UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Universidad Internacional de La Rioja (UNIR)

Escuela de Ingeniería

Máster universitario en Dirección e Ingeniería de Sitios Web

RAMA INVESTIGACIÓN

MCPDev: Metodología para la reutilización de la experiencia web a los desarrollos multiplataforma

Trabajo Fin de Máster

presentado por: Rodríguez Gómez, Rafael Eduardo

Director/a: Boubeta Puig, Juan

Ciudad: Buenos Aires, Argentina
Fecha: 26 de julio de 2018

Dedicatoria

Quiero agradecer a mi esposa por reír a mi lado en los momentos gratos y apoyarme de forma constante y paciente en los momentos más oscuros y difíciles de la investigación, a mis padres, mis colegas Juan Boubeta (director de TFM) y Nadia Gámez (directora de la maestría), mi tutora Mabel Campos que me ha acompañado durante toda esta aventura que ha sido mi primera maestría en la UNIR y en general quiero agradecer a todos los profesores y personas involucradas en este largo proceso.

Unas líneas de dedicatoria pueden resultar poco agradecimiento para tanto que he recibido de tantas personas que han estado involucradas en mi vida personal y profesional y gracias a las cuales esta investigación es una realidad. Aun así, ¡Gracias totales!

Resumen

Cada día son más las plataformas en las cuales un aplicativo de software debe tener presencia para poder llegar a sus consumidores objetivo. Dichas plataformas usualmente trabajan con distintos lenguajes de programación y sus usuarios esperan diferentes experiencias de las mismas. Desarrollar aplicaciones independientes para cada una de estas plataformas representa un incremento considerable, tanto en el presupuesto, como en la complejidad de mantenimiento de la plataforma. Los *frameworks* de desarrollo multiplataforma permiten compartir una cantidad significativa de código y lógica entre aplicaciones orientadas a distintas plataformas haciendo uso de un único lenguaje de programación. Aun así los resultados no siempre son satisfactorios por distintas razones, siendo una de estas la falta de claridad en los lineamientos a seguir a la hora de la elección de las tecnologías a utilizar y en el cómo utilizarlas. En este Trabajo Fin de Master (TFM) desarrollamos una metodología ágil denominada MCPDev que establece un *framework* adaptable basado en la modularidad y reutilización de lógica para cualquier proyecto y equipo que se embarque en los desarrollos multiplataforma, brindando nociones básicas acerca del ambiente, conceptos a tener en cuenta y pasos a seguir. El objetivo principal es lograr más tareas, en menor tiempo, con menores costos y manteniendo la calidad y consistencia. Esta metodología se ha validado a través de un caso de estudio que utilizó una pila de desarrollo basada en *JavaScript* sobre *Node.js*, *React*, *React Native* y *Electronjs*. Los hallazgos al aplicar la metodología y compararla con aplicaciones nativas para las plataformas indican una reducción considerable en las líneas de código escritas y los lenguajes de programación y *frameworks* utilizados, a la vez que se ha obtenido un producto difícil de diferenciar de una aplicación nativa hasta para usuarios avanzados.

Palabras Clave: Modularidad, Reusabilidad, Web, Móvil, Multiplataforma, Metodología Ágil.

Abstract

Every day there are more platforms in which a software application must have presence to reach their target consumers. Those platforms usually work with different languages and users expect different experiences from them. Developing independent applications for each platform represents a considerable increase in both budget and maintenance complexity. Cross-platform frameworks allow sharing a significant code and logic base between applications targeted for different platforms while using a single programming language. Even so, the results are not always satisfactory for different reasons, being one of them the lack of clear guidelines to follow for choosing and using the needed technologies. In this master´s degree project we developed an agile methodology called MCPDev that establishes an adaptable framework based on modularity and reuse of logic for any project and any team that takes on crossplatform developments, providing basic notions about the environment, concepts and steps to follow. The goal is to achieve more tasks, in less time, with lower costs and maintaining quality and consistency. For this we use a JavaScript Stack on top of Node.js, React.js, React Native and Electronjs. Our findings suggest that after applying the methodology and compare the resulting applications against native versions for each one of the platforms it resulted in a considerable reduction of code lines, programming languages and technologies implied. All while keeping enough quality and performance in the resulting product to make it difficult for professional programmers to identify if the App was native or cross-platform.

Keywords: Modularity, Reusability, Web, Mobile, Cross-platform, Agile Methodology.

Tabla de Contenidos

Índice de Tablas

Índice de Figuras

Índice de Anexos

1 Introducción

1.1 Motivación

La tecnología, como cualquier otro ámbito de la vida, evoluciona a pasos agigantados. Esta naturaleza cambiante ha traído consigo el nacimiento de nuevas plataformas (como lo son los *Smart-TV* o los *Smartphone*), otras han visto como el paso del tiempo las ha transformado en su totalidad (como la Web y los desarrollos *Standalone*) y, a su vez, todo esto ha conllevado a un cambio en la forma en la que nos relacionamos con la tecnología, con nuestros iguales y hasta con nuestro entorno (Internet de las Cosas). En la actualidad es más común ver a un joven leyendo las noticias en su teléfono de camino al trabajo, que en un periódico impreso. Por lo tanto, desde la posición de una empresa, su pobre o inexistencia en alguno de estos medios puede significar su fracaso.

A día de hoy las principales plataformas en el mercado son los sistemas operativos (OS por sus siglas en ingles) de ordenadores personales (Windows y Mac OS), los sistemas operativos móviles (iOS y Android) y la Web. Si bien las aplicaciones orientadas para la web pueden ser utilizadas prácticamente cualquier OS a través de un aplicativo llamado *Browser*, podemos asegurar que las aplicaciones realizadas para estas principales plataformas son mayormente incompatibles entre sí dada la diversidad de tecnologías y lenguajes diferentes involucrados en los desarrollos nativos para cada una de estas plataformas.

A la hora de abordar estos desarrollos para múltiples plataformas dado un aplicativo se pueden tomar distintos caminos:

Web: Realizar una aplicación web y mostrarla a través de un explorador (*browser*) independientemente del sistema operativo. Esta aproximación es bastante común sin embargo, bastante limitante dependiendo del modelo de negocio del proyecto.

Nativo: Este camino se basa en realizar una aplicación para cada plataforma sobre la que se quiera operar a pesar de que en gran medida la lógica a la que estarán sujetas dichas aplicaciones es independiente de la plataforma sobre la que se ejecute. En otras palabras, equipos de desarrollos reinventan la rueda constantemente para distintas plataformas, resultando en mayores tiempos de desarrollo, mayores costos, mayor dificultad en el mantenimiento y, finalmente, pudiendo conllevar a inconsistencias en la experiencia final del usuario.

***Cross-platform*:** La principal idea detrás de este tipo de desarrollo es: desarrollar una vez y desplegar en múltiples plataformas (OS en este caso). Existen distintos tipos de desarrollos *cross-platform* (como se muestra más adelante), pero la falta de estandarización y metodología formal para abordar este tipo de desarrollos puede representar un gran reto. Hay muchas opciones y cada una de ellas pareciera estar muy amarrada a un concepto de tecnología en lugar de a un principio reusable.

En la investigación llevada a cabo en este TFM se profundiza en este último camino para abordar los desarrollos para múltiples plataformas tiendo como motivación:

Realizar más tareas en menor tiempo, con menores costos y manteniendo la calidad y consistencia en el ámbito de los desarrollos multiplataforma.

En concreto se genera una metodología que será llamada *Modular Cross-platform Development Agile Methodology* (MCPDev por sus siglas en inglés) y que establece un *framework* adaptable basado en la modularidad y reutilización de lógica para cualquier proyecto y equipo que se embarque en los desarrollos multiplataforma, brindando nociones básicas acerca del ambiente, conceptos a tener en cuenta y pasos a seguir tanto para la elección de tecnologías como para el desarrollo y ciclo de vida del proyecto.

Para probar dicha metodología se plantea un *framework* con tecnologías actuales como lo son React Native, Node.js, Electronjs, entre otros, y un caso de estudio concreto para, a través de dicho *framework*, probar la validez y utilidad de la metodología.

1.2 Estructura del documento

A continuación se indica la estructura de esta memoria de TFM.

Capítulo 1: Se presenta la motivación de la investigación

Capítulo 2: Se realiza el planteamiento del problema y la metodología de trabajo

Capítulo 3: Estado del arte detallado mediante el contexto de la investigación y el análisis de trabajos relacionados

Capítulo 4: Cuerpo principal de la investigación. Desarrollo de la metodología y el *framework* a utilizar para el caso de estudio

Capítulo 5: Planteamiento, desarrollo y análisis de resultados del caso de estudio

Capítulo 6: Conclusiones y trabajos a futuro

2 Planteamiento del problema

En este capítulo se realiza el planteamiento formal del problema mediante la descripción de los objetivos generales y específicos de la investigación, así como de la metodología de trabajo.

2.1 Objetivo General

El objetivo general de este TFM es el desarrollo de una metodología mediante la cual se puedan reducir los tiempos y costos de desarrollo de software, a la vez que se minimizan las posibles inconsistencias en la experiencia del usuario en aplicaciones implementadas de forma multiplataforma a través de la modularidad y reutilización de lógica.

2.2 Objetivos Específicos

A continuación, se identifican los objetivos específicos de este TFM:

- Analizar el estado del arte alrededor de la temática de desarrollos multiplataforma. Identificar oportunidades de mejora de las aproximaciones actuales para tomarlas como base para dar forma a la nueva metodología (requisitos iniciales).
- Armar un ambiente de desarrollo basado en tecnologías de punta mediante las cuales podamos dar una aplicación práctica y real de la metodología.
- Identificar las personas involucradas en la metodología así como sus roles y responsabilidades en el proceso.
- Identificar los artefactos que serán generados como consecuencia de la aplicación de la metodología.
- Definir la estructura formal de la metodología (etapas y fases).
- Identificar los casos de uso de la metodología.
- Desarrollar un framework para la aplicación de la metodología.
- Evaluar la metodología mediante la aplicación del framework desarrollado a un caso de estudio.
- Presentar un análisis formal de los resultados obtenidos junto con recomendaciones a futuro para la mejora continua de la metodología.

2.3 Metodología de trabajo

Se aplicará una metodología cuantitativa basada en el método científico. En esta, se busca demostrar la validez de la hipótesis planteada a través de la relación numérica entre variables, en otras palabras, se deben identificar y recoger variables medibles las cuales puedan ser analizadas estadísticamente.

La aplicación de dicha metodología sigue el plan de trabajo presentado en la *Tabla 1 – Plan de trabajo.*

	Primer día de la semana	02/04	09/04	16/04	23/04	30/04	07/05	14/05	21/05	28/05	04/06	11/06
Tarea	Descripción / Semanas	1	2	3	4	5	6	7	8	9	10	11
1	**Elementos iniciales de la memoria**											
1.1	Portada											
1.2	Estructura de Capítulos											
1.3	Abstract											
1.4	Introducción											
2	**Planteamiento del problema**											
2.1	Objetivo general											
2.2	Objetivos específicos											
2.3	Motivación											
2.4	Metodología de trabajo											
3	**Estado del arte**											
3.1	Contexto - Investigación teórica (conceptos, patrones y metodologías)											
3.2	Contexto - Investigación tecnológica (tecnologías principales / propuestas)											
3.3	Trabajos relacionados											
4	**Desarrollo de Framework para la metodología propuesta**											
4.1	Identificación de requisitos											
4.2	Descripción de la metodología											
4.3	Usos de la metodología											
4.4	Desarrollo del Framework											
5	**Caso de estudio**											
5.1	Planteamiento del experimento											
5.2	Aplicación de la metodología en el experimento											
5.3	Presentación de los resultados											
5.4	Análisis de datos e interpretación de resultados											
6	**Conclusiones**											
6.1	Conclusiones											
6.2	Trabajos a futuro											
	Redacción de la Memoria											
	Milestones y Entregables											
	Estructura de capítulos											
	Entrega del borrador de TFM											

Tabla 1 – Plan de trabajo[1]

[1] La dedicación semanas/persona varía a lo largo de las semanas, por esta razón algunas actividades más extensas parecieran conllevar menor tiempo que algunas más triviales.

Elementos iniciales de la memoria

Estructura base de la memoria de la investigación, incluirá la temática, el *abstract* y la introducción.

Planteamiento del problema

Descripción formal de la problemática a trabajar. Se definirán los objetivos a realizar en la investigación, así como la motivación y la metodología de trabajo a utilizar.

Estado del arte

Obtención de contexto para el ámbito de la investigación mediante la revisión de conceptos e investigaciones científicas realizadas sobre el mismo. Permite conocer el estado actual en la materia para poder, partiendo de lo ya probado, brindar nuevos aportes a la comunidad.

Desarrollo de *Framework* para la metodología propuesta

Definición formal de la metodología (requisitos, usos, personas involucradas, entregables, fases y pasos) mediante la cual se dará solución al problema planteado y se cumplirán todos los objetivos esperados, y desarrollo de un *Framework* para la aplicación de dicha metodología.

Caso de estudio

Planteamiento de un experimento sobre el cual aplicar el *framework* desarrollado. Mediante el análisis e interpretación de los resultados obtenidos se busca evaluar la metodología.

Conclusiones

Apartado de cierre. Incluirá las conclusiones obtenidas como resultado del proyecto para facilitar su revisión para futuros lectores, así como las recomendaciones a futuro para aquellos que deseen continuar el trabajo acá planteado.

Redacción de la Memoria

Redacción de la memoria, la cual se llevará a cabo a lo largo de toda la investigación a medida que se vayan logrando los diferentes hitos y tareas planteados.

***Milestones* y Entregables**

Diferentes entregables a realizar como resultado de la investigación.

3 Estado del arte

En este capítulo se realiza un estudio detallado del estado del arte involucrado en la temática de la investigación. Dicho estado del arte incluye el análisis de trabajos relacionados así como la contextualización mediante la definición de conceptos importantes para la investigación.

3.1 Contextualización

A continuación se definen conceptos importantes para la investigación.

3.1.1 Arquitecturas

3.1.1.1 Arquitectura de Microservicios

Es una aproximación para el desarrollo de *software* que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP). Cada servicio se encarga de implementar una funcionalidad completa del negocio (por ejemplo, un microservicio puede encargarse en su totalidad del manejo de todo lo involucrado a usuarios como su perfil, autenticación y roles). Cada servicio es desplegado de forma independiente, pudiendo existir un mínimo de administración centralizada independiente a los servicios y pudiendo todo estar programado en distintos lenguajes y usando diferentes tecnologías de almacenamiento de datos [1].

3.1.1.2 Arquitectura Cliente – Servidor

Es un estilo de arquitectura marcado por la separación de intereses, en la cual existe una independencia total entre los proveedores de información o servicios llamados servidores y los consumidores de dichos recursos llamados clientes. De esta forma un proveedor de servicios es indiferente ante quien lo pueda consumir, permitiendo ser reutilizado por múltiples clientes (por ejemplo, clientes de una misma aplicación enfocados en distintos OS). A su vez, un cliente tiene suficiente independencia de los servicios que consume, permitiendo consumir distintas fuentes de forma simultánea [2].

3.1.2 Aplicaciones

3.1.2.1 Aplicativo de Software (clasificación según la plataforma objetivo)

- **Web:** Programa de software ejecutada sobre un explorador o *browser* realizado para cumplir un conjunto determinado de tareas. Puede correr en cualquier dispositivo independiente de su sistema operativo siempre que este posea un *browser*.
- **Standalone:** Programa realizado para realizar un conjunto determinado de tareas en equipos de escritorio. Normalmente las aplicaciones son realizadas para un sistema operativo de escritorio en particular y solo pueden ser ejecutadas sobre este.
- **Móvil:** Programa realizado para realizar un conjunto determinado de tareas en equipos móviles. Normalmente las aplicaciones son realizadas para un sistema operativo móvil en particular y solo pueden ser ejecutadas sobre este.

3.1.2.2 Aplicaciones Cross-platform

Una aplicación *Cross-platform* es aquella que, tiendo un mismo código fuente, puede correr en distintos dispositivos diferenciados bien por su naturaleza (Escritorio, Web y móvil, entre otros) o OS (Windows, Mac, iOS y Android, entre otros). Existen diversidad de categorizaciones para estas aplicaciones, sin embargo, para efectos de la investigación se tomará como base la siguiente:

- **Aplicaciones Cross-platform Nativas:** También conocidas como *Aplicaciones Cross-platform Compiladas* o simplemente como *Aplicaciones Cross-platform*. Si bien cada OS suele tener su propio kit de desarrollo de software (SDK por sus siglas en inglés) y su propia pila de tecnologías (como Java en Android y Swift en iOS) las aplicaciones *Cross-platform* nativas se realizan trabajando sobre algún *framework* que brinda un SDK y pila de tecnologías propio, creando una capa de abstracción sobre la cual los desarrolladores trabajan independientemente del OS objetivo y luego dicho *framework* se encarga de transformar la aplicación realizada por el usuario en aplicaciones nativas para los OS objetivos. Un ejemplo de esto es React Native, el cual permite a los desarrolladores crear un app utilizando el *framework* y luego obtener aplicaciones nativas tanto para Android como para iOS.
- **Aplicaciones híbridas HTML5:** También conocidas simplemente como *Aplicaciones Híbridas*. Son aplicaciones realizadas con tecnologías Web como lo son JavaScript, HTML y CSS y que se apoyan en las vistas web avanzadas que ponen al alcance las aplicaciones nativas de los distintos sistemas operativos para mostrarse y funcionar como si se tratase de una aplicación nativa aunque en realidad es una página web corriendo dentro de una aplicación nativa del dispositivo. Pueden también apoyarse

en *frameworks* como Adobe Cordoba para acceder a funciones propias del SDK del OS del equipo y poder de esta forma cumplir tareas propias de aplicaciones nativas.

3.1.3 Conceptos

3.1.3.1 Servicio Web REST (REST Web Service)

En el estilo arquitectónico REST, tanto datos como funcionalidades se consideran recursos y se accede a ellos utilizando URLs, generalmente en la Web, y está diseñado para usar un protocolo de comunicación sin estado, típicamente HTTP. Se actúa sobre estos recursos utilizando un conjunto de operaciones simples y bien definidas. Las estructuras REST imponen una arquitectura cliente / servidor, en el cual los clientes y servidores intercambian representaciones de recursos mediante el uso de una interfaz y protocolo estandarizados, desasociando los recursos de su representación para que puedan ser accedidos en variedad de formatos estandarizados con XML o JSON [3].

3.1.3.2 Interfaz de programación de aplicaciones Web (Web API por sus siglas en inglés)

Los API son interfaces mediante las cuales un proveedor permite interactuar con sus servicios a distintos clientes. Aplicado al contexto web por lo general son un conjunto de web *services*, (por ejemplo, un conjunto de *web services REST* es llamado un REST API) y en él se especifican todos los recursos (datos y funciones) que un proveedor pone al alcance de los clientes que deseen interactuar con él [4].

3.1.3.3 Modularidad

El concepto de modularidad se usa principalmente para reducir la complejidad, beneficiar la flexibilidad y promover la reusabilidad al dividir un sistema en diversos subcomponentes reusables con independencia entre ellos y de esta forma ocultar la complejidad de cada parte detrás de una abstracción y una interfaz [5].

3.1.4 Tecnologías

3.1.4.1 JavaScript

JavaScript es un lenguaje interpretado principalmente utilizado del lado del cliente que comenzó en Netscape, un navegador web desarrollado en la década de 1990. El lenguaje fue creado para permitir a los desarrolladores web incluir código ejecutable en sus páginas web, de modo que puedan hacer sus páginas web interactivas o realizar tareas simples.

JavaScript es muy útil, puede ser ejecutado en todos y cualquier navegador web *out-of-the-box*. Esto implica que una aplicación de JavaScript puede ser ejecutada en cualquier sistema operativo que disponga de un explorador, mientras que una aplicación de escritorio o móvil se ejecuta solo en la plataforma a la que está dirigida (Windows, Mac OSX, Linux, iPhone, Android). Esto le permite escribir aplicaciones multiplataforma de una manera realmente fácil. El rol de JavaScript también se ha expandido significativamente. Las plataformas como *Node.js* permiten a los desarrolladores ejecutar JavaScript en el lado del servidor. Ahora es posible crear aplicaciones web completas en las que la lógica tanto del lado del cliente como del lado del servidor esté escrita en JavaScript [6].

3.1.4.2 Node.js

Ambiente de ejecución *Open Source* y multiplataforma para JavaScript nacido en el 2009 y construido sobre el motor V8 de JavaScript de Chrome que permite utilizar JavaScript como lenguaje de programación en el lado del servidor. Node.js usa un modelo Entrada/Salida (I/O por sus siglas en inglés) no bloqueante (esto quiere decir que, si bien tiene un único hilo de ejecución para I/O, las peticiones a Node.js no bloquean el mismo y son atendidas por instancias creadas en paralelo) y orientado a eventos (por la misma naturaleza asíncrona del lenguaje de programación de JavaScript) que hacen que Node.js una plataforma ligera y eficiente [7].

Node.js como proyecto dirigido por Node.js *Foundation* y es facilitado por el programa de proyectos colaborativos de Linux *Foundation*.

