



Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería de Software

**Framework ágil para realizar pruebas funcionales de
software orientado a aplicaciones móviles, caso:
aplicación móvil para transferencias bancarias
desarrollada bajo metodología Scrum**

INFORME DE TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el Título Profesional de Ingeniero de Software

AUTOR

Stephany Gianina DANERI BELTRÁN

ASESOR

Luis Alberto ALARCÓN LOAYZA

Lima, Perú

2017



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

Referencia bibliográfica

Daneri, S. (2017). *Framework ágil para realizar pruebas funcionales de software orientado a aplicaciones móviles, caso: aplicación móvil para transferencias bancarias desarrollada bajo metodología Scrum* [Informe de Trabajo de Suficiencia Profesional de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Escuela Profesional de Ingeniería de Software]. Repositorio institucional Cybertesis UNMSM.



UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
Escuela Profesional de Ingeniería de Software

Acta de Sustentación del
Trabajo de Suficiencia Profesional

Siendo las 19:00 horas del día 22 de febrero del año 2018, se reunieron los docentes designados como Miembros de Jurado del Trabajo de Suficiencia Profesional, presidido por el Ing. Bartra More Arturo Alejandro (Presidente), Lic. Román Concha Norberto Ulises (Miembro) y el Lic. Alarcón Loayza Luis Alberto (Miembro Asesor) para la sustentación del Trabajo de Suficiencia Profesional Intitulado: **"FRAMEWORK ÁGIL PARA REALIZAR PRUEBAS FUNCIONALES DE SOFTWARE ORIENTADO A APLICACIONES MÓVILES, CASO: APLICACIÓN MÓVIL PARA TRANSFERENCIAS BANCARIAS DESARROLLADA BAJO METODOLOGÍA SCRUM"**, por la Bachiller: **Daneri Beltrán, Stephany Gianina**; para obtener el Título Profesional de Ingeniera de Software.

Acto seguido de la exposición del Trabajo de Suficiencia Profesional, el Presidente invitó a la Bachiller a dar las respuestas a las preguntas establecida por los miembros del Jurado.

La Bachiller en el curso de sus intervenciones demostró pleno dominio del tema, al responder con acierto y fluidez a las observaciones y preguntas formuladas por los señores miembros del Jurado.

Finalmente habiéndose efectuado la calificación correspondiente por los miembros del Jurado, la Bachiller obtuvo la nota de 18 (En letras) Dieciocho.

A continuación el presidente de jurados el Ing. Bartra More Arturo, declara a la Bachiller Ingeniera de Software.

Siendo las 20:15 horas, se levantó la sesión.

Presidente

Ing. Bartra More Arturo Alejandro

Miembro

Lic. Román Concha Norberto Ulises

Miembro Asesor

Lic. Alarcón Loayza Luis Alberto

DEDICATORIA:

A mis padres, Rossana Beltrán Segura y
Rómulo Daneri Francia, porque sin su
apoyo constante mis planes no se hubieran
convertido en logros.

A mis abuelos Luis Enrique Daneri y Lucía Francia
A mis abuelos Máximo Beltrán y Norma Segura.
A Enrique Palomino Solano y Mirtha Beltrán.
A Leonor Beltrán, Linda Beltrán, Rosario
Beltrán, Belinda Beltrán y Zoila Sullón.

A toda mi familia, por siempre preocuparse
por mi crecimiento profesional.

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE

**Un Framework Ágil para realizar Pruebas Funcionales de Software orientado a
Aplicaciones Móviles, Caso: Aplicación Móvil para Transferencias Bancarias
desarrollada bajo Metodología Scrum**

Titulando: DANERI BELTRÁN, STEPHANY GIANINA

Asesor: ALARCÓN LOAYZA, LUIS ALBERTO

Título: Informe, para optar el Título Profesional de Ingeniero de Software

Fecha: Diciembre del 2017

RESUMEN

La realización de pruebas sobre aplicaciones móviles son mucho más complejas que las pruebas realizadas para aplicaciones web o de escritorio, ya que existen diferentes factores que amplían drásticamente el espectro de pruebas. Al seguir metodologías ágiles se deben realizar actividades de control de calidad en cada funcionalidad construida. Idealmente cada funcionalidad debería ser completada en una o dos semanas, y ello evidencia que se tienen tiempos muy cortos para garantizar su calidad. Por todo lo anteriormente mencionado, el presente informe de experiencia profesional describe un Framework Ágil para realizar Pruebas de Software donde se establece el proceso de trabajo a seguir, se dan pautas de cómo definir casos de prueba y de cómo construir pruebas automatizadas que respalden la implementación de una funcionalidad a nivel de servicios backend y a nivel de aplicativo móvil. El caso contexto de este informe es el de una aplicación móvil para las plataformas Android y iOS cuya funcionalidad principal son las transferencias bancarias. El autor del presente fue la ingeniera de calidad del producto a cargo. Actualmente, la aplicación cumple con la calidad esperada por el cliente y se encuentra publicada en Play Store (Android) y App Store (iOS).

Palabra claves: pruebas automatizadas, testing ágil, scrum, control de calidad, pruebas de aplicaciones móviles

MAJOR NATIONAL UNIVERSITY OF SAN MARCOS
FACULTY OF SYSTEMS ENGINEERING

ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE

**An Agile Framework for performing Functional Software Testing oriented to
Mobile Applications, Case: Mobile Application for Bank Transfers developed
under the Scrum Methodology**

Titulando: DANERI BELTRÁN, STEPHANY GIANINA

Asesor: ALARCÓN LOAYZA, LUIS ALBERTO

Título: Informe, para optar el Título Profesional de Ingeniero de Software

Fecha: Diciembre del 2017

ABSTRACT

Testing on mobile applications is much more complex than testing for web or desktop applications, since there are different factors that dramatically broaden the spectrum of tests. When following agile methodologies, quality control activities must be carried out in each constructed functionality. Ideally, each functionality should be completed in one or two weeks, and this shows that there are very short times to guarantee its quality. For all the aforementioned, this report of professional experience describes an Agile Framework for Software Testing where the work process to be followed is established, guidelines are given on how to define test cases and how to build automated tests that support the implementation of functionality at the backend services level and at the mobile application level. The case context of this report is that of a mobile application for Android and iOS platforms whose main functionality is bank transfers. The author of the present was the quality engineer of the product in charge. Currently, the application meets the quality expected by the client and is published in Play Store (Android) and App Store (iOS).

Key words: automated testing, agile testing, scrum, quality control, mobile application testing

TABLA DE CONTENIDO

Carátula externa o pasta	i
Página en blanco	ii
Carátula interna	iii
Ficha Catalográfica	iv
Dedicatoria	v
RESUMEN	vi
ABSTRACT	vii
Tabla de Contenido	viii
Índice de Figuras	x
Índice de Tablas	x
INTRODUCCIÓN	1
CAPÍTULO I - TRAYECTORIA PROFESIONAL	3
1.1 EXPERIENCIA PROFESIONAL	3
1.2 FORMACIÓN	5
1.3 IDIOMAS	5
1.4 CERTIFICACIONES / CURSOS	5
1.5 CONOCIMIENTOS / HABILIDADES	6
CAPÍTULO II - CONTEXTO EN EL QUE SE DESARROLLÓ LA EXPERIENCIA	8
2.1 EMPRESA - ACTIVIDAD QUE REALIZA	8
2.1.1 Datos de la empresa	8
2.2 VISIÓN	9
2.3 MISIÓN	9
2.4 ORGANIZACIÓN DE LA EMPRESA	10
2.5 ÁREA, CARGO Y FUNCIONES DESEMPEÑADAS	10

2.6 EXPERIENCIA PROFESIONAL REALIZADA EN LA ORGANIZACIÓN	11
CAPÍTULO III - ACTIVIDADES DESARROLLADAS	11
3.1 SITUACIÓN PROBLEMÁTICA	12
3.1.1 Definición del problema	12
3.2 SOLUCIÓN	16
3.2.1 Objetivos	17
Objetivo general	17
Objetivos específicos	17
3.2.2 Alcance	17
Dentro del alcance	17
Fuera del alcance	18
3.2.3 Conceptos fundamentales	19
3.2.4 Framework desarrollado	25
3.2.4.1 Definición de la estrategia a seguir	27
3.2.4.2 Buenas prácticas para la definición de Historias de usuario	37
3.2.4.3 Lineamientos generales para el diseño y construcción de pruebas	41
3.2.5 Contribuciones obtenidas en su aplicación en el caso contexto	55
3.3 EVALUACIÓN	63
3.3.1 Evaluación económica	63
CAPÍTULO IV. REFLEXIÓN CRÍTICA DE LA EXPERIENCIA	65
CAPÍTULO V. CONCLUSIONES Y RECOMENDACIONES	66
5.1 CONCLUSIONES	66
5.2 RECOMENDACIONES	68
5.3 FUENTES DE INFORMACIÓN	69
5.4 GLOSARIO	70

Índice de Figuras

Figura	Pág
Figura 01: Organigrama de la empresa	9
Figura 02: Framework Ágil para realizar pruebas de software orientado a Aplicaciones Móviles. Fuente propia.	26
Figura 03: Acciones a realizar en la Backlog Grooming establecidas por el Framework ágil para pruebas de software. Fuente propia.	30
Figura 04: Tablero de trabajo dentro del sprint. Fuente propia.	35
Figura 05: Arquitectura del framework de pruebas automatizadas para la aplicación móvil de transferencias bancarias. Fuente propia	59
Figura 06: Ejemplo de una definición de pruebas basada en requisitos de negocio. Fuente propia.	60
Figura 07: Ejemplo del desarrollo de la capa de comunicación entre los requisitos de negocio y la definición de las vistas del aplicativo móvil. Fuente propia.	61
Figura 08: Definición de los elementos y controles de una vista de aplicativo móvil. Fuente propia.	62

Índice de Tablas

Tabla	Pág
Tabla 01: Buenas prácticas para establecer buenas definiciones de Historia de Usuario. Fuente propia.	40
Tabla 02: Estructura para especificaciones de casos de prueba y explicación de cada punto. Fuente propia.	42

INTRODUCCIÓN

El presente informe describe el esfuerzo hecho para establecer un Framework Ágil para realizar Pruebas de Software orientado a Aplicaciones Móviles.

