Estudio de la Capacidad de Mantenimiento de las Aplicaciones Móviles Híbridas

Daniel Díaz[1], Susana Herrera[1], Federico Rosenzvaig[1]

[1] Instituto de Investigación en Informática y Sistemas de Información, Universidad Nacional de Santiago del Estero, 1912 Av. Belgrano (S), Santiago del Estero, Argentina
{danieldiaz,sherrera}@unse.edu.ar, fedvaig@gmail.com

Abstract. El presente artículo muestra los resultados de una investigación aplicada sobre la capacidad de mantenimiento de las aplicaciones móviles híbridas multiplataforma. Se utilizó el *framework Ionic* para generar aplicaciones Android y iOS a partir de un entorno web basado en el núcleo de HTML5, CSS3 y Javascript. Para la evaluación, se desarrolló la aplicación *Mi día en la UNSE*, usada en el evento de difusión de la oferta académica de la Universidad Nacional de Santiago del Estero. El desarrollo se basó en la metodología Mobile-D. La capacidad de mantenimiento de las aplicaciones móviles generadas por Ionic fue evaluada en el marco de la norma ISO/IEC 25.000, usando la estrategia de medición y evaluación *Goal Oriented Context Aware Measurement and Evaluation* (GOCAME). Los resultados obtenidos conducen a afirmar que el framework Ionic permite generar aplicaciones multiplataforma híbridas con alto nivel de mantenibilidad.

Keywords: Desarrollo híbrido de aplicaciones móviles, Ionic, Mantenimiento de aplicaciones móviles, Mobile-D.

1 Introducción

A pesar de sus ventajas, las aplicaciones móviles requieren ser desarrolladas atendiendo características especiales y limitadas de los dispositivos donde se ejecutan: sistema operativo (SO), capacidad del procesador, tamaño de pantalla, duración de la batería, conectividad, entre otros. Considerando el SO, las aplicaciones móviles requieren ser desarrolladas en forma separada para cada uno, lo cual implica un alto costo de desarrollo y mantenimiento, ya que involucra formar equipos de trabajo con competencias profesionales diferentes.

Sin embargo, han surgido alternativas que permiten que estas aplicaciones se desarrollen bajo el núcleo de las tecnologías web y se ejecuten de forma nativa en los diversos SO móviles, mediante un explorador web. A este tipo de aplicaciones se las denomina aplicaciones móviles híbridas [1]. Esto da la posibilidad de diseñar una única instancia, que sea adaptable a las diferentes plataformas del mercado, optimizando la eficiencia del desarrollo. Sin embargo, este tipo de desarrollo podría arrastrar problemas en el mantenimiento de las aplicaciones generadas. Existen varios

frameworks que permiten generar aplicaciones híbridas. En este artículo se muestran resultados de un estudio sobre la capacidad de mantenimiento (o mantenibilidad) de aplicaciones híbridas usando el *framework* Ionic.

Para llevar adelante la evaluación, se construyó una aplicación móvil híbrida denominada Mi día en la UNSE. La misma fue utilizada en el evento "La UNSE para todos", en el cual la Universidad Nacional de Santiago del Estero presenta su oferta académica a alumnos de nivel secundario de toda la provincia. La aplicación posee información relacionada con la difusión de las carreras de todas las facultades, así como también brinda actividades de transmisión en vivo y de recreación (concurso fotográfico).

La calidad de un sistema puede ser vista desde un enfoque interno o externo. Desde el modelo de calidad interna, la calidad es evaluada durante la revisión del producto, considerando sus características, siendo la mantenibilidad una de las principales según el estándar ISO IEC 25000 [2].

Cabe considerar que el mantenimiento, representa más esfuerzo que cualquier otra actividad de la Ingeniería del software. Es la facilidad con la cual se puede corregir un programa si se encuentra un error, adaptarlo si su entorno cambia, o mejorarlo si el cliente desea un cambio en los requisitos.

Para evaluar la característica de mantenibilidad se tomó como referencias los pasos sugeridos por la estrategia GOCAME [3].

Este trabajo forma parte de un proyecto más amplio donde se estudian diversos aspectos de las aplicaciones móviles multiplataforma [4].