3.1.4.3 React

React es una librería de JavaScript para crear interfaces de usuario (UI por sus siglas en inglés) creada y mantenida por Facebook y una amplia comunidad de desarrolladores y empresas independientes. Algunas de las principales características de React son: [8]

- **Declarativo:** React hace que sea fácil crear UI interactivas. Se diseñan vistas simples para cada estado en la aplicación, y React actualiza y representa de manera eficiente solo los componentes correctos cuando los datos cambien. Las vistas declarativas hacen que su código sea más predecible, más simple de entender y más sencillo de depurar.
- **Basado en Componentes:** Se basa en componentes encapsulados que administran su propio estado y que posteriormente son compilados para crear interfaces de usuario complejas. Como la lógica de los componentes está escrita en JavaScript en lugar de plantillas, se puede pasar fácilmente datos enriquecidos a través de su aplicación y mantener el estado fuera del DOM.
- **Aprenda una vez, Aplique en cual lugar:** React no hace suposiciones sobre el resto de la pila de tecnologías utilizadas en un proyecto, por lo que se pueden desarrollar nuevas características en React sin reescribir el código existente. React también puede renderizarse en el servidor utilizando Node y generar aplicaciones móviles usando React Native.

3.1.4.4 React Native

React Native es una librería de JavaScript basada en React y mantenida por Facebook que permite crear aplicaciones móviles. El enfoque de React Native radica en la eficiencia del desarrollador en todas las plataformas que le interesan: aprenda una vez, escriba en cualquier lugar. Actualmente React Native suporta las versiones de OS >= Android 4.1 (API 16) y >= iOS 8.0, igualmente Facebook usa React Native en múltiples aplicaciones de producción y continuará invirtiendo en React Native [9].

- **Crear aplicaciones móviles nativas usando JavaScript y React:** React Native permite crear aplicaciones móviles usando solo JavaScript. Utiliza el mismo diseño que React, lo que permite componer una interfaz de usuario móvil rica a partir de componentes declarativos.
- **Una aplicación React Native es una aplicación móvil real:** Con React Native, no se crea una "aplicación web móvil", una "aplicación HTML5" o una "aplicación híbrida", se crea una aplicación móvil real que no se distingue de una aplicación creada con Objective-C, Java o Swift (uno u otro dependiendo del OS). React Native utiliza los mismos bloques de interfaz de usuario fundamentales que las aplicaciones normales de iOS y Android, simplemente se juntan esos bloques usando JavaScript y React.

- **No pierda tiempo recompilando:** React Native acelera el proceso de creación de aplicaciones. En lugar de volver a compilar, se puede recargar la aplicación al instante. Con la opción de *hot-reload* incluso se puede ejecutar un código nuevo mientras se conserva el estado de la aplicación.
- **Use código nativo cuando lo necesite:** React Native puede ser combinado sin problemas con componentes escritos en Objective-C, Java o Swift, manteniendo parte de la aplicación en React Native y parte usando código nativo del OS directamente.

3.1.4.5 Redux

Redux es un contenedor de estado predecible para aplicaciones de JavaScript que se puede utilizar junto con React o con cualquier otra librería de presentación de vistas.

Esta herramienta ayuda a escribir aplicaciones que se comportan de forma coherente, se ejecutan en diferentes entornos (cliente, servidor y nativo) y son fáciles de probar. Además de eso, ofrece una gran experiencia de desarrollador, como la edición de código en vivo combinada con un depurador de viaje en el tiempo.

Redux desarrolla las ideas planteadas por Facebook en la herramienta Flux y a su vez esta basado en 3 principios que lo diferencian de otras implementaciones de dicha herramienta: [10]

- **Única fuente de verdad:** El estado de toda la aplicación es almacenada en un árbol de objetos dentro de un solo *store*.
- **El estado es de solo lectura:** La única forma de modificar el estado es emitir una acción (un objeto que describa lo que sucedió).
- **Los cambios se realizan con funciones puras:** Para especificar cómo se transforma el árbol de estado por las acciones, se escriben funciones *reducers* puras.

3.1.4.6 Electronjs

Electron es un *framework open source* que permite crear aplicaciones de escritorio multiplataforma (es compatible con Mac, Windows y Linux) utilizando tecnologías Web como JavaScript, HTML y CSS. Se basa en Node.js y Chromium y es utilizado en multitud de aplicaciones comerciales y *open source* de gran impacto [11].

3.1.4.7 Semantic UI

Semantic UI es un *framework* de front-end enfocado en generar interfaces de usuario atractivas, responsive y amigables con el usuario. Semanti UI tiene integración con otros *frameworks* y herramientas para el desarrollo de front-end como es el caso de React [12].

3.1.5 Metodologías

3.1.5.1 Metodologías Ágiles

Según el PMI [13] una metodología es un sistema de prácticas, técnicas, procedimientos y normas utilizados por quienes trabajan en una disciplina. En un enfoque clásico, al aplicar las metodologías se plantea un alcance y las actividades necesarias para cumplir el mismo, controlando todas las áreas del proceso de trabajo y dejando poco si no nulo espacio para cambios o interpretaciones a lo largo del camino para de esta forma asegurar el cumplimiento de los objetivos en el plazo, con los recursos y la calidad estipulada inicialmente en las primeras etapas de aplicación de la metodología.

Ahora bien, dada la naturaleza cambiante de los proyectos (en especial, mas no únicamente, los proyectos de tecnología) muchos equipos se ven en la necesidad de cambiar el alcance, objetivos o prioridades de un proyecto en plena ejecución del mismo y tener una metodología que no pueda adaptarse a dichos cambios puede representar un gran contratiempo. Por esta razón nacen las metodologías ágiles, las cuales buscan cumplir con los siguientes principios: [14]

- *Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.*
- *Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.*
- *Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.*
- *Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.*
- *Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.*
- *El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.*

- *El software funcionando es la medida principal de progreso.*
- *Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.*
- *La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.*
- *La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.*
- *Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.*
- *A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.*

3.1.5.2 PMBoK (Project Management Body of Knowledge)

El cuerpo de conocimiento de la gerencia de proyectos (PMBoK por sus siglas en inglés) es una colección de procesos, guías y buenas prácticas propuestas por el Instituto de Gerencia de Proyectos (PMI por sus siglas en inglés) y que es actualmente aceptado como un estándar en el área de la gestión y gerencia de proyectos. No es una metodología como tal, pero brinda todas las herramientas necesarias para que un equipo formalice una metodología a su medida. El estándar es continuamente revisado y mejorado por el PMI [13].

3.1.5.3 SCRUM

Scrum es una metodología ágil para completar proyectos complejos. Scrum fue pensado originalmente para proyectos de desarrollo de software, pero funciona bien para cualquier ámbito de trabajo complejo e innovador. El framework planteado para la aplicación de Scrum es realmente sencillo y consta principalmente de los siguientes pasos: [15]

- Un product owner (representante del negocio) crea y prioriza una lista de deseos / actividades llamada product backlog (almacen de producto).
- Durante el *sprint planning* (reunión para la planificación del *sprint*), el equipo toma una pequeña parte de los elementos en el tope de la lista de deseos, que ahora llamaremos *sprint backlog*, y decide cómo implementar esas piezas.
- El equipo cuenta con una ventana finita de tiempo llamado *sprint* (generalmente de dos a cuatro semanas) para completar su trabajo, pero se reúne cada día para evaluar su progreso (*daily Scrum*).

- El ScrumMaster es la persona encargada de mantener al equipo enfocada en su objetivo a lo largo del proceso de trabajo.
- Cada *sprint* resulta en un trabajo listo para entregarse.
- Al final de cada *sprint* tiene lugar una revisión del sprint (evaluación del trabajo realizado en el mismo) y una retrospectiva.
- En lo que finaliza un *sprint* empieza inmediatamente otro repitiendo todo el proceso hasta que se completen todos los ítems en el *product backlog*, se termine el presupuesto o se llegue a una fecha de cierre. La elección de cuál de estos hitos marca el fin del trabajo es totalmente específico al proyecto, pero, sin importar la razón elegida, Scrum se asegura de culminar y entregar el trabajo con valor para el momento en el que el proyecto se detiene.

3.1.5.4 Kanban

Kanban es un framework popular utilizado para implementar el desarrollo de software ágil. Requiere la capacidad de comunicación en tiempo real y la total transparencia del trabajo. Los elementos de trabajo se representan visualmente como *Kanban cards* (tarjetas) en un *Kanban board* (tablero), lo que permite a los miembros del equipo ver el estado de cada pieza de trabajo en cualquier momento. Si bien el uso de Kanban es prominente entre los equipos de desarrollo de software ágiles de hoy en día, sus inicios se remontan a las plantas de Toyota de finales de la década de 1940 [16] (véase la *Figura 1).*

- **Kanban boards:** En kanban el trabajo gira en torno al *kanban board* (normalmente virtual por su trazabilidad, sencillez en la colaboración y accesibilidad desde múltiples ubicaciones), una herramienta que se utiliza para visualizar y optimizar el flujo de trabajo entre el equipo. Un *kanban board* básico tiene un flujo de trabajo de tres pasos: Por Hacer, En Progreso y Hecho. Sin embargo, dependiendo del tamaño, la estructura y los objetivos del equipo, el flujo de trabajo se puede hacer mas completo para cumplir con los requisitos propios del equipo / proyecto en particular.
- **Kanban cards:** En japonés, Kanban se traduce literalmente como "señal visual". Para los equipos que utilizan Kanban, cada elemento de trabajo se representa como una tarjeta en el tablero. El objetivo principal de esto es permitir a los miembros del equipo rastrear el progreso del trabajo a través de su flujo de una manera muy visual. Cada tarjeta presenta información crítica sobre un elemento de trabajo en particular, brindando al equipo una visión completa sobre quién es responsable de ese elemento, una breve descripción del trabajo esperado, cuánto tiempo se estima para su realización, entre otros, para que de esta forma cualquier miembro del equipo

pueda ver el estado de cada elemento de trabajo en un punto dado en el tiempo, así como todos los detalles asociados, asegurando un mayor enfoque, trazabilidad completa e identificación rápida de bloqueadores y dependencias.

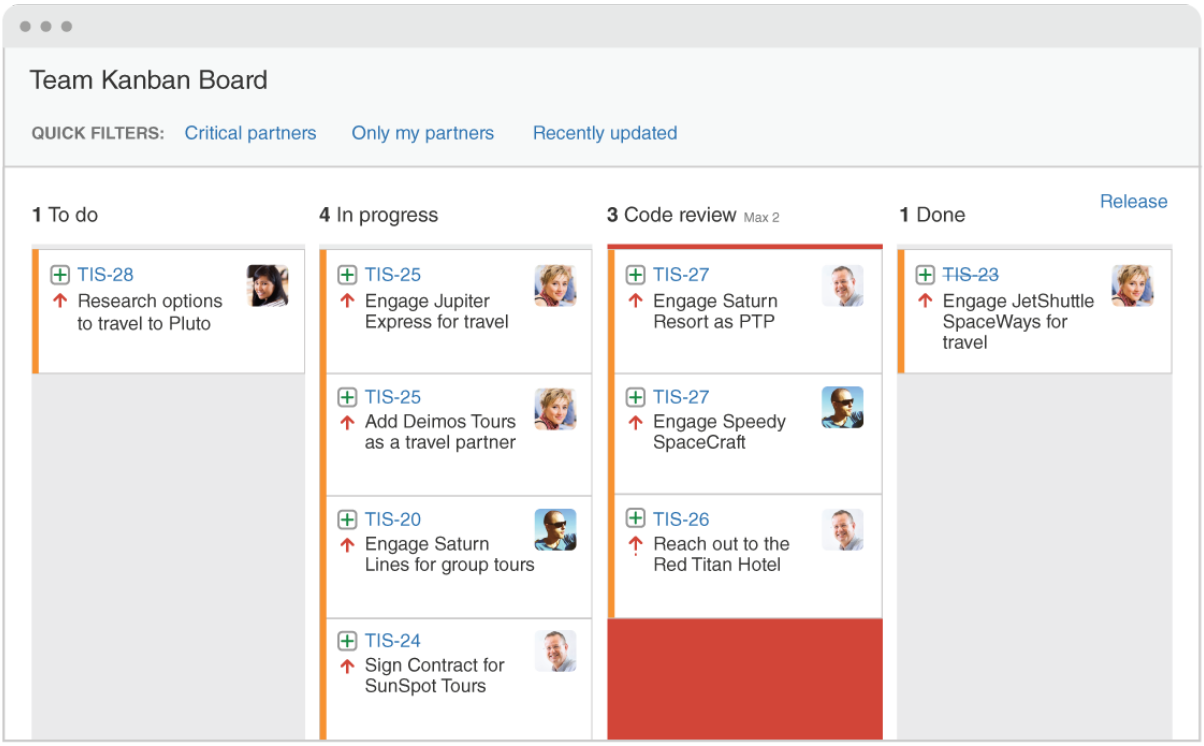

Figura 1 - Kanban board example [16]

3.1.6 Herramientas

3.1.6.1 Bitbucket

Bitbucket es un repositorio de control de versiones en forma de plataforma web creado por Atlassian y que trabaja tanto con Git como con Mercurial como sistemas de control de revisiones [17].

3.1.6.2 Trello

Trello es una aplicación web para la gestión de proyectos inspirada en Kanban y que basa su funcionamiento en tableros al igual que dicha metodología. Fue creada en 2011 y actualmente es propiedad de Atlassian [18].

3.1.6.3 Google Drive

Google Drive es un servicio de almacén y sincronización creado por Google. Drive permite a sus usuario almacenar documentos en la plataforma web para accederlos en distintos dispositivos, compartirlos con quien lo deseen y trabajarlos de forma colaborativa a través de otras herramientas de Google como Docs [19].

3.1.6.4 Visual Studio Code

Visual Studio Code (VSCode) es un editor de código creado por Microsft altamente personalizable y estable. Incluye varias herramientas *out-of-the-box*, sin embargo, su principal ventaja es su gran cantidad de plug-ins que son generalmente desarrollados por la gran comunidad que cuenta. VSCode es gratuito y código abierto [20].

3.2 Trabajos relacionados

En esta sección se describen los trabajos relacionados y se realiza una comparativa entre ellos, clasificando los trabajos en función de las características más relevantes para la investigación actual: Desarrollo multiplataforma, Framework Fullstack, Modularidad y Reusabilidad del código y lógica, y Metodología de desarrollo de software (véase *Tabla 2).*

Paper	Desarrollo multiplataforma	Framework Fullstack	Modularidad y Reusabilidad del código y lógica	Metodología de desarrollo de software
[21]	Solo en el ámbito móvil	No, solo se discuten herramientas de Frontend móvil	No	No
[22]	Solo en el ámbito móvil	No, solo se discuten herramientas de Frontend móvil	No	No
[23]	Solo en el ámbito móvil	No, solo se discuten herramientas de Frontend móvil	No	No
[24]	Solo en el ámbito móvil	No, solo se discuten herramientas de Frontend móvil	No	No
[25]	Si	No, solo se discuten herramientas de Frontend móvil	No	No

Tabla 2 - Trabajos Relacionados

3.2.1 Desarrollo multiplataforma

Si bien la gran mayoría de los *papers* tocan la temática de desarrollos multiplataforma [21] [22] [23] [24] [25], la gran mayoría los enfoca en un ámbito en particular. Dada la relevancia que ha tomado en los últimos años, uno de los ámbitos más estudiados recientemente es el móvil, viendo los desarrollos multiplataforma como aquellos que puedan correr en 2 o más OS (iOS y Android, entre otros). Algún *paper* [25] va un poco más allá e intenta abordar desarrollos entre distintos ámbitos (Escritorio, móvil, web, entre otros) analizando tecnologías como Java, pero sin entrar mucho en materia y enfocándose principalmente en el ámbito móvil.

3.2.2 Framework Fullstack

Los desarrollos de aplicaciones de software conllevan una lógica más allá de la presentación de datos a los clientes. Las arquitecturas cliente-servidor hacen una clara distinción de roles en este aspecto. La mayoría de las investigaciones multiplataforma no ignoran la existencia de estos proveedores de servicios, sin embargo, suelen enfocarse principalmente en el aspecto del frontend [21] [22] [23] [24] [25]. Para apoyar un desarrollo realmente multiplataforma hace falta sentar ciertas bases o principios de arquitectura que deben estar presentes en estos proveedores de servicios.

3.2.3 Modularidad y Reusabilidad del código y lógica

Normalmente las investigaciones realizadas hasta el momento sobre el tema de los desarrollos multiplataforma se enfocan en el que usar (*frameworks* y herramientas) y no tanto en el cómo usarlo de la mejor forma. La utilidad de una herramienta dependerá el uso que se le dé. De esta forma se toca constantemente los beneficios de utilizar un *framework* como React Native [21] [22] para mediante un único código base generar aplicaciones tanto para Android como para iOS lo cual reduce significativamente las horas de desarrollo y mantenimiento necesarias de haber realizado 2 aplicaciones nativas, pero no se discuten los beneficios de apoyarse en la estructura modular intrínseca en React como librería para de esta forma reducir la complejidad interna del desarrollo de la aplicación multiplataforma.

3.2.4 Metodología de desarrollo de software

La mayoría de los *papers* e investigaciones sobre el ámbito de los desarrollos *cross-platform* [21] [22] [23] [24] [25] toman un enfoque principalmente práctico, enfocándose en tecnologías, herramientas y paradigmas puntuales más que en el desarrollo de una metodología formal que permita aplicar dichos conocimientos en proceso bien definido. Esto deja una brecha importante entre el qué hacer y el cómo hacerlo a la hora de abordar un desarrollo multiplataforma.

4 Desarrollo de un Framework para la metodología propuesta

En este capítulo representa el cuerpo principal de la investigación. En él se realiza el desarrollo de la metodología MCPDev así como el *framework* mediante el cual se aplicarán los principios de la metodología en el caso de estudio.

4.1 Identificación de requisitos

Ni los procesos del PMBoK, la metodología ágil SCRUM o el método Kanban fue pensado únicamente para ser utilizado en proyectos multiplataforma (aun cuando sean muy utilizados en ellos) por lo que no dan prácticas, técnicas, procedimientos o normas para afrontar distintos retos muy propios de dichos proyectos.

Inspirados en ellos se propone una nueva metodología para ser utilizada en proyectos de naturaleza multiplataforma y que cumpla con los siguientes requisitos:

- **Metodología de desarrollo de Software**

A diferencia de la mayoría de las investigaciones sobre el área, se espera una metodología de desarrollo de software completa que abarque no tan solo aspectos técnicos y de implementación, sino todo el ciclo de vida del proyecto desde su concepción hasta su entrega, pasando por el desarrollo y la administración.

- **Independencia tecnológica**

Se busca una solución independiente a las tecnologías con las cuales finalmente se aplique y que se base únicamente en principios reusables como teorías y conceptos.

- **Aplicable para los desarrollos multiplataforma**

La metodología debe dar guías claras y aplicables para abordar los desarrollos multiplataforma. De igual forma debe ser aplicable independientemente de las plataformas elegidas, facilitando la elección de las herramientas correctas para el desarrollo.

- **Enfoque modular y reusable**

La metodología gira en torno a la reducción de lapsos y costos manteniendo la calidad y consistencia en el ámbito de los desarrollos multiplataforma. Este concepto no se limita a la

reutilización de una misma base de código para desarrollar aplicaciones para distintas plataformas, sino que también puede ser aplicado a cada plataforma de forma independiente. La modularidad brinda un grado de abstracción que permite la reutilización de código dentro de un proyecto, evitando invertir tiempo desarrollando una misma tarea múltiples veces.

4.2 Descripción de la metodología

La metodología ágil para el desarrollo modular multiplataforma (Modular *Cross-platform Development Agile Methodology*) o MCPDev por sus siglas en inglés, es el nombre de la metodología a plantear en esta investigación.

4.2.1 Roles

Antes de hablar de la metodología es prudente hablar de los *stakeholders* que participarán en el proceso (véase *Tabla 3).*

Rol	Descrición
Product Owner (PO por sus siglas en ingles)	Representante de la institución cliente. Se encarga de transmitir las necesidades del cliente y validar la correctitud de las implementaciones y decisiones del equipo. Concepto tomado de la metodología ágil Scrum.
Product Manager (PM por sus siglas en ingles)	Líder y facilitador del proyecto. Es aquel encargado de velar por la correcta implementación de la metodología. Se encarga de mantener al equipo enfocado en su objetivo. Similar al concepto de *ScrumMaster* existente en la metodología ágil Scrum.
Diseñador	Diseñar la interfaz del sistema en función de la navegación, los objetivos, requerimientos e información.
Líder técnico	Establecer la arquitectura y módulos del sistema. Velar por que se cumplan las buenas prácticas de la empresa en cuanto a

	código y arquitectura dentro del proyecto.
Desarrollador	Programador que realizará el producto. También estará encargado de las revisiones del código de sus iguales.
Responsable de infraestructura	Responsable de las labores de infraestructura y despliegue que puedan existir en el proyecto.
Responsable de datos	Responsable por la creación y mantenimiento de la representación de datos y lógica de negocio dentro de la plataforma.
Representante de usuarios	Representante de los consumidores. Persona encargada de representar las necesidades, dudas, inquietudes y patrones de uso de los usuarios finales del sistema.
Analista de calidad (QA por sus siglas en inglés)	Analista de la calidad. Realizará pruebas manuales y pruebas automatizadas sobre el producto. De igual forma evaluará la implementación de los diseños iniciales en el producto final.