Belatrix Software Factory es una empresa argentina que brinda servicios de desarrollo, calidad y prueba de software a clientes que en su mayoría son norteamericanos. Fue creada en 1993 y que en el 2011 estableció una sede en Perú; hoy, Belatrix cuenta con cinco sedes, tres en Argentina, una en Perú y una en Colombia; adicionalmente cuenta con oficinas comerciales en USA.

A finales del 2015, una importante entidad financiera iniciaba con la creación de su centro de innovación. A inicios del 2016, deseaba sumar personas con perfiles especializados en desarrollo, aseguramiento de calidad y prueba de software con experiencia trabajando con metodologías ágiles para diversos proyectos que se empezaban a desarrollar. El perfil de los ingenieros de calidad requería que tuvieran experiencia con Agile Testing y que hayan desarrollado esfuerzos de automatización de pruebas en diferentes niveles. Las actuales empresas proveedoras de dicha entidad financiera no encontraban personas que encajaran con el perfil, por lo que Belatrix empezó a brindarle estos talentos.

Hacer transferencias de forma intuitiva, cuyo proceso sea fácilmente entendible por un público joven y que le haga frente al uso del dinero en efectivo son las características del aplicativo móvil que bajo metodología Scrum, fue construido teniendo en mente que en su tiempo de vida debía seguir agregando funcionalidades útiles e incorporar mejoras continuas en base a la retroalimentación obtenida por los usuarios.

Un equipo multidisciplinario inició con el desarrollo de un aplicativo móvil para transferencias. La construcción de un aplicativo requería establecer una forma de trabajo que

permitiera alta coordinación entre todos los miembros del equipo para así mostrar software funcionando según lo especificado cada semana.

Es así que los esfuerzos de QA estuvieron orientados a crear un Framework Ágil para Pruebas Funcionales de Software orientado aplicaciones móviles que permitiera trabajar de la forma más cohesionada posible a Desarrollo y a QA, tal que se pudiera mostrar software funcionando cada sprint.

En el CAPÍTULO I se especifican cronológicamente roles y funciones del autor, actividades, aprendizaje empírico y formal. También se precisa la experiencia significativa.

En el CAPÍTULO II se describe parte de la historia de la empresa Belatrix Software Factory, su visión, misión, organización, los servicios que brinda y adicionalmente la organización del equipo de trabajo en el que se desarrolló la experiencia y que se encuentra en el sitio del cliente.

El CAPÍTULO III se da el detalle del trabajo realizado el cual se refiere en este caso a un Framework Ágil para realizar Pruebas Funcionales de Software orientado a una aplicación móvil de transferencias bancarias desarrollada bajo metodología Scrum.

El CAPÍTULO IV se refiere al aporte del autor, el desarrollo profesional que le demandó, las necesidades que se atendieron, la experiencia y la capacitación requeridas.

En el CAPÍTULO V se detallan las conclusiones y también se indican las recomendaciones.

CAPÍTULO I - TRAYECTORIA PROFESIONAL

Grado Académico de Bachiller en Ingeniería de Software – Escuela Académico Profesional de Ingeniería de Software – Facultad de Ingeniería de Sistemas e Informática – Universidad Nacional Mayor de San Marcos, con experiencia mayor a 4 años como Ingeniero de Aseguramiento de la Calidad Automatizador.

Experiencia en proyectos web y móviles de diferentes rubros: bancario, educación, farmacéutico y logístico, la mayor parte de estos proyectos pertenecientes a clientes de USA. Esta es la trayectoria profesional de la autora de este informe:

1.1 EXPERIENCIA PROFESIONAL	
QA Automation Engineer Senior Belatrix Software Factory Proyecto: <i>Aplicación Móvil de pagos para importante entidad financiera (desde el 01.04.2016)</i> - Responsable por la definición de casos de prueba - Ejecución de pruebas manuales en las plataformas Android & iOS - Responsable por el desarrollo de pruebas automatizadas de integración, para móviles con Appium usando BDD en Ruby. - Desarrollo de un sistema soporte mock de message broker para pruebas.	01.07.2017 - hasta la fecha

QA Automation Engineer III Belatrix Software Factory Proyecto: <i>Apertura de cuentas de ahorro para importante entidad financiera (hasta el 31.03.2016)</i> - Definición y ejecución de casos de prueba - Desarrollo de pruebas automatizadas con Robot Framework Proyecto: <i>Aplicación Móvil de pagos para importante banco peruano (desde el 01.04.2016)</i>	01.02.2016 - 30.06.2017
QA Engineer II Belatrix Software Factory Proyecto: <i>Plataforma web para una asociación médica profesional de USA</i> - Coordinación remota con otros miembros del equipo de QA en Argentina para diseño de casos de prueba. - Ejecución de pruebas y reporte de incidencias - Desarrollador de un framework para pruebas automatizadas en C#, usando Specflow	01.06.2015 - 31.01.2016
QA Engineer II Belatrix Software Factory Proyecto: <i>Plataforma backend de Home Banking para cliente de</i>	01.12.2014 - 31.05.2015

<i>USA</i> - Responsable de trabajar junto con los product owner para la definición de requisitos. - Diseño de pruebas junto con otros miembros del equipo de QA. - Ejecución de pruebas y reporte de incidencias - Desarrollador de un framework para pruebas automatizadas en Java	
QA Engineer (Professional Practitioner & Intern) Belatrix Software Factory Proyecto: <i>Aplicación Web de tendencias médicas para una compañía startup de USA</i> - Responsable del análisis y definición de requisitos con el Product Owner. - Ejecución de pruebas manuales, reporte de incidencias y regresiones - Coordinación en remoto con desarrolladores en Perú y Argentina	02.04.2014-30.11.2014

1.2 FORMACIÓN	
Grado Académico de Bachiller en Ingeniería de Software – Escuela Profesional de Ingeniería de Software – Facultad de Ingeniería de Sistemas e Informática – Universidad Nacional Mayor de San Marcos.	Mar 2015

Educación Secundaria Institución Educativa Mixta 6044 Jorge Chávez de Surco	Mar 2004 - Dic 2008
---	---------------------

1.3 IDIOMAS	
Inglés - Intermedio Avanzado Universidad Nacional Mayor de San Marcos (Ciclos I - VI)	Mar 2009 - Dic 2011

1.4 CERTIFICACIONES / CURSOS	
UX Estratégico: del Usuario al Producto	Jun 2017 - Ago 2017
ISTQB® Certified Tester – Foundation Level Licencia 17-CTFL-109785-03	Desde Mar 2017
Certified ScrumMaster® Scrum Alliance	Desde Ago 2015
Continuous Integration Workshop Fecha de la certificación: jul. de 2015 – actualidad Licencia 64701DAB5426D531D06D	Jul 2015
Web Testing - Test Automation Survival Camp Fecha de la certificación: mar. de 2015 – actualidad Licencia 15D3E4EC494A16619247	Mar 2015 - Abril 2015
Unit Testing - Test Automation Survival Camp Fecha de la certificación: mar. de 2015 – actualidad Licencia 759BC3473945741C11F9	Mar 2015 - Abril 2015

1.5 CONOCIMIENTOS / HABILIDADES	
Lenguajes de programación	Java, C#, Ruby
Para pruebas automatizadas	Cucumber (Java & Ruby) Specflow (C#) Robot Framework Selenium IDE & Web Driver (Java & Ruby) Appium (Ruby), Rest-client (Ruby) Sinatra (Ruby)
Metodologías Ágiles	Scrum (Certified Scrum Master desde el 2015)

CAPÍTULO II - CONTEXTO EN EL QUE SE DESARROLLÓ LA EXPERIENCIA

2.1 EMPRESA - ACTIVIDAD QUE REALIZA

Belatrix Software Factory es una empresa de origen argentino cuyo alcance de servicios son diseño de productos, arquitectura y desarrollo de software, aseguramiento de la calidad y prueba de software, servicios de Big Data y DevOps. Cuenta con sedes en Argentina (3, dos en Mendoza y una en Buenos Aires), Perú (una en Lima) y Colombia (una en Bogotá), y con oficinas comerciales en Fort Lauderdale, San Francisco y Nueva York.

