



PROPUESTA DE UN PLAN DE MEJORA DE CALIDAD DE UNA APLICACIÓN BASADO EN LA NORMA ISO/IEC 25010: CASO ARMADILLO

Joaquín Pabst Olmedo

joaquin.pabst@usm.cl

Mauricio Figueroa
Profesor Guía

Pedro Godoy
Profesor Correferente

Resumen: ARmadillo es un videojuego móvil que utiliza realidad aumentada, elaborado para ayudar a niños a desarrollar su comprensión lectora. El problema fue el nulo control de calidad en su desarrollo, lo que no permite tener una noción del estado del producto. Para abordar esto, se propone diseñar una evaluación siguiendo el modelo de calidad ISO/IEC 25010, realizar un plan de pruebas, ejecutarlo y analizar los resultados, para obtener un plan de futuras mejoras. Se planteó evaluar la mantenibilidad del código para facilitar el trabajo futuro por parte del equipo, la eficiencia de las escenas en realidad aumentada dado que no se contaba con experiencia en esta tecnología, y la fiabilidad del producto bajo internet inestable ya que era un requisito del cliente. Los resultados mostraron oportunidades en la mantenibilidad y eficiencia de desempeño. El plan de mejoras resultante propone realizar divisiones de responsabilidades a ciertas clases, así como también refactorizar métodos que presentaban alta complejidad, también precargar las escenas de realidad aumentada para disminuir los tiempos de espera. Este plan queda para una futura implementación, buscando mejorar la calidad de ARmadillo y mantener esta evaluación constante en el desarrollo del software.

Palabras Clave: ISO 25010, Calidad, Mantenibilidad, Eficiencia, Fiabilidad.

1 Introducción

La comprensión lectora es una habilidad que desarrolla el ser humano para entender lo que está leyendo, esta permite inferir información que se encuentra implícita en los textos. Con el fin de ayudar a los niños de primero y segundo básico a desarrollar esta habilidad se creó ARmadillo, el cual es un videojuego en Unity para dispositivos móviles que utiliza tecnología de realidad aumentada (AR) como principal diferenciador, el cual cuenta con un enfoque de actividades que se pueden desarrollar en clases como también en los hogares.

Este software se desarrolló a lo largo de cinco meses, dando como resultado una demo con la totalidad de características del producto. Esta incluye principalmente actividades AR, historietas sobre la flora y fauna chilena y recompensas por el trabajo realizado. Los datos son respaldados en una base de datos Firestore. Además, la aplicación cuenta con un tiempo límite de uso para limitar el uso del dispositivo móvil a los alumnos.

El equipo desarrollador al momento de iniciar el proyecto contaba en su mayoría con experiencia en el desarrollo de videojuegos en Unity. Por el contrario, no tenía experiencia en el uso de herramientas de AR ni en la base de datos en nube Firestore, por lo que se fue experimentando con estas tecnologías a medida que se desarrollaba el proyecto.

El problema radica en que no hubo un control de calidad exhaustivo en el desarrollo, ya que no se contó con un modelo de calidad en el que se evaluara el software de forma periódica. Esto preocupa al equipo desarrollador de ARmadillo, por las futuras implicancias que ven en las áreas de mantenibilidad del código para el trabajo a



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: PROPUESTA DE UN PLAN DE MEJORA DE CALIDAD DE UNA APLICACIÓN BASADO EN LA NORMA ISO/IEC 25010: CASO ARMADILLO

Nombre del candidato(a): Joaquín Ignacio Pabst Olmedo

Carrera / Grado: Ingeniería Civil Informática

Campus: Casa Central Valparaíso ; **Departamento:** Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Mauricio Carlos Figueroa Colarte, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 28/07/2025 ; **Firma:** 

Estudiante o Candidato(a):

Fecha: 28/07/2025 ; **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.



futuro, además del desempeño que pueda tener la tecnología de RA en los dispositivos móviles y la capacidad de almacenamiento de datos en condiciones de internet inestable.

La calidad en los productos de software es de suma relevancia, debido a que cada vez más las organizaciones, empresas e individuos dependen de estos para llevar a cabo sus vidas, por lo que en caso de fallar, podrían enfrentarse a grandes consecuencias, por lo que es fundamental que el software sea de alta calidad [1]. Para controlar la calidad se utilizan los modelos de calidad del software, los cuales son una herramienta útil para cumplir con los objetivos de calidad, confiabilidad y pruebas del software en distintos proyectos. Estos permiten identificar módulos del programa con alto riesgo [2].

En este trabajo se realizará una evaluación de calidad al software ARmadillo siguiendo el modelo de calidad ISO/IEC 25010 y la metodología de evaluación ISO/IEC 25040. Se determinará el alcance de la evaluación (características del modelo a evaluar), diseñar y ejecutar esta misma, analizar los datos arrojados y finalizar con una propuesta de plan de futuras mejoras.

1.1 Objetivos

El objetivo general es analizar la calidad del software ARmadillo (videojuego móvil) en base a la norma ISO/IEC 25010, con el fin de elaborar un plan de futuras mejoras. Este se compone de los siguientes objetivos específicos:

- Diseñar un plan para medir la calidad del software en base a los criterios relevantes de ISO/IEC 25010.
- Realizar el plan de medición de calidad.
- Analizar resultados para detectar oportunidades de mejora.
- Obtener un plan de mejoras de calidad para ARmadillo.

1.2 Estructura del informe

El capítulo N°2 se dan las bases teóricas relacionadas a la evaluación de calidad del software, así como trabajos previos que han recopilado métricas o realizado evaluaciones de este tipo. En el capítulo N°3 se diseña la evaluación de calidad para ARmadillo y en el capítulo N°4 se muestran los resultados. En el capítulo N°5 se realiza un análisis de los resultados para luego en el capítulo N°6 presentar el plan de futuras mejoras. Finalmente, en el capítulo N°7 contiene las conclusiones del trabajo realizado.

2 Marco Conceptual

2.1 Calidad del Software

En la industria del desarrollo de software se define **calidad** como el grado en que un sistema, componente o proceso cumple con los requisitos especificados, así como también con las necesidades o expectativas del cliente o usuario [3].

En las normas ISO (*International Organization for Standardization*) la **calidad del software** se define como un conjunto de características que determinan la satisfacción de necesidades tanto implícitas como explícitas del software [4]. En Ingeniería de Software: un enfoque práctico [1], se define la **calidad del software** como un proceso eficaz de software que se aplica de manera que crea un producto útil para aquellos que lo producen y aquellos que lo utilizan.

El **aseguramiento de la calidad** del software proviene de un patrón planificado y sistemático de acciones necesarias para proporcionar la confianza de que un producto cumple con lo requerido, esto a través de tareas diseñadas para evaluar el proceso a través del que se desarrolla software [3]. Las mediciones de software buscan la comprensión de este, logrando cuantificar el estado del producto [5].

La **gestión de la calidad** es la aplicación de procesos y verificación de que estos continúen en el desarrollo, además de que los resultados estén alineados con los requerimientos. El problema es que la valoración del software es subjetiva, mientras que el uso de estándares establece una base para decidir si se alcanzó el nivel de calidad requerido [6].

Una **métrica de calidad** es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado de calidad [3].

2.2 Modelos de calidad

Un modelo de calidad, o modelo de evaluación de calidad, se define como un modelo analítico que proporciona una evaluación cuantitativa de características de calidad seleccionadas. Estos modelos ayudan a obtener una evaluación objetiva del producto [7].

Existen dos tipos de enfoques referentes a modelos de calidad del software [8]:

- **Modelos generales/básicos:** desarrollados para ser utilizados en toda clase de software, son atributos de calidad que se podrían aplicar a cualquier aplicación.
- **Modelos específicos:** desarrollados para ser utilizados en una clase en particular de software, sus atributos de calidad cubren todos los aspectos relevantes de la clase en cuestión.

Algunos **modelos básicos** son: McCall, Boehm, Dromey, FURPS, ISO 9126, ISO 25010. Estos seis han sido los modelos más destacados históricamente [9]. En [10] se realiza un análisis comparativo de los diferentes modelos de calidad de software. Se concluye con la recomendación del uso del modelo ISO/IEC 25010 para la ingeniería de software.

El **Modelo de McCall** fue creado en el año 1977, en este se detalla una jerarquía de factores, criterios y métricas. Donde un factor de calidad es una característica de comportamiento del sistema, un criterio de calidad es un atributo del factor relacionado con la producción y una métrica de calidad es una medida que captura algún aspecto del sistema [11]. Los factores de calidad que presenta este modelo son: mantenimiento, facilidad de pruebas, flexibilidad, interoperabilidad, portabilidad, reusabilidad, corrección, fiabilidad, eficiencia, integridad y facilidad de uso.

