Proyecto Intermodular

Gestión de la configuración, análisis y gestión del riesgo y aseguramiento de la calidad

Redactado por: Santiago Quiceno Vargas

2ºDAM

Índice

Contenido

Gestión de la configuración

El texto de **Gestión de la Configuración** tiene como objetivo establecer aquellos procedimientos necesarios para controlar todo elemento del proyecto que sean susceptibles de sufrir modificaciones, así como gestionar las incidencias que puedan surgir durante su ejecución una vez que el programa sea funcional por el usuario. Este plan garantiza la trazabilidad, integridad y coherencia del producto final desarrollado, además de facilitar el control de versiones y la resolución ordenada de problemas.

Elementos de la Configuración (EC)

Los **Elementos de la Configuración** son todos aquellos componentes del proyecto que puedan verse afectados por cambios, errores o mejoras. Dichos componentes requieren control de versiones. Los principales EC identificados para proyecto son los siguientes:

- **Código fuente de la aplicación Android:** toda la arquitectura interna de la aplicación escrita en lenguaje Java, XML y Kotlin.
- **Documentación técnica:** texto relacionado a la especificación de requisitos, manuales de usuarios y diagramas UML.
- **Base de datos:** estructura de tablas, código de las mismas en lenguaje SQL, y sus consecuentes scripts de creación y migración.
- **Recursos multimedia:** imágenes, iconos, archivos de audio, GIFs y otros elementos usados en el proyecto.
- **Archivos de configuración del entorno de desarrollo:** archivos *gradle* y *manifest* con su respectivo código Kotlin y XML usados para definir la el diseño del proyecto.

Nomenclatura de versiones

Para la clara y coherente identificación de cada versión de un Elemento de Configuración (EC), se ha de utilizar la siguiente nomenclatura:

[NombreEC]_v[Major].[Minor].[Patch]

La explicación de cada apartado de la nomenclatura es el siguiente:

- **NombreEC:** referido a cada elemento especificado en el apartado *Elementos de la Configuración (EC);* por ejemplo, ***ManualDeUsuario_v1.0*.**
- **Major:** todos aquellos cambios estructurales importantes o funciones críticas que comprometan el flujo natural de la aplicación.
- **Minor:** incluye mejoras menores y ampliaciones que expandan la experiencia del usuario, más no comprometen la usabilidad total o parcial de la aplicación.

- **Patch:** parches o correcciones sobre aspectos puntuales y que no necesitan tanta atención como una de las actualizaciones *Major* o *Minor* anteriormente mencionadas.

Así pues, la nomenclatura sobre uno de los posibles elementos clave en el desarrollo de la app tendría la siguiente apariencia:

MainActivity_v1.2.3

Sistema de almacenamiento de versiones

Todas las versiones de los EC explicadas en el apartado anterior se almacenarán en un sistema de control de versiones distribuido. Para ello, se usará la herramienta **Git** en complemento con la plataforma **GitHub,** esta última siendo usada como repositorio central. Se han de establecer las ramas específicas para cada fase del desarrollo del proyecto, siguiendo un lenguaje común establecido en este documento.

Se usarán **cuatro ramas principales** sobre las que se desarrollarán las subramas que complementarán el proyecto: la rama ***main*** será la versión estable, lista para enviar a producción; la rama ***develop*** será para guardar las versiones en desarrollo activo por el equipo; la rama ***feature/**** serán las ramas para añadir nuevas funcionalidades a la aplicación base; y en la rama ***hotfix/**** se incluirán aquellas correcciones que necesiten de extrema atención por parte del equipo desarrollador al ser incidencias que comprometen el desarrollo de la aplicación.

Cada uno de los *commits* que se realicen al actualizar alguna de las ramas o subramas del Git deberá incluir un mensaje descriptivo que indique el cambio realizado por el autor/desarrollador, el nombre del autor/desarrollador realizador del cambio, y la fecha y hora a la que el cambio fue realizado en la rama.