Tabla 3 - Roles involucrados

Los roles no hacen referencia directa a personas en el equipo, en otras palabras, un integrante del equipo puede tener más de un rol. Lo importante con los roles es entender los perfiles y responsabilidades existentes en la metodología.

4.2.2 Artefactos

Artefactos, sean o no de software, que serán generados, mantenidos o utilizados como consecuencia de la aplicación de la metodología a un proyecto de software.

- **Tablero Kanban:** El objetivo principal es permitir a los miembros del equipo rastrear el progreso del trabajo a través de su flujo de una manera muy visual.

Los elementos de trabajo se representan visualmente como tarjetas Kanban en el tablero Kanban. Cada tarjeta presenta información crítica sobre el elemento de trabajo, asegurando un mayor enfoque, trazabilidad completa e identificación rápida de bloqueadores y dependencias.

Un punto de mucha importancia en la metodología es la estimación de esfuerzos involucrados para cada actividad plasmada en forma de tarjeta Kanban. Dicha estimación dependerá mucho de la experiencia y experticia del equipo, sin embargo, para la actual metodología se basará en la cantidad de horas/persona requeridas por la actividad, de esta forma, una actividad con un puntaje de 8 quiere decir que esta estimado que tome 8 horas en realizarse. Las primeras estimaciones pueden ser un poco erráticas, pero su precisión va mejorando con las siguientes iteraciones y la experiencia conseguida.

Tanto las tajetas como el tablero Kanban son conceptos traídos de la metodología Kanban. El tablero Kanban es una representación 'visual' del backlog del sprint (concepto traído de Scrum). El tablero Kanban debe incluir como mínimo las siguientes columnas:

- Por Hacer: Trabajo por realizar.
- En Progreso: Trabajo en progreso actualmente.
- En Revisión de Código: Trabajo realizado pero en revisión.
- En QA: Trabajo revisado y asignado al responsable de QA.
- Listo: Trabajo listo. Solo los roles de QA, PM y PO pueden mover los tickets a esta columna.

- **Calendario:** A pesar de lo poco restrictivo que es MCPDev en cuanto a los lapsos de trabajo por la misma naturaleza ágil de la metodología, siempre hay ciertas fechas claves o hitos a tener en cuenta. Por esta razón se debe contar con un calendario único y al acceso de todos los involucrados (preferiblemente a través de alguna herramienta web) donde se representen de forma clara y distintiva cualquier hito importante para el equipo así como:
  - Las fechas de inicio y fin (de existir) del proyecto, así como de cada uno de los sprints.
  - Las fechas y horarios de los *Grooming* de cada sprint.
  - Las fechas y horarios del *Planning* de cada sprint.
  - Las fechas y horarios de la Retrospectiva de cada sprint.
  - Las fechas y horarios de los *Code Freeze* de cada sprint.
  - Las fechas y horarios de los *Release* de cada sprint.
  - Los horarios de las reuniones diarias (DSU).

    - Las fechas en caso de vacaciones o ausencias por parte de algún miembro del equipo.
- **Wiki:** Única fuente de la verdad. En el mismo deben incluirse todas las directrices que puedan necesitarse en el equipo. A pesar de ser una metodología ágil, en mayor o menor grado siempre hay estándares que deben seguirse y esta es la mejor forma de asegurarlos. Cabe destacar que dichos estándares y/o directrices pueden cambiar a lo largo del tiempo, por lo que es importante mantener actualizado el wiki. Más allá de las requerimientos en particular del equipo aplicando la metodología, acá deben mantenerse:
    - Herramientas y tecnologías involucradas en el proyecto así como sus versiones.
    - Documentos de interés para el equipo y el proyecto.
    - Plantillas de formularios.
- **Retrospectivas:** Documento formal que se genera al final de cada sprint e incluye qué cosas el equipo piensa se hicieron bien en el sprint actual, qué cosas pudieron hacerse mejor y, finalmente, llegar a acuerdos para tener un mejor desempeño el sprint siguiente.
- **Acta de cierre de Sprint:** Documento formal que incluye información sobre el sprint en cuestión. La estructura y detalles dependen del equipo que esté utilizando la metodología, sin embargo, debe incluir como mínimo las actividades acordadas inicialmente, las actividades realizadas al final del sprint y las actividades que van a arrastrarse para un futuro sprint y el porqué.
- **Plataforma para el manejo de versiones:** Es vital para el proceso de desarrollo de software contar con una herramienta que permita la accesibilidad y edición colaborativa del código del proyecto, así como una trazabilidad en sus versiones. GIT y Mercurial son algunos ejemplos. Además se espera que:
    - Haya una rama de desarrollo en la cual se encuentren todas las funcionalidades culminadas durante un sprint, y una rama maestra donde se encuentre todo el código probado y listo para ser entregado.
    - Cada tarjeta Kanban a trabajar va a ser desarrollada en una nueva rama que se realizará partiendo de la rama de desarrollo y posteriormente al ser completada se unirá a dicha rama a través de un *Pull Request* (PR) que debe ser evaluado por sus pares y aprobado como mínimo por el líder técnico del proyecto.
    - Los PR a la rama maestra se harán de forma controlada por el líder técnico del proyecto y únicamente desde la rama de desarrollo.

- **Código funcional:** Al final de cada sprint se debe producir un entregable de software como consecuencia del trabajo colaborativo, el cual debe incluir el alcance funcional acordado para el sprint correctamente desarrollado y probado. El código siempre debe ser entregado, sin embargo, esta entrega puede estar acompañada por un aplicativo o no dependiendo del proyecto y sus requerimientos.
- **Acta de constitución del Proyecto:** Documento formal normalmente más enfocado a la parte administrativa pero que debe definir, entre otras cosas, acuerdos y compromisos por parte del equipo y del cliente, temática del proyecto, alcance inicial, lapsos de tiempo disponibles (en caso de existir) y cualquier otra entrada necesaria para asegurar el compromiso inicial y éxito a futuro del proyecto.
- **Lecciones aprendidas:** Si bien al final de cada sprint existente se trabaja un acta para mejorar el trabajo de cara a cada siguiente sprint, este documento va más enfocado a los aprendizajes y lecciones del proyecto como un todo para intentar aplicar en proyectos futuros. Es un documento realizado por y para el equipo, no para el cliente, sin embargo, es importante tomar el cuenta la conformidad del mismo con el proceso de desarrollo y el producto final.
- **Acta de cierre del proyecto:** Documento formal que incluye información sobre el proyecto. La estructura y detalles dependen del equipo que esté utilizando la metodología, sin embargo, debe incluir como mínimo las actas de cierre de todos los sprints donde se puede ver reflejado todo el trabajo realizado por el equipo y aprobado por el PO, así como información de la entrega formal del producto final, de la conformidad del cliente y posibles recomendaciones a futuros del producto.

4.2.3 Etapas y Fases

MCPDev es una metodología ágil basada en conceptos planteados en el PMBoK, la metodología ágil SCRUM y el método Kanban pero enfocada principalmente para ser utilizada en proyectos multiplataforma.

Siguiendo lo planteado por el PMBoK, la metodología MCPDev plantea 10 áreas de conocimiento:

- Gestión de la integración

Áreas básicas:

- Gestión del alcance
- Gestión de la programación
- Gestión de los costos
- Gestión de los recursos
- Gestión de la calidad

Áreas de apoyo:

- Gestión de las comunicaciones
- Gestión de los interesados

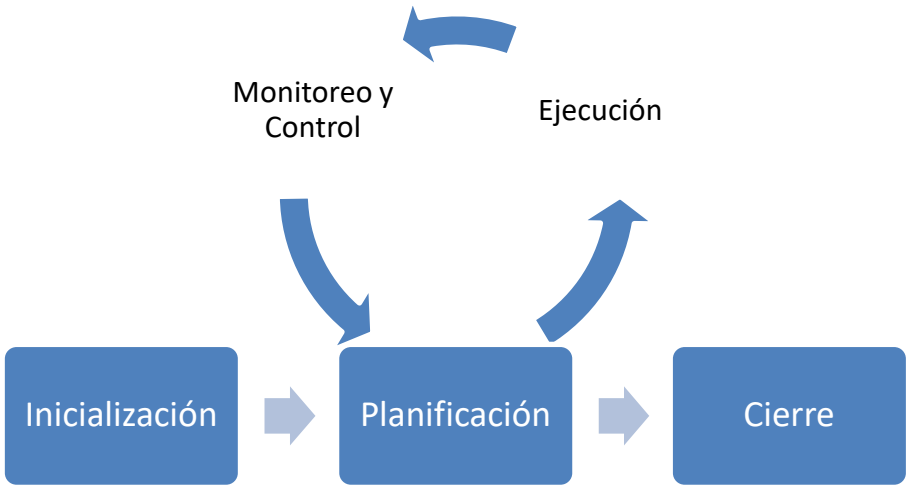

Figura 2 - Ciclo de vida MCPDev

Entrando en detalle sobre la implementación de la metodología y su ciclo de vida, MCPDev es una metodología incremental, que plantea diversos procesos los cuales divide en grupos o fases, teniendo una fase de Inicialización y una de Cierre que se ejecutan una única vez (al principio y final del proyecto respectivamente), y entre ellas genera iteraciones llamadas *Sprint* con las fases de Planificación, Ejecución y Monitoreo y control, las cuales van resultando en entregables y componentes de software funcionales que serán entregados al cliente (véanse *Tablas 4 y 5 e Figura 2).*

Fases / Areas	Inicialización	Planificación	Ejecución
Integración	* Desarrollar el acta de constitución del proyecto. * Crear el tablero kanban. * Crear el calendario. * Crear el wiki. * Crear el formato de las retrospectivas y el formato de acta de cierre de sprint. * Configurar la plataforma de manejo de versiones. * Dar inicio formal al proyecto	* Empezar el Sprint	* Dirigir el trabajo del proyecto
Alcance	* Identificar plataformas objetivo del proyecto. * Elegir tecnologías a utilizar en base a las plataformas objetivo del proyecto y los requerimientos iniciales del mismo. * Identificar las prioridades iniciales del proyecto, elaborar las tarjetas kanban correspondientes, agregarlas al backlog de productos y ordenarlas según las prioridades el negocio. * Grooming inicial	* Recopilar requisitos, elaborar las tarjetas kanban correspondientes, agregarlas al backlog de productos y ordenarlas según las prioridades el negocio. * Grooming	-
Cronograma	* Definir fechas y horas para las diferentes reuniones y actividades necesarias en los sprints.	* Planning	* Poner en marcha el desarrollo del software
Costo	* Generar presupuesto preliminar del proyecto	* Generar costo estimado del Sprint	-
Calidad	-	* Elaborar el plan de pruebas	* Aplicar plan de pruebas
Recursos	* Definir tamaño y perfil del equipo en base a las tecnologías a utilizar	* Establecer el equipo de trabajo necesario para el proyecto * Asignar roles y responsabilidades	-
Comunicaciones	* Comunicar la descripción del proyecto y el presupuesto	* Elaborar el plan de comunicaciones generando calendarios y canales de comunicación para el equipo	* Notificar a los clientes involucrados del inicio del proyecto
Interesados	* Identificar interesados considerando las exigencias del proyecto y las habilidades / disponibilidad del equipo	-	-

Tabla 4 - Procesos de MCPDev. Parte 1 de 2

Fases / Areas	Monitoreo y control	Cierre
Integración	* Actualizar plan del proyecto * Mantener cambios en el wiki y calendario * Demo * Retrospectiva	* Cierre del proyecto * Gestionar lecciones aprendidas para futuros proyectos
Alcance	* Evaluar el cumplimiento del plan del sprint	-
Cronograma	* Evaluar el cumplimiento del plan del sprint	-
Costo	-	-
Calidad	* Evaluar plan de pruebas * Evaluar código en los Pull Request	
Recursos	-	-
Comunicaciones	* Reuniones diarias de seguimiento	* Notificar a los interesados del fin del proyecto
Interesados	-	-

Tabla 5 - Procesos de MCPDev. Parte 2 de 2

4.2.3.1 Fase de Inicialización

- **Duración:** Se espera que la duración sea lo más corta posible, sin embargo, esto puede variar a gran medida por la complejidad y conocimiento previo que se tenga del proyecto, así como las negociaciones iniciales entre el equipo de trabajo y el cliente. Con esto dicho, se espera que esta fase tenga una duración mínima de 1 semana.
- **Acta de constitución del Proyecto**: Definición de formal del documento que da inicio al proyecto.
- **Formatos de documento:** Generar los diferentes formatos a utilizar a lo largo del proyecto, haciendo principal énfasis en los documentos de:
  - Retrospectivas
  - Acta de cierre del sprint
- **Wiki:** Crear, mediante alguna herramienta especializada, este repositorio o fuente única de la verdad. Definir una estructura de fácil acceso y navegación para fomentar el uso de la herramienta.
- **Calendario**: Creación del calendario y configuración inicial del mismo definiendo:
  - Las fechas de inicio y fin (de existir) del proyecto, así como de cada uno de los sprints. Todo sprint debe empezar un lunes.

  - Definir un horario y crear un evento recurrente para los *Grooming*, el *Planning*, la Retrospectiva, el *Code Freeze*, y los DSU de cada sprint.
  - Hacer una carga inicial de las vacaciones o ausencias conocidas de aquellos miembros del equipo que hayan sido definidos hasta este momento.
- **Tablero Kanban:** Crear, mediante alguna herramienta especializada, el tablero Kanban con las secciones requeridas por el negocio.
- **Product Backlog:** Identificar las prioridades iniciales del proyecto y generar en base a ellas las tarjetas Kanban asociadas para tener una fuente suficiente de trabajo con la cual dar inicio a las actividades en el primer sprint.
- **Grooming inicial:** El último viernes de esta fase tendrá lugar el primer *grooming* del proyecto o reunión en la cual se evalúa la complejidad asociada a cada tarjeta de Kanban y se les otorga un puntaje que nos ayudará a controlar la carga de trabajo real (basada en dificultad y tiempo de trabajo y no solo en número de actividades, debido a que esto no es necesariamente representativo del esfuerzo) que será tomada para cada iteración. Dado el posible conocimiento limitado del equipo sobre la temática del proyecto, sobre el modo de trabajo de la metodología o sobre el resto de los integrantes del equipo, este *grooming* es de especial interés y conviene tomarse tiempo para repasar los principios de la metodología y del proyecto antes de dar comienzo formal al *grooming* y posteriormente analizar con gran detalle cada tarea discutida. Este es un concepto traigo del Scrum. Duración 4 horas.
- **Definir equipo de trabajo:** Definir el tamaño y perfil del equipo en base a las tecnologías a utilizar y los requerimientos del proyecto. No hay un tamaño ideal para los equipos, sin embargo existen ciertas recomendaciones para mejorar el funcionamiento del mismo:
  - Una misma persona puede tener varios roles, sin embargo, sería ideal que aquella persona encargada de realizar una actividad no sea la misma a cargo de evaluarla. También tener en cuenta que hacer tareas en paralelo suele afectar la eficiencia.
  - Equipos pequeños son más fáciles de gerenciar y esto facilita la aplicación de la metodología. En caso de tener un alcance muy amplio y requerir gran número de involucrados, es recomendable dividir el proyecto en diversas áreas o “sub-proyectos” y separar la fuerza de trabajo en varios equipos que trabajen de forma interrelacionada pero cada uno enfocado en un área en particular.
  - Diversos equipos pueden tener objetivos en común y trabajar de forma interrelacionada para lograrlos, sin embargo, siempre se debe mantener la independencia de cada unidad de trabajo (equipo). Cada equipo debe

encargarse de la aplicación de la metodología y selección de trabajo en base a sus prioridades.

- **Generar el presupuesto preliminar del proyecto:** Dada la naturaleza de la metodología, el alcance puede variar a lo largo del tiempo y por tanto también los esfuerzos involucrados. Esto modificará directamente los costos. Con esto dicho, es importante dar a todos los involucrados (equipo y cliente) una aproximación inicial a los esfuerzos requeridos por los requerimientos iniciales del proyecto. En el caso del cliente este "esfuerzo requerido" se muestra en forma de un presupuesto preliminar.
- **Configurar la plataforma para el manejo de versiones.**
- **Selección de la base tecnológica a utilizar.**

4.2.3.2 Iteraciones o Sprints (Fases de Planificación, Ejecución y Monitoreo y control)

- **Duración:** Tienen duración de 2 a 4 semanas empezando siempre los días lunes. Se da preferencia a la duración más corta posible.
- **Daily Stand Up** (DSU)**:** Se tiene una reunión diaria de corta duración en la que todos los miembros del equipo hacen una pausa activa, se colocan de pie y comparten su estatus del progreso y participa cualquier duda o elemento que le esté bloqueando su trabajo para que la persona encargada de dicho aspecto (dependerá de la naturaleza de la duda o elemento) le pueda brindar la ayuda necesaria. Duración: 15 o 30 minutos máximo. Es un concepto tomado del *daily Scrum* de la metodología Scrum.
- **Backlog de productos:** A lo largo de todo el ciclo de vida del proyecto, el *Product Owner* estará encargado recopilar los requisitos del negocio, elaborar las tarjetas Kanban correspondientes (siempre intentando hacerlas lo más unitarias posibles), agregarlas al backlog de productos y ordenarlas según las prioridades el negocio. El concepto del backlog de productos es tomado de la metodología Scrum.
- **Planning:** El primer lunes del Sprint ocurre el *Planning* o reunión en la cual el equipo toma una pequeña parte de las tarjetas Kanban del tope de backlog de productos para ser trabajadas en el Sprint actual. Dichas tarjetas son repartidas entre el equipo de desarrollo dependiendo de su naturaleza. El concepto de *Planning* es tomado de la metodología Scrum. Duración: 1 hora.
- **Grooming:** Cada viernes se realiza el *Grooming* del sprint. Las tarjetas son evaluadas por orden de prioridad desde el principio del product backlog. Duración: 1 hora cada viernes.

- **Code Freeze:** El último miércoles de cada sprint toma lugar el *Code Freeze*, o momento máximo para entregar los avances del Sprint por parte del equipo de desarrollo. A partir de este momento la rama de desarrollo queda estable para que el equipo de pruebas pueda terminar de realizar todas las pruebas (manuales y/o automáticas) restantes.
- **Release:** El último viernes de cada sprint se pasa todo el código previamente probado a fondo de la rama de desarrollo a la maestra y se vuelven a ejecutar todas las pruebas para asegurar que el nuevo código no produjo cambios o daños en alguna otra parte del sistema.
- **Cierre de sprint:** De igual forma, el último viernes de cada sprint tendrá lugar una revisión del sprint donde se revisará el acta de cierre de sprint (previamente realizada por el PM durante el mismo día), se presentará un demo para mostrar los avances conseguidos y tendrá lugar la retrospectiva. El documento para la retrospectiva debe ponerse a la disposición del equipo como mínimo un par de días antes de la reunión para que coloquen sus ideas en los apartados de “cosas hechas de manera correcta” y “cosas que se pudieron hacer mejor” de forma anónima. El apartado final de la retrospectiva “acuerdos para el siguiente sprint” será llenado de forma conjunta en la reunión. Tanto las retrospectivas como las actas de cierre de sprint deben ser guardadas en el wiki. Duración: 1 hora.
- **Adaptación a cambios:** Como toda metodología ágil, el alcance del proyecto, necesidades y requerimientos pueden cambiar en cualquier momento durante el ciclo de vida del mismo y la metodología reaccionará a estos cambios. Con esto dicho, las actividades a realizar dentro de un sprint no pueden cambiar libremente. Una vez empezado un sprint las tarjetas Kanban pueden cancelarse si luego de una investigación pertinente se determina que el trabajo ya no es necesario, pueden sufrir pequeñas modificaciones, pero no pueden sufrir una gran modificación o agregarse nuevas tarjetas. En esta caso deben agregarle al backlog de productos y serán tomados en cuenta para un sprint futuro.
- **Manejo de costos:** El manejo de costos (pago de recursos, equipos, entre otros) se manejara en base a sprint para de esta forma darle mayor gestión sistemática del Retorno de Inversión (ROI).
- **Iteración:** En lo que finaliza un sprint empieza inmediatamente otro repitiendo todo el proceso hasta que se completen todos los ítems en el backlog de productos, se termine el presupuesto o se llegue a una fecha de cierre. Al final de cada sprint las tarjetas Kanban que no hayan podido ser culminadas deberán ser arrastradas hasta el siguiente sprint, afectando la carga de nuevo trabajo que se pueda aceptar.

- **Calendario:** En todo momento el equipo debe mantener el calendario actualizado con toda la información relevante.
- **Wiki:** En todo momento el equipo debe mantener el wiki actualizado con toda la información relevante.
- **Plan de pruebas:** A medida que el equipo de desarrollo va trabajando en las funcionalidades para un sprint, el equipo de QA va trabajan en paralelo en un plan de pruebas que aplicarle a dichas funcionalidades y mediante las cuales se pueda asegurar la correctitud y buen funcionamiento de la implementación.