El artículo se estructura de la siguiente manera. En el apartado 2 se presentan los conceptos básicos que sustentan la investigación: aplicaciones móviles híbridas, calidad del software, mantenibilidad, estrategias para la evaluación de la mantenibilidad. En el apartado 3 se presenta el desarrollo de la aplicación multiplataforma usando la metodología Mobile-D y el *framework* Ionic. Posteriormente, en el apartado 4, se muestra la estrategia seguida para la evaluación de la mantenibilidad y los resultados obtenidos. Finalmente, en el apartado 5 se presentan las conclusiones alcanzadas.

2 Marcos referenciales

2.1 Aplicaciones móviles híbridas

Las aplicaciones móviles son aplicaciones informáticas diseñadas para ser ejecutadas en teléfonos inteligentes, tabletas y otros dispositivos móviles [5]. Se encuentran disponibles a través de plataformas de distribución, operadas por las compañías propietarias de los SO móviles como Android, iOS, BlackBerry OS, Windows Phone, entre otros. En la actualidad, el mercado de aplicaciones móviles está dominado por dos grandes empresas: Google y Apple, con sus SO móviles Android y iOS, respectivamente [6].

Las aplicaciones móviles son diferentes para cada SO. Son aplicaciones nativas que se desarrollan en el lenguaje nativo del propio dispositivo. Son dependientes del SO donde residen. Las aplicaciones no son portables, hay que desarrollar una por cada

SO. El lenguaje de programación para aplicaciones Android es Java. Mientras que las aplicaciones iOS se programan con Objective C y Swift.

En los últimos años, ha surgido como alternativa el desarrollo móvil multiplataforma, que trata de compartir la misma codificación para los diferentes SO. Entre sus ventajas sobresalen: menor tiempo y costo de desarrollo, permite acceso al hardware del dispositivo, disponibilidad de entornos potentes de desarrollo (Delphi, Visual Studio, etc.), utilización de tecnologías web (HTML5, Javascript y CSS). Sin embargo, el rendimiento de las aplicaciones y sus interfaces de usuario, pueden afectar la experiencia de usuario [5].

Según Lisandro Delía [1, 7] las aplicaciones móviles multiplataforma pueden ser de diferentes tipos:

- Aplicaciones web responsivas. Se ejecutan dentro de un navegador, no dependen de ninguna plataforma y su ejecución es rápida y fácil. Una desventaja con respecto al desarrollo nativo, además de limitar accesos a todos los recursos de un dispositivo, se hace necesario acceder a internet para su funcionamiento. Existen *frameworks* que son marcos de desarrollo que hacen más simple el desarrollo de aplicaciones web con diseño adaptable, entre éstos están CodeIgniter [8], Synphony [9,10] y Laravel [11].
- Aplicaciones híbridas. Usan tecnologías web pero se ejecutan en un "contenedor web (*webview*)", es parte de una aplicación nativa, desde la cual se puede acceder a todos los recursos del dispositivo. Existen diversos *frameworks* que permiten es tipo de aplicación: PhoneGap [12], CocoomJS, Ionic [13], Sench Touch.
- Aplicaciones interpretadas. Usan un lenguaje base que traduce gran parte a código nativo, y el resto se interpreta cuando se ejecuta. Por un lado, esto es una ventaja, pero se depende en forma exclusiva de una plataforma.
- Aplicaciones generadas por compilación cruzada. Las aplicaciones se compilan en forma nativa para cada plataforma que se va a usar. Entre las plataformas de desarrollo se encuentran Xamarin [14], Embarcadero Delphi 10 Seattle, Ruby Motion.

2.2 Calidad del software y mantenibilidad

La norma ISO/IEC 25000 [2], conocida como SQuaRE (*System and Software Quality Requirements and Evaluation*), es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del software. Este estándar define tres vistas diferenciadas en el estudio de la calidad del producto:

- Vista interna: se ocupa de las propiedades del software como tamaño, complejidad o la conformidad con las normas de orientación a objetos.
- Vista externa: analiza el comportamiento del software en la producción y estudia sus atributos.
- Vista en uso: mide la productividad y efectividad del usuario final al utilizar el software.

La norma ISO/IEC 25010 contempla los modelos: modelo de calidad de producto y modelo de calidad en uso.