2.1.1 Datos de la empresa

- Razón social: BELATRIX PERÚ S.A.C.
- RUC: 20392810316
- Dirección: Av. República de Panamá 3591 San Isidro, Lima
- Teléfono: +51 (1) 717-3350

Principales Clientes:

La gran mayoría de los clientes de Belatrix son empresas norteamericanas tales como mFoundry, Chatham Financial, NuSkin, entre otras. Adicionalmente, en el Perú, cuenta con dos clientes nacionales, una aseguradora y uno de los bancos más grandes del medio. Dicho importante banco fue el escenario en donde se desarrolló la experiencia a ser relatada en el presente informe.

Sobre la entidad financiera donde se desarrolló la experiencia

Como parte de sus planes de Transformación Digital, el banco cliente de Belatrix creó su propio centro de innovación en donde se formaron equipos con talentos de diferentes empresas proveedoras con el objetivo de construir diferentes productos de forma muy diferente a como lo hace su actual área de sistemas. El autor de este informe fue asignado como QA a un equipo donde el producto a construir buscaba permitir hacer transferencias de forma intuitiva, cuyo proceso sea fácilmente entendible por un público joven y que le haga frente al uso del dinero en efectivo.

2.2 VISIÓN

“Nuestra visión es ser reconocidos como la mejor empresa latinoamericana de desarrollo de productos de software e innovación.

Del inglés original *“Our vision is to be recognized as the best Latin American software product development & innovation firm.”*

2.3 MISIÓN

“Nuestra misión es ser un socio global apasionado que ofrece innovación de software.”

Del inglés original: *“Our mission is to be a passionate global partner delivering software innovation.”*

2.4 ORGANIZACIÓN DE LA EMPRESA

A continuación se muestra el organigrama de la empresa Belatrix Software Factory:

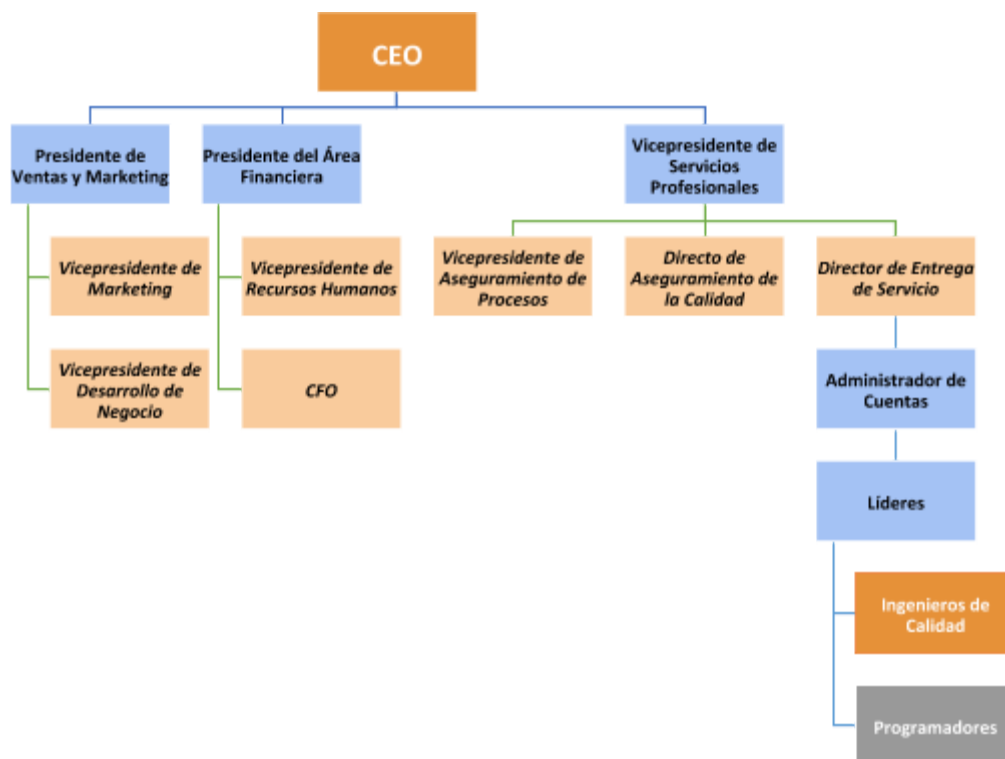


Figura 01: Organigrama de la empresa

2.5 ÁREA, CARGO Y FUNCIONES DESEMPEÑADAS

El autor de este informe se desempeña como Ingeniera de Aseguramiento de Calidad especializada en Automatización de pruebas con grado Senior en el laboratorio responsable del desarrollo de la aplicación móvil de pagos. A lo largo de su carrera ha sido la responsable por la calidad de 5 proyectos diferentes. Tres de los proyectos fueron desarrollados por equipos Scrum en la sede de Belatrix y dos de ellos, en el sitio del cliente.

2.6 EXPERIENCIA PROFESIONAL REALIZADA EN LA ORGANIZACIÓN

Las funciones realizadas como Ingeniero de Calidad fueron:

- Apoyo a los Product Owners de los diferentes proyectos en la especificación de requisitos funcionales y no funcionales.
- Análisis y diseño de casos de prueba.
- Construir frameworks de pruebas en diferentes lenguajes de programación dependiendo de las necesidades del cliente y creación de stubs para reemplazo de dependencias y así facilitar la validación de funcionalidades solicitadas.
- Implementación de pruebas de integración (a nivel de servicios backend), pruebas web, pruebas móviles para Android & iOS.
- Ejecución de casos de prueba y realización de pruebas de regresión.
- Reporte de incidencias y seguimiento de las mismas.
- Sugerencia de mejoras funcionales / procesos en los proyectos.

CAPÍTULO III - ACTIVIDADES DESARROLLADAS

3.1 SITUACIÓN PROBLEMÁTICA

3.1.1 Definición del problema

Contexto del problema a nivel organizacional

La importante entidad financiera, cliente de Belatrix, enfrenta una serie de obstáculos:

- Entorno técnico con abundante tecnología heredada y de difícil mantenimiento.
- Requisitos funcionales establecidos meses antes y comunicados a los responsables del desarrollo mediante documentos que contienen descripciones ambiguas u omisiones críticas.
- Toma de decisiones lentas debido a la burocracia propia de una organización con alcance nacional.
- Debido a la naturaleza del negocio de una entidad financiera se requiere establecer medidas de seguridad muy fuertes, las cuales requieren ser validadas por un área diferente a la de certificación.
- Debido a la metodología tradicional en cascada seguida en el Área de Sistemas, la probabilidad de que el equipo cree un producto cuyo resultado final ya no sea el necesitado por negocio es alta.
- Los procesos tradicionales de control de calidad de aplicaciones, llevados a cabo por el Área de Certificación, toman mucho tiempo y, en casos extremos,

es posible tener hasta 10 ciclos de retesting (es decir, se repiten los esfuerzos de: encontrar defectos en la implementación, devolver a los responsables del desarrollo para realizar las correcciones y devolver a certificación para revisar si las correcciones arreglaron el problema).

- La realidad actual, donde la competencia ya no es solamente con otras entidades financieras, sino que ahora también aparecen las empresas Fintech que son capaces de avanzar rápidamente y de proveer innovación mediante productos y servicios digitales.

Contexto del problema a nivel del equipo de trabajo

El centro de innovación de la entidad financiera nació con el objetivo de desarrollar productos rápidamente y que mejoren significativamente la experiencia de sus clientes.

Pero la construcción de un producto bajo este entorno también propone retos, donde los mayores son:

- **Desarrollo ágil del producto:** que refiere a la forma de trabajo dentro de un equipo conformado por profesionales con diferentes especialidades tales como: expertos de negocio, expertos en estrategias de marketing y comunicaciones, diseñadores de experiencia de usuario, diseñadores visuales, desarrolladores front-end / mobile, desarrolladores backend, ingenieros de calidad expertos en automatización de pruebas.
- **Control de calidad en desarrollo ágil:** la cual involucra que el diseño de pruebas se inicie en el momento que se empiezan a trabajar los requisitos de las historias de usuario, sean diseñadas mientras desarrollo construye una

funcionalidad y sean ejecutadas al recibir la funcionalidad entregable dentro del sprint.

- **Cambios constantes en el diseño del producto:** los especialistas de diseño plantean cambios constantes después de obtener algún comentario y debido a que la metodología ágil con la que se construye el producto permite realizar cambios, el equipo responsable (desarrollo e ingenieros de calidad) debe implementarlos y ello afecta directamente a las pruebas y en muchas oportunidades a la arquitectura de la aplicación.
- **Presiones de tiempo:** debido a que el mundo donde se desenvuelve la organización es altamente competitivo, se requiere lanzar productos y/o actualizaciones periódicas con nuevas funcionalidades para los clientes.