4.2.3.3 Fase de Cierre

- **Duración:** Se espera que la duración sea lo más corta posible, sin embargo, esto puede variar a gran medida por la complejidad del proyecto, o los trámites previamente definidos por el equipo. Con esto dicho, se espera que esta fase tenga una duración mínima de 1 semana.
- **Acta de cierre del Proyecto**: Definición de formal del documento que da fin al proyecto.
- **Lecciones aprendidas:** Se redacta el documento de lecciones aprendidas durante el proyecto para ser utilizadas por el equipo / empresa en una futura implementación de la metodología.
- **Transferencia de conocimiento**: Se hace entrega formal de todo el material pertinente en posesión del equipo. Entre dicho material se encuentra:
    - Acta de cierre del proyecto
    - Propiedad de los repositorios del proyecto
- **Fin de actividades:** Se dan por finalizadas las actividades del proyecto y:
    - Se cierra el tablero Kaban
    - Se cierra el calendario
    - Se cierra el wiki
    - Se libera el equipo de trabajo

4.2.4 Consideraciones tecnológicas: Modularidad y reutilización del código

4.2.4.1 Fase de Inicialización

En la fase de Inicialización ocurren algunas de las actividades más diferenciables de la metodología.

No hay un lenguaje o *framework* perfecto, solo hay opciones más capacitadas para afrontar problemas particulares. Por esta razón, MCPDev asegura independencia de tecnologías por lo que la elección de las mismas queda por parte del equipo.

La base elegida condicionará a gran medida el proyecto y su capacidad de reaccionar ante cambios. Una vez elegida se pueden agregar nuevas herramientas a medida que aparezca la necesidad, pero esta base no cambiará una vez empezado el desarrollo, por eso es importante hacer un análisis inicial y ordenado de los siguientes puntos:

- **Plataformas objetivo:** *¿Cuáles son las plataformas objetivo del proyecto?* Una vez definidas las plataformas se buscan *frameworks* que permitan trabajar con todas las plataformas objetivo. En este apartado no se toma en cuenta otras características de los *frameworks*, solo que permitan desarrollo en las múltiples plataformas.
- **Naturaleza y requerimientos del proyecto:** *¿Qué se espera del proyecto?* Cada *framework* tiene sus características propias y esto afecta directamente aquellas aplicaciones desarrolladas con ellos. De igual forma cada proyecto tiene sus características propias por lo que deben analizarse los requerimientos iniciales del proyecto y asegurarse de que las tecnologías analizadas cumplan con ellas.
    - **Aplicaciones Cross-platform Nativas:** Suelen resultar en aplicaciones nativas con buen desempeño general a cambio de mayor complejidad en el desarrollo. Por tanto la experiencia de usuario es nativa y suele diferir por plataforma.
    - **Aplicaciones híbridas HTML5:** Suelen sacrificar un poco la experiencia de usuario nativa y el desempeño en funcionalidades avanzadas a cambio de un menor tiempo de desarrollo y una base compartida de código mayor. Muchos video juegos y aplicaciones sencillas suelen optar por esta opción, ya que no brindan al usuario experiencia nativa o funcionalidades avanzadas.
- **Reutilización de código:** *¿Qué tanto permite el framework tener una base compartida de código para múltiples plataformas?* Cada *framework* multiplataforma tiene sus peculiaridades en este apartado y estas pueden ir desde compartir poco más que el lenguaje de programación hasta tener un código único y desplegar la aplicación en distintas plataformas sin ningún cambio extra. En este apartado se filtrarán las herramientas para mantener solo aquellas que permitan el mayor porcentaje de reutilización del código.

- **Áreas de conocimiento del equipo de desarrollo:** *¿Qué tecnologías maneja el equipo?* Como último punto se toma en cuenta el área de experticia del equipo. Aquellas herramientas en las cuales el equipo ya tengan experiencia serán una ventaja para la implementación final del proyecto.

4.2.4.2 Iteraciones o Sprints (Fases de Planificación, Ejecución y Monitoreo y control)

Uno de los requerimientos de la metodología es el mantener una base de código altamente reusable entre distintas plataformas. Para esto se propone un patrón de desarrollo de software basado en un enfoque modular en el cual se establecen métricas puntuales para calificar y dividir el código en módulos según su naturaleza. Antes de discutir la clasificación de los módulos se introducen algunos conceptos relevantes para el patrón de desarrollo:

- **Componente:**
  - Son la unidad básica de trabajo en la metodología.
  - Dentro del patrón planteado son sinónimo de módulo de frontend.
  - Un componente debe ser autosuficiente y debe encargarse de su funcionamiento y estado de forma total.
  - Un componente puede estar compuesto de otros componentes.
  - Un componente suele recibir atributos de otros componentes.
  - Deben ser lo más unitarios posibles.
  - Deben ser funciones puras, es decir, no deben verse afectadas por su entorno (dado unos parámetros de entrada siempre obtendremos los mismos parámetros de salida).
  - El flujo de datos entre componentes debe ser mediante un enlace unidireccional del padre a sus hijos. Un componente puede afectar los componentes que lo forman al variar el valor de los atributos con los que los instancia, sin embargo, estos componentes “hijos” no pueden modificar dichos valores buscando afectar la representación de los datos en el padre. Esto establece un estándar de comunicación y evita posibles comportamientos inesperados en la implementación de proyectos.
  - Ahora plantearemos un ejemplo cotidiano que iremos referenciando y ampliando a lo largo de la sección, un formulario de registro de

usuario. Una primera división unitaria en componentes de dicho requerimiento pudiese ser:

- **Un componente para cada tipo de input.** La personalización de las interfaces de usuario es un escenario común en los desarrollos modernos y es de especial interés mantener consistencia de estos cambios a lo largo de la aplicación (que todos los inputs de un tipo se vean igual independientemente de la vista en la que nos encontremos dentro del sistema). Por esta razón es buena práctica mantener un componente unitario para cada tipo de input.
- **Un componente para el formulario.** Este componente da estructura formal al formulario, instanciando todos aquellos componentes de input que sean necesarios.

- **Proveedor de servicios (Provider)**
  - Se enfoca en la lógica de negocio, la fuente de los recursos.
  - Tanto datos como funcionalidades se consideran recursos y la fuente de los mismos normalmente son los diferentes APIs web de los que se nutre la aplicación. Sin embargo, también pueden ser bases de datos locales, caché, u otro sistema de centralizado de recursos que afectará de forma global el App.

A su vez, dichos componentes o módulos son discriminados según su naturaleza, perteneciendo únicamente a una de las siguientes categorías:

- **Componentes de presentación:**
  - Son aquellos componentes encargados de representar los elementos de UI.
  - No incluyen lógica de negocio, en otras palabras, definen vistas mas no comportamientos.
  - Suele ser particular de cada plataforma sobre la cual se desarrolla.
  - Es buena práctica organizar los componentes de presentación en un mismo directorio dentro del proyecto para reconocerlos fácilmente de los envoltorios.
  - Volviendo al ejemplo del formulario, es sencillo reconocer que:

        - Ambos componentes planteados anteriormente son de presentación.
        - Los elementos HTML input no necesariamente son compatibles con las áreas de ingreso de texto propias de otras plataformas como las móviles.

- **Envoltorios de componentes (Wrappers):**
    - Son aquellos componentes que dan funcionamiento, mas no vistas.
    - Brindan control y funcionamiento.
    - Solo los envoltorios pueden comunicarse con los *providers.*
    - Por lo general están compuestos de componentes de presentación y especifican a estos como reaccionar ante las distintas situaciones que se puedan presentar. En pocas palabras, son componentes que reciben otros componentes y retornan una versión extendida del mismo.
    - Suele ser independiente de la plataforma sobre la cual se desarrolla, por esta razón suelen constituir en gran medida la base de código reutilizable entre proyectos.
    - Es buena práctica organizar los envoltorios en un mismo directorio dentro del proyecto para reconocerlos fácilmente de los componentes de presentación.
    - Volviendo al ejemplo del formulario, no es común ver un formulario tal y como ha sido planteado en un desarrollo moderno real, sin tener validación alguna y realización el envío de la información directamente desde el formulario. Ampliando la estructura del ejemplo para cumplir con requisitos comunes de los formularios modernos tendríamos como resultado:
        - **Un componente de presentación para cada tipo de input.** Tal como fue planteado con anterioridad, se encarga únicamente de la representación gráfica del input así como de sus mensajes de error.
        - **Un envoltorio para input.** Envoltorio encargado de aplicar la lógica de validaciones pertinente a cada input, y a su vez, de especificar al componente de presentación cuándo y qué mensaje de error/ayuda mostrar al usuario. A diferencia del apartado anterior donde cada tipo de input es controlado por un componente de presentación distinto, la lógica del envoltorio

es similar, si no igual, independientemente del tipo de input por lo que es suficiente con un envoltorio genérico para input en el cual se especifique el componente de presentación a utilizar y la lógica de validaciones a aplicar sobre la data introducida por el usuario.

- **Un componente de presentación para el formulario.** Define la estructura formal del formulario. En él se instancian las distintas ocurrencias del envoltorio de input especificando toda la información que estos necesitan para su correcto funcionamiento.
- **Un envoltorio para el formulario.** Envoltorio encargado de la interacción con el proveedor de servicios. Recibe del componente de presentación del formulario toda la información ya validada en lógica de los envoltorios de inputs, y se encarga de hacerla llegar al proveedor de servicios objetivo.

Un escenario común de la reutilización de los distintos tipos de componentes es:

- Reutilizar componentes de presentación dentro de un mismo proyecto, pero con distintos fines. Por ejemplo, los **componentes de presentación para cada tipo de input,** planteados anteriormente, son reusables dentro de una misma plataforma, pero difícilmente lo sea en el caso de buscar una versión móvil del formulario y una versión web.
- Reutilizar los envoltorios de componentes entre las distintas plataformas. Por ejemplo, el **envoltorio para el formulario** de registro de usuario planteado anteriormente difícilmente será usado en dos lugares dentro de una misma plataforma, pero si puede ser reutilizada en el caso de que se busque generar una versión móvil y una versión web.

Por último se deben tener en cuenta algunas consideraciones especiales:

Backend

Como toda estructura Cliente-Servidor, los proveedores de servicios son los encargados de nutrir las aplicaciones mediante el intercambio de recursos. Ahora bien, este backend debe cumplir algunas especificaciones.

- **Microservicios:** Para fomentar la independencia de las ramas del negocio, facilitar el trabajo en paralelo de múltiples equipos aplicando la metodología y para fomentar la estabilidad de los servicios (aquellos microservicios que no reciban cambios no verán afectada su disponibilidad, facilitando el trabajo al equipo de *frontend*) se fomenta utilizar una arquitectura basada en microservicios, en lugar de una estructura monolítica donde toda la lógica de negocio recide en un único proyecto.
- **API web:** Cada microservicio debe contar con un API web que sirva como puente de comunicación para cualquier que desee comunicarse con él. Es importante el uso correcto de estándares REST a la hora de crear los diferentes servicios web que compondrán el API. Se recomiendo el uso del formato JSON para el intercambio de recursos independientemente de la representación final que se tenga de dicha data en el cliente o en el servidor.
- **Lenguajes:** Suele ser ventajoso utilizar el mismo lenguaje de programación a lo largo de todas las áreas de desarrollo. Sin embargo, una de las principales ventajas de las arquitecturas basadas en microservicios es su capacidad de independencia tecnológica que nos permite elegir la mejor herramienta para el trabajo más indicado. Esta característica es algo importante a tener en cuenta siempre y cuando el costo de la curva de aprendizaje de dicha tecnología no sea mayor que el beneficio que puede traer al equipo su utilización en lugar de alguna con la que se encuentren más familiarizados.

Pull Request:

Para hacer un monitoreo continuo de la calidad del *software* y el cumplimiento de todos estos conceptos, así como para mantener al equipo informado de cada nueva funcionalidad y componente desarrollado (con la idea de que en caso de necesitarlo lo reutilicen en lugar de rehacerlo) se utilizan las revisiones de código en los *Pull Requests.*

Más allá de esto, el proceso de entrega de avances a través de revisiones de código también ayuda a compartir los conocimientos existentes en el equipo sobre las tecnologías involucradas, lo que funciona para una mejora continua de la formación de los involucrados.

Indiferentemente del rol o la experiencia de un desarrollador, al terminar su código no puede unirlo de forma inmediata a los repositorios principales (desarrollo y maestro), en su lugar, debe crear una solicitud para que su código sea unido a alguno de estos repositorios (*Pull Request*). Para aprobarlo, sus pares y el líder técnico deben revisar el trabajo realizado y dar fe de su correctitud.

4.3 Usos de la metodología

MCPDev está pensada para apoyar los desarrollos multiplataforma independientemente del tamaño o características del equipo o el proyecto. Sin embargo, y para obtener los mayores beneficios de la implementación de la metodología, se recomienda un tamaño mínimo de equipo de 3 personas y estos son el PM, el líder técnico y un desarrollador, los demás roles pueden ser repartidos entre ellos.

Cabe acotar que para poder aplicar esta o cualquier otra metodología en una empresa / equipo hay que tener en cuenta varios aspectos como:

- Competencias personales de los integrantes del equipo (no solo las del líder de proyecto).
- Técnicas, plantillas, estándares, herramientas y procesos existentes en la empresa las cuales puedan afectar directamente la forma que tome la metodología y los procesos que esta lleve a cabo dentro de un proyecto para la empresa.
- Cultura organizacional y contexto (Madurez de la empresa).

Con esto dicho, acá se plantea un mapa conceptual del ciclo de vida para la metodología de forma tal que la misma pueda ser aplicada en empresas / equipos de tecnología involucrados en desarrollos multiplataforma indiferentemente de su tamaño y de los valores obtenidos al analizar los aspectos anteriores. En otras palabras, se plantea una metodología incrementable y personalizable a cada empresa y que está centrada en los desarrollos multiplataforma (véase la *Figura 3*).

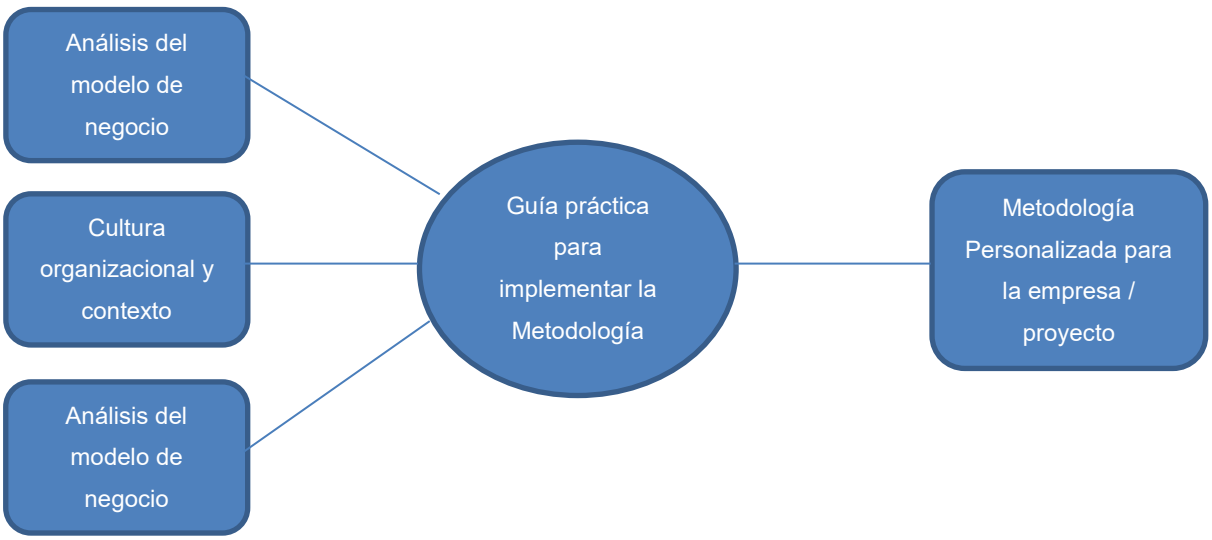

Figura 3 - Mapa conceptual de la aplicación de la metodología en instituciones reales

Otro aspecto importante a tener en cuenta es la naturaleza del proyecto en sí. Dependiendo del valor y la complejidad del proyecto, más allá de la estructura del mismo, podemos necesitar más o menos herramientas y/o técnicas para mejorar nuestra adecuación de la metodología al proyecto (véase la *Figura 4).*

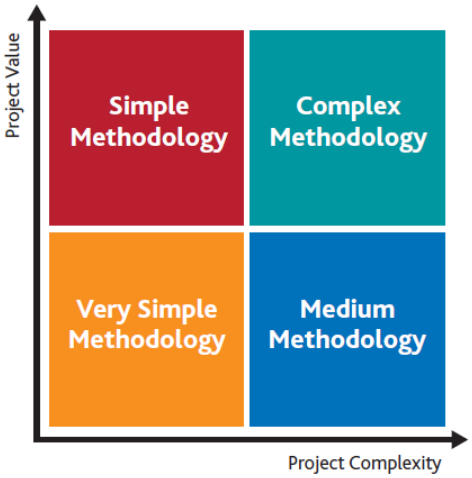

Figura 4 - Matriz de clasificación de complejidad de las metodologías en dirección de proyectos [26]

4.4 Desarrollo del Framework

Con el fin de realizar un caso de estudio de la metodología, se desarrollo un *framework* tomando en cuenta todos los principales aspectos planteados en MCPDev.

Es importante destacar que para muchos de los hitos de la metodología se eligieron herramientas ya existentes y la metodología simplemente especificará la forma de interactuar con ellas, mas no estará realizando nuevas versiones de las mismas.

4.4.1 Wiki – Google Drive

Para el Wiki se eligió la plataforma web de Google Drive, mediante la cual se estructuró la fuente única de recursos para el equipo:

- **Directorio del Proyecto:** Todo el wiki está creado bajo un directorio con el nombre del proyecto. Dentro de este directorio se encuentra la información básica del proyecto. En la actual implementación de la metodología dicha información básica está compuesta por el 'Acta de constitución del proyecto', el 'Acta de cierre del proyecto' y el documento de 'Tecnologías y accesos'. Dada

la naturaleza del caso de estudio se dejará a un lado todo lo relacionado a gestión de costos. De igual forma se encuentran los subdirectorios de 'Formatos' y 'Sprints' (véase *Figura 5).*

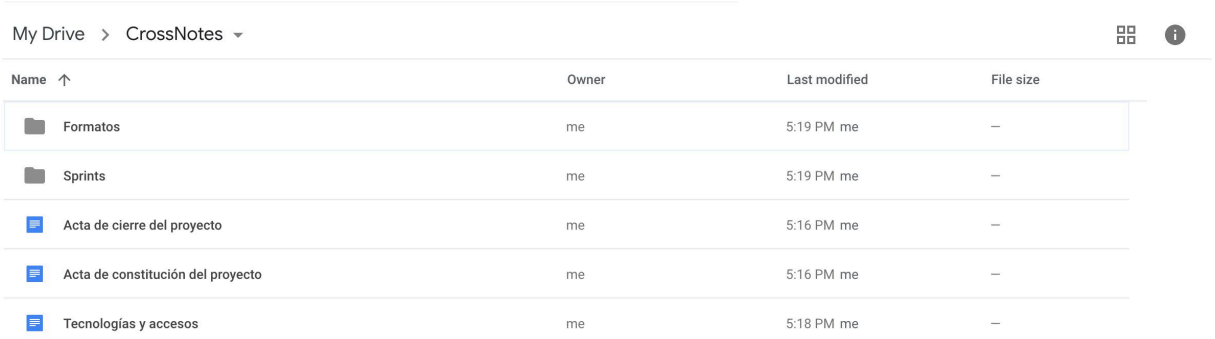

Figura 5 - Directorio de Proyecto

- **Directorio de Formatos:** Dentro de este directorio se encuentran todos los formatos de archivos a utilizar por el equipo de forma que cualquiera pueda acceder a ellos. En la actual implementación de la metodología dichos formatos son 'Acta de cierre de Sprint' y 'Retrospectiva' (véase *Figura 6).*

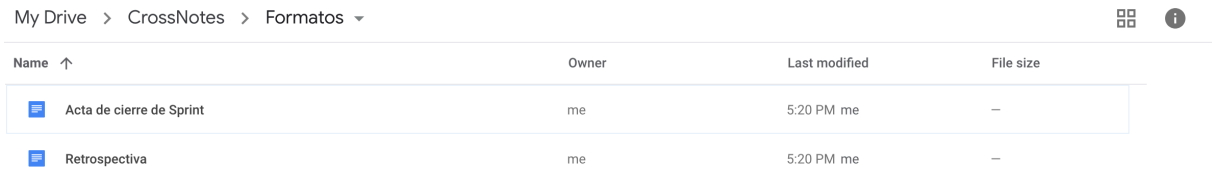

Figura 6 - Directorio de Formatos

- **Directorio de Sprints:** Conjunto de directorios de *Sprint*. Cada *Sprint* que tome lugar en el proyecto tendrá su subdirectorio dentro del directorio de *Sprints* e incluirá la información básica de gestión de *Sprint* 'Acta de cierre de Sprint' y 'Retrospectiva') más cualquier otra documentación que se considere pertinente compartir durante el mismo (véase *Figura 7).*

Figura 7 - Directorio de Sprint

4.4.2 Tablero Kanban – Trello

Se utilizará la herramienta de Trello como tablero Kanban para gestionar el proyecto (su ciclo de vida y diferentes requerimientos) y cumplir con directrices de la metodología.

4.4.3 Calendario – Google Calendar

Se utilizará la herramienta de Google Calendar como calendario para el control y gestión de citas y actividades del equipo siguiendo las directrices de la metodología.