El Modelo de Calidad de Producto (vista interna/externa) categoriza las propiedades de la calidad del producto o sistema en ocho características:

funcionalidad, eficiencia, compatibilidad, usabilidad, confiabilidad, seguridad, **mantenibilidad** (o capacidad de mantenimiento) y portabilidad. Cada característica está compuesta por un conjunto de subcaracterísticas.

El Modelo de Calidad en Uso (vista en uso). Define cinco características referidas a las salidas de la interacción con el sistema: efectividad, eficiencia, satisfacción, libre de riesgos y contexto. Cada característica puede ser asignada a diferentes actividades de los *stakeholders*.

Este trabajo se enmarca en el Modelo de Calidad del Producto. Y, teniendo en cuenta el estándar, la mantenibilidad se define en función de las siguientes subcaracterísticas:

- Modularidad: Propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.
- Reusabilidad: Es el grado en que un programa (o partes de este) se puede reusar en otras aplicaciones.
- Analizabilidad: Facilidad con la que se puede evaluar el impacto de un determinado cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software, o identificar las partes a modificar.
- Cambiabilidad: Capacidad del producto que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.
- Capacidad de ser probado: Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.

3 Desarrollo de aplicación híbrida con Ionic

El análisis, diseño, codificación y prueba de la aplicación híbrida multiplataforma “Mi día en la UNSE” se hizo con Mobile-D [15]. Esta metodología se compone de diversas fases por las cuales pasa el producto a realizarse: exploración, inicialización, producción, estabilización y pruebas. Cada una de las fases posee un día de planificación y un día de entregas.

Una vez finalizada todas las fases, se obtuvo un prototipo de aplicación móvil distribuible para múltiples plataformas, en particular, Android, iOS y Windows Phone.

En la fase de exploración se definieron los requisitos iniciales, el alcance y los recursos a usar. Teniendo en cuenta las necesidades del Centro de Orientación de la UNSE, la aplicación debía ser usada en el evento “La UNSE abre sus puertas”, evento en el cual se muestra la oferta académica de la UNSE a los aspirantes de las escuelas secundarias. Las principales funcionalidades de la aplicación serían:

- Gestión de usuarios: Permitir el registro de usuarios para monitorear sus interacciones con el prototipo y para la generación de reportes estadísticos solicitados por el área del COEP.
- Difusión de carreras: utilizar la cámara del dispositivo para leer datos por medio de códigos QR y visualizar en pantalla su resultado. Este código posee información específica de los eventos que ocurren en la jornada de difusión de carreras.

- Concurso fotográfico: Subir una fotografía tomada con la cámara del dispositivo. El usuario que captó dicha fotografía participa de un sorteo por un premio sorpresa.
- Transmisión en vivo: Posibilidad de seguir el evento vía streaming por el canal de Youtube de UNSE Tevé.

En cuanto a los recursos tecnológicos necesarios, se definieron los siguientes:

- Entorno de desarrollo: Ionic *Framework*
- Lenguaje de programación: Java, TypeSript
- Tecnologías web: HTML5, SASS, ANGULAR JS, JS, PHP
- IDE de desarrollo: Visual Studio Code
- IDE de diseño y maquetado: Adobe Photoshop CS6
- Gestor de Base de Datos: Firebase
- Bibliotecas: Node JS v8.11.2, Apache Cordova, Java SE Development Kit 8, Android Studio 3.1.2, Xcode 7, SDK WP8, Microsoft Visual Studio 2015

En la fase de inicialización, se prepararon los recursos tecnológicos (instalación y configuración) y humanos. Además, se definió la arquitectura de la aplicación y se realizó la planificación de las fases y de sus iteraciones.

En la fase de Producción se llevó a cabo toda la implementación de la aplicación. Se definieron 7 historias de usuario: registro de usuarios, acceso de usuarios, escanear código QR, registro y captura de imagen, transmisión en directo, programación de actividades. A partir de ello, se confeccionaron las *Task Card* en las cuales se asignaron las tareas a los desarrolladores. Posteriormente, se diseñó el modelo de la Base de Datos y se lo implementó en Firebase. Luego, se diseñaron las pantallas de usuario. Finalmente, se llevaron a cabo la codificación y las pruebas de aceptación (a cargo de los desarrolladores). En las figuras 1 y 2 se muestran pantallas correspondientes a algunas de las funcionalidades de la aplicación.