El **Modelo de Boehm** se crea en el año 1978, el cual tiene como objetivo definir cualitativamente la calidad del software a través de un conjunto de atributos y métricas, este siendo jerárquico al igual que el modelo de McCall, la diferencia radica en que Boehm se amplía en el concepto de mantenibilidad, mientras que McCall se centra en *As-is Utility*. Se organiza en características de nivel alto, intermedio y bajo. Las cualidades de alto nivel se refieren a la utilidad actual que presenta el software si se usara (*As-is Utility*), la capacidad de mantención y la portabilidad en caso de cambiar de entorno [12]. Las características intermedias corresponden a los siguientes factores de calidad: portabilidad, confiabilidad, eficiencia, usabilidad, testabilidad, comprensión y flexibilidad. Mientras que las de bajo nivel o primitivas son la base para las métricas de calidad.

El **Modelo de Dromey** nace en 1995 de la idea de que los modelos jerárquicos de software como el de McCall y Boehm suelen ser demasiado ambiguos en las descripciones de los niveles más bajos, lo que ofrece poca ayuda a los desarrolladores. En cambio, afirma que no se pueden incorporar atributos de calidad directamente, sino que en cambio se inicia de componentes del software de los cuales emergen los atributos de



calidad [11]. Se mencionan los siguientes atributos de calidad: funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad, portabilidad y reusabilidad [13].

El **Modelo FURPS+** clasifica las características en dos tipos, funcionales y no funcionales. Las características no funcionales son: usabilidad, fiabilidad, rendimiento y soportabilidad. El símbolo “+” agrega nuevos requisitos, los cuales son de diseño, de implementación, interfaz y físicos [8].

El **Modelo ISO 9126** fue derivado del modelo de McCall, esta norma internacional, publicada en el año 1991, propone un modelo de calidad que consta de dos partes, la calidad interna/externa y la calidad de uso del software. Los atributos internos son aquellos que pueden evaluarse sin ejecutar el software, mientras que los externos son al ejecutar el software. Mientras que los de calidad de uso se refieren a aquellos que experimenta el usuario. Los atributos de calidad internos/externos son: eficiencia, funcionalidad, mantenibilidad, portabilidad, confiabilidad y usabilidad [8]. Las cualidades de uso son: eficiencia, productividad, seguridad y satisfacción. Para evaluar correctamente estos atributos se requiere identificar los requisitos de calidad del usuario [14].

El **Modelo ISO 25010** norma internacional perteneciente a la familia de normas ISO/IEC 25000 o SQuaRE (*System and Software Quality Requirements and Evaluation*), fue presentada en el año 2011, siendo una evolución del modelo de calidad presentado por la norma ISO 9126. En esta se describe el modelo de calidad del producto y de uso del software, en el que se describen características y subcaracterísticas a evaluar [4]. Han existido dos versiones del modelo de calidad ISO 25010, el primero en el año 2011 presentaba las siguientes características de calidad en su modelo de calidad de producto: adecuación funcional, eficiencia de desempeño, compatibilidad, usabilidad, fiabilidad, seguridad, mantenibilidad y portabilidad [15]. Para el año 2023 se lanzó una nueva versión de la norma, en la cual los cambios más relevantes son el añadido de protección como una nueva característica, usabilidad se renombró como capacidad de interacción y la portabilidad fue renombrada como flexibilidad [16].

2.3 ISO/IEC 25000

La norma internacional ISO/IEC 25000 es una familia de normas que tiene como objetivo establecer un marco de trabajo común para la evaluación de la calidad de un producto de software [17]. Se origina a partir de las normas ISO/IEC 9126 e ISO/IEC 14598 [15]. Cuenta con las siguientes divisiones:

- ISO/IEC 2500n es la división de gestión de calidad, en la que se define la arquitectura de la familia de normas, terminología y definiciones que serán utilizadas por el resto de las normas.
- ISO/IEC 2501n es la división de modelos de calidad.
- ISO/IEC 2502n es la división de medición de calidad interna, externa y de uso.
- ISO/IEC 2503n es la división de requisitos de calidad basados en modelos de calidad y medidas de calidad.
- ISO/IEC 2504n es la división de evaluación de calidad. Proporciona requisitos, recomendaciones y directrices para la evaluación del producto de software.
- ISO/IEC 25050 a ISO/IEC 25099 constituye la extensión de la familia de normas, incluye principalmente requisitos, instrucciones y formatos comunes en la industria.

2.4 ISO/IEC 25010

El modelo de calidad perteneciente a la familia SQuaRE se compone de nueve características, cada una con sus propias subcaracterísticas [4]:

- **Adecuación Funcional:** La capacidad de un producto para proporcionar funciones que satisfagan las necesidades explícitas e implícitas del usuario.

- **Eficiencia de desempeño:** capacidad de un producto para ejecutar sus funciones dentro de los parámetros de tiempo y rendimiento, siendo eficiente en el uso de recursos.
- **Compatibilidad:** capacidad de un producto de intercambiar información con otro producto y ejecutar sus funciones mientras comparte el entorno y recursos.
- **Capacidad de interacción:** capacidad de un producto para intercambiar información entre el usuario y la interfaz del producto para completar una tarea.
- **Fiabilidad:** capacidad de un producto de ejecutar funciones específicas bajo condiciones específicas en un período especificado de tiempo sin fallar.
- **Seguridad:** capacidad de un producto para proteger la información y los datos de manera que las personas u otros productos tengan el grado de acceso a los datos apropiado a sus tipos y niveles de autorización, y para defenderse contra patrones de ataque por actores maliciosos.
- **Mantenibilidad:** capacidad de un producto de ser modificado con efectividad y eficiencia.
- **Flexibilidad:** capacidad de un producto para adaptarse a cambios de requisitos, contexto de uso o entorno de ejecución.
- **Protección:** capacidad de un producto de evitar un estado en el cual la vida humana, la propiedad o el medio ambiente se vean en peligro.

2.5 ISO/IEC 25040

La norma propone un modelo de referencia general para la evaluación, que considera las entradas al proceso de evaluación, las restricciones y los recursos necesarios para obtener las correspondientes salidas [17]. Está dirigida a desarrolladores, integradores, aseguradores de calidad y evaluadores de calidad independientes, sean internos o externos a la organización que desarrolla o mantiene el producto, servicio o datos [18]. El modelo general que propone la ISO 25040 consta de cinco pasos generales [17]:

- **Definir la evaluación de calidad:** se define el alcance establecido de la evaluación, los criterios de evaluación, las entidades destinatarias y otros factores.
 - **Establecer el objetivo:** definir el objetivo de la evaluación.
 - **Identificar las entidades objetivo:** identificar todas las entidades objetivo de la evaluación.
 - **Definir los criterios de evaluación de la calidad:** identificar los criterios de evaluación de calidad, los cuales son requisitos específicos utilizados para evaluar la calidad. Estos pueden incluir factores como la fiabilidad, eficiencia del rendimiento, la compatibilidad, etc.
 - **Definir los requisitos para el rigor de la evaluación:** se definirá el rigor de las evaluaciones para garantizar la exactitud, fiabilidad y validez de los resultados.
- **Diseñar la evaluación:** se diseña la evaluación, incluyendo las entidades objetivo y componentes que deben ser evaluados. Los métodos que se aplicarán para la medición, calificación y análisis de la calidad.
 - **Identificar los componentes a evaluar:** el evaluador deberá identificar las entidades y componentes que deben someterse a una evaluación.
 - **Seleccionar e implementar módulos de evaluación de calidad:** se seleccionan e implementan los módulos de evaluación de calidad, teniendo en cuenta las características y subcaracterísticas de calidad, las necesidades de información y las medidas de calidad aplicables.
 - **Determinar los métodos de análisis de la calidad:** se determinan los métodos por lo que se llevará a cabo la evaluación de calidad.
 - **Definir los resultados:** se definen los resultados que tendrá una evaluación de calidad (Ej.: puntuación, aprobado o no aprobado, puntos fuertes y débiles, etc.).



- **Planificar la evaluación:** se definen, acuerdan y comunican a las partes interesadas el alcance y enfoque que se adoptará en la evaluación de calidad.
 - **Identificar las actividades de la evaluación:** se identifican las actividades para llevar a cabo las evaluaciones del paso anterior.
 - **Determinar el personal y la programación:** las pruebas de evaluación deberán programarse y dotarse de personal, teniendo en cuenta los recursos disponibles.
 - **Registrar el plan de evaluación de calidad:** se registra el plan de evaluación de calidad.
 - **Conseguir el consenso sobre el plan de evaluación:** validar el plan con terceros.
- **Ejecutar la evaluación:** se ejecuta el plan de evaluación para obtener los resultados esperados.
 - **Aplicar módulo de calificación de la calidad:** se implementan los módulos elegidos y se aplicará la evaluación de las entidades objetivo para producir los resultados de calificación de la calidad.
 - **Analizar los resultados de la calificación de calidad:** se analizan los resultados para obtener la evaluación de las características y subcaracterísticas.
- **Concluir la evaluación:** se generan los resultados de la evaluación de calidad, satisfaciendo las necesidades de las partes interesadas, de forma que garantiza su validez y utilidad.
 - **Revisar los resultados de la evaluación:** el evaluador y el solicitante realizan una revisión de los resultados de la evaluación para validar su justificación.
 - **Elaborar el informe de evaluación:** una vez revisados los resultados se realiza el informe de evaluación, el que incluye requisitos, resultados y el análisis realizado.
 - **Revisar el proceso de evaluación de calidad:** el evaluador realizará una revisión exhaustiva del proceso de evaluación llevado a cabo, los indicadores y las medidas utilizadas. La información obtenida se deberá utilizar en la mejora del proceso y eficacia de las medidas realizadas.
 - **Gestionar los datos de la evaluación:** finalizada la evaluación, debe realizar el adecuado tratamiento de los datos y objetivos de evaluación, ya sea almacenarlos o eliminarlos.

