de deuda técnica de pruebas.
5. 18-20 de marzo de 2022 : Utilización de herramientas de análisis estático de código.
6. 25-27 de marzo de 2022 : Introducción de pipelines de CI (Continuous integrations).
7. 01-03 de Abril de 2022 : Análisis de deuda de arquitectura.
8. 22-24 de Abril de 2022 : Análisis de deuda ATAM + QAW.

### II-B. Flujo de trabajo

Como parte del desarrollo del contenido del curso, se desempeñaron las siguientes actividades en la base de código [3], en cada etapa se implementaron de forma práctica distintas actividades que apalancaron la identificación integral de la deuda técnica de la solución, siguiendo el siguiente esquema de trabajo y cronograma.



Figura 1: Flujo de trabajo

### II-C. Detalles técnicos del proyecto

El reproductor de audio utiliza el siguiente stack de tecnología :

1. Java 16+ [5]
2. JavaFX 17 [6]
3. Maven [7]
4. PostgreSQL [8]
5. CSS
6. Docker

Características disponibles:

1. Reproducir musica en formato MP3
2. Clasificación de canciones por género o estado de ánimo (Mood)
3. Búsqueda de canciones por álbum, artista o nombre
4. Búsqueda de letras de las canciones en Azlyrics
5. la capacidad de conectarse automáticamente a YouTube y reproducir la canción allí

Documentación de la solución:

Se presentan cómo referencia las diferentes vistas relevantes en los diferentes niveles de alcance de la solución, persistencia, interacción, composición lógica y colaboración .