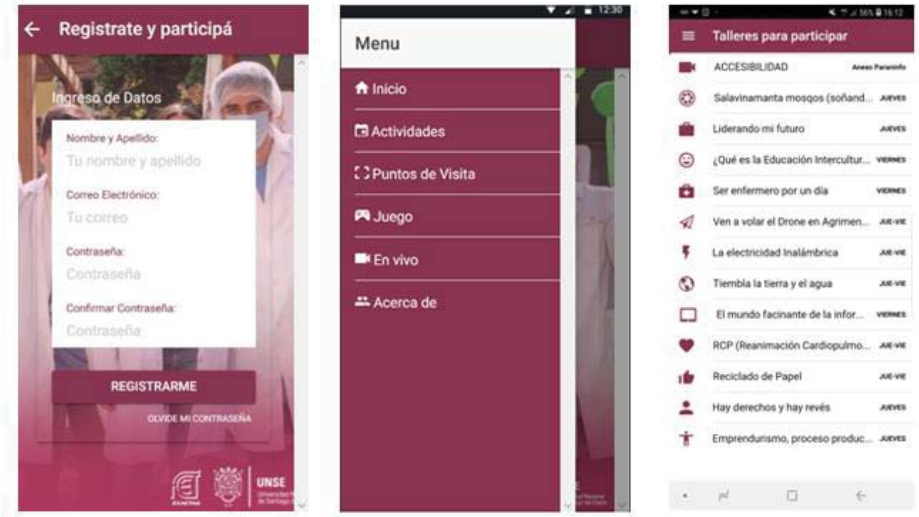

Figura 1. Principales pantallas de la aplicación Mi día en la UNSE: registro de usuarios, menú principal, actividades.

En la fase de estabilización, se llevaron a cabo las últimas acciones de integración para asegurar que el sistema completo funcione correctamente. Una vez integrado el sistema, se procedió a generar las aplicaciones nativas para los sistemas operativos móviles Android y iOS, mediante el empaquetador proporcionado por Ionic *Framework*. Este proceso es iterativo. Por cada modificación o integración que se realizada, se repitió el proceso de compilación y liberación de la aplicación nativa resultante.

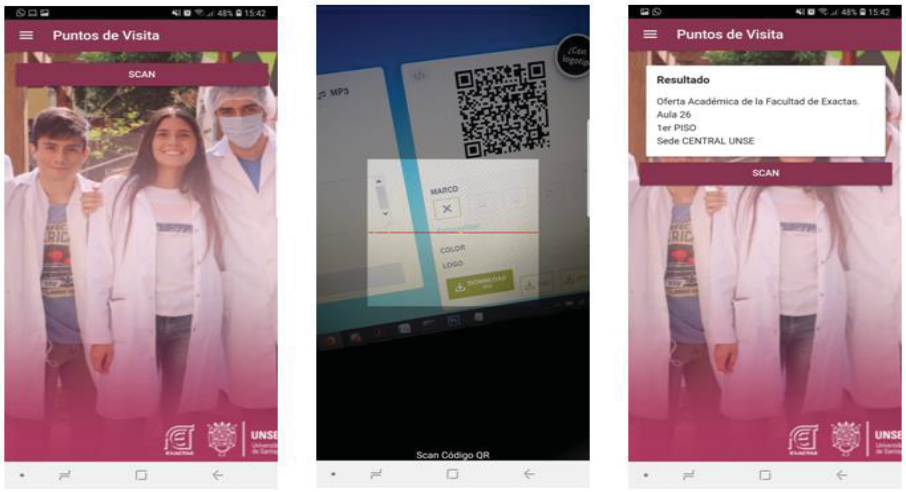

Figura 2. Pantallas de "Puntos de vista". Los usuarios obtienen información leyendo un código QR.

Por último, en la fase de prueba, se validan las funcionalidades del producto terminado, considerando los requisitos. Se utilizó la técnica de observación directa [16] sobre una muestra aleatoria de usuarios.