2.6 Mantenibilidad

La mantenibilidad es la facilidad con la que un sistema o componente de software puede ser modificado para corregir fallos, mejorar el rendimiento u otros atributos [3]. En [4] se definen cinco subcaracterísticas:

- **Modularidad:** Capacidad para limitar los efectos de un cambio en otros componentes.
- **Reusabilidad:** Capacidad para ser utilizado como activo en otro sistema o en la construcción de otros activos.
- **Analizabilidad:** Capacidad de ser evaluado de forma eficiente con relación a los cambios previstos y el impacto en sus partes, para identificar deficiencias, fallos o partes que deben modificarse.
- **Modificabilidad:** Capacidad de un producto para ser modificado de forma eficiente sin introducir defectos.
- **Testabilidad:** Capacidad para permitir que pruebas objetivas sean diseñadas y realizadas para determinar si se cumple un requisito.

2.6.1 Métricas de Mantenibilidad

En [19] se realiza un análisis de herramientas de código con las cuales se pueden medir métricas básicas de mantenibilidad de software. En su estudio trabajan bajo los acuerdos de la norma ISO/IEC 9126. Realizaron

una recopilación de métricas de diferentes fuentes, clasificándolas en las subcaracterísticas asociadas a la característica de calidad, las cuales son analizabilidad, cambiabilidad, estabilidad y testabilidad.

En [20] se realiza un estudio de métricas utilizadas en el sector del software automotriz. Su problema era que, existiendo métricas de calidad, en la práctica estas sufren dificultades al ser asignadas a un atributo de calidad. Por lo mismo se realizó un estudio el cual buscaba métricas y sus valores límite, la relación de la métrica con los estándares, los atributos de calidad que abordan y cómo estas son soportadas por herramientas. El análisis contabilizó un total de 38 artículos, de los cuales se extrajeron 112 métricas. Se destacan, para mantenibilidad la complejidad ciclomática, líneas de código, falta de cohesión en métodos (LCOM) y acoplamiento entre métodos (CBO).

En [21] se desarrolló un modelo para evaluar la mantenibilidad de microservicios, con un enfoque en la familia de normas SQuaRE. Se seleccionaron métricas de código para evaluar la testabilidad y modificabilidad, para luego utilizar un algoritmo de *clustering* que clasifica los microservicios en tres grupos diferentes (bajo, medio y alto) dependiendo de su grado de mantenibilidad.

2.6.1.1 Complejidad Ciclomática

En el año 1976 Thomas McCabe introduce una métrica conocida como Complejidad Ciclomática (*Cyclomatic Complexity*), la cual mide la cantidad de caminos independientes en un fragmento de código. Medir la CC proporciona grandes beneficios, ya que ayuda a prevenir cuándo un componente del software tendrá una alta tasa de defectos o será difícil de probar y mantener [22]. Para el cálculo de CC se tiene la siguiente fórmula [23]:

$$CC = E - N + 2P \quad (1)$$

Donde E es el número de aristas, N el número de nodos y P el número de componentes conexos. La fórmula se simplifica para programas estructurados de la siguiente forma:

$$CC = \text{Número de predicados} + 1 \quad (2)$$

Algunas variaciones relacionadas con la complejidad en las clases son WMC y AC. La WMC representa la suma de la complejidad ciclomática de todos los métodos de una clase. En cambio, AC representa el promedio de complejidad ciclomática en todos los métodos. Para el lenguaje de programación C# se estima un umbral de 36 para WMC y de 3 para AC [24].

2.6.1.2 Líneas de código

Métrica para evaluar el tamaño del software. En la programación orientada a objetos se usa la métrica AMS para evaluar el promedio de líneas de código por método. Se estima un umbral de 20 para el lenguaje de programación C# de [24].

2.6.1.3 Falta de cohesión en los métodos

Esta métrica mide la falta de cohesión en los métodos de una clase. Se basa en el principio de que, si una variable aparece en muchos métodos de una clase provoca que su valor de cohesión sea mayor, en cambio si no comparte método su valor es menor [25]. El valor de LCOM se obtiene de la siguiente fórmula [26]:

$$LCOM = |P| - |Q| \quad (3)$$



Donde P es el conjunto de pares de métodos los cuales no comparten variables entre sí y Q es el conjunto de pares de métodos los cuales si comparten variables entre sí. Un nivel alto de LCOM indica que los métodos están poco relacionados entre sí (falta de cohesión), mientras que un bajo nivel indica que están altamente cohesionados. En caso de que el valor sea negativo este se ajusta a cero. Se estima un umbral para LCOM en programación C# de 74% [24]. El valor umbral en este caso corresponde al porcentaje de conjuntos P en el total existente.

2.6.1.4 Numero de métodos

Esta métrica NM indica el número total de métodos en una clase, puede indicar que una clase realiza demasiado trabajo [25]. Para el lenguaje de programación C# el umbral puede considerarse de 60 [24].

2.6.1.5 Acoplamiento entre objetos

La métrica CBO mide el nivel de acoplamiento de una clase con otras. Una clase esta acoplada a otra cuando utiliza sus métodos dentro de sus propios métodos. El acoplamiento excesivo es perjudicial para el diseño modular, por lo que impide la reutilización. Mientras más independiente es, más fácil es reutilizarla [26]. Se estima un umbral para CBO en programación C# de 23 clases referenciadas [24].

2.7 Eficiencia de desempeño

En [4] se definen tres características de eficiencia de desempeño:

- **Comportamiento temporal:** Capacidad para ejecutar funciones bajo condiciones específicas cumpliendo el tiempo de respuesta y la tasa de rendimiento.
- **Utilización de recursos:** Capacidad de no utilizar más que la cantidad especificada de recursos para ejecutarse.
- **Capacidad:** Capacidad para cumplir con los requisitos de límites máximos de un parámetro del producto.

En [27] se realiza un investigación sobre la utilización del modelo de calidad ISO/IEC 25010 en dos aplicaciones desarrolladas en el motor Unity con uso de herramientas de realidad virtual (Vuforia y Wikitude). El estudio se enfocó en la característica de eficiencia de desempeño, en específico en las subcaracterísticas de comportamiento en el tiempo y utilización de recursos. Esto con el objetivo final de proporcionar información sobre las diferencias de estas dos herramientas antes de ser seleccionadas para la elaboración de un producto.

Las métricas utilizadas estan basadas en la familiad de normas SQuaRE. En comportamiento temporal se utilizan las metricas de Tiempo de respuesta, Tiempo de espera y Rendimiento. En el caso de Utilizacion de recursos se utilizo el Uso de memoria. Por último la Capacidad no fue medida debido a que el software de prueba no seria ejecutado en linea.

2.7.1 Tiempo de respuesta

El objetivo de esta métrica es estimar el tiempo necesario para completar una tarea, lo que se traduce a tomar el tiempo desde que se envía la petición, hasta obtener la respuesta. La fórmula es la siguiente:

$$TR = B - A \quad (4)$$

En donde A es el tiempo de envío de la petición y B es el tiempo en que se recibe la primera respuesta. En el análisis de las aplicaciones Unity, esta métrica corresponde al tiempo que transcurre desde que se activa la cámara, hasta la visualización de los objetos 3D.

2.7.2 Tiempo de espera

El objetivo de esta métrica es obtener el tiempo desde que se envía una instrucción, hasta que se realice. Lo que se traduce a tomar el tiempo desde que se inicia un trabajo hasta que este es completado. La fórmula es la siguiente:

$$TE = B - A \quad (5)$$

En donde A es el tiempo cuando se inicia el trabajo y B es el tiempo cuando el trabajo se ha completado. En el análisis de las aplicaciones Unity, esta representa el tiempo desde que se presiona un botón (el cual deberá hacer que se instancie un objeto en el entorno) hasta que el objeto 3D es visualizado.

2.7.3 Uso de memoria

El objetivo de esta métrica es verificar la cantidad de memoria usada para realizar una tarea. Lo que se traduce a medir la cantidad total de memoria y la cantidad de espacio de memoria que es usado para realizar una tarea. La fórmula es la siguiente:

$$UM = B - A \quad (6)$$

En donde A es la cantidad de espacio de memoria que es usado para realizar una tarea y B es la cantidad total de espacio de memoria. En el análisis de las aplicaciones Unity se mide el uso de memoria antes de iniciar la aplicación y el uso de memoria una vez visualizado el objeto 3D.