Metadatos de versión

Se deberán de almacenar una serie de metadatos para cada una de las versiones registradas de un EC. Estos metadatos son el identificador único de versión, el nombre del EC, la versión asignada (según su nomenclatura), la fecha de creación, el autor del cambio, la descripción del cambio y el estado de la versión, distinguiendo este último apartado entre *Estable*, *En revisión* u *Obsoleto*.

Estos metadatos se deberán de gestionar dentro del propio repositorio de Git mediante etiquetas (*tags*) y archivos de documentación complementaria (*CHANGELOG.md*) para una mayor claridad y orden en el proyecto.

Procedimiento de gestión de incidencias

Ante la aparición de una incidencia durante la ejecución del proyecto (independientemente del ámbito o EC en el que esta incidencia haya ocurrido) deberá llevarse a cabo el siguiente procedimiento previamente estructurado:

1. **Registro de la incidencia:** se documentará la incidencia en el sistema de seguimiento (apartado exclusivo para este objetivo, llamado **GitHub Issues**); se asignará un identificador único, una descripción detallada sobre el problema, el EC afectado y la fecha de detección; y se clasificará la incidencia según su tipo, distinguiendo entre funcional, técnica, documental o de integración.
2. **Evaluación de la incidencia:** el equipo deberá evaluar analizar la causa raíz del impacto en el proyecto. Se deberán determinar el nivel de prioridad y la urgencia de resolución, clasificando el problema en ***alto, medio*** o ***bajo.*** Además, se asignará a un responsable del equipo desarrollador para el tratamiento del inconveniente.
3. **Resolución de la incidencia:** si la incidencia afecta directamente al código fuente de la aplicación base, se deberá crear una subrama específica para tratarla y documentarla dentro de la rama principal ***hotfix/*** (correcciones de extrema atención). El responsable encargado de la solución de la incidencia aplicará todas las correcciones necesarias y validará la posible solución mediante pruebas en compañía de uno o más miembros del equipo desarrollador para dejar constancia. Una vez que el problema se haya mitigado, se actualizará la versión del EC afectado y se documentará el cambio realizado. Por último, se deberá cerrar la incidencia en el sistema de seguimiento, indicando la solución aplicada y la fecha de cierre.

Nota: *dependiendo de la gravedad de la incidencia, el líder del equipo desarrollador podrá designar a más de un responsable encargado de la mitigación del problema siempre y cuando su colaboración con la solución de dicho problema no afecte al flujo natural de desarrollo del proyecto.*

Análisis y gestión del riesgo

En base al proyecto de la aplicación de libros electrónicos, el **Plan de Gestión de Riesgo** tiene como finalidad identificar, evaluar y controlar todos los posibles eventos que puedan afectar al desarrollo del trabajo de una manera negativa. El siguiente plan establece una serie de procedimientos claros para anticipar problemas, minimizar el daño del impacto en caso de que estos problemas lleguen a surgir, y garantizar una continuidad clara para con el proyecto.

Identificación de riesgos

Analizando el trabajo, se han identificado los siguientes riesgos principales asociados al proyecto:

1. **Retrasos en el desarrollo:** por la cantidad de funciones que se pretenden implementar en la versión base de la aplicación (lectura, foros, *Marketplace*), se prevé que los retrasos ocurrirán durante el desarrollo del proyecto.
2. **Fallos técnicos en Android Studio:** abarca tanto errores en el código que produzcan excepciones como también incompatibilidad con librerías externas.
3. **Problema de integración con la base de datos:** es posible que surjan desde errores en relaciones y pérdidas de datos.
4. **Baja coordinación del equipo:** tiene que haber comunicación entre los integrantes del proyecto para que este pueda crecer orgánicamente y así evitar repetición de trabajo (que dos desarrolladores creen una funcionalidad básicamente idéntica en lugar de enfocarse en dos cuestiones diferentes) o que algún miembro del equipo se quede colgado sin nada que hacer.
5. **Riesgos de seguridad:** desde gestión de contraseñas hasta protección de datos de los usuarios.
6. **Sobrecarga de funcionalidades:** se tendrá especial cuidado con no implementar apartados declarados *En hibernación* antes de tiempo.
7. **Falta de pruebas suficientes:** serán fundamentales las pruebas para evitar que excepciones no previstas salten en producción.
8. **Riesgos externos no previstos:** no surgen por culpa del propio equipo desarrollador, pero afectan al producto entregado de igual manera (cambios en requisitos académicos, plazos de entrega o disponibilidad de recursos).