4 Evaluación de la capacidad de mantenimiento

Como se mencionó en 2.2, la calidad de un sistema puede ser vista desde un enfoque interno o externo. Desde el modelo de calidad interna, la calidad es evaluada durante el desarrollo del producto, considerando sus características, siendo la mantenibilidad una de las principales.

Para efectuar la medición y evaluación de la mantenibilidad de las aplicaciones móviles generadas a partir de la utilización de *Ionic*, se seleccionó la estrategia GOCAME. Para la elección se tuvo en cuenta sus numerosos casos de aplicación, la base conceptual, la definición precisa del proceso de medición y evaluación y la solidez de sus métodos y herramientas. GOCAME es una estrategia para la medición y evaluación de proyectos de cualquier especie basada en propósitos y objetivos [3]. Fue elaborada sobre las bases del *Framework Contextual Information Need, Concept Model, Attribute, Metric, Indicator* (C-INCAMI) y utiliza la metodología *Web Quality*

Evaluation Method (WebQEM) para el proceso de construcción de métricas e indicadores y proceso de ejecución de mediciones. Sus principales etapas son: Definición de los requerimientos no funcionales, Diseño de la evaluación, Implementación de la medición y evaluación, Análisis y Recomendaciones.

4.1 Definición de requerimientos no funcionales

Como lo indica la estrategia, en esta etapa se definió el Árbol de Requerimientos. Partiendo de la característica de calidad que se desea evaluar, capacidad de mantenimiento, se definieron las sub-características y los atributos (Ver Tabla 1). Se utilizaron las subcaracterísticas especificadas en la norma ISO IEC 25000 para la mantenibilidad: analizabilidad, modificabilidad, capacidad de ser probado y reusabilidad.

Tabla 1. Árbol de requerimientos (etapa 1 de GOCAME).

Mantenibilidad
1.1 Analizabilidad
1.1.1 Complejidad ciclomática
1.1.2 Código repetido
1.1.3 Densidad Comentarios
1.2 Modificabilidad
1.2.1 Densidad de ciclos
1.2.2 Nombres de las variables
1.3 Capacidad de ser probado
1.3.1 Densidad de pruebas
1.4 Reusabilidad
1.4.1 Estructura del programa

Complejidad ciclomática: se basa en el análisis del diagrama de flujo determinado por las estructuras de control de un determinado código. Cuanto más compleja sea la lógica del código, más difícil será de entender, mantener y probar.

Código Repetido: mide la relación entre la cantidad de código repetido de un producto y su tamaño (sin contar comentarios). Cuando un código tiene errores y este es repetido, también se duplican los errores.

Densidad de comentarios: mide la relación entre la cantidad de comentarios (documentación) de un producto y su tamaño (incluyendo comentarios). Cuanto mayor sea la densidad de comentarios, más fácil de mantener será el software examinado.

Densidad de ciclos: mide la relación entre el número de dependencias cíclicas que presenta el producto y la cantidad de módulos existentes.

Nombre de variables: los nombres de variables, funciones, constantes y procedimientos son omnipresentes deberían ser nombres significativos que tengan lógica y sean explicativos.

Densidad de pruebas: mide la relación entre el número de pruebas unitarias y la complejidad ciclomática, es decir, el número de pruebas unitarias codificadas en relación con el número de pruebas unitarias deseable.

Estructura del programa: el código fuente de una aplicación debe estar escrito de tal manera que un componente pueda leerse fácilmente desde lo general a lo particular. Esto reduce el tiempo de comprensión por parte de las personas que lean el código fuente y por ende resulta ser más mantenible.

4.2 Diseño de la medición

La medición del software se logra aplicando métricas. El diseño de las mediciones se llevó a cabo considerando: la definición de requisitos no funcionales (el árbol) las métricas existentes: la complejidad ciclomática, el total de líneas de un programa (LOC) y la densidad de comentarios. A partir de la definición de ésta última, se determinaron las restantes medidas, las cuales siguen una especificación similar.

4.3 Diseño de la evaluación

En esta etapa se definieron los indicadores para cada uno de los atributos y los valores considerados para evaluar el resultado (Ver Tabla 2).