2.8 Fiabilidad

En [4] se definen cuatro características de fiabilidad:

- **Ausencia de fallos:** Capacidad de ejecutar funciones sin errores durante su operación.
- **Disponibilidad:** Capacidad de un producto de estar operativo cuando se requiere.
- **Tolerancia a fallos:** Capacidad de funcionar como se espera, aunque ocurran fallos de software o hardware.
- **Capacidad de recuperación:** Capacidad de recuperar datos afectados y restablecer el estado deseado en caso de interrupciones o fallos.

3 Diseño plan de evaluación

3.1 Metodología

Se utilizará la norma ISO/IEC 25040 como base y guía a la hora de planificar, ejecutar y evaluar la calidad del software ARmadillo.

1. Establecer los objetivos

Se deberán establecer los objetivos a la hora de evaluar ARmadillo. Identificar las entidades que se evaluarán, estas pueden ser scripts C#, componentes del juego, envío de datos, etc. Definir el criterio con el que se evaluarán, en relación con el modelo propuesto en la norma ISO/IEC 25010.

2. Diseñar la evaluación

Se deberá identificar los componentes del software a evaluar, lo que se evaluará y cómo se clasificarán los resultados.



3. Planificar la evaluación

Se deberá identificar las actividades de la evaluación y se registrarán en un plan de evaluación, el cual deberá contar con el objetivo de cada prueba.

4. Ejecutar la evaluación

Se aplicarán los módulos previamente definidos, ejecutando las herramientas sobre el código y las pruebas sobre los casos, con el fin de obtener resultados que se analizarán respecto a los rangos o valores esperados.

5. Concluir la evaluación

Analizar los resultados obtenidos y revisar el proceso de evaluación para proponer mejoras a futuro. Finalmente, se debe almacenar los resultados para futuras comparaciones.

3.2 Objetivos de Evaluación ARmadillo

A la fecha de publicación de este trabajo, el software ARmadillo corresponde a una versión demo que incluye funciones y características que se propusieron en el inicio del desarrollo. Estas funcionalidades fueron probadas en el contexto de un aula de clases y supervisada por un docente durante el desarrollo del contenido. Por lo que el objetivo de esta evaluación no se centrará en las características relacionadas con la adecuación funcional y la capacidad de interacción.

Debido a que el software no gestiona datos privados que requieran estándares de seguridad (como datos bancarios o información personal relevante) no se realizará una evaluación de seguridad, debido a que no se han implementado medidas por el contexto actual del sistema.

El software ARmadillo no coexiste con ningún otro software a la hora de realizar sus funciones, solo interactúa con una base de datos en la nube, pero debido a que esta no comparte recursos de la aplicación se descarta evaluar las características relacionadas con la compatibilidad.

La evaluación de la calidad que se busca en este trabajo tiene por objetivo evaluar el software en su estado actual, en el sistema operativo para el cual se construyó, por lo que la característica de flexibilidad no será considerada en la evaluación. Debido a que es un videojuego educativo que no presenta riesgo para el entorno físico donde se ejecuta, la característica de protección será descartada igualmente.

ARmadillo fue desarrollado sin el control de calidad de ningún modelo, tampoco se emplearon patrones de diseño de forma consciente o rigurosa. El equipo desarrollador tiene la preocupación de que la forma en que el código está estructurado sea poco entendible, analizable y difícil de modificar, tanto para un desarrollador externo que busque escalar el código, como para los miembros del equipo desarrollador actual que busquen involucrarse en código en el cual no tuvieron participación previa. Por lo que la característica de mantenibilidad es de importancia evaluarla en estos momentos, debido a que el desarrollo aún se encuentra en una fase que lo permite y sería de importancia marcar un precedente en esta área, para futuras mejoras y evaluaciones.

En el desarrollo se trabajó con dos tecnologías desconocidas previamente por el equipo desarrollador, estas son la AR y Firestore. La poca experiencia que cuenta el equipo con estas dos tecnologías motiva cuestionamientos sobre su rendimiento que estas tienen. En el caso de RA, se debe evaluar su eficiencia en el desempeño, ya que el uso de dispositivos móviles debe ser fluido para no afectar la experiencia del usuario. Por otro lado, se debe evaluar la fiabilidad de la conexión con Firestore, esto debido a que la aplicación está orientada al trabajo en las aulas de clase, en donde la conexión a internet puede ser inestable. Se busca evaluar la capacidad de ARmadillo para enfrenar interrupciones en dicha conexión.

3.2.1 Objetivos de Mantenibilidad

En la característica de mantenibilidad las entidades a evaluar serán los *scripts* desarrollados por el equipo hasta la fecha, por lo que no se incluirá el código externo relacionado con la tecnología AR. Las subcaracterísticas involucradas serán: analizabilidad, reusabilidad, modularidad, modificabilidad y testabilidad.

3.2.2 Objetivos de Eficiencia de Desempeño

En la característica de eficiencia de desempeño las entidades a evaluar serán las escenas AR. Se buscan medir los recursos que consume la aplicación al levantar la escena con los objetos 3D, además del tiempo que requiere este proceso. Las subcaracterísticas involucradas serán: Comportamiento temporal y Utilización de recursos.

3.2.3 Objetivos de Fiabilidad

En la característica de fiabilidad la entidad que será evaluada es la conexión con Firestore y el manejo de los datos en caso de pérdida de conexión. Las subcaracterísticas involucradas serán: Tolerancia a Fallos y Capacidad de Recuperación

4 Diseño de evaluación

4.1.1 Pruebas de Mantenibilidad

Para la evaluación de métricas de mantenibilidad se utilizará Roslyn el cual es la implementación de código abierto de los compiladores de C# y Visual Basic, con una superficie de API para construir herramientas de análisis de código [28]. Para la validación de los datos se deberá verificar los resultados arrojados por el programa evaluador en al menos 10 clases. Una vez verificada la veracidad de los resultados obtenidos en esos casos, se llevará a cabo la evaluación completa del código. Se seleccionaron un total de seis métricas de evaluación, estas son: AC, WMC, NM, AMS, LCOM y CBO. Los Pasos por seguir son:

1. Seleccionar *scripts* relevantes para la evaluación, en este caso serán aquellos que haya realizado el equipo, descartando los propios de Unity y la tecnología AR.
2. Realizar la programación utilizando Roslyn para la obtención de las métricas.
3. Validar los resultados con una muestra de al menos 10 scripts (cuyos valores de las métricas se conozcan) comparando el resultado del programa con la realidad. En caso de discordancia se hacen los ajustes pertinentes al programa de cálculo de métricas, si los resultados coinciden se procede con la evaluación de la totalidad de los scripts.
4. Filtrar y sintetizar los resultados de cada métrica. Se espera obtener el valor de la métrica en cada clase, además si esta aprueba o no.

Las condiciones de evaluación y aprobación se ven en la Tabla 1.

Tabla 1. Métricas de Mantenibilidad
Fuente. Elaboración propia

Métrica	Condición Evaluación	Condición Aprobación	Subcaracterística ISO/IEC 25010
AC	Uno o más métodos	AC <= 3	Testabilidad/Modificabilidad
WMC	Uno o más métodos	WMC <= 36	Analizabilidad
NM	-	NM <= 60	Modularidad/Analizabilidad
AMS	Uno o más métodos	AMS <= 20	Testabilidad/Analizabilidad
LCOM	Dos o más métodos	LCOM <= 74%	Modularidad/Modificabilidad
CBO	Dos o más métodos	CBO <= 23	Reusabilidad



4.1.2 Pruebas de Eficiencia de Desempeño

Respecto al umbral de uso de memoria RAM en su ejecución se tomará en cuenta que Unity aconseja no usar más del 70% de la memoria física total del dispositivo [29]. Se estima que este uso de memoria es excesivo, por lo que para estas pruebas el umbral se encontrará en el 35% de la memoria RAM total. El dispositivo que se utilizará será un Galaxy Tab A9+, modelo SM-X210. Esta cuenta con 4GB de RAM totales, por lo que el umbral para el límite de uso de memoria será 1.4GB.

Para el tiempo de carga de los objetos 3D y el tiempo de espera para el levantamiento de la escena se utilizaron umbrales arbitrarios, definidos en base al comportamiento esperado en el funcionamiento del software. Para la validación de los resultados por cada prueba se deberán verificar 10 escenas distintas, esto para contrastar resultados en diferentes configuraciones de los entornos AR.

Se deberán seleccionar las escenas a evaluar, para luego medir los tiempos de instanciación de los modelos 3D utilizados en la aplicación, así como también el tiempo de espera para levantar todas las escenas AR. Por último, la utilización de memoria de la aplicación. Los pasos por seguir para los casos de tiempo de carga y espera son los siguientes:

1. Seleccionar las escenas a evaluar.
2. Programar la medición de tiempo en las distintas actividades, una vez pulsado el botón para iniciar una actividad AR, una vez se instancia la cámara AR y una vez se instancian todos los objetos de la escena.
3. Verificar que los tiempos registrados sean válidos y correspondan a los eventos medidos.
4. Recopilar los datos y dar como resultado un promedio de tiempo de espera y de carga.