Evaluación de riesgos

Para este apartado, se ha diseñado una tabla ilustrativa para explicar el nivel de **Probabilidad, Impacto y Nivel de riesgo** de cada riesgo. Se calificarán, de arriba abajo, los riesgos más importantes a tratar hasta los riesgos menos importantes o secundarios.

Riesgo	Probabilidad	Impacto	Nivel de riesgo
Retrasos en el desarrollo	Alta	Alta	Critico
Fallos técnicos	Media	Alta	Alto
Problemas de integración BBDD	Media	Alta	Alto
Falta de pruebas suficientes	Media	Alta	Alto

Riesgos de seguridad	Baja	Alta	Alto
Sobrecarga de funcionalidades	Alta	Media	Alto
Baja coordinación	Media	Media	Medio
Riesgos externos	Baja	Media	Medio

Planificación de respuestas

Para cada riesgo analizado y previsto se han de establecer acciones preventivas y de contingencia para intentar subsanar el daño provocado de la manera más efectiva posible.

Para los **retrasos en el desarrollo** se aplicará la medida preventiva resumida en un **cronograma realista, especificado con el diagrama de Gantt,** requerido además para la presentación del proyecto, donde se asignarán las tareas a realizar de manera clara y concisa. El responsable sobre este cronograma de actividades será el **líder del equipo desarrollador.** Por otra parte, las medidas de contingencia ante los retrasos que puedan surgir será **priorizar nuevamente las funcionalidades,** y en otros términos, **posponer todos los apartados en hibernación,** o **descartarlos definitivamente** si se prevé que su inclusión será incompatible con la fecha de entrega del proyecto.

Respecto a los **fallos técnicos** (tanto en Android Studio como en otras plataformas de desarrollo) se intentará **mantener actualizado el entorno y las librerías** que se estén usando. Ante esta preventiva, **realizar pruebas de compatibilidad** también seria ideal. El **equipo técnico** será el responsable de este apartado. Como medidas de contingencia, será de fundamental prioridad **documentar errores y buscar soluciones alternativas** (*stack overflow y repositorios oficiales*). El responsable de esta tarea será el **desarrollador asignado para esta tarea.**

Para evitar los **problemas de integración de BBDD,** se tendrá que **revisar el modelo entidad-relación** y **realizar pruebas unitarias de consultas** para asegurar el correcto funcionamiento. El **miembro o miembros encargados de la base de datos** será el designado para dicha tarea. Como medidas de contingencia, se priorizarán las **realizaciones de copias de seguridad** de la base de datos designada. También se deberán **aplicar correcciones en migración** si fuese necesario. El **equipo técnico** será el encargado de mitigar este riesgo.

Con la **coordinación del equipo** se presenta un posible problema de falta de disponibilidad, además de que cada persona miembro del equipo desarrollador también tiene otras responsabilidades. Para evitar la falta de sincronía entre los miembros del equipo, se **organizarán reuniones semanales de seguimiento** (preferiblemente al inicio de semana o al final de la misma); también se usará **GitHub Projects** para la asignación de tareas semanales o mensuales y la asignación de quién tiene que hacer qué. Esta medida, por obvias razones, **incluye a la totalidad del equipo desarrollador.** En el caso de que haya descoordinación

en el equipo, se **organizará una reunión prioritaria en la que se redistribuirán las tareas.** También se ha de **reforzar la comunicación interna** mediante cualquier medio que los desarrolladores estén dispuestos a usar (WhatsApp, Discord, entre otros).