Luego se diseñaron los indicadores globales para cada una de las subcaracterísticas, para ello se utilizó la sumatoria de los indicadores multiplicada por su peso. Ver Formula 1.

$$IG = \sum_{i=1}^{n} p_i * IE_i \quad (1)$$

Donde,

IG es cada indicador global.
n es la cantidad de indicadores elementales de una subcaracterística.
pi es el peso de cada atributo.
IEi es cada indicador elemental.

Se asignaron “pesos” a cada uno de los atributos de acuerdo a la relevancia que tiene cada uno de ellos, teniendo en cuenta la calidad del producto y la experiencia de desarrolladores. Sumados todos los de una subcaracterística, dan 1 (ver Tabla 3).

Luego, se definieron los indicadores globales (IG) y los umbrales para establecer un criterio de decisión. Dada la fórmula presentada anteriormente, el indicador global siempre puede tomar un valor perteneciente al rango real [1-100] y no se requiere mapeo alguno. En la Tabla 4 se presentan los indicadores globales.

IG para la característica de mantenimiento se define de la siguiente manera:

NCM=0,20 * NA + 0,50 * NM + 0,05 * NCP + 0,25 * NR

Siendo los criterios de decisión los siguientes:

10<=NCM<=40	Revisar código
40<NCM<=50	Marginal
NCM>50	Nivel de mantenimiento satisfactorio

Tabla 2. Indicadores y sus valores de decisión.

Indicador	Criterios de toma de decisión
CC = E-N +2 CC: Complejidad ciclomática E: número de aristas N: número de nodos	Criterios tabulados (McCabe,T., 1976) 1-10 Programa simple, sin mucho riesgo 11-20 Más complejo, riesgo moderado 21-50 Complejo, Programa de alto riesgo >=50 Programa con muy alto riesgo
DCR= (1-(LC/LT)) *100 DCR: Densidad de código repetido LC: Líneas de código duplicadas LT: Líneas totales de código – líneas comentadas	2<=DCR<=20 Insatisfactorio. Programa con mucho código repetido. 20<DCR<=50 Moderado, revisar código DCR>50 Satisfactorio.
DCOM= (LC/LOC) *100 DCOM: Densidad de comentarios LC: Líneas comentadas LOC: Líneas totales de código	10<=DCOM<20 Programa con pocos comentarios 21<=DCOM<=35 Programa con comentarios moderado DCOM>35 Programa correctamente documentado
CYCD= (1-(CEM/M))*100 CYCD: Densidad de ciclos CEM: Ciclos entre módulos M: Total de módulos	10<=CYCD<=25 Programa con muchos ciclos 26<CYCD<=50 Programa con ciclos moderado CYCD>50 Programa con pocos ciclos.
DNCS= NVCS/NVT *100 DNCS: Densidad de nombres con sentido NVCS: Número de variables con sentido NVT: Número de variables totales	15<=DNCS<50 Variables con poco sentido 51<=DNCS<=70 Variables con sentido moderado DNCS>70 Nombres correctos con sentido
UTD= (1-((UN/CC)/100))*100 UTD: Densidad de pruebas unitarias UN: Número de pruebas unitarias CC: Complejidad ciclomática	1<=UTD <=25 Insatisfactorio 26<UTD<50 Caminos de prueba moderado UTD>=50 Fácil de ser probado
EP=NMBE/M*100 EP: Estructura del programa NMBE: Número de módulos bien escritos M: Cantidad de módulos	15<=EP<=25 Programa con dificultad de entender 26<EP<=50 Programa moderado EP>50 Programa bien estructurado

4.4 Implementación de la medición y evaluación

Los proyectos desarrollados bajo el entorno Ionic generan una gran cantidad de archivos y directorios. El proyecto de la aplicación “Mi día en la UNSE” tiene el siguiente tamaño: 9 descriptores de acceso (número de funciones “get” y “set”), 9 clases, 4117 directorios, 35851 archivos, 537 líneas de código físicas, 16 métodos.

Los archivos que se tomaron para realizar la medición fueron los que contienen la mayor parte del código del proyecto: Registro.ts (funcionalidades relacionadas con el registro de usuarios), Login.ts (permite el acceso o denegación de usuarios en la

aplicación móvil), List.ts (despliega la lista de actividades), Reset.ts (implementa los mecanismos para la recuperación de contraseñas).

Tabla 3. Pesos asignados a los indicadores de los atributos.