Los pasos por seguir para el uso de memoria de ARmadillo:

5. Seleccionar las escenas donde se tomarán las medidas del nivel de uso de memoria.
6. Reiniciar el dispositivo móvil y tomar medidas sin uso de aplicaciones.
7. Tomar medidas en escenas AR de ARmadillo.
8. Recopilar los datos y dar como resultado el uso de memoria de ARmadillo.

Las condiciones de aprobación se ven en la Tabla 2.

Tabla 2. Métrica Eficiencia de Desempeño

Fuente. Elaboración propia

Métrica	Condición Aprobación	Subcaracterística ISO/IEC 25010
TR	TC $\leq$ 0.1 [seg]	Comportamiento Temporal
TE	TE $\leq$ 1.5 [seg]	Comportamiento Temporal
UM	UM $\leq$ 1.4 [GB]	Utilización de Recursos

4.1.3 Pruebas de Fiabilidad

Para el área de fiabilidad se evaluarán tres casos relacionados con internet intermitente en el uso de la aplicación. Si bien ARmadillo requiere conexión para ingresar a una cuenta, debiera soportar la pérdida de conexión de esta al estar usándola. Se debe evaluar si se logra finalizar una actividad, si los datos de esta una vez recuperada la conexión se envían a FireStore y si el tiempo límite de uso sigue en funcionamiento. Para la validación de los datos en las pruebas de fiabilidad se realizarán 15 mediciones en las pruebas relacionadas con la realización de actividades y recuperación de datos. En cambio, solo 5 en la de tiempo límite, debido a que las iteraciones son prácticamente similares.

Los pasos por seguir al hacer una actividad con internet intermitente son los siguientes:

1. Ingresar a ARmadillo con conexión a internet e iniciar una actividad. Mientras se desarrolla la actividad cortar la conexión a internet y continuar normalmente.
2. Realizar y recopilar las observaciones

Los pasos por seguir para verificar el límite de tiempo son:

3. Ingresar a ARmadillo con conexión a internet. Una vez en el menú cortar la conexión a internet. Esperar que el tiempo límite de la aplicación termine y verificar si se bloquea o no el uso de la aplicación.
4. Realizar y recopilar las observaciones

Los pasos por seguir para verificar la recuperación de los datos son:

5. Ingresar a ARmadillo con conexión a internet e iniciar una actividad. Mientras se desarrolla la actividad cortar la conexión a internet y continuar normalmente. Cerrar la aplicación y volver a ingresar con conexión a internet.
6. Verificar tanto en la aplicación como en Firestore si se almacenaron los datos.
7. Realizar y recopilar las observaciones

Las condiciones de aprobación se ven en la Tabla 3.

Tabla 3. Pruebas Fiabilidad

Fuente. Elaboración propia

Prueba	Condición Aprobación	Subcaracterística ISO/IEC 25010
Finalizar actividad con internet inestable.	Se aprueba si se logra finalizar la actividad sin internet	Tolerancia a Fallos
Envío de datos con internet inestable	Se aprueba si retomada la conexión, se envían los datos a Firestore	Capacidad de Recuperación
Tiempo límite con internet inestable	Se aprueba si se bloquea la aplicación, sin estar conectado a internet	Tolerancia a Fallos

4.1.4 Cronograma ejecución de pruebas

Se planea realizar la totalidad de pruebas de calidad en un lapso de diez días laborales. Distribuyendo las tareas como se ve en la Tabla 4.

Tabla 4. Cronograma ejecución de pruebas

Fuente. Elaboración propia

	Pasos	Dia 1	Dia 2	Dia 3	Dia 4	Dia 5	Dia 6	Dia 7	Dia 8	Dia 9	Dia 10
Mantenibilidad	1										
	2										
	3										
	4										
Eficiencia de Desempeño	1										
	2										
	3										
	4										
	5										
	6										
	7										



	8										
Fiabilidad	1										
	2										
	3										
	4										
	5										
	6										
	7										

5 Realización plan de evaluación

La evaluación se llevó a cabo de acuerdo con el cronograma propuesto.

5.1 Evaluación de Mantenibilidad

Los *scripts* que fueron seleccionados corresponden a aquellos desarrollados por el equipo desarrollador de ARmadillo. La cantidad de clases se presenta en la Tabla 5.

Tabla 5. Clases evaluadas
Fuente. Elaboración propia

Clases	Cantidad	Porcentaje
Total	114	100%
Con un o más métodos	86	75.4%
Con dos o más métodos	70	61.4%

De la totalidad de clases evaluadas el 75.4% aproximadamente corresponde a clases que cuentan con al menos un método, por lo que en estas se podrán evaluar las métricas: NM, WMC, AC y AMS. Por otro lado, el 61.4% de las clases posee dos o más métodos, en donde sumado a las métricas anteriores se podrá evaluar LCOM y CBO.

5.1.1 Métrica AC

Se evaluaron 86 clases respecto al nivel de complejidad ciclomática promedio de los métodos, los resultados generales se ven en la Tabla 6.

Tabla 6. Resultados AC
Fuente. Elaboración propia

Mantenible	Cantidad	% de la Muestra	% del Total
Si	69	80.2%	60.5%
No	17	19.8%	14.9%

Las clases que no son mantenibles bajo los criterios de AC se ven en Tabla 7.

Tabla 7. Clases que desaprobaron AC
Fuente. Elaboración propia

Clase	Valor AC
PalabraDiccionario	6
InicioDeSesion	4
VerificacionDeUsuario	3.75

PersonajeTouchMapa	3.2
PuntosTouchMapa	5
TiendaManager	4
MostrarResultados	5
DeteccionEscena	3.5
MostrarEstrellasOrdenamiento	4
OrdenamientoManager	3.2
ResultadosActividad	4
InfoBoton	4
FormularioNivelPrimero	3.5
EstrellasSelector	3.2
GetInventarios	4.5
EleccionDeAlternativa	3.5
EleccionMultipleSinOrden	6

5.1.2 Métrica WMC

Se evaluaron 86 clases respecto al nivel de complejidad ciclomática total de los métodos, el resultado se visualiza en la Tabla 8.

Tabla 8. Resultados WMC

Fuente. Elaboración propia

Mantenible	Cantidad	Porcentaje	Porcentaje Total
Si	85	98.8%	74.6%
No	1	1.2%	0.1%

Las clases que no son mantenibles bajo los criterios de WMC se ven en Tabla 9.

Tabla 9. Clases que desaprobaron WMC

Fuente. Elaboración propia

Clase	Valor WMC
GameManager	118

5.1.3 Métrica AMS

Se evaluaron 86 clases respecto a las líneas de código promedio por método, el resultado se visualiza en la Tabla 10.

Tabla 10. Resultados AMS

Fuente. Elaboración propia

Mantenible	Cantidad	Porcentaje	Porcentaje Total
Si	80	93%	70.1%
No	6	7%	5.2%

Las clases que no son mantenibles bajo los criterios de AMS se ven en Tabla 11.

Tabla 11. Clases que desaprobaron AMS

Fuente. Elaboración propia

Clase	Valor AMS
PalabraDiccionario	26
InicioDeSesion	21.67



VerificacionDeUsuario	27.75
MostrarResultados	24
GetInventarios	29
EleccionMultipleSinOrden	34

5.1.4 Métrica LCOM

Se evaluaron 70 clases respecto al nivel de cohesión entre los métodos de una clase, el resultado se visualiza en Tabla 12.

Tabla 12. Resultados LCOM

Fuente. Elaboración propia

Mantenible	Cantidad	Porcentaje	Porcentaje Total
Si	46	65.7%	40.4%
No	24	34.3%	21.1%

Las clases que no son mantenibles bajo los criterios de LCOM se ven en Tabla 13.

Tabla 13. Clases que desaprobaron LCOM

Fuente. Elaboración propia

Clase	Valor LCOM (%)
TestButton	100
WordRotation	100
WordSinusoidalMov	100
AudioManager	100
DefinicionesPalabras	100
GameManager	91.62
InicioDeSesion	100
ListaNombres	95.24
SwipeDetection	85.71
@TouchControlsMapa	100
BotonAbrirMenuUnidad	74.55
GuardarTiempoManager	100
Productinfo	100
FloatingWord	100
LenguaControl	83.33
CheckWritten	100
MostrarEstrellasOrdenamiento	100
PlacementModified	93.33
FloatingLocation	100
ResultadosActividad	100
FormularioNivelPrimero	100
pruebaFirestore	100
EleccionDeAlternativaAR	100
EstrellasSelector	80

5.1.5 Métrica NM

Se evaluaron 114 clases respecto al número de métodos, el resultado se visualiza en Tabla 14.