Para los posibles **riesgos de seguridad** se ha de incluir un correcto **cifrado de contraseñas, control de accesos por rol** y, en ultimada estancia, **uso de buenas practicas de seguridad.** Como medidas de contingencia se aplicarán **correcciones inmediatas de vulnerabilidad y notificación al equipo.** El equipo desarrollador será el responsable de estas dos tareas.

Sobre las **sobrecargas de funcionalidades,** lo más importante será **dejar los apartados extras en hibernación hasta consolidar el núcleo funcional.** Como medida de contingencia, sencillamente, será la **eliminación de funcionalidades no críticas o vitales,** y reasignar las tareas que cada miembro tiene que hacer. De la revisión y el proceso orgánico del proyecto se encargará el **líder del equipo desarrollador.**

Para evitar la falta de pruebas suficientes se realizará un **plan de pruebas unitarias e integración desde fases tempranas.** Estas pruebas pueden incluirse como una "subrama" dentro del cronograma. Se intentarán priorizar las pruebas para evitar la **corrección de errores surgidos en producción,** y si esto ocurre, **corregirlos lo antes posible para darles el visto bueno y continuar con el proceso de producción.**

Por último, los **riesgos externos** son más difíciles de prever debido a su carácter aleatorio. La mejor medida que se podrá aplicar será un **continuo monitoreo de plazos y requisitos académicos,** así como **mantener la comunicación con los técnicos analistas del proyecto** (tutores, empresarios, accionistas, etc.). Como posible medida de contingencia, dependerá del riesgo específico que surja. Para esto, es necesaria la comunicación y la **adaptabilidad del equipo desarrollador ante posibles imprevistos.**

Seguimiento y control

A lo largo de la ejecución del proyecto, se realizarán una serie de actuaciones para asegurar que el proyecto avanza orgánicamente y que las incidencias se logran disolver de manera eficaz.

Primeramente, se ha de llevar a cabo una **revisión periódica de riesgos** analizados en reuniones semanales y actualizando el plan según su propia evolución. También se incluirán los **registros de incidencias** con ayuda de GitHub Issues, asignando los desarrolladores responsables y su posible fecha de resolución. Se llevará acabo un **control de calidad** mediante pruebas continuas y validación de funcionalidades críticas, que forman parte de la aplicación base. En el repositorio de Git, se añadirán una **documentación con los cambios realizados** para mayor trazabilidad.

Se establecerá la siguiente jerarquía de control y seguimiento para el proyecto:

1. **Líder de proyecto:** encargado de la supervisión general y las actualizaciones de plan de desarrollo.
2. **Equipo desarrollador:** identificación y resolución de incidencias técnicas.
3. **Responsable de control de BBDD:** control de integridad y seguridad de datos.
4. **Equipo de QA:** encargado de la validación de pruebas y de la fase de aseguramiento de la calidad.

Aseguramiento de la calidad

El objetivo del **Plan de Aseguramiento de la Calidad** tiene como objetivo garantizar que los procesos y productos generados por el equipo desarrollador cumplan con unos estándares de calidad previamente definidos. Este plan abarca todo el procedimiento de desarrollo de la app, desde la fase de análisis hasta la entrega final del producto al equipo de producción. También se incluirá la validación de los usuarios y la documentación asociada a la aplicación.