Subcaracterística	Atributos	Peso de atributos	Peso de Subcaracterística
Analizabilidad	Complejidad ciclomática	0,30	0,20
	Código repetido	0,40	
	Densidad de comentarios	0,30	
Modificabilidad	Densidad de ciclos	0,25	0,50
	Densidad de nombres con sentido	0,75	
Capacidad de ser probado	Densidad de pruebas	1	0,05
Reusabilidad	Estructura del programa	1	0,25

Tabla 4. Indicadores Globales y sus valores de decisión.

Indicador Global	Criterios de toma de decisión	
Nivel de Analizabilidad (NA) NA=0,30 * CC + 0,40 * DCR + 0,30 * DCOM	1 <= NAM <=20 20<NAM<=50 NAM>50	No es analizable Analizable moderado Satisfactorio
Nivel de Modificabilidad (NM) NM=0,25 * DYCD + 0,75 * DNCS	15<=NM<=40 41<NM<=70 NM>70	Insatisfactorio Moderado Satisfactorio
Nivel de Capacidad de ser probado (NCP) NCP=1* UPD	1<=NCP<=20 20<NCP<=50 NCP>50	Difícil de probar Moderado Programa fácil de probar
Nivel de Reusabilidad (NR) NR=1* EP	15<=NM<=40 41<NM<=70 NM>70	Insatisfactorio Moderado Satisfactorio

En la Tabla 5 se presentan los valores obtenidos a partir de la medición de los atributos de la capacidad de mantenimiento, teniendo en cuenta el conjunto de archivos seleccionados del proyecto Ionic. Estos valores están mapeados en un rango de 1 a 100 y expresados en valor de porcentaje (%), ya que se aplica una función de proporción a cada de uno de ellos, a excepción de la complejidad ciclomática, cuyos valores ya están predeterminados, según la literatura. En la misma tabla se muestra, además, los resultados de la evaluación de los indicadores globales y también el valor final obtenido para la Capacidad de Mantenimiento. En todos los casos se indica con verde cuando los indicadores adoptan un valor positivo, con amarillo cuando adoptan un valor marginal (intermedio). En ningún caso se obtuvieron resultados negativos.

El valor de la Capacidad de Mantenimiento obtenido es 87,84.

Tabla 5. Mediciones obtenidas y evaluación.

Árbol de Requerimientos	**Archivos del Proyecto IONIC**			
	Registro.ts	Login.ts	List.ts	Reset.ts
Mantenibilidad		**87,84**		
Analizabilidad		44,87		
Complejidad ciclomática (CC)		4		
Densidad de Código Repetido (DCR)		92,92		
Densidad de Comentarios (DCOM)		21,53		
Modificabilidad		97,91		
Densidad de Ciclos (CYCD)		91,66		
Nombres de Variables con Sentido (DNCS)		100		
Capacidad de ser probado		98,5		
Densidad de Pruebas (UTD)		98,5		
Reusabilidad		100		
Estructura del Programa (EP)		100		

5 Conclusiones

A partir de la aplicación de la estrategia GOCAME, se pudo medir y evaluar la mantenibilidad de la aplicación multiplataforma desarrollada con Ionic. Para ello, se ha invertido un esfuerzo considerable en la definición de las métricas e indicadores propuestos y en el proceso de medición manual, ya que no se utilizaron herramientas automatizadas para el análisis estático del código fuente. Como se mencionó anteriormente, la medición y evaluación se realizó en base a un grupo de cuatro archivos que contienen la mayor parte del código del proyecto Ionic.

Como se mencionó en el marco teórico, el desarrollo híbrido de las aplicaciones permite, utilizando un único *framework* (en este caso Ionic), generar aplicaciones móviles para diversos SO móviles. Durante el proceso de desarrollo, se utilizó el mismo código fuente para la liberación de aplicaciones compatibles con las plataformas Android y iOS. Al compilar la aplicación para cada plataforma, los componentes de Ionic se adaptaron estéticamente a las reglas de cada una. Al finalizar la compilación se obtuvieron los ejecutables. Esto se realizó cada vez que se lanzaba una nueva versión.