Contexto del problema a nivel del producto

El producto a construir es una aplicación móvil cuyo operación principal es la realización de transferencias bancarias, lo que conlleva a lidiar con:

- **Múltiples sistemas operativos base:** Actualmente, el mercado de los móviles es dominado por Android y iOS como sistemas operativos base para la inmensa gama de dispositivos móviles usado por el público y a su vez, cada sistema operativo cuenta con múltiples versiones donde cada una, debe contar con soporte. Ejemplo:
 - Android cuenta con 4.4, 5.0, 5.1, 6.0, 7.0, 7.1.1
 - iOS, con 9.3, 10.1, 10.2, 10.3, 11

- **Múltiples fabricantes:** Apple es el único propietario del sistema iOS, pero los dispositivos móviles con Android son fabricados por diferentes empresas como Samsung, LG, Motorola, Huawei, y cada uno de ellos hace personalizaciones a sus productos, lo que implica que una funcionalidad puede trabajar perfectamente en Samsung y puede no funcionar en un Motorola.
- **Múltiples dimensiones de pantalla por dispositivo:** cada dispositivo cuenta con diferentes tamaños de pantalla y resoluciones, lo que puede devenir en que algunos elementos de la aplicación, como botones o cuadros de textos se muestren descuadrados o entrecortados en algunos dispositivos.
- **La necesidad de validar tres grandes componentes que influyen en el aplicativo:** el aplicativo muestra lo que su backend procesa y, a su vez, backend procesa acciones de acuerdo al componente de procesamiento intermedio entre él y los servicios de Message Broker de la entidad financiera.
- **Dependencia de servicios externos al desarrollo del aplicativo:** la aplicación depende de dos tipos de servicios, los propios de la entidad (en su sistema Message Broker) y los de terceros. En el caso de los servicios bancarios, si alguno de los servicios es modificado se requeriría una regresión en el aplicativo para corroborar que no existe ningún impacto; de la misma forma, con respecto a los servicios de terceros, se debe estar alerta por si ellos hacen algún cambio con respecto a su funcionamiento.

El problema

El reto a superar es el de realizar control de calidad de un aplicativo móvil que involucra sistemas críticos y cuya calidad impacta directamente en la imagen del banco. Su problemática se expresa en tres grandes aristas:

- A. **La organización** (de la que dependen los servicios bancarios usados por el aplicativo).
- B. **El equipo de trabajo multidisciplinario** (la forma de trabajo, la definición de los requisitos y la toma de decisiones).
- C. **El producto** y características propias de su arquitectura.

3.2 SOLUCIÓN

Crear un framework ágil que sirva de base para realizar pruebas de software orientada a aplicaciones móviles que:

- Permita el fácil diseño, implementación y ejecución de las pruebas.
- Muestre que las pruebas pueden usarse como documentación propia del aplicativo.
- Permita obtener retroalimentación constante con respecto al status de la calidad del producto mediante la ejecución de requisitos implementados.
- Demuestre que las herramientas construidas por los ingenieros de calidad son de gran ayuda al desarrollo de la aplicación.

En esta sección se establecen el objetivo general y los específicos, el alcance, los fundamentos usados y el aporte.

3.2.1 Objetivos

Objetivo general

Establecer un framework ágil que sirva como base para la realización de pruebas de software y ésta sea llevada de forma exitosa para otros proyectos de la misma o de similar naturaleza.

Objetivos específicos

- A. Establecer el proceso de trabajo a seguir por los dueños del producto, los ingenieros de desarrollo, ingenieros de calidad y expertos en diseño que permitan realizar el mejor control de calidad posible.
- B. Crear las herramientas a usar como soporte para el desarrollo y la ejecución de pruebas.
- C. Establecer la forma de trabajo de los ingenieros de calidad con respecto al diseño de los casos de prueba, la construcción de las pruebas automatizadas que respalden la implementación de una funcionalidad a nivel de servicios backend y a nivel de aplicativo móvil y la ejecución, tal que los tiempos de ejecución de pruebas se reduzcan dramáticamente en tiempos.

3.2.2 Alcance

Dentro del alcance

El alcance de este proyecto cubre la descripción de la forma de trabajo a seguir por un equipo de trabajo ágil al lidiar con el testing del aplicativo, la definición, criterios de elección, diseño y construcción de herramientas de

soporte a construir, implementación y ejecución de pruebas funcionales realizadas durante el proceso de desarrollo.

Fuera del alcance

Se encuentran fuera de alcance las pruebas no funcionales y proceso de aseguramiento de calidad.

3.2.3 Conceptos fundamentales

Metodologías ágiles

Es el nombre con el que se denomina a la práctica que promueve iteraciones continuas de desarrollo y testing a través del ciclo de vida del desarrollo de software [5]. Esta práctica orienta a que los esfuerzos de desarrollo de software se hagan mediante [5]:

- **Planificación adaptativa**

El equipo de desarrollo y los representantes del negocio planifican cuál será el próximo entregable que aporte valor y definen los alcances de la misma.

- **Desarrollo evolutivo**

No se asume que el producto debe tener desde su concepción todos los detalles de sus requerimientos, sino por el contrario, se construye sobre funcionalidades antes construidas.

- **Entrega temprana**

Es mejor hacer entregas continuas al negocio, para que ellos puedan evaluar si el resultado de lo que solicitaron satisface sus expectativas. Si ellos creen que el resultado no es tan satisfactorio como ellos pensaron que lo sería al momento de pensar en los requisitos, tienen la oportunidad de corregir pronto la situación.

- **Mejora continua**

Si hubieron detalles en los requisitos que pudieron ser previstos, los mismos son considerados en una próxima iteración.

- **Respuesta rápida y flexibilidad al cambio**

Si circunstancialmente, hubiera alguna necesidad de cambiar la planificación previamente hecha, esto se podría hacer.

El término “ágil” fue popularizado por el Manifiesto por el Desarrollo Ágil del Software, la cual establece 04 principios [1]:

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

Cabe recalcar que este manifiesto establece que aunque se valoran mucho los elementos de la derecha, se valoran aún más los de la izquierda [1].

Scrum

Es una metodología dentro del cual las personas pueden abordar complejos problemas de adaptación, mientras entregan productos del mayor valor posible. Se ha utilizado para gestionar el trabajo en productos complejos desde principios de la década de 1990. Scrum no es un proceso, técnica o método definitivo, es más bien, es un marco dentro del cual puede emplear varios procesos y técnicas. Scrum deja en claro la eficacia relativa de las técnicas de gestión y trabajo de su producto para que pueda mejorar continuamente el producto, el equipo y el entorno de trabajo [3].

El marco de Scrum consiste en Equipos Scrum, roles asociados, eventos, artefactos y reglas. Cada componente dentro del marco sirve para un propósito específico y es esencial para su éxito [3].

Las reglas de Scrum unen los roles, eventos y artefactos, y rigen las relaciones e interacciones entre ellos [3].

Las reglas de Scrum se describen en todo un documento llamado Scrum Guide, cuyos autores son Ken Schwaber and Jeff Sutherland.

Testing ágil

Del inglés, *Agile Testing*, es una práctica de testing que sigue los principios del desarrollo ágil.

Bajo esta práctica:

- Las pruebas no son una etapa, sino por el contrario, se prueba continuamente en cada iteración.
- Se provee retroalimentación constante a los responsables del negocio sobre el status del producto que se viene desarrollando.
- Se mantiene la documentación mínima necesaria, la suficiente que se requiere, ni más, ni menos.
- Se promueve el desarrollo de pruebas unitarias y el desarrollo orientado a pruebas.

- Se busca que todo el equipo sea responsable por la calidad de los entregables, y no sólo lo sean los testers.

Automatización de pruebas

Mientras que las pruebas manuales involucran que una persona ejecute una serie de pasos previamente establecidos en un caso de pruebas, producto del ejercicio de realizar el diseño pruebas, la automatización de pruebas refiere al esfuerzo que involucra desarrollar un programa que simule el comportamiento humano al realizar una prueba y así, ahorrar esfuerzos en situaciones que requieren varios ciclos de retesting.

Las pruebas automatizadas son importantes porque:

- Ejecutar pruebas manuales en múltiples ocasiones de todos los flujos de trabajo, todos los campos y todos los escenarios negativos consumen mucho tiempo y dinero.
- Es difícil probar casos extremos de límites en valores, como por ejemplo, los límites de movimientos en transferencias y estas situaciones son más sencillas de simular mediante un programa.
- No requieren intervención humana, una vez que éstas se encuentren ya desarrolladas, ya que pueden ser ejecutadas fuera del horario laboral, ahorrando horas-hombre.
- Aumentan la velocidad de ejecución de prueba.
- Ayudan a tener una buena cobertura de pruebas en sistemas complejos.

- Las pruebas manuales pueden volverse tediosas para el tester, y, por lo tanto, existe el riesgo de que el ejecutor de pruebas omita algún caso de pruebas.

Evidentemente, el desarrollo de pruebas automatizadas requieren un esfuerzo de profesionales especializados en este campo, con conocimientos en desarrollo de software, en estrategias de simulación de dependencias y experiencia con el manejo de herramientas.

Testing de aplicaciones móviles

Del inglés, *Mobile Application Testing*. En la actualidad, la tecnología móvil y los dispositivos inteligentes son tendencia y se crean miles de aplicativos móviles todos los días a nivel mundial.

Como todo desarrollo, los aplicativos móviles también son sometidos a pruebas y en este caso, las pruebas son mucho más complejas de lo que sería probar una aplicación de escritorio o una web, y esto debido a que:

- Existen diferentes Sistemas Operativos para móviles, tales como Android y iOS; y , a su vez, éstos cuentan con diferentes versiones por cada una, a las que la aplicación debe dar soporte. Por ejemplo: Android cuenta como 4.4, 5.0, 5.1, 6.0, 7.0, 7.1, 8.0 y iOS cuenta con 9.2, 9.3, 10.0, 10.2, 10.3, 11.0, 11.1.
- Existen múltiples fabricantes de dispositivos móviles, como por ejemplo, Samsung, Motorola, LG, Huawei, y demás; y, a su vez, cada uno de ellos hace customizaciones específicas a las versiones base de Android.
- Cada dispositivo móvil tiene diferentes tamaños y resoluciones de pantalla, lo cual puede impactar en la calidad visual del aplicativo.