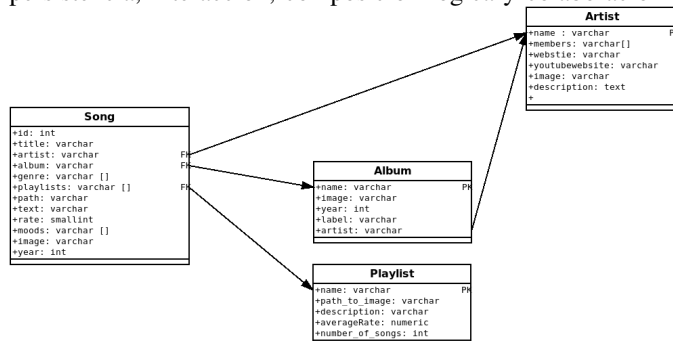


Figura 2: Modelo entidad-relación

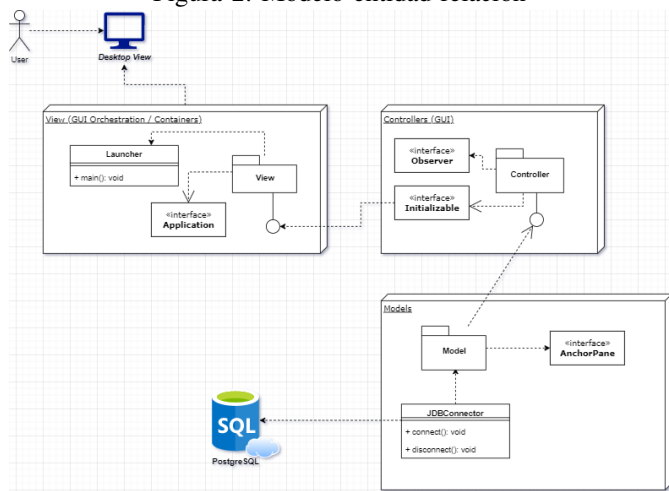


Figura 3: Modelo de interacción

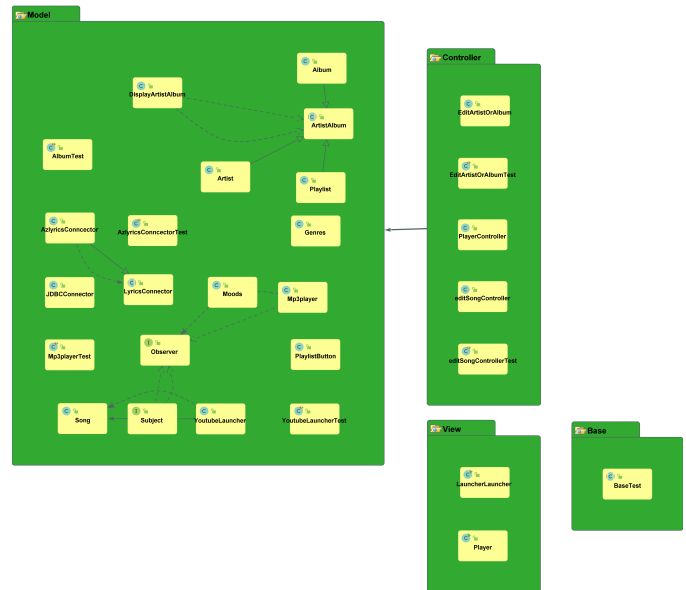


Figura 4: Vista por módulos

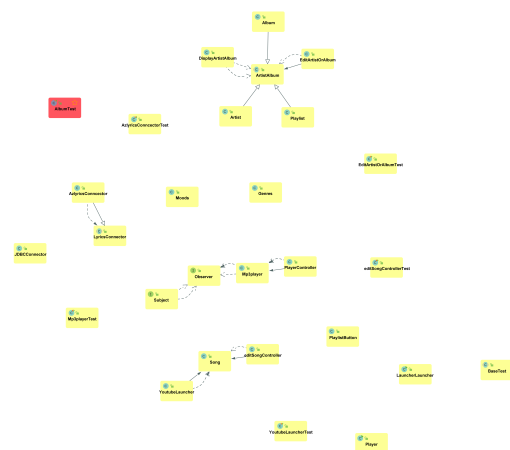


Figura 5: Vista por dependencias

### III. RESULTADOS

### III-A. Identificación de Code-Smells

Se detectan dos clases que acumulan gran parte de la deuda técnica de código fuente, estas son JDBCConector y PlayerController que concentrarán la mayor parte de los esfuerzos de refactorización.

En general se encuentran falencias en los principios SOLID, encapsulamiento y segregación de responsabilidades.

Se recomiendan aplicar las estrategias de composición de métodos y movilización de características: Extract Method Replace Method with Method Object Extract Class Replace Temp with Query por mencionar algunas.[1]

Se recomienda crear una suite de pruebas unitarias previo a cualquier actividad de refactorización. [9]

### III-B. Valoración de características de Clean Code y prácticas XP

**Código enfocado:** En Clean-code el código debería ser enfocado y tener un propósito específico en donde la simplicidad es un patrón común. Para el estado actual de la solución, se evidencia que a pesar de que las clases están bien divididas y en teoría los propósitos específicos están bien definidos, aún hay clases extensas en tamaño con múltiples propósitos.

**Recomendación XP:** Para estos casos es recomendable usar la práctica XP de refactoring para segregar y segmentar el propósito de esta clase, en clases auxiliares con alcances bien definidos.

**Regla del boy-Scout:** Esta característica no se cumple, pues hasta este punto la solución ha sido construida por una sola persona (El autor original). Una recomendación en el plan de refactorización sería mantener los principios de Clean Code después de cada iteración, garantizando que el código fuente mejora en simplicidad, legibilidad y abstracción después de cada modificación.

**Recomendación XP:** En caso de involucrar a un equipo de desarrollo en versiones posteriores, se recomienda usar estrategias XP como Pair-Programming o Peer reviewing, lo que evitará el sesgo por confirmación (El creador siempre cree que su construcción no pudo haberse hecho mejor) y garantizará la regla del boy scout en cada modificación (commit).

**Entendible :** En este segmento se evidencia buen manejo del lenguaje de dominio en el código fuente original, hay pocas funciones que no se lean naturalmente. Se detectan algunas oportunidades de mejora, como métodos largos que disminuyen la legibilidad.