4.4.4 Plataforma para el manejo de versiones – Bitbucket

Se utilizará la plataforma de Bitbucket como plataforma de manejo de versiones para la cumplir con las necesidades de gestión centralizado de código, trabajo mediante ramas y utilización de los *Pull Request* para asegurar la calidad del código, entre otras directrices de la metodología.

5 Caso de estudio

En este capítulo se realiza el planteamiento, desarrollo, recolección de datos y análisis de resultados del caso de estudio mediante el cual se prueba la metodología.

5.1 Planteamiento del experimento - CrossNote

El agente diferenciador para decidir utilizar la metodología MCPDev en un proyecto es el hecho de que este implique múltiples plataformas objetivo. Teniendo esto en cuenta, el experimento a realizar se enfocará más en el alcance en cuanto a plataformas objetivo que en cuanto a funcionalidades del aplicativo. De igual forma, se hará énfasis en la calidad del aplicativo resultante como opción válida ante los desarrollos nativos.

5.1.1 Temática

La temática del caso de estudio será una aplicación multiplataforma para la gestión de notas en las que un usuario va a poder visualizar y modificar sus notas en cualquiera de los dispositivos soportados. El aplicativo debe contar con:

- **Manejo de usuarios:**
  - Estructura de usuarios: Nombres, apellidos, nombre de usuario (hasta 16 caracteres), correo y contraseña (8 – 16 caracteres).
  - Acciones permitidas a los usuarios: Inscribirse, iniciar sesión, cerrar sesión y modificar sus datos.
- **Manejo de categorías:**
  - Las notas están divididas por categorías y son propias para cada usuario. El único atributo propio de las categorías es su nombre (hasta 40 caracteres)
  - La plataforma agrega una categoría por defecto:
    - Todas: Muestra todas las notas independientemente de la categoría a la que pertenezcan.
  - Acciones permitidas: Crear categoría, modificar categoría, eliminar categoría.
- **Manejo de notas:**
  - Tienen título y contenido.

- El contenido no tiene límite definido de caracteres, mientras que el título sí (40 caracteres).
- Pueden o no tener una categoría, pero en caso de tenerla es única por nota.
- Acciones permitidas: Crear, modificar, eliminar nota, asociar y desasociar nota a una categoría.

5.1.2 Plataformas

Como resultado de la aplicación de la metodología en el proyecto se deben obtener versiones para las siguientes plataformas:

Escritorio: MacOS 10.13 o mayores y Windows 10 o mayores.

Móvil: Android 8.0 y mayores y iOS 11 y mayores.

Web: Para efectos de la prueba solo se trabajará con Google Chrome desde la versión 64.0.0 en adelante.

Si bien tanto la metodología como las tecnologías utilizadas pudiesen soportar otras plataformas (como las distribuciones GNU/Linux dentro de la categoría de plataformas de escritorio) en el experimento nos enfocaremos únicamente en las acá mencionadas.

5.1.3 Pruebas

- **Calidad de la experiencia de usuario:**

  Mediante encuestas evaluar la calidad de la experiencia de usuario de los aplicativos resultantes comparado con aplicativos nativos (véase el *anexo 1)*. La muestra de usuarios a encuestar estará formada por 20 usuarios especializados (desarrolladores, QA, PM, PO y demás stakeholders usualmente involucrados en el proceso de desarrollo de software) y 20 usuarios consumidores de software elegidos al azar sin tomar en cuenta su experticia, pero que sean usuarios del OS objetivo. A cada usuario seleccionado probará una plataforma móvil (iOS o Android) y una de escritorio (Windows o MacOS).

  La metodología parte del desarrollo de una versión web utilizando tecnologías netamente web para posteriormente, y a partir de esta, desarrollar las demás versiones. Por esta razón este apartado se enfocará en evaluar los aplicativos para los distintos sistemas operativos más no para la versión web.

- **Complejidad tecnológica**

  Calcular el número de tecnologías y lenguajes bases involucrados en este desarrollo multiplataforma y compararlo con el número de tecnologías bases utilizados en los desarrollos nativos de dichas aplicaciones.

  Por tecnologías bases entendemos aquellas que componen la pila de desarrollo mínima utilizada para lograr el desarrollo, acá no son tomadas en cuenta diferente librerías utilizadas por elección de los desarrolladores para facilitar su trabajo en aspectos particulares del proyecto.
- **Reutilización de código entre aplicaciones**

  Para la interpretación de las pruebas se partirá de la web a las demás plataformas. En otras palabras, se hará la aplicación web y, tomando esta como base, se realizarán las demás versiones especificando qué porcentaje del código y lógica de negocio de esta fue reutilizado para cada nueva plataforma.
- **Reutilización de código entre aplicaciones para un mismo tipo de plataforma**

  Cuánta lógica de negocio y código se pudo compartir entre las aplicaciones móviles y cuánto entre las de escritorio.
- **Reutilización de código dentro de las aplicaciones**

  Cuánta lógica de negocio y código se pudo reutilizar dentro de cada aplicación analizada de forma independiente por la aplicación de los principios de modularidad planteados en la metodología.
- **Mantenimiento**

  La complejidad de mantenimiento viene directamente asociada al esfuerzo necesario para implementar un cambio en la lógica de negocio y ver este reflejado en todas las plataformas objetivos del proyecto. Para esto se agregará un atributo ‘descripción’ a cada nota y se analizará el costo en cuanto a líneas de código de hacer efectivo este cambio en la web y posteriormente, cuánto de este código pudo ser reutilizado para aplicar dicho cambio en todas las demás plataformas.

5.2 Aplicación de la metodología en el experimento

5.2.1 Fase de Inicialización

- **Duración**

Tal y como se recomienda en la metodología, se tomó la primera semana del proyecto para la fase de inicialización.

- **Acta de constitución del Proyecto**

Como resumen y una de las partes centrales de la etapa se desarrollo el Acta de constitución del Proyecto donde se incluye la temática, alcance, plataformas objetivo, tecnologías a utilizar, fechas relevantes, compromisos y por último los participantes y sus roles (véase *Anexo 2)*.

- **Formatos de documento**

De acuerdo a la especificación del *framework* desarrollado para la implementación de la metodología, se desarrollaron los formatos de 'Acta de cierre de *Sprint*' y 'Retrospectiva' (véanse *Anexos 3* y *4)*.

- **Consideraciones tecnológicas**

Para obtener la pila tecnológica base nos apoyamos en los pasos listados en la metodología para el análisis de requerimientos y características proyecto / equipo.

Plataformas objetivo: Las versiones mínimas de los sistemas operativos son: MacOS 10.13, Windows 10, Android 8.0 y iOS . De igual forma, en cuanto a la versión web se trabajará con Google Chrome desde la versión 64.0.0 en adelante.

- Teniendo en cuenta que haremos un desarrollo web se analizaron tecnologías que permitieran tomar esta aplicación como base y reutilizar la mayor capacidad de lógica posible.
- Se eliminaron *frameworks* de desarrollo multiplataforma que no fueran compatibles con desarrollos web o de escritorio como Appcelerator y Xamarin.
- **Electronjs** fue elegido para las versiones de escritorio de MacOS y Windows.

Naturaleza y requerimientos del proyecto: Proyecto de corto alcance y sin restricciones en cuanto al lapso de tiempo que pueda tomar el desarrollo. La calidad del aplicativo en cuanto a desempeño, calidad de experiencia de usuario (UX) e interfaz de usuario (UI) son prioritarias para el proyecto y se esperan que sean

similares al que se pudiese obtener de realizar las aplicaciones con tecnologías nativas.

- Los *frameworks* de aplicaciones híbridas basadas en HTML5 como ionic o phonegap fueron descartadas por los sacrificios en UX y desempeño que suelen traer como resultado.

Reutilización de código:

- **React** es una librería de renderizado de vistas enfocada en la modularidad, reusabilidad y desempeño. Es una herramienta a la cual se pueden extrapolar las teorías de modularidad la metodología sin mayor esfuerzo.
- **React Native** nos permite utilizar la misma sintaxis de React para los desarrollos móviles lo que nos permitiría una reutilización importante de lógica.
- React al ser una tecnología web es compatible con **Electronjs.**
- Un punto importante a tener en cuenta es la reutilización de componentes de presentación por lo cual se decidió utilizar un *framework* que brindará distintas opciones de uso común, reusables y autocontenidas. En este apartado fue elegido **Semantic UI**.

Áreas de conocimiento del equipo de desarrollo: El equipo ha estado envuelto en investigaciones en el área de los desarrollos web del lado del cliente durante los últimos años, por lo que el lenguaje de JavaScript es un punto en común entre ellos.

- Con el fin de aprovechar las fortalezas del equipo y unificar el lenguaje de programación en el proyecto fueron elegidos **Node.js** y **ExpressJS** para el desarrollo de backend y **MongoDB** para el almacenamiento de datos.

Finalmente la pila de desarrollo elegida quedo de la siguiente forma:

Lenguaje: JavaScript

Backend: Node.js, ExpressJS, MongoDB.

Frontend: React, Redux, React Native, Electronjs, Semantic UI.

Por último, tal como lo especifica la metodología, el desarrollo de backend será realizar siguiendo una arquitectura basada en microservicios. Para esto se analizaron todas las funcionalidades a realizar y se dividieron en dos bloques distintivos:

Microservicio de gestión de usuarios: Manejará toda la lógica inherente a los usuarios incluyendo más no limitándose a la autenticación y autorización.

Microserivio de gestión de notas: Manejará toda la lógica involucrada en la administración de notas. Las categorías están estrechamente relacionadas a la lógica de las notas, al punto que pudieran verse como una características de estas, por dicha razón estarán incluidas en el mismo microservicio.

- **Calendario**

Sobre la herramienta ya seleccionada para el desarrollo del calendario se procede a incluir todas las fechas, reuniones e hitos de importancia para el proyecto (véase *Figura 8).*

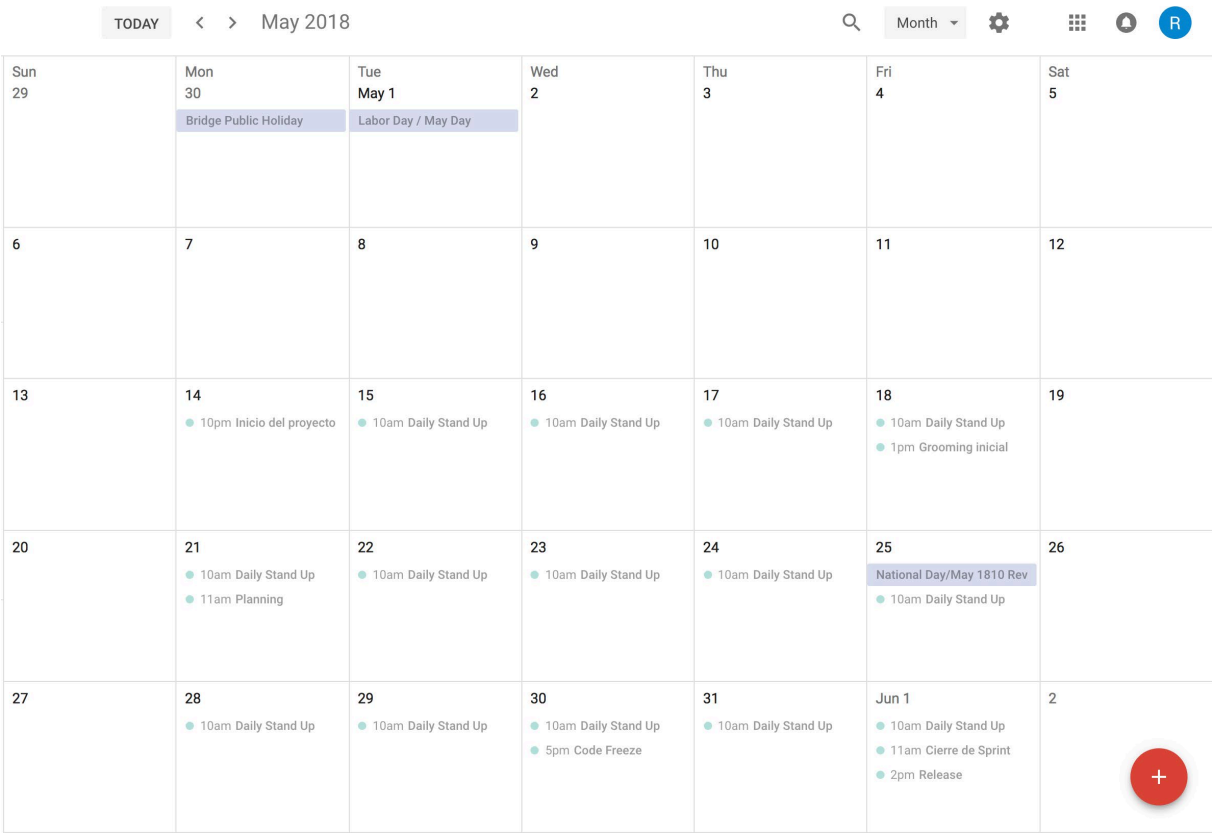

Figura 8 - Calendario de actividades

- **Wiki**

Sobre la estructura de wiki ya definida en el *framework* incluimos los formatos recién generados de 'Acta de constitución del Proyecto', 'Acta de cierre de Sprint' y 'Retrospectiva', así como actualizar la lista de tecnologías y herramientas involucradas en el trabajo diario del equipo (véase *Anexo 5)*.

- **Tablero Kanban, Product Backlog y Grooming inicial**

Apoyados en la herramienta de Trello se genera el tablero a utilizar en el proyecto, se hace la carga inicial de las actividades del proyecto y se procede a realizar el grooming inicial con todo el equipo para dar puntaje a las actividades iniciales del proyecto (véase *Figura 9).*

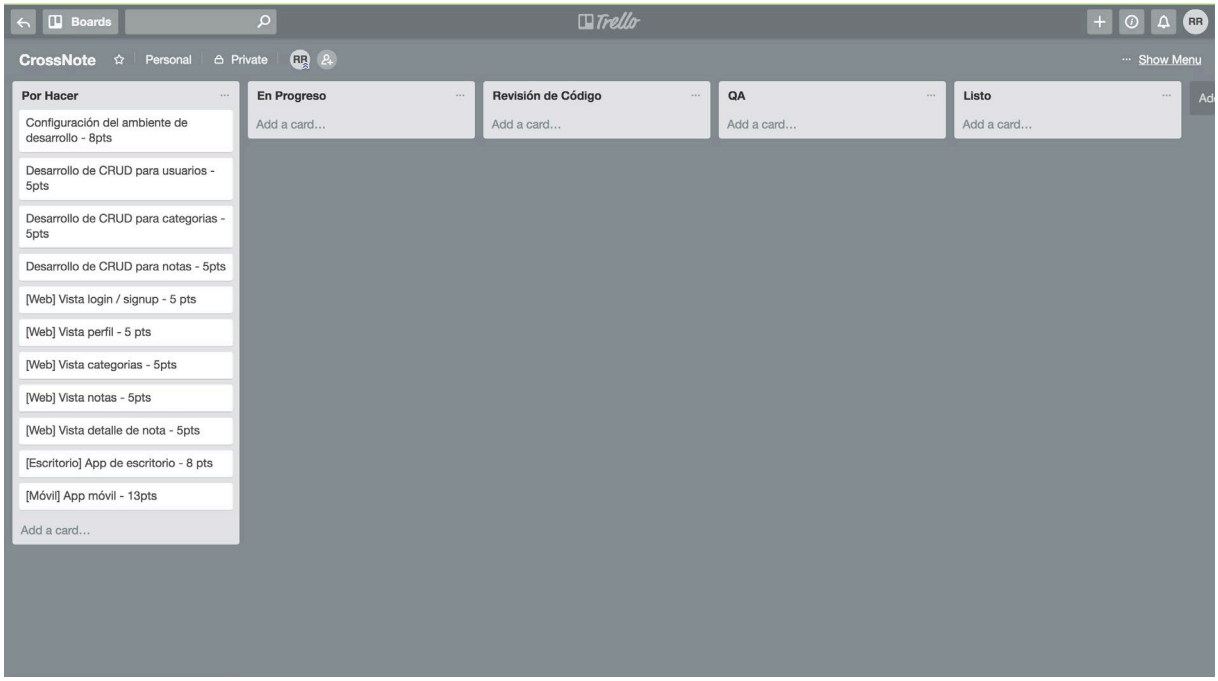

Figura 9 - Tablero Kanban de CrossNote

- **Configurar la plataforma para el manejo de versiones**

Se crea el repositorio del proyecto en la plataforma de Bitbucket. De igual forma se crean las ramas de desarrollo y maestra y se generan los accesos necesarios al equipo de trabajo (véase *Figura 10)*.

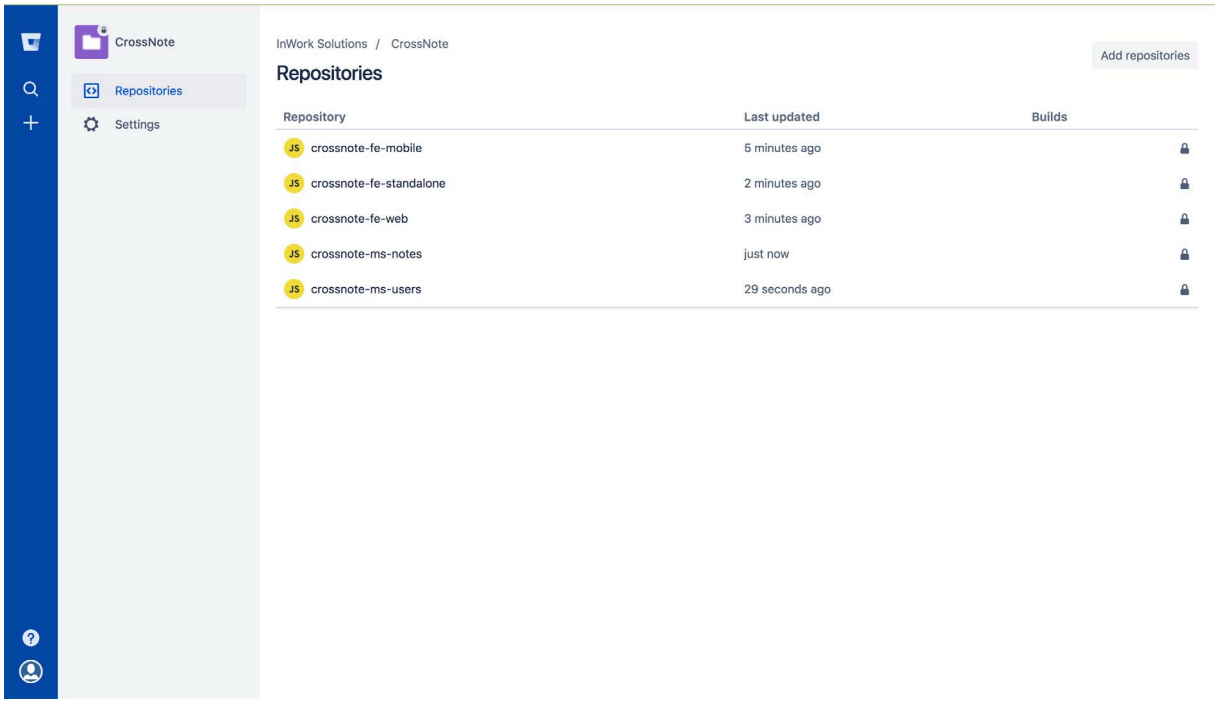

Figura 10 - Proyecto en bitbucket

- **Equipo de trabajo**

Por la naturaleza del experimento el equipo de trabajo es el mismo investigación.

5.2.2 Iteraciones o Sprints (Fases de Planificación, Ejecución y Monitoreo y control)

- **Duración e Iteración**

Por la naturaleza del experimento y lo reducido de su alcance se realizó una pequeña modificación a la especificación en cuanto a la duración de los Sprints. El proyecto se realizó en base a Sprints de 1 semana iniciando un día lunes. Luego del Grooming inicial del proyecto se concluyo que el desarrollo del mismo estaría compuesto por 1 sprint únicamente por lo que la duración total de esta etapa fue de 1 semana.

- **Daily Stand Up** (DSU)**:**

Esta reunión diaria de análisis de la situación actual ayudó al equipo a mantener en contexto el desarrollo del proyecto y los retos presentados así como sus posibles soluciones.

- **Product Backlog, Planning y Grooming**

Dado el corto alcance del proyecto, durante esta fase no fue necesario mantener un *Backlog* de productos o realizar *Groomings* adicionales al inicial dado el corto alcance del proyecto. De igual forma, solo fue realizado un *Planning* durante esta etapa.

- **Code Freeze y Release**

Otra de las excepciones realizadas para adaptar la metodología al alcance y lapsos de tiempo del experimento fue colocar el *Code Freeze* el último día del sprint, es decir, el viernes de la semana de desarrollo, realizando el *Release* inmediatamente después. El proyecto contó con un único *Release* que incluía la totalidad del código por lo que no fueron necesarias pruebas unitarias o de integración (manuales o automatizadas) antes de realizarlo.

- **Calendario y Wiki**

Dado que el proyecto se realizo en solo 1 Sprint no fue necesario realizar mayores modificaciones al calendario o Wiki.