Tabla 14. Resultados NM
Fuente. Elaboración propia

Mantenible	Cantidad	Porcentaje	Porcentaje Total
Si	113	99.1%	99.1%
No	1	0.9%	0.9%

Las clases que no son mantenibles bajo los criterios de NM se ven en Tabla 15.

Tabla 15. Clases que desaprobaron NM
Fuente. Elaboración propia

Clase	Valor NM
GameManager	118

5.1.6 Métrica CBO

Se evaluaron 70 clases respecto al nivel de acoplamiento con otras clases, el resultado se visualiza en Tabla 16.

Tabla 16. Resultados CBO
Fuente. Elaboración propia

Mantenible	Cantidad	Porcentaje	Porcentaje Total
Si	70	100%	61.4%
No	0	0%	0%

Bajo los criterios de CBO todas las clases fueron mantenibles.

5.2 Evaluación de Eficiencia de Desempeño

5.2.1 Tiempo de Carga y Espera

Se realizó un total de 10 muestras de datos en distintas actividades AR, lo cual se ejecutó utilizando librerías de Unity para toma de tiempos, luego se calculó la diferencia entre estos, los resultados se pueden ver en Tabla 17.

Tabla 17. Evaluación TR y TE
Fuente. Elaboración propia

ID Actividad	TR [seg]	TE [seg]
B1U001A000	0.004882813	1.311523
B1U002A000	0.007324219	3.874023
B1U006A000	0.002996445	1.575865
B2U005A000	0.004882813	1.264648
B1U001A001	0.002254486	0.8793983
B1U001A002	0.007293701	0.9243393
B1U001A003	0.001159668	0.8617401
B1U002A001	0.001953125	0.8642578
B1U002A002	0.003314972	0.8325691
B1U002A003	0.004821777	0.8372192



El análisis de los resultados se presenta en la Tabla 18, donde se observa que en las pruebas de TR se aprobaron todos los casos con amplia diferencia respecto al umbral propuesto, por otro lado, dos casos en los que no se aprobó la prueba TE, siendo uno de ellos considerablemente superior al valor permitido.

Tabla 18. Resultados TR y TE

Fuente. Elaboración propia

Prueba	% Aprobados	% Desaprobados
TR	100%	0%
TE	80%	20%

5.2.2 Utilización de Memoria

Se tomaron un total de 10 muestras en diferentes escenas de RA, los resultados se encuentran en la Tabla 19. Donde se muestra la medición en los MB utilizados de la memoria RAM.

Tabla 19. Evaluación UM

Fuente. Elaboración propia

ID Actividad	UM (MB)
B2U002A000	391
B1U002A002	397
B2U001A001	373
CLU001	390
B1U002A000	394
TNL001	508
B1U001A000	395
CLU002	390
B1U003A003	392
B1U002A003	392

El resultado fue aprobado el 100% de los casos de prueba.

5.3 Evaluación de Fiabilidad

5.3.1 Realización de actividades y envío de datos

Se seleccionaron 15 actividades dentro de ARmadillo para las pruebas referentes al funcionamiento con internet intermitente durante la realización de actividades y la posterior recuperación de datos una vez restablecida la conexión. Los resultados de estas se pueden ver en la Tabla 20.

Tabla 20. Resultados realización de actividades y envío de datos

Fuente. Elaboración propia

ID Actividad	Termino actividad	Recuperación de Datos
B1U002A002	Aprobado	Aprobado
B2U001A001	Aprobado	Aprobado
CLU001	Aprobado	Aprobado
B1U003A003	Aprobado	Aprobado
CLU003	Aprobado	Aprobado
B1U003A002	Aprobado	Aprobado
ESP001	Aprobado	Aprobado
B2U002A000	Aprobado	Aprobado

B1U002A000	Aprobado	Aprobado
B1U005A000	Aprobado	Aprobado
TNL001	Aprobado	Aprobado
B1U002A003	Aprobado	Aprobado
B2U001A000	Aprobado	Aprobado
B1U003A003	Aprobado	Aprobado
CLU002	Aprobado	Aprobado

El 100% de los casos de prueba en ambas evaluaciones fueron aprobados.

5.3.2 Tiempo límite

Para evaluar el funcionamiento del tiempo límite con internet intermitente, se realizaron cinco casos distintos. En todos los casos se debía bloquear la aplicación una vez que el usuario se encontrara en el menú de actividades. En el caso 1 se esperó en el menú hasta que se finalizara el tiempo, en los casos 2 y 3 se ingresó a actividades y una vez finalizado el tiempo se regresó al menú, por último, en los casos 4 y 5 se ingresó a la sección “Casa” y una vez finalizado el tiempo, se volvió al menú. Los resultados se presentan en la Tabla 21.

Tabla 21. Resultados tiempo límite
Fuente. Elaboración propia

Casos	Resultado
Caso 1	Apruebo
Caso 2	Apruebo
Caso 3	Apruebo
Caso 4	Apruebo
Caso 5	Apruebo

En los cinco casos de prueba se bloqueó la actividad, una vez finalizado el tiempo límite y vuelto al menú de actividades. La prueba se aprobó en el 100% de los casos.

6 Análisis de resultados

Finalizadas las evaluaciones de calidad se procede a analizar los resultados obtenidos.

6.1 Análisis de Mantenibilidad

El proceso de obtención de métricas se desarrolló de acuerdo con lo planeado, logrando evaluar un total de 115 clases, en las cuales se identificó que el 31.3% de las clases presentan problemas de mantenibilidad según los criterios aplicados. Esto abre una serie de oportunidades para realizar ajustes en el código y en la distribución de lógica en las clases.

Las clases que desaprobaban la métrica AC presentan problemas a la hora de testear su funcionamiento y ser modificadas, debido a su alta complejidad promedio, estas presentan una oportunidad de ser simplificadas. Se debieran evaluar los métodos para refactorizarlos, buscando simplificar la lógica.

La clase que desaprobó la métrica WMC (GameManager) presenta problemas en su analizabilidad. Esta maneja una gran cantidad de lógica, debido a que también fue la única que desaprobó la métrica NM, lo cual afecta su modularidad. La clase representa una oportunidad de ser dividida en nuevas clases o métodos que cuenten con responsabilidades más específicas.



El caso de las clases que desaprobaron la métrica AMS ven perjudicada su analizabilidad y testabilidad, debido a que los métodos de estas clases cuentan con una gran cantidad de líneas de código. Presentan la oportunidad de dividir la lógica en nuevas clases o métodos.

La métrica LCOM es el factor que debiera determinar si la reubicación de la lógica de una clase debe darse a una nueva clase o un nuevo método, ya que si la clase desaprobó esta métrica presenta una baja cohesión, lo que implica que sus métodos manejan distintas lógicas, permitiéndole a una clase múltiples responsabilidades, afectando la modularidad del código. Por lo que, en caso de tener que dividir la lógica de una clase se debiera analizar su LCOM, en caso de ser aprobado, la reubicación de lógica se recomendaría realizarla en un nuevo método, en cambio si se desaprueba, la reubicación debiera hacerse en una nueva clase.

Las clases que desaprobaron solamente LCOM deben evaluarse para determinar si requieren la división de su lógica en más clases, esto debido a que en muchos casos donde se desaprobó esta métrica la clase ya es lo suficientemente pequeña, por lo que dividirla no afectaría realmente a la modularidad o analizabilidad del código, además se corre el riesgo de sobrepoblar el código de clases. Por lo que en casos de solo desaprobarse LCOM se debiera evaluar el tamaño de la clase, esto con el criterio de NM, en caso de ser bajo, no valdría la pena distribuir.

6.2 Análisis de Eficiencia de Desempeño

El proceso se llevó a cabo como se esperaba, se lograron obtener las medidas planeadas desde un inicio. En el caso de la utilización de memoria (UM), se aprobaron todos los casos de prueba, por lo que se concluye que al menos en el dispositivo de prueba, se utiliza un nivel de memoria RAM adecuado.

En la prueba de carga de objetos en relación con la carga de la cámara, se concluye que la diferencia es insignificante, debido a que se aprobaron las pruebas por un amplio margen. Lo que lleva a pensar que la diferencia de carga entre la cámara AR y los objetos 3D no varía significativamente. Este hecho puede estar relacionado con que en la preproducción del proyecto se planteó un límite de polígonos para los modelos 3D, el cual buscaba que los modelos fueran fáciles de cargar por los dispositivos móviles, en ese aspecto se puede determinar que el tiempo de carga de los objetos es ideal.

En relación con la prueba de tiempo de espera de la escena AR, se obtuvo como resultado que, en algunos casos, se excede el tiempo umbral de espera. Al no deberse al tiempo de carga de los objetos (debido al análisis de la anterior evaluación), este tiempo se relaciona directamente con la capacidad del equipo móvil para montar el entorno AR. El inicio de la carga de este entorno inicia una vez el usuario aprieta el botón correspondiente al inicio de esta parte de la actividad. Esto representa una oportunidad de mejora, debido a que sabemos de antemano que previo a la interacción con el botón se iniciará a continuación el entorno AR.