De acuerdo con las mediciones efectuadas, la capacidad de mantenimiento de la aplicación multiplataforma “Mi Día en la UNSE” fue evaluada usando el indicador global NCM, igual a 87,84%. Este valor indica que la capacidad de mantenimiento es satisfactoria, ya que supera el valor establecido como objetivo, 50% (apartado 4.3).

Teniendo en cuenta que el valor del NCM representa la medición de un único desarrollo individual (por cada plataforma), el coste y tiempo de desarrollo implicaría

el aumento del esfuerzo de mantenimiento dos veces más, ya que se debería de disponer de los recursos necesarios para programar en entornos específicos de cada plataforma. Por lo tanto, si se requiere que el proceso de mantenimiento sea corto y menos costoso, entonces lo mejor es desarrollar una aplicación multiplataforma híbrida.

En síntesis, si bien este estudio recae sobre el desarrollo de una única aplicación ("Mi día en la UNSE"), el resultado obtenido conduce a la siguiente afirmación**: el desarrollo híbrido usando Ionic permite obtener aplicaciones móviles multiplataforma con capacidad de mantenimiento satisfactorio.**

Asimismo, se observaron algunas desventajas en cuanto el uso del *framework* Ionic, entre las que se destaca la dificultad para trabajar con bibliotecas nativas que requieren configuraciones diferentes para cada SO móvil. En este caso, como solución se optó por dejar de lado bibliotecas de terceros que no ofrezcan compatibilidad con los SO móviles trabajados en este proyecto. Se utilizaron bibliotecas estándares del repositorio de Ionic.

Referencias

1. Delía, L. Trabajo final de especialista en Ingeniería Web: *Desarrollo de aplicaciones móviles multiplataforma.* Facultad de Informática, Universidad Nacional de la Plata. 2017.
2. International Standards Organization. ISO 25000. https://iso25000.com/index.php/normas-iso-25000 [Consultado 20/08/2018].
3. Becker P., Lew P., Olsina L.: *Strategy to Improve Quality for Software Applications: A Process View.* 2011.
4. Herrera, S., Fennema, C., Morales, M., Maldonado, M., Palavecino, R., Rosenzvaig, F., Macedo, A., Suárez, C., Pellicer, R., Villavicencio, R. *Sistemas móviles multiplataforma con realidad aumentada. Aplicaciones en educación y salud*. XXI Workshop de Investigadores en Ciencias de la Computación. ISBN: 978-987-3619-27-4. San Juan. Disponible en: http://sedici.unlp.edu.ar/handle/10915/77150. 2019.
5. Santiago, R. et al. *Mobile learning: nuevas realidades en el aula*. Grupo Océano. pp. 8-26-27, 22-29. ISBN 9788449451454. 2015.
6. Statcounter, Globalstats, http://gs.statcounter.com/os-market-share/mobile/worldwide (2019)
7. Delía, L. N., L. Galdamez, P. Corbalán, P. Pesado and P.Thomas. *Approaches to Mobile Application Development: Comparative Performance Analysis*. Computing Conference 2018 18-20 July 2018 | London, UK. 2018.
8. EllisLab, CodeIgniter, https://codeigniter.com. 2002.
9. Symfony SAS, Symfony, https://symfony.com. 2019.
10. Hernández, M. T. Symfony Framework. Desarrollo Rápido de Aplicaciones Web. IT Campus Academy. 2015.
11. Taylor Otwell, Laravel, https://laravel.com. 2019.
12. PhoneGap. https://www.phonegap.com [Consultado 01/07/2018]. 2018.
13. Sperry, B. Lynch, M. Ionic Framework. https://ionicframework.com/docs/v2/getting-started/concepts. 2012.
14. Xamarin. (2018). Sitio Web: https://www.xamarin.com. [Consultado 11/08/2018].
15. Abrahamsson, P., Hanhineva, A., Hulkko, H., Ihme, T., Jäälinoja, J., Korkala, M., Koskela, J., Kyllönen, P., Salo, O. *Mobile D: An Agile Approach for Mobile Application Development.* OOPSLA 2004, Canadá. Online: http://agile.vtt.fi/mobiled.html. 2004.
16. Hernández Sampieri, R. *Metodología de la Investigación*. México. Mc Graw-Hill. Quinta edición. ISBN: 978-607-15-0291-9. 2010.