- **Plan de pruebas y Cierre de sprint**

A medida que se iban realizando las diferentes funcionalidades del aplicativo el equipo fue redactando un detallado *Plan de pruebas* que posteriormente fue aplicado durante el demo de la reunión de *Cierre de sprint* quedando grabado en video para posteriormente poder ser compartido con los interesados. De igual forma durante esta reunión se realizaron los documentos de ‘Acta de cierre de Sprint’ y ‘Retrospectiva’ del Sprint (véanse *Anexo 3* y *Anexo 4)*.

- **Consideraciones tecnológicas**

La estructura y el como aplicar los principios planteados por la metodología dependerá mucho de las tecnologías a utilizar. De igual forma, el desarrollo modular y los conceptos de componentes y envoltorios está un poco más trabajado en algunos entornos de desarrollo como en los basados en ReactJS (nuestra principal tecnología en el desarrollo de *frontend* independientemente de la plataforma), en donde tenemos claras referencias a patrones de desarrollo de *software* mediante distintos principios como los *Containers* en Redux. [10]

Apoyándonos en esta nomenclatura ya existente en el entorno de desarrollo llamamos '*Containers'* a los envoltorios de componentes y estructuramos los proyectos acoplándonos a las prácticas comunes dentro de las tecnologías a utilizar pero manteniendo una clara distinción entre *Containers*, Componentes y Proveedores (*Providers*) (véase la *Figura 11).*

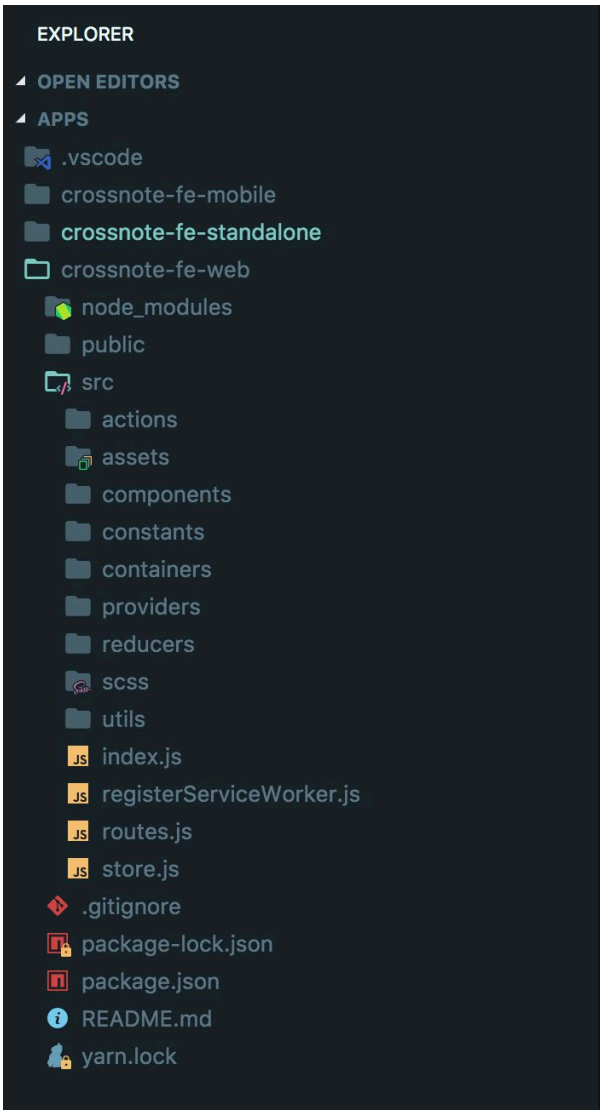

Figura 11 - Estructura de proyecto

Tal como en el ejemplo utilizado en la descripción del paradigma de programación utilizado por la metodología, el proyecto de CrossNote incluye un formulario para el registro de usuarios, en este se puede apreciar de forma práctica la interacción y beneficios de los diferentes tipos de componentes planteados:

- **Componente de presentación para inputs.** La utilización de Componentes de presentación permitió en gran medida la reutilización de código dentro de un mismo proyecto así como la consistencia en la experiencia de usuario. Estos componentes fueron específicos para cada plataforma ya que se trabajaban con elementos y/o librerías propias de la misma (véanse *Figuras 12* y *13).*

```
import { Form, Message } from 'semantic-ui-react'

const RenderField = ({ input, label, type, meta: { touched, error } }) => {
  const hasError = !!(touched && error)
  return (
    <div className='field field-container'>
      <Form.Input
        fluid
        {...input}
        label={label}
        placeholder={label}
        type={type}
        error={hasError}
      />
      {hasError && (
        <Message
          error
          header={error}
        />
      )}
    </div>
  )
}
```

Figura 12 - [Web] Componente de presentación para inputs

```
import { StyleSheet } from 'react-native'
import { Input } from 'react-native-elements'
import getStyles from './styles'
const styles = StyleSheet.create(getStyles())

const RenderField = ({ input: { onChange, ...restInput }, label, type, meta: { touched, error } }) => {
  const hasError = !!(touched && error)
  return (
    <Input
      containerStyle={styles.root}
      label={label}
      placeholder={label}
      textContentType={type}
      onChangeText={onChange}
      {...restInput}
      errorMessage={hasError ? error : ''}
    />
  )
}
```

Figura 13 - [Mobile] Componente de presentación para inputs

- **Envoltorio para input.** Este componente permitió realizar las validaciones en tiempo real sobre el contenido de los inputs y fue reutilizado en las diferentes plataformas de forma indistinta entre los diferentes tipos de inputs de usuarios, solo debiendo especificar el componente de presentación del tipo input que se estaría extendiendo (véase *Figura 14).*

```
          <Field
        name="first_name"
        type="text"
        label="First Name"
        component={RenderField}
      />
```

Figura 14 - Envoltorio de componente para inputs

Como último punto dentro de las consideraciones tecnológicas esta el *Backend*. Es de especial importancia la apropiada utilización de estándares para permitir la correcta y

sencilla interacción entre el servidor y sus distintos cliente, por esta razón el API a utilizar es el resultado de la unión de los Servicios Web Rest generados en ambos microservicios entre los cuales resaltan los servicios contenidos en las *Tablas 6, 7, 8 y 9.*

- **Manejo de Sesión - Microservicio de gestión de usuarios**

Descripción	Método	URL
Iniciar sesión	POST	`{SERVER}/api/v1/auth/login`
Finalizar sesión	POST	`{SERVER}/api/v1/auth/login`

Tabla 6 - Endpoints de manejo de sesión

- **Manejo de Usuarios - Microservicio de gestión de usuarios**

Descripción	Método	URL
Registrar nuevo usuario	POST	`{SERVER}/api/v1/user `
Obtener detalles de un usuario	GET	`{SERVER}/api/v1/user/:id `
Modificar detalles de usuario	PUT	`{SERVER}/api/v1/user/:id`

Tabla 7 - Endpoints de manejo de usuario

- **Manejo de Categorías - Microserivio de gestión de notas**

Descripción	Método	URL
Obtener todas las categorías	GET	`{SERVER}/api/v1/category `
Crear categoría	POST	`{SERVER}/api/v1/category `
Modificar categoría	PUT	`{SERVER}/api/v1/category/:id `
Eliminar categoría	DELETE	`{SERVER}/api/v1/category/:id `

Tabla 8 - Endpoints de manejo de categorías

- **Manejo de Notas - Microserivio de gestión de notas**

Descripción	Método	URL
Obtener todas las notas	GET	`{SERVER}/api/v1/note `
Crear nota	POST	`{SERVER}/api/v1/note `
Obtener detalles de una nota	GET	`{SERVER}/api/v1/note/:id `
Modificar nota	PUT	`{SERVER}/api/v1/note/:id `
Eliminar nota	DELETE	`{SERVER}/api/v1/note/:id `

Tabla 9 - Endpoints de manejo de notas

5.2.3 Fase de Cierre

A pesar de que la duración mínima recomendada es de 1 semana, dada la naturaleza del experimento esta fue de solo 1 día y se puede resumir en las siguientes actividades:

- **Acta de cierre del Proyecto**:

Sirve de cierre formal del proyecto. Da evidencia de la conformidad de las partes con el desarrollo realizado, así como de la entrega del código funcional. Tiene una estructura similar al 'Acta de constitución del proyecto' con el objetivo de poder comparar lo inicialmente acordado con lo finalmente realizado, no dejando lugar a dudas o confusiones (véase *Anexo 6)*.

- **Lecciones aprendidas**

Se redacta el documento de lecciones aprendidas durante el proyecto para ser utilizadas por el equipo / empresa en una futura implementación de la metodología (véase *Anexo 7).*

- **Transferencia de conocimiento**

El alcance del proyecto no incluía la puesta en producción o publicación de aplicativos para ningún mercado de aplicaciones por lo que únicamente se hizo entrega del acta de cierre de proyecto previamente desarrollada y se cedió la propiedad de los repositorios de código (véase *Anexo 6 y Anexo 7)*.

- **Fin de actividades**

Se dieron por finalizadas las actividades del proyecto y:

  - Se cerró el tablero Kanban
  - Se cerró el calendario
  - Se cerró el wiki
  - Se liberó el equipo de trabajo

5.3 Presentación de resultados

- **Calidad de la experiencia de usuario:**

Como resultado de las encuestas realizadas se recibieron un total de 10 opiniones para cada plataforma por cada grupo de usuarios. Los resultados se muestran en las *Tablas 10, 11, 12 y 13* separados por grupo de usuarios:

- **Usuarios especializados**

	Desarrollo Nativo	Desarrollo Multiplataforma
Dispositivo Móvil - Android	8	2
Dispositivo Móvil - iOS	9	1
Dispositivo de Escritorio - MacOS	10	0
Dispositivo de Escritorio - Windows	10	0

Tabla 10 - Encuesta de naturaleza del aplicativo para usuarios especializados

	Desempeño	UX	UI	Otra
Dispositivo Móvil - Android	0	2	0	0
Dispositivo Móvil - iOS	0	1	0	0
Dispositivo de Escritorio - MacOS	0	0	0	0
Dispositivo de Escritorio - Windows	0	0	0	0

Tabla 11 - Encuesta de indicativo de desarrollo multiplataforma para usuarios especializados

- **Usuarios consumidores de software**

	Desarrollo Nativo	Desarrollo Multiplataforma
Dispositivo Móvil - Android	9	1
Dispositivo Móvil - iOS	10	0
Dispositivo de Escritorio - MacOS	10	0
Dispositivo de Escritorio - Windows	10	0

Tabla 12 - Encuesta de naturaleza del aplicativo para usuarios consumidores

	Desempeño	UX	UI	Otra
Dispositivo Móvil - Android	0	1	0	0
Dispositivo Móvil - iOS	0	0	0	0
Dispositivo de Escritorio - MacOS	0	0	0	0
Dispositivo de Escritorio - Windows	0	0	0	0

Tabla 13 - Encuesta de indicativo de desarrollo multiplataforma para usuarios consumidores

Esta data puede ser un poco más representativa al ser observada de forma gráfica (véanse *Figuras 15 y 16).*

- **Usuarios especializados**

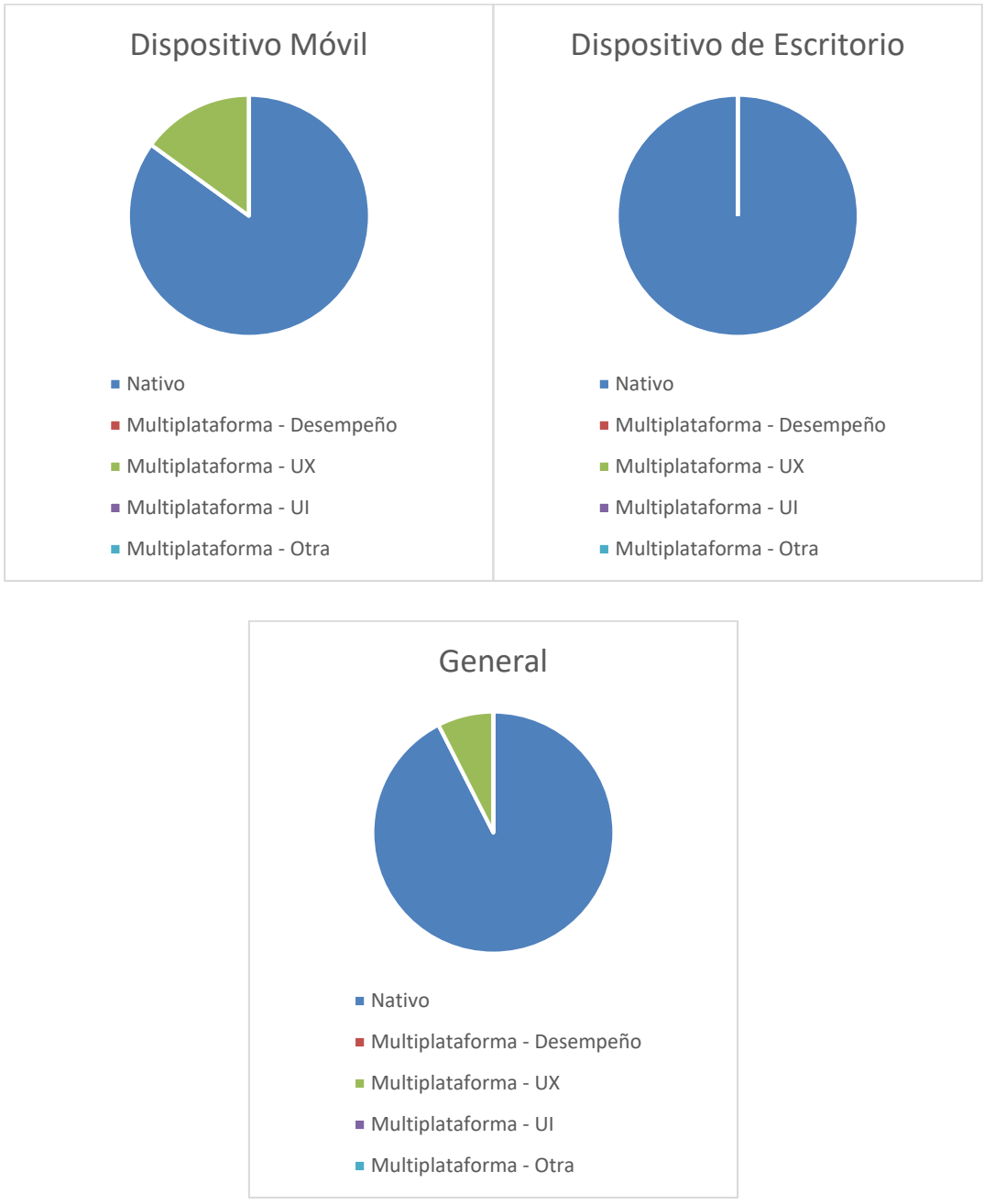

Figura 15 - Gráficas de encuestas para usuarios especializados

- **Usuarios consumidores de software**

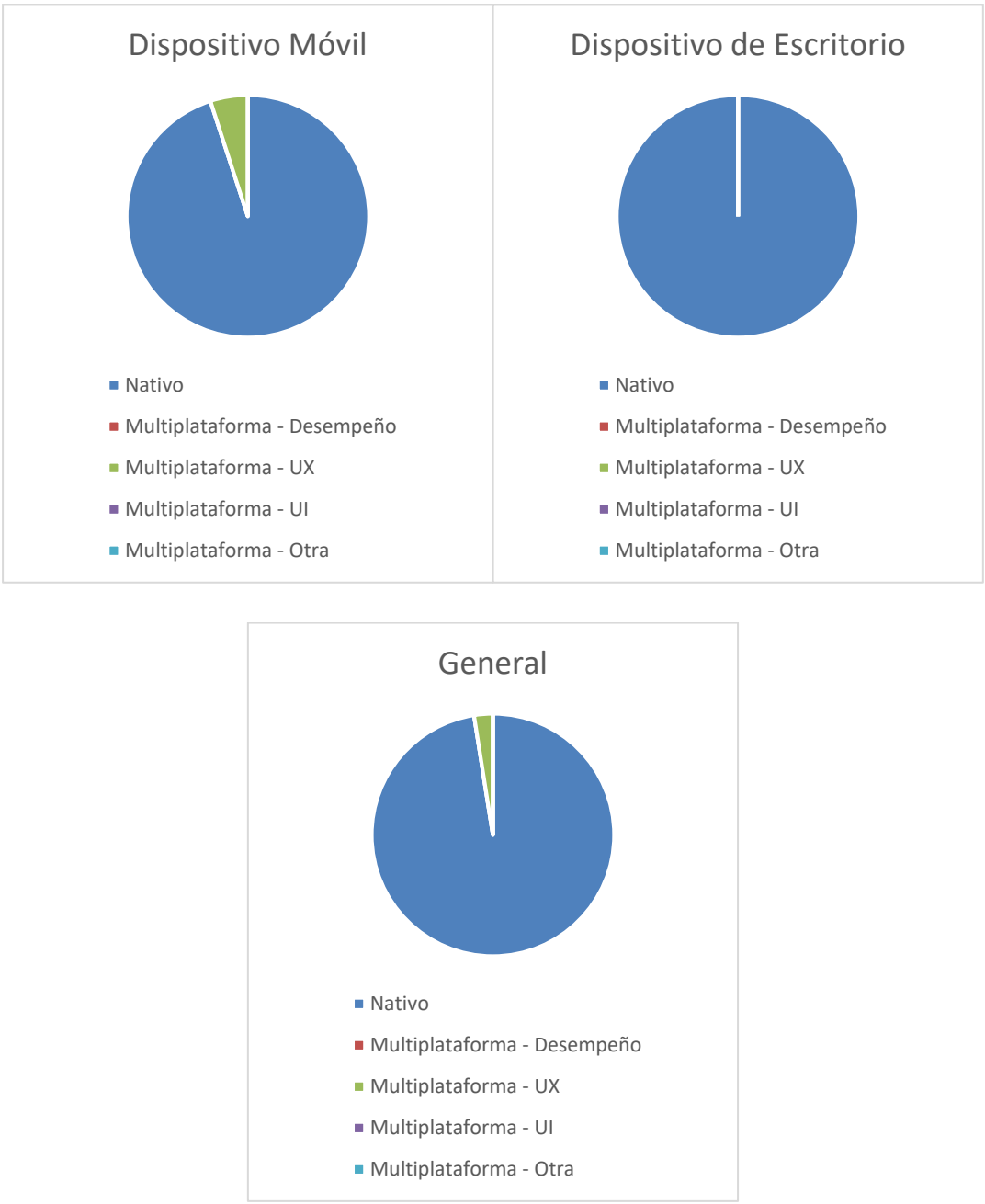

Figura 16 - Gráficas de encuestas para usuarios consumidores

- **Complejidad tecnológica**

Se utilizaron variedad de librerías y tecnologías para facilitar el desarrollo, sin embargo, la base de la pila de desarrollo utilizada fue:

Lenguaje de Programación: JavaScript

Tecnologías: React, React Native, Electronjs, Node.js y MongoDB.

- **Reutilización de código entre aplicaciones**
  - **Backend**
    - **Total:** 763 líneas de código
  - **Aplicación Web**
    - **Total:** 2.065 líneas de código
  - **Aplicaciones de escritorio**
    - **Total:** 2.115 líneas de código
    - **Reutilizado de la web:** 2.065 líneas de código (aprox. 97,6% del código) (véase *Figura 17).*
  - **Aplicaciones móviles**
    - **Total:** 2.294 líneas de código
    - **Reutilizado de la web:** 1.307 líneas de código (aprox. 57% del código) (véase *Figura 17).*

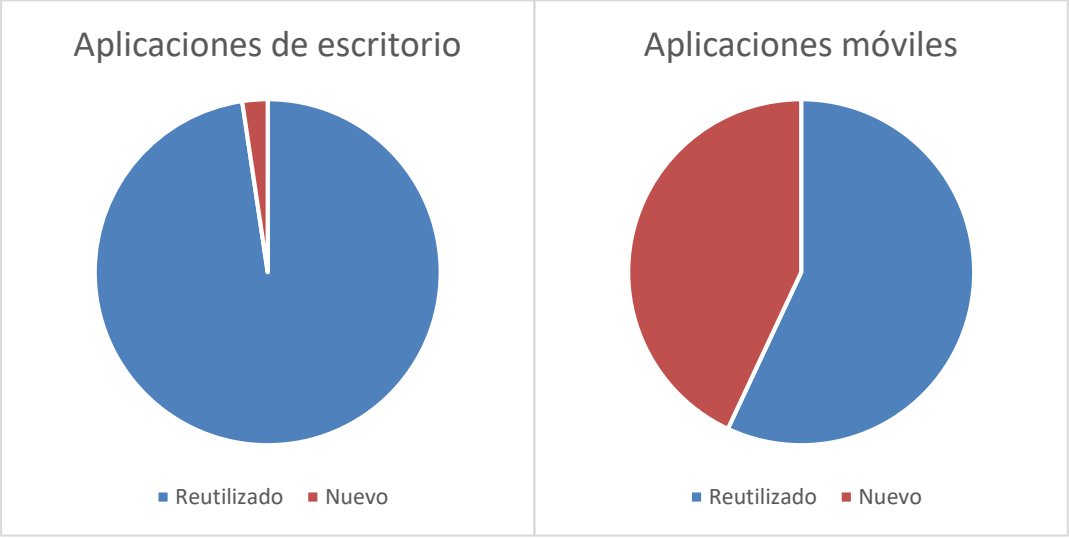

Figura 17 - Reutilización de código entre aplicaciones

- **Reutilización de código entre aplicaciones para un mismo tipo de plataforma**

Dentro de las aplicaciones para un mismo tipo de dispositivo (Escritorio o móvil) la reutilización del código fue de un 100%, en otras palabras, se utilizó el mismo código base

para generar las aplicaciones de un mismo tipo de dispositivo independientemente del sistema operativo objetivo.