6.3 Análisis de Fiabilidad

Las pruebas de fiabilidad se llevaron de acuerdo con lo estimado, obteniendo un 100% de aprobación en los tres casos de prueba. En conclusión, el software ARmadillo se encuentra al momento de realizado este trabajo, con los estándares definidos al iniciar el proyecto sobre el funcionamiento de la aplicación con internet inestable.

7 Plan de futuras mejoras

El plan de futuras mejoras busca aprovechar las oportunidades encontradas en el análisis de la evaluación. Debido a que en las pruebas de fiabilidad no se desaprobó ningún caso, este plan de futuras mejoras solo contemplará el área de mantenibilidad y eficiencia de desempeño.

7.1 Mantenibilidad

Se busca mejorar la testabilidad, analizabilidad, modularidad y modificabilidad actuales del código. Para esto, se plantean dos tareas: refactorizar métodos y reubicar código. En la Tabla 22 se observan las clases que deben ser modificadas, además se filtraron aquellas donde desaprobaron LCOM, pero que no contaban con un tamaño significativo como para ser modificadas.

Tabla 22. Métricas por modificar

Fuente. Elaboración propia

ID	Clase	NM	WMC	AC	AMS	LCOM	CBO
1	GameManager	X	X			X	
2	InicioDeSesion			X	X	X	
3	PalabraDiccionario			X	X		
4	VerificacionDeUsuario			X	X		
5	MostrarResultados			X	X		
6	GetInventarios			X	X		
7	EleccionMultipleSinOrden			X	X		
8	MostrarEstrellasOrdenamiento			X		X	
9	ResultadosActividad			X		X	
10	FormularioNivelPrimero			X		X	
11	EstrellasSelector			X		X	
12	LenguaControl					X	
13	PlacementModified					X	
14	ListaNombres					X	
15	SwipeDetection					X	
16	BotonAbrirMenuUnidad					X	
17	PersonajeTouchMapa			X			
18	PuntosTouchMapa			X			
19	TiendaManager			X			
20	DeteccionEscena			X			
21	OrdenamientoManager			X			
22	InfoBoton			X			
23	EleccionDeAlternativa			X			

La clase GameManager debe ser dividida, distribuyendo su lógica en clases más específicas. Actualmente, maneja la lógica de la obtención, actualización y envío de datos, además del tiempo límite de uso de la aplicación y la lógica de las compras en la tienda. Se deben crear dos nuevas clases, una que se encargue de la lógica del tiempo límite de la aplicación y otra de la lógica de compras. Con esto, se espera definir responsabilidades únicas para cada clase.

La clase InicioDeSesion cuenta con solo dos métodos. El problema es que uno de estos tiene una gran cantidad de lógica, la cual debe ser refactorizada buscando no modificar su comportamiento, para esto se proponen la creación de nuevos métodos para distribuir su lógica. En este caso, a pesar de haber desaprobado LCOM no se propone la división en nuevas clases, debido a que cuenta solo con dos métodos y estos se encuentran orientados a la misma lógica.

El procedimiento anterior es el mismo que deben seguir las clases con ID del 3 al 7. Estas deben ser refactorizadas y distribuidas en nuevos métodos, debido a su valor de AMS. Por otro lado, las clases de ID del 17 al 23 solamente se deberá implementar una refactorización de sus métodos y, en caso de no poder simplificarse se buscará la creación de nuevos métodos.



Las clases 8, 9 y 10 deberán ser refactorizadas, pero debido a su tamaño no se realizará una distribución en nuevas clases, en cambio, la clase 11 deberá ser refactorizada y dividida en nuevas clases. Actualmente, esta última clase presenta la lógica de selección de puntaje, se debe dividir de acuerdo con la lógica específica de cada tipo de actividad.

La clase 13 dividirá su lógica en dos clases, una para la ubicación del objeto 3D y otra para moverlo. La clase 14 se dividirá en dos clases, una para la lógica de tratamiento de datos y otra para mostrar la pantalla. La clase 16 se dividirá en tantas clases como tipos de actividad tenga la aplicación, cada uno con la lógica perteneciente a su actividad. Por último, la clase 12 y 15 no se modificarán debido a que la lógica que manejan debe permanecer en la misma clase.

7.2 Eficiencia de Desempeño

La mejora que se busca respecto a la eficiencia es la disminución del tiempo de espera para la carga del entorno. Bajo este objetivo, se propone la precarga del entorno en la pantalla previa al inicio de la etapa en realidad aumentada.

Actualmente el inicio del entorno AR se da en dos casos, la primera es que la actividad inicie con una etapa 2D para luego iniciar la AR y la segunda es que directamente la actividad inicie con el entorno AR, en este caso se debiera agregar una pantalla 2D inicial que oculte la precarga del entorno.

Los pasos por seguir serían:

1. Elaborar pantalla 2D previa al inicio de actividad AR (esto solo para el segundo caso).
2. Movilizar lógica de carga a la pantalla previa del escenario AR.
3. Al pulsar el inicio de actividad se desactiva la pantalla 2D.

Esta modificación permitirá a los componentes de la escena AR cargarse en la pantalla 2D previa, la cual cumple la función de telón, por lo que una vez pulsado el botón no se dirigirá a otra escena, si no que se desactivará la pantalla, ya que ahora en este punto ya se estará en la escena AR.

La prioridad con la que se deben implementarse las modificaciones se presenta en la Tabla 23.

Tabla 23. Prioridad de implementación

Fuente. Elaboración propia

Alta	Media	Baja
GameManager InicioDeSesion	PalabraDiccionario VerificacionDeUsuario MostrarResultados GetInventarios EleccionMultipleSinOrden EstrellasSelector PlacementModified ListaNombres BotonAbrirMenuUnidad	PersonajeTouchMapa PuntosTouchMapa TiendaManager DeteccionEscena OrdenamientoManager InfoBoton EleccionDeAlternativa MostrarEstrellasOrdenamiento ResultadosActividad FormularioNivelPrimero

8 Conclusiones

Se propuso una puesta en práctica del modelo ISO/IEC 25010 para el desarrollo de videojuegos y tecnología de realidad aumentada, utilizando enfoques de evaluación de software en general. La evaluación de calidad elaborada para ARmadillo tuvo un alcance de tres características relevantes para el modelo ISO 25010, siendo estas: mantenibilidad, eficiencia de desempeño y fiabilidad. El desarrollo del proceso reveló el estado en que se encuentra el software, limitándose a estas áreas, que eran el enfoque en esta oportunidad y quedando para el futuro la evaluación de más características que enriquezcan el proceso de evaluación de calidad.

En relación con los resultados de la evaluación, se puede concluir que ARmadillo en el aspecto de la fiabilidad cuenta con los estándares de funcionamiento bajo condiciones de internet inestable que se esperan de su correcto funcionamiento. Respecto a la eficiencia de desempeño se detectó una oportunidad de mejora en los tiempos de espera de las escenas de realidad aumentada. Por último, en mantenibilidad se detectaron múltiples oportunidades de mejora, respecto a la complejidad y modularidad del código.

Las oportunidades de mejora detectadas afirman que el uso de modelos de calidad para la evaluación de software permite identificar debilidades del producto, ya sea en condiciones que no se cumplen, como también en aspectos que se pueden mejorar.

En relación con el proceso de evaluación en general, se recomienda mantener esta instancia a lo largo de todo el desarrollo de ARmadillo, ejecutando las evaluaciones de forma periódica. Realizar un seguimiento del estado en el que se encuentra el software semana a semana, comparando los resultados de las pruebas con las evaluaciones anteriores y analizando las variaciones percibidas en el periodo de desarrollo. Esto podría permitir al equipo anticiparse a futuras complicaciones y resolverlas una vez se detecten, disminuyendo los costos que podrían traer dichos problemas.

Respecto a las pruebas que se realizaron, estas debieran someterse constantemente a evaluación, buscando un proceso de mejora continua. Esto contempla tanto las especificaciones de las pruebas, como los métodos con los que se llevaron a cabo. La evolución de las pruebas debiera ser enfocada tanto en reflejar de mejor forma el estado de la aplicación como también en la eficiencia con que se lleva a cabo el proceso evaluador.

En el apartado de las **pruebas de mantenibilidad** se logró visualizar el estado actual del código, pero para una futura implementación de las métricas se debiera ajustar el umbral de aprobación en función del contexto del equipo y la naturaleza del producto. Cada software y equipo programador es distinto, por lo que un marco personalizado de evaluación, considerando umbrales límite personalizados para la evaluación de ARmadillo reflejaría el estado en que se encuentra el software de forma más valiosa para el equipo. Para esto se debiera realizar un estudio de las clases en busca de aquellas que no sean mantenibles desde el punto de vista de los desarrolladores, luego realizar un análisis de las métricas para buscar correlación entre estas, definiendo así nuevos umbrales límites.