Productos sujetos a control de calidad

En este apartado quedarán detallados los productos que serán objeto de control de calidad junto con sus posibles indicadores y procedimientos de evaluación:

1. **Especificación de requisitos:** para este apartado, los indicadores de calidad se medirán en el **porcentaje de requisitos aprobados por el usuario de la aplicación sobre el total propuesto**, así como el **número de iteraciones necesarias** hasta llegar a una aprobación final. Como procedimiento de evaluación**,** se deberán llevar a cabo **revisiones formales** por parte del equipo, una **validación explícita hecha por el usuario** (por ejemplo, mediante una *checklist* después de testear la aplicación y su funcionamiento), y un **registro de observaciones y cambios solicitados** por los usuarios de prueba en cada iteración.
2. **Modelo de datos y diagrama UML:** los indicadores de calidad aquí serán la **coherencia entre el modelo entidad-relación y los requisitos funcionales** (para asegurar la calidad de este apartado, se deberá corregir el modelo para ajustarlo a los requisitos funcionales, o bien, hacerlo a la inversa). Otro indicador será la **ausencia de ambigüedades o relaciones incorrectas,** que se corregirán conforme el proyecto vaya tomando forma. Los procedimientos de evaluación serán **revisiones entre los miembros del**

equipo y la **validación mediante casos de prueba de integridad** hechos por todo el equipo de desarrollo.

3. **Código fuente de la aplicación:** para medir la calidad del código, se **contará el porcentaje de funcionalidades implementadas respecto al total planificado;** cuanto mayor sea este porcentaje, más seguro estará el requisito de calidad de este apartado. También se contarán el **numero de errores detectados** en pruebas unitarias o de integración. El procedimiento de evaluación se llevará a cabo a través de una **revisión del código** (*code review*) en GitHub, y una serie de **pruebas automatizadas y manuales** para abarcar todos los casos de uso y localizar posibles nuevas vulnerabilidades.
4. **Manual de usuario:** los indicadores de un buen manual de usuario serán, por evidentes motivos, la **claridad y comprensión del contenido por parte de un usuario no técnico** y que no forme parte del equipo de testeo. Otro indicador será la **cobertura final de todas las funcionalidades implementadas** incluyéndolas en el manual. Los procedimientos de evaluación del manual de usuario serán las **revisiones de usuarios externos al equipo** y su visto bueno sobre el entendimiento del manual. A través de estas revisiones, se llevarán a cabo las **correcciones basadas en *feedback*** recibido.
5. **Producto software final:** las acciones que medirán la calidad serán los **porcentajes de casos de uso que se ejecuten correctamente,** probando la aplicación en las plataformas previstas y localizando los errores que surjan en cada uno. También se tomará en cuenta el **grado de satisfacción del usuario** para con la aplicación en las pruebas de validación. Para evaluar el producto final, la mejor forma serán las **pruebas funcionales con usuarios reales** y un **registro de incidencias** y mejoras propuestas por estos usuarios externos al equipo.

Procedimiento de participación del usuario en la evaluación

Para garantizar una correcta satisfacción del usuario con el uso del producto final, se establecerá un **procedimiento de evaluación participativa**, que incluirá tres fases principales.

La primera fase será la **sesión de validación funcional** con usuarios representativos, en la que se probarán los principales flujos de trabajo de la aplicación, que incluirán **lectura, publicación de libros, creación de reseñas y participación en foros.**

La segunda fase se llevará a cabo a través de **cuestionarios de evaluación** diseñados para recoger la opinión de los usuarios seleccionados sobre tres apartados fundamentales de la aplicación clasificados en la **facilidad de uso e**

interfaz, el **cumplimiento de requisitos funcionales** y la **utilidad del manual de usuario.**

La tercera y última fase incluirán una serie de **documentos de validación,** donde los usuarios podrán aprobar o rechazar funcionalidades específicas, así como sugerir nuevas ideas para la expansión o la mejora de la aplicación, dejando así constancia formal de su conformidad.

Los resultados de estas tres fases de evaluación se documentarán y, en caso de detectar desviaciones significativas, se aplicarán acciones correctivas antes de la entrega del producto final.