**Recomendación XP:** Una práctica XP que permitiría mantener este atributo de calidad vigente, sería la definición de un estándar de codificación para el proyecto. Esto permitiría mantener un patrón consistente de nombramiento de variables, clases, métodos. Allí se definirían los patrones de diseño a usar, la arquitectura general y definiría una línea base de implementación. [10]

### III-C. Creación de pruebas de unidad, análisis de deuda técnica de pruebas

**Hallazgos:** El proyecto tiene una dependencia fuerte con los componentes de JavaFx.

Estos al ser componentes de UI utilizan multi-threading para la mayoría de sus invocaciones, en este caso para generar pruebas en componentes de FX se utiliza la suite de pruebas de unidad TestFX la cual permite crear entornos virtuales de pruebas para componentes FX, lo que habilita las pruebas de las unidades de negocio y middle-ware de la aplicación.

**Propuestas:** En contraposición a la librería de Jafafx, se recomienda, para mantener el producto en el tiempo, incluir una secuencia de pasos de pruebas funcionales que complementen estas pruebas de unidad.

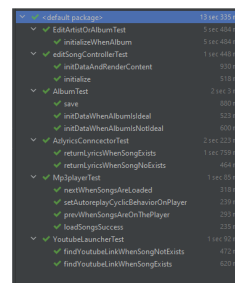
**Limitaciones:** JavaFx sólo puede verificar la composición de interfaces gráficas a nivel estructural, a nivel de orquestación de eventos, pero palidece a la hora de hacer una verificación de UX y emular algunos escenarios que sólo pueden ser verificados vía pruebas funcionales como por ejemplo : La reproducción de audio de la canción seleccionada, la funcionalidad de repetición del autoreplay, el comportamiento circular de la lista de reproducción .

**Trabajo futuro:** Otro punto focal de la aplicación está en el JDBCConnector, para realizar pruebas de esta clase será necesario emular el ambiente de persistencia de la aplicación. Para ello se implementará el uso de Mocks de JDBC con Mockito, esto permitirá simular ambientes de persistencia y mocks de objetos DAOs de forma simple. [11]

### Resultados de la implementación de pruebas de unidad [12]:

#### Resumen de pruebas ejecutadas

En total se implementaron 6 clases de pruebas de unidad cubriendo un total de 14 escenarios críticos para el funcionamiento del reproductor. Se prueban integraciones externas con servicios de reproducción de videos de Youtube y obtención de letras de canciones por medio de AZLyrics.com



Test Name	Duration
TestNG package	15 sec 359 ms
✓ EditArtistOnAlbumTest	1 sec 484 ms
✓ initializeOnAlbum	1 sec 484 ms
✓ addSongControllerTest	1 sec 484 ms
✓ setDataSourceAndContent	410 ms
✓ initialize	510 ms
✓ AlbumTest	2 sec 540 ms
✓ save	500 ms
✓ setDataWhenAlbumIdeal	500 ms
✓ setDataWhenAlbumNotIdeal	600 ms
✓ AsyncConnectorTest	2 sec 222 ms
✓ return_syncWhenSongExists	1 sec 738 ms
✓ return_syncWhenSongNotExists	440 ms
✓ MediaPlayerTest	1 sec 10 ms
✓ nextWhenSongNotLoaded	210 ms
✓ setAutoplayCycleBehaviourOnPlayer	210 ms
✓ previousWhenSongNotOnThePlayer	210 ms
✓ loadSongsFromData	210 ms
✓ YoutubeAutosheetTest	1 sec 50 ms
✓ findYoutubeLinkWhenSongNotExists	470 ms
✓ findYoutubeLinkWhenSongExists	420 ms

#### Resumen acumulado de Coverage de la suite implementada

Se logra cubrir un total del 55% de las clases, con una cobertura general del 42% de la funcionalidad de la aplicación, se excluyen modelos no utilizados y aquellos que se dedican exclusivamente a actividades de persistencia JDBC.