- El desarrollo del aplicativo puede ser nativo (es decir, es una aplicación cuyo desarrollo es propio para una plataforma con un lenguaje o tecnología específica como Java o Kotlin para Android y Objective-C o Swift para iOS) o híbrido (en otras palabras, este desarrollo es hecho con tecnologías en Javascript y se ejecutan en Android o iOS dependiendo de su compilación).
- Diferentes condiciones a las que un aplicativo es sometido, como por ejemplo, límite de almacenamiento en el smartphone, calidad de conexión de internet, comportamiento en casos de batería baja, consumo de memoria, permisos aceptados o denegados a ciertas características del dispositivo.

Adicionalmente, se debe considerar que una funcionalidad específica del aplicativo puede comportarse como se espera para un tipo de modelo de smartphone y puede fallar para otro, aunque compartan hasta la misma versión de sistema operativo.

Las pruebas móviles son extremadamente complejas y su casuística se multiplica dependiendo de la cantidad y variedad de dispositivos que se encuentren dentro del alcance de pruebas. Por todo lo anteriormente mencionado, no es aconsejable establecer a las pruebas manuales como única estrategia de prueba.

Mocking: estrategia de simulación de dependencias

En las pruebas de software, es común que el artefacto de software bajo prueba dependa de otros componentes. Por lo tanto, cuando se desea probar una unidad, se suelen construir pruebas unitarias que ayuden a validar que, ésta, de forma aislada funcione correctamente; pero después se vuelve necesario probarla junto a todas sus dependencias para evaluar si la comunicación con otros componentes es correcta [4].

Al probar todas las dependencias, se refleja con más realismo el comportamiento en producción. Sin embargo, estando en el proceso de desarrollo, algunas dependencias (como bases de datos y servicios web) ralentizarían la ejecución de la prueba, agregarían un alto costo en tiempos de configuración para la prueba y requerirían que los desarrolladores y/o testers tengan control total sobre tales dependencias externas, situación que es casi imposible debido a restricciones propias de seguridad en entidades que manejan data sensible [4].

Las razones anteriormente mencionadas son la causa por la cual se establecen estrategias de simulación de dependencias, ya que al hacer esto los desarrolladores ganan un gran enfoque: la prueba cubrirá las interacciones esperadas con sus dependencias; además, las ineficiencias de probar dependencias son mitigados [4].

3.2.4 Framework desarrollado

El framework ágil para realizar pruebas de software orientado a aplicaciones móviles es una compilación de las mejores prácticas de lo que fue el esfuerzo a nivel de control de calidad en un proyecto cuya duración fue de aproximadamente un año y medio.

El contexto en donde se desarrolló fue en la de un equipo de trabajo que seguía Scrum como metodología de trabajo.

En la Figura 01, se muestra las acciones y artefactos que componen el framework desarrollado.

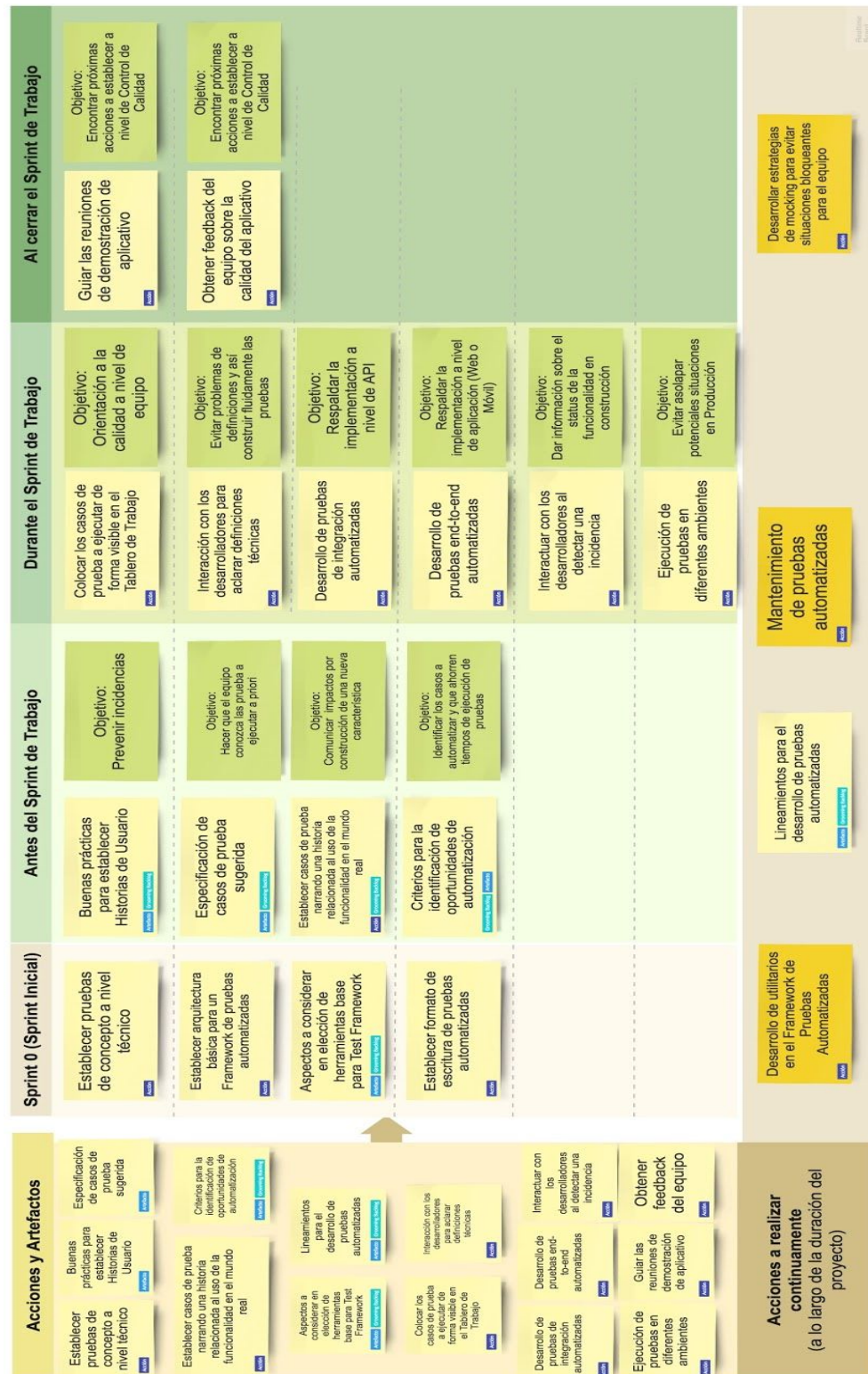


Figura 02: Framework Ágil para realizar pruebas de software orientado a Aplicaciones Móviles. Fuente propia.

3.2.4.1 Definición de la estrategia a seguir

La metodología de trabajo a seguir por el equipo fue Scrum. La práctica de testing ágil establece seguir prácticas orientadas al desarrollo ágil, y por ello, desde el punto de vista del autor de este informe:

- Es mejor ejecutar medidas preventivas sobre las historias de usuarios a desarrollar, con el fin de encontrar la menor cantidad de incidencias y evitar ciclos de retesting.
- Es necesario hacer que los ciclos de retesting duren lo menos posible con el fin de dar a conocer el estado de la historia de usuario en desarrollo lo más pronto posible a todos los involucrados, y esto se logra con la implementación de pruebas automatizadas a nivel de integración y end-to-end.

Para lograr los puntos mencionados líneas arriba, el framework establece que se deben realizar diferentes acciones en pos del Control de Calidad en diferentes etapas del proyecto:

- En el sprint inicial del proyecto (también llamado sprint 0).
- Antes de iniciar el desarrollo en un sprint de trabajo.
- Durante el sprint de trabajo.
- Al cierre del sprint de trabajo.

Durante el sprint inicial o sprint 0

Este sprint es el espacio que se da para aclarar asuntos que aún no se conocen o de los que aún no se han tomado decisiones. Ejemplos de estas situaciones son: definición de servidores, definición del tipo de aplicación a construir, tecnologías tentativas a emplear, construcción de pruebas de concepto, definición de diseños preliminares, esclarecer la visión del objetivo del aplicativo a construir y más.

En esta etapa, desde el punto de vista del framework establecido, se debe realizar la definición preliminar de las estrategias de pruebas, de la arquitectura de un framework de pruebas y deben establecerse pruebas de concepto para evaluar tecnologías.

Antes de iniciar el sprint de trabajo

Antes de que el equipo de trabajo comience con el desarrollo, es necesario realizar 02 esfuerzos, el primero consiste en reuniones de Backlog Grooming y el segundo, es el Sprint Planning. A continuación, se detallan las acciones establecidas por el framework en cada una.