- **Reutilización de código dentro de las aplicaciones**
  - **Aplicación Web**
    - **Total de líneas ahorradas:** 718 líneas de código (representa un aumento de aprox. 34,8% del código) (véase *Figura 18).*
  - **Aplicaciones de escritorio**
    - **Total de líneas ahorradas:** 718 líneas de código (representa un aumento de aprox. 34% del código) (véase *Figura 18).*
  - **Aplicaciones móviles**
    - **Total de líneas ahorradas:** 806 líneas de código (representa un aumento de aprox. 35,1% del código) (véase *Figura 18).*

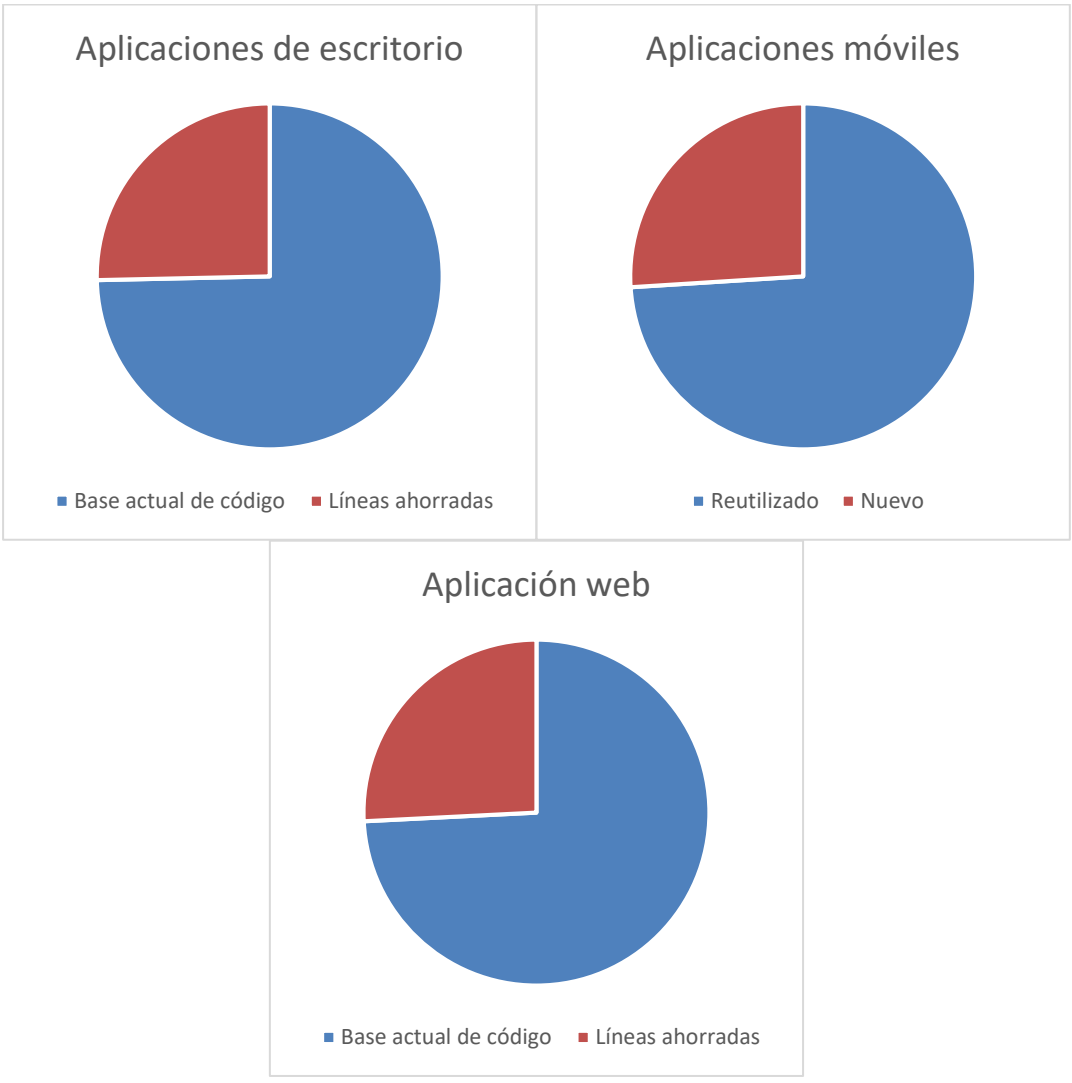

Figura 18 - Reutilización de código dentro de las aplicaciones

- **Mantenimiento**

Una característica importante de la metodología era su adaptabilidad al cambio y esta va directamente relacionada a la facilidad de mantenimiento para los productos realizados con la misma. En este apartado se realizo una adición de un atributo 'descripción' a las notas en todas las plataformas (véanse *Figuras 19, 20 y 21).*

Para realizar este cambio fue necesario incluir **13 líneas de código** en la versión web y ningún cambio extra sobre las versiones móviles o de escritorio ya que el cambio fue realizado en un archivo de la base compartida de código entre plataformas.

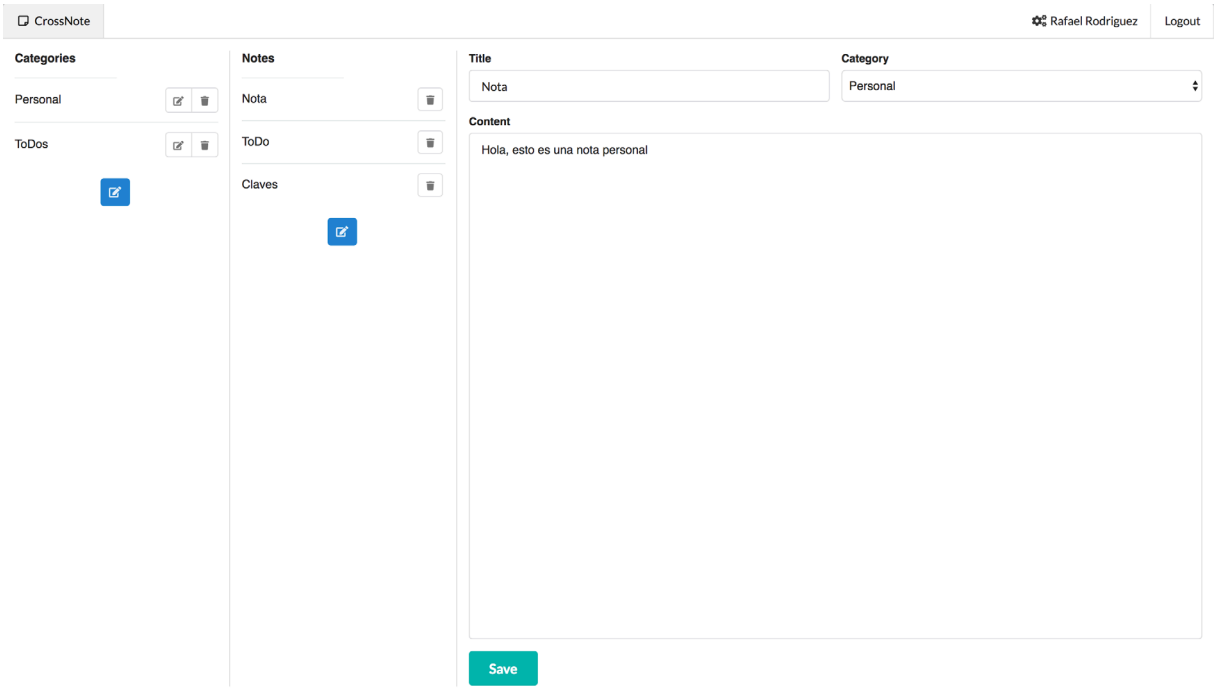

Figura 19 - [Web] Detalle de nota

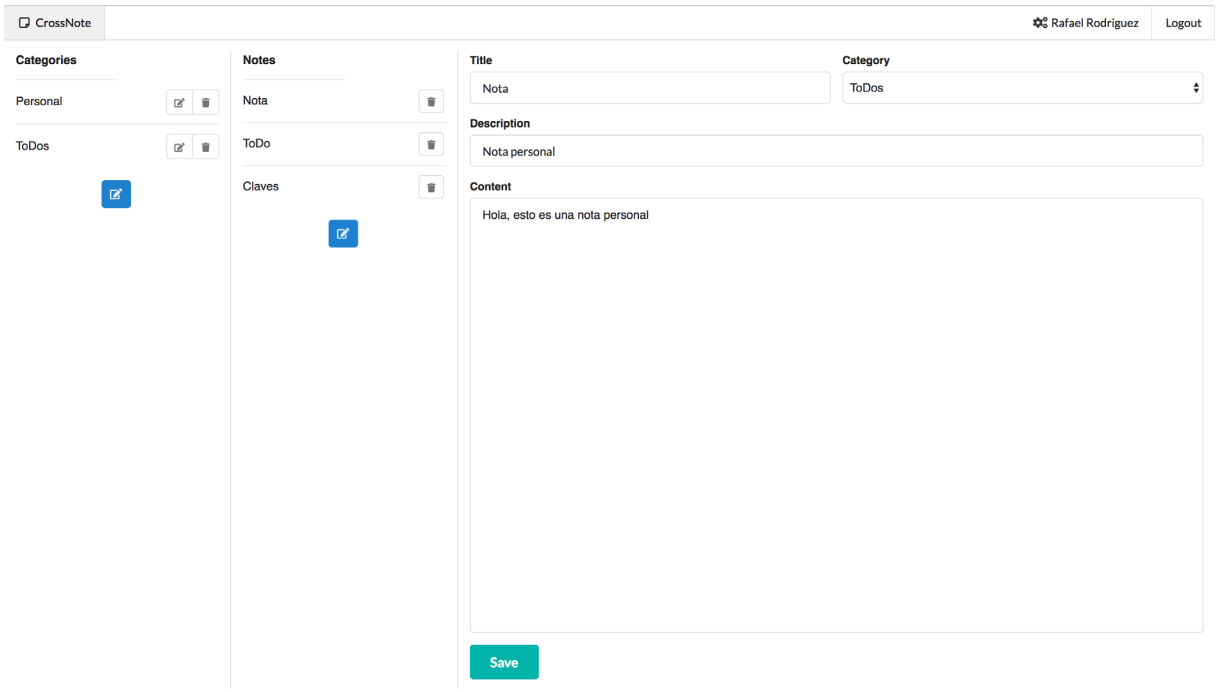

Figura 20 - [Web] Detalle de nota con descripción

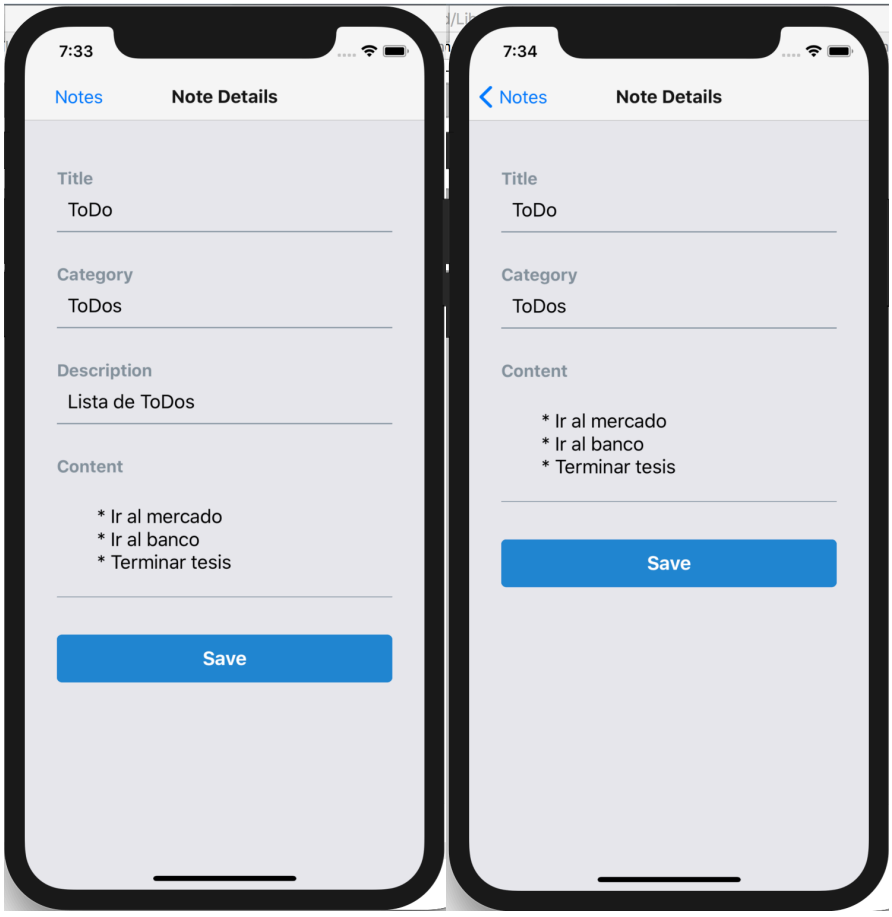

Figura 21 - [iOS] Detalle de nota con y sin descripción

5.4 Análisis de datos e interpretación de resultados

- **Calidad de la experiencia de usuario:**

En las figuras 15 y 16, y en las tablas 9, 10, 11 y 12, se puede apreciar como independientemente de la plataforma objetivo, mediante la aplicación de la metodología es posible realizar una aplicación que cumpla con lo esperado de un aplicativo nativo al punto que esta sea difícil de reconocer como una solución multiplataforma. Los usuarios no pudieron reconocer el desarrollo como una solución multiplataforma en aproximadamente un 95% si tomamos en cuenta a todos los tipos de usuarios y todas las plataformas objetivo evaluadas.

De igual forma se evidencia como la percepción de los usuarios con respecto a la naturaleza de la aplicación evaluada varía poco sin importar la experticia que estos tengan en el área. Los usuarios especializados no pudieron reconocer el desarrollo como una solución multiplataforma en aproximadamente un 92,5% tomando en cuenta todas las plataformas objetivo evaluadas, mientras que los usuarios consumidores de software en un 97,5%.

Por último, también se evidencia que la totalidad de casos en los que la solución planteada fue reconocida como multiplataforma fue en las aplicaciones móviles, principalmente en el sistema operativo Android y por temas de experiencia de usuario. Aún así, los porcentajes de acierto en la identificación de la solución como multiplataforma son bajos. Tomando en cuenta únicamente las plataformas móviles, los usuarios especializados no pudieron reconocer el desarrollo como una solución multiplataforma en aproximadamente un 85% de los casos, mientras que los usuarios consumidores de *software* en un 95%.

- **Complejidad tecnológica**

Se pudo identificar una pila de desarrollo con una base en un único lenguaje de programación y 5 tecnologías (todas ellas usuarias del lenguaje de programación identificado). Para dar contexto a este número vamos a compararlos con una pila de desarrollo nativa. Cabe destacar que Node.js es la tecnología sobre la cual se desarrolló el *backend* mientras que MongoDB es un manejador de bases de datos y supondremos que los usaremos ambos independientemente de la plataforma objetivo para tener una única fuente centralizada de data:

  - **Windows**
    - **Lenguaje de Programación:** C#

        - **Tecnologías:** .NET
    - **MacOS**
        - **Lenguaje de Programación:** Swift
        - **Tecnologías:** Xcode
    - **iOS**
        - **Lenguaje de Programación:** Swift
        - **Tecnologías:** Xcode
    - **Android**
        - **Lenguaje de Programación:** Kotlin
        - **Tecnologías:** Android KTX
    - **Web**
        - **Lenguaje de Programación:** JavaScript
        - **Tecnologías:** React

En caso de haber optado por un desarrollo nativo para las plataformas seleccionadas se habría utilizado una pila de desarrollo con una base de aproximadamente 4 lenguajes de programación y 4 tecnologías, 6 si incluimos Node.js y MongoDB para el desarrollo del *backend* y almacenamiento de data (véase *Figura 22).*

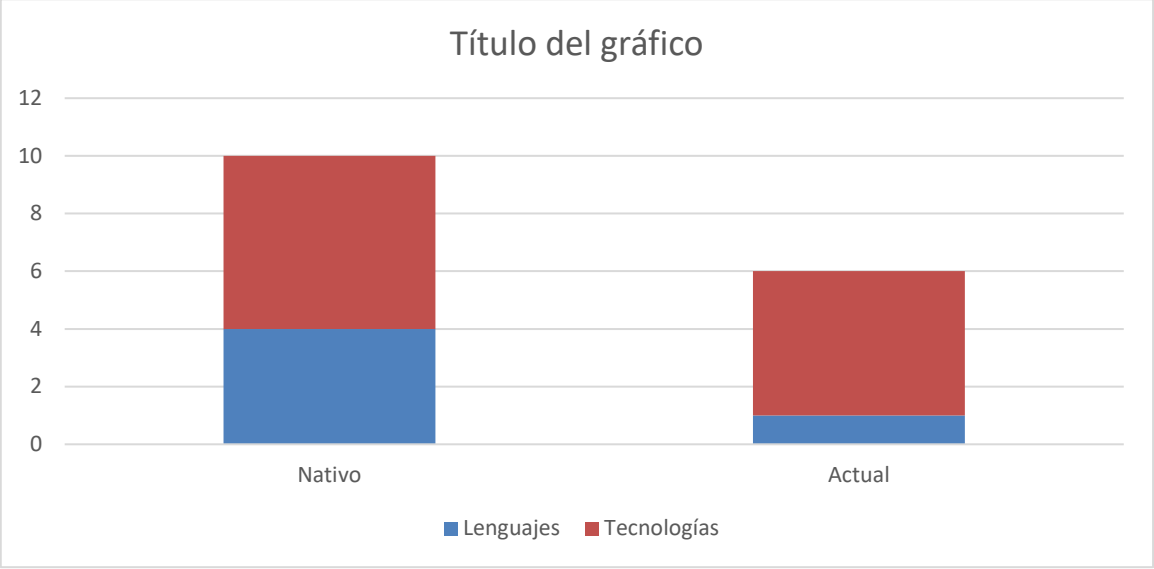

Figura 22 - Comparación de tecnologías

La cantidad de lenguajes y/o tecnologías a utilizar en un proyecto representa retos directos al éxito del proyecto. Algunos de estos retos se pueden ver desde la formación del

equipo de trabajo en el que se deben tener más número de desarrolladores o desarrolladores con mayor número de habilidades y, sin importar el camino a elegir esto repercutirá en forma de aumento en los costos y la complejidad de mantenimiento del proyecto.

Por último cabe destacar que independientemente de la pila de desarrollo a utilizar (nativa o multiplataforma) en el desarrollo suelen utilizarse en mayor o menor medida variedad de librerías auxiliares para lograr con propósitos puntuales como la administración del estado dentro de las aplicaciones y librerías con componentes gráficos, entre otros, sin embargo, estas no son tomadas en cuenta en este análisis.

- **Reutilización de código entre aplicaciones**

Lo primero que podemos observar es que el porcentaje de código reutilizado varía según la plataforma, más específicamente según el tipo de dispositivo. El porcentaje de código reutilizado en las aplicaciones de escritorio es de aproximadamente un 97,6% mientras que en las aplicaciones móviles es de aproximadamente un 57%. Este hallazgo puede ser comprendido por la similitud esperada entre la UI y UX de las aplicaciones web y las aplicaciones de escritorio.

Por último, si tenemos en cuenta las importantes diferencias tanto entre la interfaz como en la experiencia de usuario esperada entre una aplicación web y una móvil el tener una base de código compartida del 57% entre ellas es una cifra importante. Más aún si la aplicación móvil generada tiene la suficiente calidad para difícilmente ser diferenciada de una nativa.

Cabe destacar que la reutilización de código pudiese ser un poco mayor de lo acá planteado ya que partimos de la premisa que los archivos son reutilizados en su totalidad o no son reutilizados. En otras palabras, si un archivo es modificado, sin importar lo pequeño que haya sido el cambio, su código será tomado como si no fue reutilizado en absoluto.

De igual forma al haber utilizado una arquitectura cliente-servidor orientada a Servicios Web Rest en el desarrollo del *Backend* se logró reutilizar en su totalidad las funcionalidades acá generadas en forma de API para todas las aplicaciones independientemente de su plataforma objetivo.

- **Reutilización de código entre aplicaciones para un mismo tipo de plataforma**

Esto puede ser entendido al analizar el funcionamiento de las herramientas de desarrollo multiplataforma y del proyecto CrossNote en si.

Si bien la gran mayoría de las tecnologías de desarrollo multiplataforma generan aplicativos con ligeras variaciones entre si para acoplarse a los elementos de UI y UX propios de cada plataforma (por ejemplo, un picker nativo no es igual en iOS que el Android), dichas tecnologías también permiten el desarrollo de funcionalidades y comportamientos específicos por plataforma. Esta característica dependerá mucho del alcance y características del producto a realizar.