Se cubre un 10% a nivel de controlador, esto debido al fuerte cumplimiento con librerías de JFX que impiden probar las clases de forma programática.



Category	55% (11/20)	54% (94/120)	42% (255/605)
ModelTests			
Controller	10% (2/12)	8% (2/20)	19% (13/67)

Figura 6: Resultado de pruebas unitarias - coverage

### III-D. Utilización de herramientas de análisis estático de código

Herramienta seleccionada: La herramienta seleccionada para la aplicación de modelos de Calidad cómo SQALE fue SonarQube en su versión Cloud SonarCloud.

SonarCloud fue seleccionada del grupo de aplicaciones por su facilidad de ser integrada a entornos de proyectos GitHub, sus mecanismos de análisis recaen en la creación de flujos de trabajo con GitHubActions lo que permite integrar la herramienta al flujo de CI/CD (Continuous Integration / Continuous Delivery).

Resumen de resultados :

Paquete	Clase	Deuda Técnica Acumulada	Riesgo
Model	JDBCController	5h 40min	Alto
Model	AZLyricsConnector	1h 37min	Alto
Model	EditArtistOrAlbum	25min	Alto
Model	Song	10min	Medio
Controller	PlayerController	5h 5min	Medio

Figura 7: Deuda técnica acumulada

Paquete	Clase	Bugs	Riesgo
Model	JDBCController	12	Alto
Model	AZLyricsConnector	1	Alto
Model	Song	1	Medio

Figura 8: Métricas de confiabilidad

Action items : Revisando estos bugs se encuentra que la mayoría de estos se deben a que muchas de las operaciones de JDBCController, no realizan el cierre adecuado de las conexiones con las fuentes de datos (BD Postgres). Otro patrón que se encuentra es que las clases que son ejecutadas en Threads de forma asíncrona, no realizan un buen tratamiento de las excepciones de interrupción de proceso, por lo que se recomienda cerrar el Thread principal (Quien invoca la función) para resolver estos hallazgos.

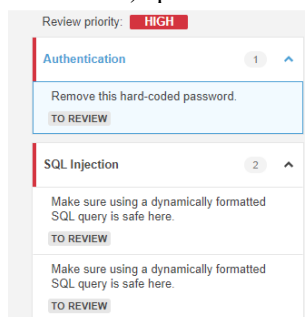


Figura 9: Análisis de seguridad

Action items : Añadir un sistema de logs a la solución que concentren los errores de la misma y que ofusquen información sensible.

Controlar la ejecución de sentencias SQL por medio de variables seguras y procedimientos almacenados. (Revisar la opción de incluir un ORM)

Las contraseñas o tokens de fuentes externas deben colocarse cómo variables de entorno o configurarse a nivel de ambiente, no deberían estar expuestas en el código fuente.

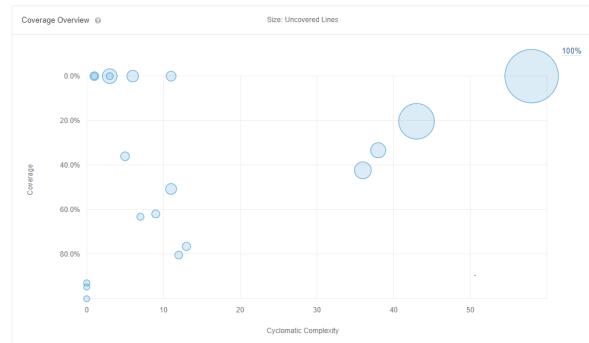


Figura 10: Resultado de análisis de coverage

Action items : Implementar pruebas de unidad para la clase PlayerController.

Aumentar escenarios de prueba para la clase JDBCController.

Mejorar la implementación de pruebas para las clases Song y MP3 Player.

El proceso detallado de implantación de está herramienta, se encuentra detallado en la bitácora del repositorio [13].