- Backlog Grooming o reuniones de refinamiento(s) del backlog

Estas reuniones establecen el espacio donde se discuten las historias de usuario con el fin de comprender la visión del Product Owner, preguntar y contestar dudas que cada uno de los involucrados tenga, desglosar historias y establecer los casos de prueba relacionados a la funcionalidad en discusión.

Desde el punto de vista del framework ágil, esta reunión es crítica, ya que en este punto:

- Se tiene la posibilidad de prevenir la mayor cantidad de incidencias discutiendo con el Product Owner los requisitos descritos en la Historia de Usuario.
- Es posible comunicar a todo el equipo qué funcionalidades se verían impactadas por la construcción (sobre éstas funcionalidades se deberán correr pruebas de regresión antes de establecer una historia de usuario como “done” o terminada).

Los ingenieros de calidad deben realizar 03 acciones en estas reuniones:

- Establecer los casos de prueba del flujo básico.
- Plantear los casos de prueba para flujos alternos y obtener los resultados esperados al consultarlo con el Product Owner; se recomienda establecer ejemplos de uso en el mundo real para obtener más detalle.
- Comunicar los casos de prueba al equipo con el fin de que su revisión permita al equipo visualizarlas y descubrir tareas potenciales para cubrirlas en la implementación.

Como resultado final de la discusión, al evaluar la cantidad y dificultad de las tareas desglosadas por todos los miembros del equipo, se establece un peso relativo de cuánto le costaría al equipo implementar esta historia, considerando los esfuerzos de desarrollo y control de calidad. Estos pesos se consideran en la Planning Meeting para tomar la decisión de cuántas historias el equipo se compromete a terminar en la iteración próxima a comenzar. En la Figura 02, se puede ver la mecánica de las sesiones de Grooming.

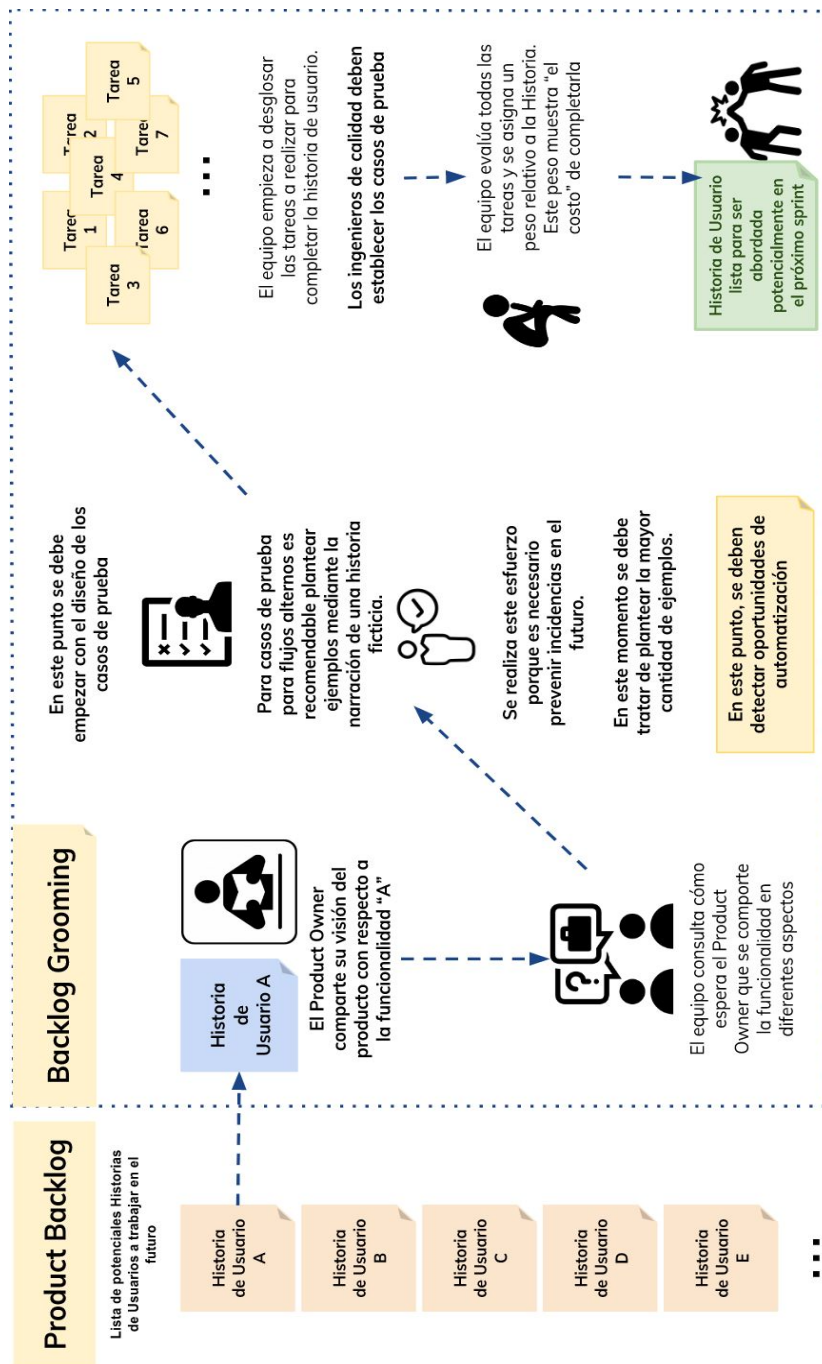


Figura 03: Acciones a realizar en la Backlog Grooming establecidas por el Framework ágil para pruebas de software. Fuente propia.

- Planning Meeting o Reunión de Planificación

Después de que las reuniones de refinamiento de backlog, justo al terminar el sprint anterior (o al enfrentar el primer sprint de trabajo de toda la historia del proyecto), se realiza una reunión en la que el equipo de trabajo revisa la lista de historias de usuario previamente ya discutidas en Backlog Groomings y listadas en orden de mayor a menor de prioridad para Product Owner. En base a los pesos establecidos, el equipo decide qué historias podrían llegar a completar en el periodo de un sprint. Una vez que la decisión ha sido tomada, el sprint inicia.

Durante el sprint de trabajo

Debido a que es necesario tener visibilidad de los esfuerzos realizados por cada miembro del equipo, se establece un tablero en el cual se encuentran:

- Las historias de usuario a trabajar durante el Sprint
- Las tareas asociadas a cada historia
- El responsable asignado a cada una de las tareas.

El tablero de trabajo contiene las siguientes columnas:

- “To-do”:

En esta sección se encuentran todas las historias a implementar durante el sprint, y por cada una de ellas, las tareas desglosadas con el equipo en la reunión de Planning.

Es indispensable que junto a las tareas, también se encuentren los casos de prueba a ejecutar descritas como tareas de forma que los desarrolladores

puedan realizar una implementación considerando los posibles escenarios de caída.

- “Doing”:

En esta columna se colocan las tareas que se van realizando durante el día. Si éstas, son terminadas, se mueven directamente a la columna “Done”.

Desde el punto de vista de calidad, en este punto, mientras los desarrolladores se encuentran construyendo las funcionalidades, los responsables del control de calidad, deben implementar en forma automatizada las pruebas previamente mapeadas.

Si en esta etapa se descubren nuevas pruebas, se deben comunicar al Product Owner y al equipo lo antes posible.

- “Ready for Testing”:

Cuando los desarrolladores sienten que su implementación está lista para ser revisada por los responsables del control de calidad, ellos deciden colocar la historia en esta columna. Para esto, ellos colocan su desarrollo en una rama de desarrollo, no en la rama master (rama en donde se encuentra la versión estable del aplicativo).

En este punto, los responsables del control de calidad ejecutan las pruebas sobre la implementación. De encontrarse incidencias sobre la historia en desarrollo, ésta es devuelta a “Doing” y se hacen todos los esfuerzos posibles por establecer hipótesis de la causa raíz y apoyar a que los desarrolladores puedan reproducir la incidencia.

Adicionalmente, en este punto, se deben ejecutar una prueba de regresión restringida a las funcionalidades construidas sprints atrás pero que pudieron resultar afectadas por la nueva funcionalidad.

Una vez que el desarrollo ha pasado por un riguroso proceso de code review y la funcionalidad deje de presentar incidencias, se otorga el visto bueno para unir esta implementación en la rama principal, en donde se encuentra la versión estable del aplicativo.

- “Testing”

Cuando la historia ya recibió la venia para poder ser unificada a la rama principal, en la que reside la versión estable del aplicativo, se vuelven a ejecutar las pruebas relacionadas a la funcionalidad construida y se ejecutan pruebas de regresión para asegurar que la nueva versión estable se encuentra funcionando como se espera.

Si por alguna razón se encontrase alguna incidencia, esta se reporta de inmediato a los desarrolladores y se crea una nueva rama de desarrollo para su corrección y se vuelven a repetir los esfuerzos de ejecución de pruebas. Debido a que estos esfuerzos de ejecución de pruebas son costosos en tiempos, la implementación de pruebas automatizadas en flujos críticos y de alta prioridad no son opcionales.

- “PO Review”

Una vez que todo los responsables de calidad ya no encuentran ninguna incidencia, la historia se mueve a la columna de PO Review, y en esta etapa, la revisión de la historia de usuario la realiza el Product Owner en un ambiente lo más real posible (ambiente de certificación o preproducción, dependiendo de la naturaleza de la historia y del criterio del equipo).