En el caso de CrossNote no fue necesaria la inclusión de ningún comportamiento diferente por sistema operativo más allá de las diferencias nativas brindadas por la tecnología como tal, demostrando que con la elección correcta de herramientas es posible atender de forma automática muchas de esas pequeñas diferencias de UX y UI esperadas por los usuarios de los diferentes sistemas operativos.

- **Reutilización de código dentro de las aplicaciones**

La primera conclusión que se puede obtener es que, siempre y cuando se tenga el mismo alcance, el porcentaje de código reutilizado dentro de las mismas aplicaciones como consecuencia de la aplicación del enfoque modular de la metodología no varía en gran medida por la plataforma objetivo. En promedio se reutiliza aproximadamente un 34% - 35% del código independientemente de la plataforma.

La segunda conclusión que se obtiene es que mediante el enfoque modular de la aplicación se logra reutilizar un significativo 35% del código base, lo que facilita en gran medida el mantenimiento futuro de la aplicación y reduce los tiempos de desarrollo, tamaño del proyecto y por tanto los costos del mismo.

- **Mantenimiento**

Gracias a la naturaleza multiplataforma del desarrollo y la gran base de código compartido existente entre las diferentes aplicaciones generadas se pueden realizar cambios como este, en el cual mediante la modificación de un único archivo compartido logramos agregar características nuevas a todas las plataformas.

En caso de haber sido desarrollos independientes pudiéramos suponer que el cambio hubiese sido 5 veces el actual (uno por cada plataforma, Web, Windows, MacOS, iOS y Android) y, en el caso de ser un cambio de mayor envergadura, hubiese involucrado un importante esfuerzo para asegurar mantener la consistencia entre la experiencia en las diferentes plataformas.

6 Conclusiones

En este capítulo se resumen las principales consecuencias y conclusiones de la investigación. De igual forma se mencionan posibles trabajos, recomendaciones o líneas de investigación a futuro.

6.1 Conclusiones

El presente proyecto de investigación cumple con los objetivos planteados de estudiar la situación actual de los desarrollos multiplataforma y, en base a esta, desarrollar, probar y ejemplificar una metodología incrementable y personalizable mediante la cual se puedan reducir los tiempos y costos de desarrollo de *software*, a la vez que se minimizan las posibles inconsistencias en la experiencia del usuario en aplicaciones implementadas de forma multiplataforma a través de la modularidad y reutilización de lógica.

La metodología propuesta fue llamada Modular Cross-platform Development Agile Methodology (MCPDev por sus siglas en inglés) y brinda a cualquier desarrollador interesado de conceptos, consideraciones tecnológicas, paradigmas de programación, artefactos a generar y estructura formal (etapas y fases) a utilizar en los desarrollos multiplataforma. De igual forma se identifican todas las personas involucradas o de interés para el desarrollo así como sus roles y responsabilidades en el proceso.

En particular, las principales características y contribuciones de esta metodología son:

- **Desarrollo multiplataforma:** Su alcance, lejos de estar limitado a un ámbito en particular, busca poder ser aplicado en cualquier proyecto multiplataforma independientemente de las plataformas objetivo.
- **Framework Fullstack:** No ve las aplicaciones como únicamente el cliente mediante el cual interactúan los usuarios o las capaz de datos y lógica de negocio. Ve las aplicaciones como un todo y se involucra en todo su ciclo de vida.
- **Modularidad y Reusabilidad del código y lógica:** Brinda lineamientos para la selección tecnológica pero se fundamenta en conceptos y practicas generales con el fin de lograr una independencia tecnológica y de plataforma. Se enfoca en el como usar más que en el que usar.
- **Metodología de desarrollo de software:** En lugar de enfocarse únicamente en los aspectos prácticos de un proyecto, brinda una metodología formal que permite gestionar el ciclo de vida de un proyecto mediante un proceso bien definido. Cierra la

brecha entre el qué hacer y el cómo hacerlo a la hora de abordar un desarrollo multiplataforma.

Por último, la evaluación de la metodología fue realizada mediante el desarrollo de un caso de estudio y la revisión de los siguientes puntos críticos sobre el producto obtenido:

- Calidad de la experiencia de usuario
- Complejidad tecnológica
- Reutilización de código entre aplicaciones
- Reutilización de código entre aplicaciones para un mismo tipo de plataforma
- Reutilización de código dentro de las aplicaciones
- Mantenimiento
-

Los resultados obtenidos confirman que la metodología es adecuada para ser aplicada en la gestión y desarrollo de proyectos multiplataforma, permitiendo reducir los tiempos y costos involucrados y manteniendo la calidad y consistencia deseada en la experiencia de usuario.

Cabe destacar que el objetivo principal son los desarrollos multiplataforma, por lo que queda pendiente estudiar su adecuación en desarrollos netamente web o de alguna otra plataforma en particular.

6.2 Recomendaciones a futuro

Existen principalmente dos líneas de trabajo futuro de este trabajo. Por un lado, podrían introducirse algunas mejoras en la metodología propuesta, tal y como se enumera a continuación:

- Agregar conceptos y buenas prácticas para implementar integración continua y entrega continua a la metodología.
- Agregar conceptos y buenas prácticas a la hora de implementar capas de *Backend For Frontend* (BFF por sus siglas en inglés).
- Agregar / generar herramienta que facilite el uso compartido de componentes y envoltorios entre diferentes aplicaciones, facilitando el mantenimiento y el desarrollo de los mismos.

- Agregar el concepto de **Diccionario de Estilos**, cuyo objetivo es ser un repositorio centralizado de estilos (colores, tipos de fuente, entre otros) para todas las aplicaciones, apoyando la consistencia de UI y facilitando la mantenibilidad futura.

Por otra parte, podrían extenderse las pruebas de la metodología de la siguiente forma:

- Incluir funcionalidades específicas de cada plataforma como las notificaciones *Push*, el acceso a cámara y archivos del sistema, entre otros.
- Revisar el desempeño de las aplicaciones obtenidas por la metodología desde el punto de vista del consumo de recursos y desempeño real del aplicativo mediante herramientas especializadas (más allá de lo que el usuario pueda percibir en su interacción), y tomar las medidas necesarias para incluir este aspecto como característica a controlar por la metodología.
- Realizar un experimento en el cual, adicional a las aplicaciones obtenidas como resultado de la metodología, se realicen aplicaciones nativas para cada una de las plataformas objetivo de forma tal que se puedan realizar comparaciones entre ellas.

Anexos

Encuesta de Investigación

Objetivo

La presente sirve como instrumento estadístico para el análisis de calidad en la experiencia de usuario en aplicaciones multiplataforma.

Instrucciones

El encuestador le facilitará uno o más dispositivos en los cuales probará una o más versiones de una aplicación para la gestión de notas personales (*CrossNote*).

Sin recibir más detalles del encuestador, usted debe realizar todas las pruebas que considere necesarias sobre el aplicativo hasta sentirse familiarizado con el mismo y, posteriormente proceda a llenar el siguiente formulario.

Al finalizar por favor retorne el documento y el equipo al encuestador.

Es importante que recuerde que debe llenar un formulario por cada aplicación que pruebe.

Tipo de dispositivo:

- ☐ Computador personal
- ☐ Móvil

Sistema Operativo:

- ☐ Android
- ☐ iOS
- ☐ MacOS
- ☐ Windows

Tipo de desarrollo:

- ☐ Nativo
- ☐ Multiplataforma

En caso de haber elegido la opción '*Multiplataforma*' en el apartado anterior, ¿cuál de las siguientes opciones describe mejor su justificación?:

- ☐ Desempeño
- ☐ Experiencia de usuario (UX)
- ☐ Interfaz de usuario (UI)
- ☐ Otra:

Anexo 1 - Encuesta de calidad de la experiencia de usuario

Acta de constitución del Proyecto

Fecha:
11 de mayo de 2.018

Participantes (equipo):
CLIENTE
EQUIPO

Fecha estimada de inicio:
14 de mayo de 2.018

Fecha estimada de fin (opcional):
03 de junio de 2.018

Proyecto:
CrossNote

Plataformas objetivo

Escritorio: MacOS 10.13 o mayores y Windows 10 o mayores.
Móvil: Android 8.0 y mayores y iOS 11 y mayores.
Web: Para efectos de la prueba solo se trabajará con Google Chrome desde la versión 64.0.0 en adelante.

Temática

Aplicación multiplataforma para la gestión de notas en las que un usuario va a poder visualizar y modificar sus notas en cualquiera de los dispositivos soportados.

Alcance

Manejo de usuarios:

- Estructura de usuarios: Nombres, apellidos, usuario, correo, contraseña.
- Acciones permitidas a los usuarios: Inscribirse, iniciar sesión, cerrar sesión, eliminar su cuenta y modificar sus datos.

Manejo de categorías:

- Las notas están divididas por categorías y son propias para cada usuario.
- La plataforma agrega 2 categorías por defecto:
    - Todas: Muestra todas las notas independientemente de la categoría a la que pertenezcan.
    - Recientemente agregadas: Muestra las notas agregadas en los últimos 3 días.
    - Recientemente eliminadas: Muestra las notas eliminadas en los últimos 3 días.
- Acciones permitidas: Crear categoría, modificar categoría, eliminar categoría (solo si no tiene notas asociadas).

Manejo de notas:

- Tienen título y contenido.
- El contenido no tiene límite definido de caracteres mientras que el título si (60 caracteres)
- Pueden o no tener una categoría pero en caso de tenerla es única por nota.

- Acciones permitidas: Crear, modificar, eliminar nota, asociar y desasociar nota a una categoría.

Tecnologías

Lenguaje: JavaScript
Backend: Node.js, ExpressJS, MongoDB.
Frontend: React, Redux, React Native, Electronjs, Semantic UI.

Compromisos

El CLIENTE y el EQUIPO se comprometen a una unión laboral de un (1) único sprint de desarrollo en el cual, previo conceso entre las partes se definirá un alcance funcional del proyecto a ser desarrollado y entregado.

Anexo 2 - Acta de constitución del Proyecto

Acta de cierre de Sprint

Proyecto	Sprint
CrossNote	1

Puntos acordados	Puntos realizados	Puntos pendientes
69	69	0

Actividades acordadas

- Configuración del ambiente de desarrollo - 8pts
- Desarrollo de CRUD para usuarios - 5pts
- Desarrollo de CRUD para categorias - 5pts
- Desarrollo de CRUD para notas - 5pts
- [Web] Vista login / signup - 5 pts
- [Web] Vista perfil - 5 pts
- [Web] Vista categorias - 5pts
- [Web] Vista notas - 5pts
- [Web] Vista detalle de nota - 5pts
- [Escritorio] App de escritorio - 8 pts
- [Móvil] App móvil - 13pts

Actividades realizadas

- Configuración del ambiente de desarrollo - 8pts
- Desarrollo de CRUD para usuarios - 5pts
- Desarrollo de CRUD para categorias - 5pts
- Desarrollo de CRUD para notas - 5pts
- [Web] Vista login / signup - 5 pts
- [Web] Vista perfil - 5 pts
- [Web] Vista categorias - 5pts
- [Web] Vista notas - 5pts
- [Web] Vista detalle de nota - 5pts
- [Escritorio] App de escritorio - 8 pts
- [Móvil] App móvil - 13pts

Actividades pendientes por realizar

Anexo 3 - Acta cierre del sprint

Retrospectiva

Sprint
1

¿Qué se hizo bien?

- Se cumplieron todos los ítems acordados.
- La metodología se aplicó al pie de la letra a pesar de ser la primera experiencia del equipo utilizándola.

¿Qué se pudo hacer mejor?

- La estimación de esfuerzo para las actividades se pudo realizar mejor. En general, todas las actividades conllevaron mas horas/hombre de las pensadas. Las actividades hubiesen podido ser fácilmente divididas en 2 sprints con los recursos actuales del equipo.
- No se tomó en cuenta el esfuerzo para desarrollar pruebas unitarias en la estimación por lo que no pudieron ser realizadas.

Compromisos y acuerdos para el próximo sprint

- Hacer una mejor estimación de esfuerzos
- Incluir pruebas unitarias para todas las actividades a desarrollar y aquellas realizadas en este sprint.

Anexo 4 - Retrospectiva Sprint 1

Tecnologías

Repositorio de tecnologías involucradas que deben ser gestionadas por el equipo y sus versiones. Existen más tecnologías involucradas, pero estas y sus versiones son manejadas por el proyecto en si. Este es un documento interno de referencia para el equipo del proyecto.

Editor de código: VSCode (1.24)
Lenguaje: JavaScript (ES6)
Backend: Node.js (8.11.2), MongoDB (2.1.20)

Accesos

Repositorio de herramientas involucradas y accesos. Este es un documento interno de referencia para el equipo del proyecto. En caso de no tener acceso a alguna de las herramientas por favor comunicarse con soporte.

Comunicaciones – Hangouts
Correo – Gmail
Tablero Kanban – Trello
Wiki – Google Drive
Calendario – Google Calendar
Repositorios – Bitbucket: crossnote_ms_user, crossnote_ms_notes, crossnote_fe_web, crossnote_fe_desktop, crossnote_fe_mobile

Anexo 5 - Tecnologías y Accesos

Acta de cierre del Proyecto

Fecha:
03 de Junio del 2.018

Participantes (equipo):
CLIENTE
EQUIPO

Fecha de inicio:
14 de Mayo del 2.018

Fecha estimada de fin (opcional):
03 de Junio del 2.018

Proyecto:
CrossNote

Fecha de fin:
03 de Junio del 2.018

Plataformas objetivo

Escritorio: MacOS 10.13 o mayores y Windows 10 o mayores.
Móvil: Android 8.0 y mayores y iOS 11 y mayores.
Web: Para efectos de la prueba solo se trabajará con Google Chrome desde la versión 64.0.0 en adelante.

Temática

Aplicación multiplataforma para la gestión de notas en las que un usuario va a poder visualizar y modificar sus notas en cualquiera de los dispositivos soportados.

Alcance

Manejo de usuarios:

- Estructura de usuarios: Nombres, apellidos, usuario, correo, contraseña.
- Acciones permitidas a los usuarios: Inscribirse, iniciar sesión, cerrar sesión, eliminar su cuenta y modificar sus datos.

Manejo de categorías:

- Las notas están divididas por categorías y son propias para cada usuario.
- La plataforma agrega 2 categorías por defecto:
  - Todas: Muestra todas las notas independientemente de la categoría a la que pertenezcan.
  - Recientemente agregadas: Muestra las notas agregadas en los últimos 3 días.
  - Recientemente eliminadas: Muestra las notas eliminadas en los últimos 3 días.
- Acciones permitidas: Crear categoría, modificar categoría, eliminar categoría (solo si no tiene notas asociadas).

Manejo de notas:

- Tienen título, descripción y contenido.
- El contenido no tiene límite definido de caracteres mientras que el título si (60 caracteres)
- Pueden o no tener una categoría pero en caso de tenerla es única por nota.
- Acciones permitidas: Crear, modificar, eliminar nota, asociar y desasociar nota a una categoría.

Tecnologías

Lenguaje: JavaScript
Backend: Node.js, ExpressJS, MongoDB.
Frontend: React, Redux, React Native, Electronjs, Semantic UI.

Conformidad

Mediante la presente aquellos acá expuestos, CLIENTE y EQUIPO, dan por finalizado el proyecto y reconocen como real toda la información presentada sobre la naturaleza y alcance del mismo. De igual forma, se reconoce que se hizo entrega de los respaldos de todo el trabajo realizado y aprobado en cada iteración del proceso (actas de cierre de sprints) y del producto final de este proyecto de desarrollo de software por medio del traspaso de la propiedad de los repositorios del código.

se comprometen a una unión laboral de un (1) único sprint de desarrollo en el cual, previo conceso entre las partes se definirá un alcance funcional del proyecto a ser desarrollado y entregado.

Firma CLIENTE **Firma EQUIPO**

Anexo 6 - Acta de cierre del Proyecto

Lecciones Aprendidas

Fecha	Proyecto
03 de Junio del 2.018	CrossNote

- Las aplicaciones hijas no deben tomarse como una única actividad. En el proyecto actual se realizo la plataforma base (web) y luego cada una de las plataformas hijo (todas las demás versiones) fueron tomadas como una única actividad dentro del proceso de desarrollo. En su lugar debió crearse una actividad por cada funcionalidad o flujo. Los desarrollos multiplataforma suelen compartir una base importante de código, sin embargo, suele se requerido un trabajo extra de adaptación a cada nueva plataforma y es más sencillo manejarlo al dividir las tareas de una forma más unitaria.
- En la medida de lo posible es bueno cumplir con la recomendación de 3 integrantes mínimo por equipo. El trabajo en paralelo suele afectar el desempeño del proyecto.

Anexo 7 - Lecciones Aprendidas

Bibliografía

[1] J. Lewis y M. Fowler, «MartinFowler.com,» 25 March 2014. [En línea]. Available: https://martinfowler.com/articles/microservices.html. [Último acceso: 05 Junio 2018].

[2] C. Cubillos, «Pontificia Universidad Católica de Valparaíso,» [En línea]. Available: http://ocw.pucv.cl/cursos-1/arquitectura-de-sistemas-de-software/materiales-de-clases/web-cliente-servidor. [Último acceso: 05 Junio 2018].

[3] Oracle, «The Java EE 6 Tutorial - What Are RESTful Web Services?,» 2013. [En línea]. Available: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html. [Último acceso: 05 Junio 2018].

[4] C. Rudrakshi, A. Varshney, B. Yadla, R. Kanneganti y K. Somalwar, «API-fication | Core Building Block of the Digital Enterprise,» 08 2014. [En línea]. Available: http://dsonline.computer.org/portal/site/dsonline/menuitem.9ed3d9924aeb0dcd82ccc6716bbe36ec/index.jsp?&pName=dso_level1&path=dsonline/2008/09&file=w5gei.xml&xsl=article.xsl. [Último acceso: 05 Junio 2018].

[5] C. Baldwin y K. Clark, Design Rules: The power of modularity, 2000.

[6] «University Stanford - What is JavaScript?,» [En línea]. Available: http://web.stanford.edu/class/cs98si/slides/overview.html. [Último acceso: 05 Junio 2018].

[7] «Linux Foundation Collaborative Projects - Node.js,» [En línea]. Available: https://nodejs.org/. [Último acceso: 05 Junio 2018].

[8] «Facebook Open Source - React,» [En línea]. Available: https://reactjs.org/. [Último acceso: 05 Junio 2018].

[9] «Facebook Open Source - React Native,» [En línea]. Available: https://facebook.github.io/react-native/. [Último acceso: 05 Junio 2018].

[10] D. Abramov, «Redux,» [En línea]. Available: https://redux.js.org/. [Último acceso: 05 Junio 2018].

[11] Electronjs, «Electronjs,» [En línea]. Available: https://electronjs.org/. [Último acceso: 05 junio 2018].

[12] Semantir-UI, «Semantir-UI React,» [En línea]. Available: https://react.semantic-ui.com/introduction. [Último acceso: 05 Junio 2018].

[13] PMI, «Project Management Institute,» [En línea]. Available: https://www.pmi.org/learning/featured-topics/methodology. [Último acceso: 05 Junio 2018].

[14] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. Martin, S. Mellor, K. Schwaber, J. Sutherland y Tho, «Agile Manifesto - Principles,» 2001. [En línea]. Available: http://agilemanifesto.org/iso/es/principles.html. [Último acceso: 05 Junio 2018].

[15] «Scrum Alliance - Learn About Scrum,» [En línea]. Available: https://www.scrumalliance.org/learn-about-scrum. [Último acceso: 05 Junio 2018].

[16] D. Radigan, «Atlassian Agile Coach - Kanban,» [En línea]. Available: https://www.atlassian.com/agile/kanban. [Último acceso: 05 Junio 2018].

[17] Atlassian, «Bitbucket,» [En línea]. Available: https://bitbucket.org/. [Último acceso: 05 Junio 2018].

[18] Atlassian, «Trello,» [En línea]. Available: https://trello.com/. [Último acceso: 05 Junio 2018].

[19] Google, «Google Drive,» [En línea]. Available: https://www.google.com/drive/. [Último acceso: 05 Junio 2018].

[20] Microsoft, «Visual Studio Code,» [En línea]. Available: https://code.visualstudio.com/. [Último acceso: 05 Junio 2018].

[21] T. A. Majchrzak, A. Biørn-Hansen Westerdals y T.-M. Grønli Westerdals, «Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks,» de *50th Hawaii International Conference on System Sciences*, Hawaii, 2017.

[22] M. Martinez y S. Lecomte, «Towards the quality improvement of cross-platform mobile applications,» de *IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems*, Buenos Aires, 2017.

[23] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef y A. M. Wahba, «Taxonomy of Cross-Platform Mobile Applications Development Approaches,» *Ain Shams Engineering Journal,* vol. 8, nº 2, pp. 163-190, 2017.

[24] L. Gaouar, A. Benamar y F. Tarik Bendimerad, «Desirable Requirements of Cross Platform Mobile Development Tools,» 2016.

[25] M. C. Enache, «Cross-Platform Technologies,» *Annals of Dunarea de Jos University. Fascicle I : Economics and Applied Informatics,* vol. 23, nº 1, p. 100, 2017.

[26] S. Whitaker, 2014. [En línea]. Available: https://www.pmi.org/learning/library/making-project-methodology-fit-guide-6085. [Último acceso: 05 Junio 2018].