### III-E. Introducción de pipelines de CI (Continuous integrations).

Se implementan los siguientes pipelines de integración continua usando git-hub actions [14]:

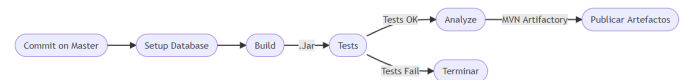


Figura 11: Flujo de integración continua



Figura 12: Flujo de integración de herramientas colaborativas

El proceso detallado de implantación de estos pipelines, se encuentra detallado en la bitácora del repositorio [15].

### III-F. Análisis de deuda de arquitectura

Métricas en consideración:

Métrica	Abreviación	Explicación
Lines Of Code	LOC	Cantidad de líneas de código fuente del artefacto
Lack of cohesion of methods	LCOM	Falta de cohesión en los métodos en % (escala de 0 - 1)
Weighted methods per class	WMC	Suma de la complejidad ciclomática de la clase
Fan-In	FIN	Dependencia de uso de la clase (Cuántas clases usan la definición)
Fan-Out	FOU	Dependencias propias de la clase (Cuántas definiciones externas usa la clase)
Depth of inheritance Tree	DIT	Profundidad en el árbol de herencia (permite detectar delegación de comportamientos de alta profundidad)

Figura 13: Métricas de análisis de arquitectura

Resultados:

Package Name	Type Name	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
Controller	EditArtistOrAlbum	33	5	0	0	0.5	1	1
Controller	editSongController	72	10	0	0	0	1	4
Controller	PlayerController	396	62	0	0	0.16129032	0	1
Controller	EditArtistOrAlbumTest	23	2	0	0	0	0	2
Controller	editSongControllerTest	37	3	0	0	0	0	2
Model	Album	75	12	0	1	0	2	1
Model	Artist	42	11	0	1	0.36363636	0	0
Model	ArtistAlbum	31	8	3	0	0.28571429	2	0
Model	AzlyricsConnector	81	13	0	1	0.33333333	2	0
Model	DisplayArtistAlbum	58	3	0	0	-1	0	2
Model	Genres	21	0	0	0	-1	1	0
Model	JDBCCConnector	277	42	0	0	0	11	2
Model	LyricsConnector	4	2	1	0	-1	0	0
Model	Moods	16	0	0	0	-1	1	0
Model	MP3player	101	35	0	1	0.11111111	1	1
Model	Observer	3	1	0	0	-1	2	0
Model	Playlist	11	3	0	1	1	0	0
Model	PlaylistButton	17	1	0	0	-1	0	0
Model	Song	162	38	0	0	0.08333333	6	4
Model	SongBuilder	53	12	0	0	0.9	1	1
Model	Subject	5	3	1	0	-1	0	1
Model	YoutubeLauncher	46	8	0	0	0	2	1
Model	AlbumTest	44	4	0	0	0	0	2
Model	AzlyricsConnectorTest	12	2	0	0	-1	0	1
Model	MP3playerTest	64	5	0	0	0	0	3
Model	YoutubeLauncherTest	14	2	0	0	-1	0	2
View	Launcher	5	1	0	0	-1	0	1
View	Player	44	6	0	0	0.66666667	1	1
Base	BaseTest	13	2	0	0	-1	0	1

Figura 14: Resultados de análisis de ARQ

Hallazgos generales:

- Vemos que las clases JDBCCConnector y Song son críticas para la solución juzgando por sus métricas de Fade-In.
- Otro punto notable es que la clase JDBCCConnector sea crítica y su complejidad ciclomática general WMC sea de 42, lo que implica que se debe trabajar en esta abstracción y podemos estar evidenciando un anti-patrón de clase Dios/Demasiada responsabilidad.
- Tenemos clases con complejidades grandes por fuera del JDBCCConnector, cómo el PlayerController, MP3Player y Song, sus WMC son muy superiores a la media y deben ser evaluados, trayendo un análisis anterior en mención estás clases concentran gran cantidad de los code-smells, por lo que podemos decir que las tareas de mejora futuras (Sprints de remediación) deben estar focalizadas en ellas.
- En general no vemos clases con falta de cohesión, exceptuando SongBuilder, que debería verificarse la posibilidad de reemplazar la clase cómo parte de las características propias de Song.
- La clase Player se muestra bajamente cohesiva, pero se ve cómo una falsa alarma por ser el punto de entrada de GUI de la solución (Debe revisarse).
- Se ve la presencia de muchos Magic Numbers lo largo de la implementación, lo que hace pensar en la necesidad de abstraer una clase de configuración con estos atributos generalmente gráficos o de frecuencia.
- Se había mencionado anteriormente en la introducción de herramientas, se reitera la necesidad de añadir un mecanismo de tratamiento de errores cómo un Logger para la solución, en este caso desde las perspectiva del Logger cómo componente.

El proceso detallado de análisis, se encuentra detallado en la bitácora del repositorio [16].

### III-G. Análisis de deuda ATAM + QAW

ATAM es un método altamente estructurado para evaluar un diseño de arquitectura el cuál permite detectar, de manera temprana, riesgos técnicos, conflictos entre atributos, puntos sensitivos del diseño y soluciones. [17]

QAW (Quality Attribute Workshops) por su parte proporciona un método para la determinación de un sistema crítico basado en el análisis de los atributos de calidad, al igual que disponibilidad, rendimiento, seguridad, interoperabilidad y modificabilidad. En este orden, QAW complementa la arquitectura ATAM, puesto que identifica los atributos relevantes de calidad y permite establecer los requisitos del sistema antes de la existencia de una arquitectura de software.[18]

En este análisis se presentará una primera aproximación a la documentación arquitectónica de esta solución, así cómo una evaluación inicial a sus atributos claves de negocio (Drivers) cómo input inicial a las operaciones y mantenimiento de arquitectura futuros de la misma.

Presentación de la arquitectura: La documentación mediante vistas es una forma de documentar el diseño de arquitectura considerando los intereses de los interesados en el producto. Existen variadas maneras de expresar las vistas y sus interrelaciones, en esta oportunidad y considerando el alcance del producto, se presentarán las vistas de componentes combinado con una vista a las interacciones a alto nivel entre los mismos.

Algunas de estas vistas se generaron utilizando la herramienta Code Iris para IntelliJ y fueron mostradas en la sección de documentación técnica.

Se muestra a continuación un ejemplo de la aplicación del ciclo de validación de atributos de calidad en formato QAW.

Evaluación de atributos de calidad 	
Elemento de trabajo	Detalle
Escenario	La completitud de la reproducción de una canción, la adición de un album y de una lista deberían tomar menos de 5 minutos
Objetivo de Negocio	Un reproductor de música atractivo y usable
Atributos de calidad relevantes	Usabilidad
Estímulo	Los usuarios reproducen una canción, navegan por los menús y hacen listas de reproducción
Fuente del estímulo	Un usuario en la aplicación de escritorio
Ambiente	La aplicación de escritorio ha sido iniciada y los sistemas de almacenamiento se encuentran en línea.
Artefacto	Aplicación de escritorio
Respuesta	Las actividades de los usuarios se completan satisfactoriamente en el tiempo requerido
Medida de la respuesta	$[(\text{Tiempo de la tarea}) / (\# \text{ de tareas})] < 5m$
Preguntas	¿Qué sucede en la GUI cuando la canción cargada toma mucho tiempo de almacenamiento o la cantidad de listas de reproducción es muy grande?
Inconvenientes	Los usuarios deben estar capacitados para usar la aplicación correctamente y actualmente el sistema no cuenta con tutoriales interactivos o guías de uso