El Product Owner en este punto puede hacer observaciones y puede plantear mejoras, las cuales, si existiera tiempo suficiente en el sprint se pueden implementar o en caso contrario, él crearía nuevas historias de usuario que contemplen las nuevas características solicitadas. Una vez que el Product Owner, se encuentre satisfecho con los resultados, establece que la Historia de Usuario se encuentra en “Done”.

- “Done”

En esta columna se colocan todas las historias que pasaron por PO Review y que se encuentran terminadas de forma satisfactoria.

En la Figura 03, se muestra la mecánica del trabajo dentro del sprint en el formato que tiene del tablero de trabajo.

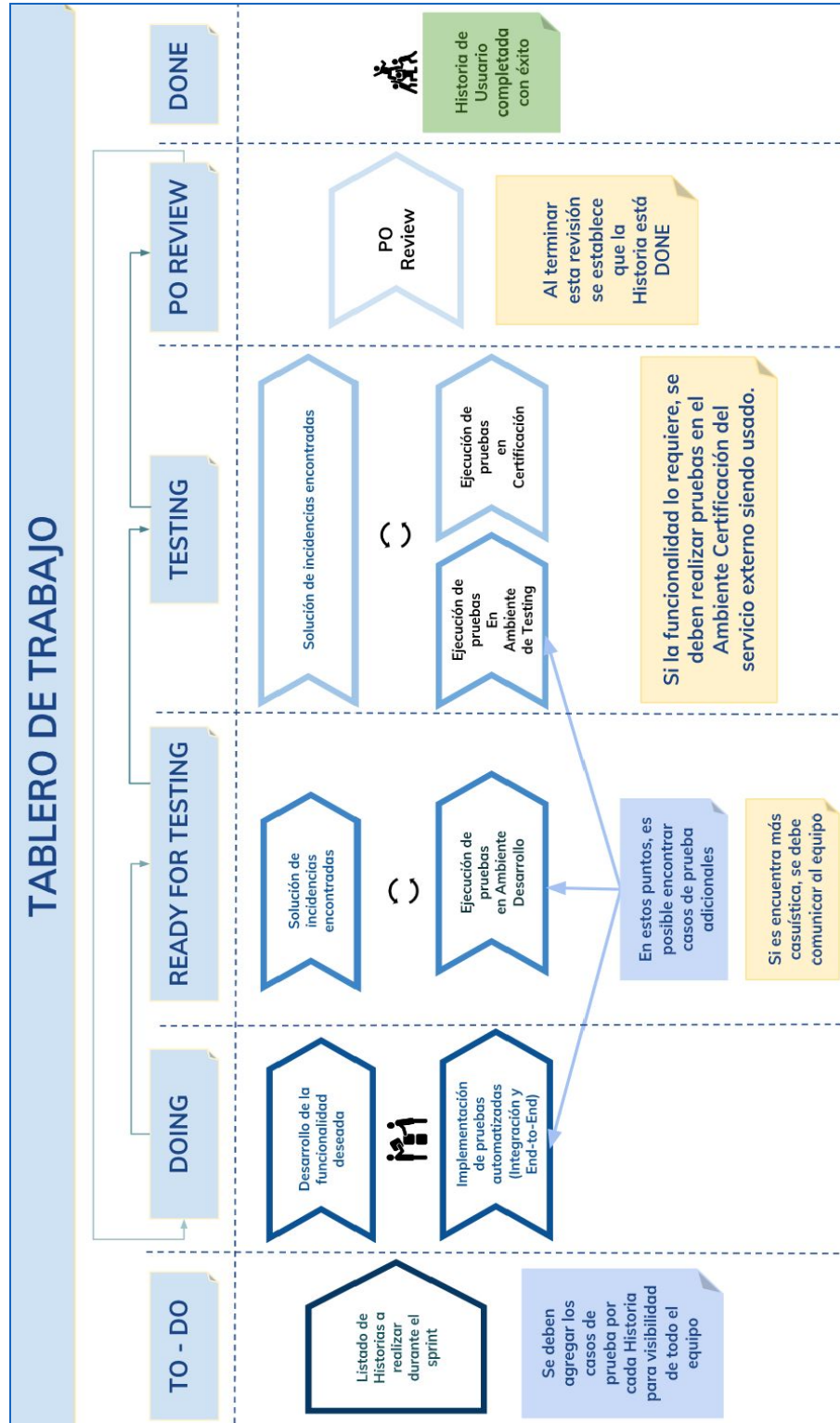


Figura 04: Tablero de trabajo dentro del sprint. Fuente propia.

Al cierre del sprint

Al terminarse el periodo establecido para el sprint (01 o 02 semanas, dependiendo de las circunstancias del proyecto), se realizan una reunión a modo de demostración de lo todo lo avanzado, de forma que todos los miembros del equipo estén en la misma página.

Si alguna historia de usuario no ha logrado ser completada, se aborda en el siguiente sprint.

Se lleva a cabo inmediatamente después una reunión de retrospectiva, la cual tiene por objetivo, encontrar posibilidades de mejora a implementar en la forma de trabajo.

Luego de terminar el ejercicio de retrospectiva, se lleva a cabo la reunión de Sprint Planning, en donde el equipo junto con el Product Owner deciden los siguientes pasos a dar en el nuevo sprint que está a punto de comenzar.

No se debe olvidar que las reuniones de backlog grooming se realizan dentro del sprint en desarrollo, para que de esta forma se cumpla que la regla de llevar a cabo las reuniones de grooming y de planning antes de iniciar el nuevo sprint.

3.2.4.2 Buenas prácticas para la definición de Historias de usuario

Los requisitos son parte crítica para el desarrollo de las funcionalidades y por ello, si no son comprendidos por el equipo se corre el riesgo de fracasar.

El Product Owner otorga su visión de la funcionalidad a incorporar en el producto, pero ellos no son expertos en desarrollo de software, por lo que él no podría dar una solución técnica y quienes sí podrían darle forma a la solución es el equipo de trabajo.

Con el fin de crear funcionalidades de calidad, el equipo realiza consultas relacionadas a la definición de la Historia de Usuario y este es el momento en donde se deben prevenir posibles incidencias en el futuro por falta de especificaciones y/o por tomar asunciones incorrectas.

Para lograr tener una buena definición de Historia de Usuario, se creó una lista de puntos a abordar donde se busca responder a las siguientes preguntas:

1. ¿La definición de esta historia se encuentra lista de forma que el equipo de trabajo pueda terminarla en un Sprint?
2. ¿Permite esta definición realizar un buen Control de Calidad?

A continuación, se presenta la Tabla 01, donde se establecen buenas prácticas para definir Historias de Usuario. En cada punto se explica por qué es importante no omitirlo.

BUENAS PRÁCTICAS PARA ESTABLECER BUENAS DEFINICIONES DE HISTORIAS DE USUARIO	
Código Identificador	<i>De preferencia, usar el código que la herramienta de gestión de historias de usuario provee.</i>
Título significativo para la Historia de Usuario	<i>El título debe ser lo suficientemente significativo como para permitir recordarla aún a posteriori.</i>
Descripción General de la funcionalidad a desarrollar	<i>Ésta debe mostrar la visión del Product Owner. Desde el punto de vista de pruebas, en esta etapa es posible identificar si esta funcionalidad afecta potencialmente a otras ya construidas.</i>
Diseños visuales de la funcionalidad esperada	<i>Esta debe esclarecer el flujo a seguir por el usuario y las interacciones esperadas. Desde el punto de vista de pruebas, esto es crítico debido a que es posible detectar muchos casos de prueba en este punto.</i>
Status de la Historia	<i>Establecer en qué situación se encuentra la historia.</i>
<i>Status 0</i>	<i>Con blockers / Sin visión clara: No se debe iniciar su avance.</i>
<i>Status 1</i>	<i>Con Visión / Descripción a alto nivel del Product Owner.</i>
<i>Status 2</i>	<i>Con diseño adjunto.</i>
<i>Status 3</i>	<i>Con desglose de tareas.</i>
<i>Status 4</i>	<i>Completamente entendido por todo el equipo y sin blockers.</i>
Objetivo a alcanzar	<i>Descripción del objetivo a alcanzar.</i>

con la implementación	<i>Desde el punto de vista de pruebas, esto es necesario debido a que es posible detectar si la definición de la historia se desvía del objetivo que se persigue.</i>
Alcance de la Historia	<i>Descripción de lo que está fuera del alcance de la historia. Si es posible, agregar referencia del código identificador de la historia que sí contempla estos aspectos.</i> <i>Desde el punto de vista de pruebas, esto es necesario debido a que se determina que se encuentra dentro y fuera del scope de pruebas. También es posible advertir a negocio sobre riesgos al establecer ciertas características como fuera de alcance.</i>
Ejemplos de lo que espera de la funcionalidad una vez construida	<i>Listar los ejemplos más significativos posibles dados por el product owner a modo de narración desde la perspectiva de usuario final.</i> <i>Desde el punto de vista de pruebas, esto es necesario debido a que se determina que se encuentra dentro y fuera del scope de pruebas. También es posible advertir a negocio sobre riesgos al establecer ciertas características como fuera de alcance.</i>
Desglose del esfuerzo en modo de tareas	<i>Tomar como premisa que las tareas listadas deben durar menos de un día.</i>
Desglose de Ingenieros Frontend - Mobile	<i>Escribir las tareas relacionadas al esfuerzo a realizar por los desarrolladores frontend - mobile.</i>
Desglose de Ingenieros Backend	<i>Escribir las tareas relacionadas al esfuerzo a realizar por los desarrolladores backend.</i>
Desglose de Ingenieros	<i>Escribir las tareas relacionadas al esfuerzo a realizar por los</i>