Se recomienda para aquellos que busquen realizar una evaluación de mantenibilidad en software desarrollado en Unity, que utilicen herramientas de evaluación que permitan personalizar el cálculo de las métricas. Para esta evaluación se utilizó Roslyn, que pese a complicarse la programación en esta para métricas más avanzadas, permitió personalizar la toma de datos logrando aislar el código fuente de Unity y las librerías propias de su programación. Esto, con el fin de enfocar la evaluación al trabajo de los desarrolladores sin incluir aspectos de la naturaleza de Unity, dándole mayor valor a los datos que se obtuvieron.

Una mejora en el proceso de evaluación de mantenibilidad sería evaluar las métricas como un conjunto y no de forma separada. Como se visualizó en esta primera evaluación, la relevancia de las métricas varía según cada clase, un ejemplo de esto fue LCOM, el cual dio valores de 100% de desacoplamiento en una gran cantidad de clases, pero estas contaban con un NM bajo, lo que generaba que la métrica LCOM perdiera relevancia debido



a la naturaleza de dichas clases. Por lo que se debiera ajustar el proceso considerando una evaluación conjunta de métricas, que retorne un grado de mantenibilidad respecto a la totalidad de las métricas.

Las **pruebas de eficiencia de desempeño** del software se llevaron a cabo según lo esperado y se logró evaluar los tiempos y recursos utilizados en el entorno de desarrollo. Las pruebas se desarrollaron en el dispositivo con que contaba el equipo para el desarrollo, esto es un punto de mejora hacia futuras pruebas a futuro, debido a que se debiera realizar un estudio de los aparatos móviles con los que cuentan las instituciones de educación básica en Chile y en base a los resultados realizar prueba con dichos dispositivos.

Respecto a las **pruebas de fiabilidad** estas debieran ampliarse, manteniendo aquellas que buscan validar la fiabilidad del manejo de la aplicación en casos de internet inestable y agregando otro tipo de pruebas relacionadas con fallas del dispositivo móvil.

En relación con el **plan de futuras mejoras**, este queda pendiente para una futura implementación, donde posterior a su implementación debiera llevarse a cabo una evaluación similar para comparar los resultados con la versión sin mejoras implementadas. Por último, este plan de mejoras es de gran valor para el equipo desarrollador y se recomienda ponerlo en práctica antes de buscar escalar la aplicación, esto para generar un proceso de evaluación constante en el desarrollo del software, permitiendo aminorar posibles costos, con el fin de entregar un producto de calidad.

Agradecimientos

Me gustaría agradecer a mi familia por apoyarme durante todos estos años, en especial a mi madre Claudia y a mi abuelo Carlos; sin ellos no habría podido llegar a esta instancia. También agradezco a los profesores que me tendieron una mano en el camino y a los amigos que me acompañaron durante este proceso. Finalmente, quiero agradecer a Fernando Alonso, cuyo ejemplo de perseverancia me ayudó a no bajar los brazos, y a Brandon Sanderson, cuyas historias me acompañaron a lo largo de todo este trayecto. A todos, muchas gracias.

Referencias

- [1] R. S. Pressman, Ingeniería de software: un enfoque práctico, 7 ed., México: McGraw-Hill, 2010.
- [2] B. Singh y S. Prasad Kannoja, «A Review on Software Quality Models,» de *2013 International Conference on Communication Systems and Network Technologies*, Gwalior, India, 2013.
- [3] Institute of Electrical and Electronics Engineers, «IEEE Standard Glossary of Software Engineering Terminology,» IEEE, New York, 1990.
- [4] ISO/IEC, «ISO/IEC 25010:2023 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models,» 2023.
- [5] S. Rifkin, «Software measurement,» *IEEE Software*, p. 70, 2009.
- [6] I. Sommerville, Ingeniería del software, 9 ed., Madrid: Pearson Educación, 2011.
- [7] D. Samadhiya, S.-H. Wang y D. Chen, «Quality Models: Role and Value in Software Engineering,» de *2010 2nd International Conference on Software Technology and Engineering*, San Juan, PR, USA, 2010.
- [8] Y. Boukouchi, A. Marzak, H. Benlahmer y H. Moutachaouik, «Comparative study of software quality models,» *International Journal of Computer Science Issues (IJCSI)*, vol. 10, n° 6, p. 309–316, Noviembre 2013.
- [9] J. P. Miguel, D. Mauricio y G. Rodríguez, «A review of software quality models for the evaluation of software products,» *International Journal of Software Engineering & Applications (IJSEA)*, vol. 5, n° 6, p. 31–53, Noviembre 2014.

- [10] R. Shonubi, «Sustainability Analysis of Software Quality Model and Collection of Real Time Metrics for Software Process Improvement,» Lappeenranta-Lahti University of Technology (LUT), Lappeenranta, 2019.
- [11] B. Kitchenham y S. L. Pfleeger, «Software Quality: The Elusive Target,» *IEEE Software*, vol. 13, nº 1, p. 12–21, 1996.
- [12] D. Milicic, «Software quality models and philosophies,» de *Software quality attributes and trade-offs*, Blekinge Institute of Technology, 2005.
- [13] R. G. Dromey, «A model for software product quality,» *IEEE Transactions on Software Engineering*, vol. 21, nº 2, p. 146–162, 1995.
- [14] H. Al-Kilidar, K. Cox y B. Kitchenham, «The use and usefulness of the ISO/IEC 9126 quality standard,» de *2005 Australian Software Engineering Conference (ASWEC)*, Noosa Heads, QLD, Australia, 2005.
- [15] P. Roa, C. Morales y P. Gutiérrez, «Norma ISO/IEC 25000,» *TIA*, vol. 3, nº 2, pp. 26-32, 2015.
- [16] arc42 Quality Hub, «Actualización de la norma ISO 25010, versión 2023,» 2 Octubre 2023. [En línea]. Available: <https://quality.arc42.org/articles/iso-25010-update-2023>. [Último acceso: 12 04 2025].
- [17] ISO25000.com, «La familia de normas ISO/IEC 25000,» 2024. [En línea]. Available: <https://iso25000.com/index.php/normas-iso-25000>. [Último acceso: 14 04 2025].
- [18] ISO/IEC, «ISO/IEC 25040:2024 Software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Evaluation process,» 2024.
- [19] E. Irrazábal y J. Garzás, «Análisis de métricas básicas y herramientas de código libre para medir la mantenibilidad,» *Revista Española de Innovación, Calidad e Ingeniería del Software*, vol. 6, nº 3, pp. 56-65, 2010.
- [20] M. Vogel, P. Knapik, M. Cohrs, B. Szyperrek, W. Pueschel, H. Etzel, D. Fiebig, A. Rausch y M. Kuhrmann, «Metrics in Automotive Software Development: A Systematic Literature Review,» *Journal of Software: Evolution and Process*, vol. 32, nº 3, 2020.
- [21] O. Özdemir y F. Buzluca, «Evaluating Microservices Maintainability: A Classification System Using Code Metrics and ISO/IEC 250xy Standards,» de *13th International Conference on Software and Computer Applications (ICSCA 2024)*, 2024.
- [22] C. Ebert, J. Cain, G. Antoniol, S. Counsell y P. Laplante, «Cyclomatic Complexity,» *IEEE Software*, vol. 33, nº 6, p. 27–29, Noviembre/Diciembre 2016.
- [23] T. McCabe, «A Complexity Measure,» *IEEE Transactions on Software Engineering*, Vols. %1 de %2SE-2, pp. 308 - 320, 31 December 1976.
- [24] T. Beranič, V. Podgorelec y M. Heričko, «Towards a Reliable Identification of Deficient Code with a Combination of Software Metrics,» *Applied Sciences*, vol. 8, nº 10, p. 1902, 2018.
- [25] R. Harrison, S. Counsell y R. R. Nithi, «Overview of object-oriented design metrics,» de *Software Technology and Engineering Practice, 1997. Proceedings., Eighth IEEE International Workshop on [incorporating Computer Aided Software Engineering]*, 1997.
- [26] S. R. Chidamber y C. F. Kemerer, «Overview of Object-Oriented Design Metrics,» de *1994 Conference on Software Engineering Education*, New Orleans, LA, USA, 1994.
- [27] F. A. Salazar Fierro, C. A. Pineda Manosalvas, N. N. Cervantes Rodríguez y P. Landeta-López, «Análisis de la eficiencia de desempeño en aplicaciones de Realidad Aumentada utilizando la normativa ISO/IEC 25010,» *Revista Ibérica de Sistemas e Tecnologías de Informação (RISTI)*, nº E22, p. 256–267, 2019.
- [28] Microsoft, «Roslyn,» 2025. [En línea]. Available: <https://github.com/dotnet/roslyn>.
- [29] Unity Technologies, «Memory Profiler: Memory on Device,» 2024. [En línea]. Available: <https://docs.unity3d.com/Packages/com.unity.memoryprofiler@1.1/manual/memory-on-device.html>.