Figura 15: Escenario de validación 1

Escenario #2

Elemento de trabajo	Detalle
Escenario	El reproductor de audio se distribuirá a lo largo del mundo / Público general
Objetivo de Negocio	Un reproductor de música disponible para todos
Atributos de calidad relevantes	Internacionalización
Estímulo	La GUI responde al idioma del sistema en el que se encuentre instalado
Fuente del estímulo	Un usuario en la aplicación de escritorio
Ambiente	La aplicación de escritorio ha sido iniciada y los sistemas de almacenamiento se encuentran en línea.
Artefacto	Aplicación de escritorio
Respuesta	Los usuarios pueden entender el contenido de la UI en su idioma de sistema
Medida de la respuesta	[ (Σ idiomas de las regiones de uso ) / # de países en donde se usa el reproductor ] >= 0.90
Preguntas	¿Se pretenden utilizar lenguajes de uso general o lenguajes de minorías étnicas / inclusión de dialectos?
Inconvenientes	

Figura 16: Escenario de validación 2

Escenario #3

Elemento de trabajo	Detalle
Escenario	Se requiere que el reproductor pueda ser usado en las diferentes plataformas Windows e IOS
Objetivo de Negocio	Un reproductor de música disponible para todos
Atributos de calidad relevantes	Interoperabilidad
Estímulo	Los usuarios pueden usar el sistema en los diferentes dispositivos/plataformas con una funcionalidad mayor al 80%
Fuente del estímulo	Un usuario en la aplicación de escritorio
Ambiente	La aplicación de escritorio no ha sido iniciada y los sistemas de almacenamiento se encuentran en línea.
Artefacto	Aplicación de escritorio
Respuesta	La aplicación se inicializa de forma correcta y permite realizar las actividades a los usuarios en distintas plataformas
Medida de la respuesta	[ (Σ funcionalidades exitosas en la plataforma en uso ) / # de funcionalidades totales ] > 0.80
Preguntas	¿Que sucede con las plataformas móviles o los OS de uso no general?
Inconvenientes	

Figura 17: Escenario de validación 3

REFERENCIAS

[1] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[2] —, “Technical debt quadrant,” *Martin Fowler*, pp. 14–0, 2009.

[3] D. Kurosz, “Player,” <https://github.com/DanielVeB/Player>, 2019.

[4] S. C. M. Reyes, “Player,” <https://github.com/sc-martinez/Player>, 2022.

[5] Oracle, “Java,” <https://www.oracle.com/java/technologies/javase/jdk16-archive-downloads.html>, 2022.

[6] OpenJFX, “Openjfx,” <https://openjfx.io/>, 2022.

[7] Apache, “Maven,” <https://maven.apache.org/>, 2022.

[8] T. P. G. D. Group, “Postgresql,” <https://www.postgresql.org/>, 2022.

[9] S. C. M. Reyes, “Player,” <https://github.com/sc-martinez/Player/blob/master/Code-Smells.md>, 2022.

[10] —, “Player,” <https://github.com/sc-martinez/Player/blob/master/CleanCode-XP-Assessment.md>, 2022.

[11] M. Grzejszczak, *Mockito Cookbook*. Packt Publishing Ltd, 2014.

[12] S. C. M. Reyes, “Player,” <https://github.com/sc-martinez/Player/blob/master/TechnicalDebt%20-Tests.md>, 2022.

[13] —, “Player,” <https://github.com/sc-martinez/Player/blob/master/IntroducingToolsOnTheProcess.md>, 2022.

[14] C. Chandrasekara and P. Herath, “Introduction to github actions,” in *Hands-on GitHub Actions*. Springer, 2021, pp. 1–8.

[15] S. C. M. Reyes, “Player,” <https://github.com/sc-martinez/Player/blob/master/IntroducingCI.md>, 2022.

[16] —, “Player,” <https://github.com/sc-martinez/Player/blob/master/ArchitecturalDebt.md>, 2022.

[17] R. Kazman, M. Klein, and P. Clements, “Atam: Method for architecture evaluation,” Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, Tech. Rep., 2000.

[18] M. R. Barbacci, R. Ellison, A. J. Lattanze, J. A. Stafford, and C. B. Weinstock, “Quality attribute workshops (qaws),” CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 2003.