de Calidad	<i>responsables de la calidad del aplicativo.</i> <i>Es aconsejable que también se coloquen tareas relacionadas a la historia en la herramienta de gestión de historias de usuario.</i> <i>Las tareas a desglosar deben comprender:</i> - <i>Los esfuerzos de automatización de pruebas a nivel de integración.</i> - <i>Los esfuerzos de automatización de pruebas a nivel de end-to-end (UI - Mobile).</i> - <i>Los casos de prueba a ejecutar (de forma que los desarrolladores conozcan qué es lo que se va a probar y así, puedan ellos validar estos casos en su implementación y en consecuencia, minimizar los ciclos de retesting).</i>
Observaciones	<i>Listar todas las observaciones que el equipo encuentre en el tiempo, junto con sus fechas de forma que:</i> - <i>en el futuro sea posible ver su progreso</i> - <i>no se olvide que deben ser absueltas en algún momento como parte de la definición de la historia o como parte de otra.</i> <i>Las observaciones hechas por los responsables de la calidad deben ser absueltas antes de comenzar con el desarrollo de la Historia de Usuario.</i>

Tabla 01: Buenas prácticas para establecer buenas definiciones de Historia de Usuario

3.2.4.3 Lineamientos generales para el diseño y construcción de pruebas

> Lineamientos para el diseño de pruebas

En la reunión de Grooming, se revisa la Historia de Usuario de forma concienzuda, y en base al detalle que el Product Owner pueda brindar se van armando casos de prueba. Con la definición preliminar de la Historia de Usuario se arman casos de pruebas que cubran los flujos básicos.

Luego de ello, se procede a ahondar en más detalle, y la forma seguida en estas situaciones es narrar una historia en la que se usa a un personaje ficticio, de forma que se pueda imaginar el comportamiento de la funcionalidad en el mundo real. Al realizar esta acción, es posible sacar casos de prueba extremos.

Las metodologías ágiles establecen que el software funcionando está por encima de la documentación, pero ello no quiere decir que la documentación se deje de lado; por ello, se debe ver la forma de hacer que las pruebas automatizadas no sean sólo scripts que simulan acciones, sino que también se conviertan en documentación sobre la funcionalidad construida.

Los casos de prueba deben colocarse dentro de la definición de la Historia de Usuario, y en el tablero de trabajo, también deben colocarse de forma gráfica para dar visibilidad a los miembros del equipo.

La especificación de un caso de prueba debe sólo incluir lo necesario. La tabla 02 muestra los puntos mínimos a completar.

ESPECIFICACIÓN DE UN CASO DE PRUEBA	
Código identificador	<i>Se recomienda usar el de la herramienta de Gestión de Historias de Usuario</i>
Nombre significativo	<i>Lo suficientemente significativo como para que el equipo la recuerde fácilmente sin leer el flujo a seguir</i>
Historia de Usuario a la que está ligada	<i>Colocar el código identificador de la Historia de Usuario</i>
Descripción	<i>Breve, lo suficiente como para poder diferenciarla de otro caso de prueba</i>
Precondiciones	<i>Listar aquellas que requieren ser simuladas</i>
Flujo a seguir	<i>Pasos a seguir para establecer el caso de prueba</i>
Resultado esperado	<i>Interacciones a nivel de Interfaz de usuario, información a mostrar, respuesta esperada de una API, etc.</i>
Tipo de flujo	<i>Flujo base de Historia de Usuario (Happy Path) / Flujo alterno</i>
¿Debe ser automatizado?	<i>Mandatorio / Opcional / No requiere</i>

Tabla 02: Estructura para especificaciones de casos de prueba y explicación de cada punto.

Fuente propia.

> *Criterios para la identificación de oportunidades de automatización*

Una vez que se tienen mapeados todos los casos de prueba descubiertos, se debe tomar la decisión sobre qué flujos automatizar. Tal y cómo los principios de pruebas establecen, no es posible hacer pruebas exhaustivas y ello también se aplica para la automatización de pruebas. Por experiencia se sabe que no todos los casos de prueba son automatizables y no todos los casos lo requieren; casos que no lo requieren son aquellos en los que conllevaría mucho esfuerzo en tiempo de construcción y poco ahorro de tiempo.

Como parte del framework de pruebas ágiles dirigidas a aplicaciones móviles, se establecieron 05 criterios para determinar si un caso de prueba debe ser automatizado:

- Nivel de uso esperado por parte de usuarios finales es alto (estimado por negocio).
- El flujo establecido por el caso de prueba representa un costo alto en tiempos de testing y retesting.
- El flujo establecido puede convertirse en precondition de alguna funcionalidad futura.
- El caso de prueba evalúa un caso límite de uso y tiene un nivel de dificultad alto para establecer manualmente sus precondiciones.
- Si el caso de prueba manifiesta un resultado esperado a nivel de interfaz de usuario y es crítico para negocio que la parte visual de esa funcionalidad se aprecie bien en el aplicativo.

La última consideración se toma en cuenta por la naturaleza propia de los dispositivos móviles, y el hecho de que la funcionalidad puede verse y funcionar como se espera en un tipo de dispositivo, y puede no comportarse como se espera en otro, por ello, para aminorar los tiempos de regresión de pruebas, es más productivo establecer una prueba automatizada, donde la variable sería el dispositivo objetivo de la prueba.

Si alguna de las situaciones anteriormente listadas tiene potencial de producirse, entonces se automatiza de forma mandatoria. Si hay algún otro caso de prueba que, se considera, podría ahorrar tiempo en la regresión de pruebas, entonces, se establece como automatizable opcional.

> Aspectos a considerar en la elección de herramientas base para un framework de pruebas automatizadas

Cuando se establece que se deben construir pruebas automatizadas, el punto de partida es el framework técnico que debe darle soporte a su desarrollo y ejecución.

Hay muchas herramientas comerciales que ofrecen funcionalidades ya hechas, pero no se suelen usar todas ya que no son necesarias o no terminan funcionando como se espera y cuando se tienen problemas técnicos en dichas herramientas, se requiere establecer contacto con su equipo de soporte y ello puede tardar horas y hasta días, lo que deviene en la percepción en los responsables del negocio de que la inversión monetaria se convirtió en gasto. Por esta razón, se suelen seguir trabajando en frameworks en los que se construye lo que sólo lo que el proyecto requiere, de forma que se permita hacer lo que negocio espera.

Gracias al Open Source y la gran comunidad de desarrolladores a nivel mundial, muchas librerías y utilitarios en general se han creado y se puede hacer uso de ellos para mejorar la productividad de los productos a construir. Esto no es diferente para los productos relacionados a testing de aplicaciones móviles.

El framework para pruebas automatizadas construido para el caso de este informe de experiencia profesional requirió que en el Sprint inicial (también llamado Sprint 0) se hicieran muchas pruebas de concepto, lidiando con diferentes tecnologías y diferentes lenguajes de programación. Después de tener varias pruebas de conceptos, exitosas y fallidas, se llegó a tomar la decisión de qué combinación de tecnologías tomar.

Los aspectos considerados para la evaluación y selección de herramientas base fueron:

- Experiencia del responsable del control de calidad a cargo

Es importante que quien lleve a cargo el esfuerzo de pruebas automatizadas tenga experiencia en su desarrollo para poder terminarlo dentro del sprint comprometido. Es necesario, también evaluar con qué herramientas ya ha trabajado porque permite tener una perspectiva más amplia y ver qué opciones son viables y cuáles no .

- Lenguaje de programación que permita desarrollo rápido y poco esfuerzo en configuraciones

Java es un lenguaje de programación muy usado en por gran parte de los desarrolladores, pero requiere mucho esfuerzo en configuraciones que otros lenguajes no; por ejemplo, Ruby, es un lenguaje que tiene como uno de sus principios “Convención sobre configuración”, lo que ayuda a ser más productivos en otros aspectos. Cabe resaltar que si se requiriera realizar una configuración personalizada siempre será posible con Ruby.

- Plataformas y ambiente de ejecución de pruebas

Cuando se realizan pruebas para aplicaciones móviles, es importante considerar qué plataformas serán parte del alcance del aplicativo. Si parte del alcance del aplicativo es iOS en iPhones o iPads, entonces se requiere tener portátiles Mac para poder compilar el aplicativo y poder hacer uso de simuladores facilitados por su entorno de desarrollo XCode.

- Soporte y mantenibilidad del framework en el tiempo

Si se hubiese optado por la adquisición de herramientas comerciales, el único soporte obtenido hubiese sido el de servicio al cliente; mientras que al usar herramientas open source, el soporte lo da la comunidad de desarrolladores que están detrás de su construcción y mantenimiento y otros desarrolladores que cómo el autor de este informe también se encuentran implementando esfuerzos similares.

Por otro lado, es mandatorio que del responsable de la creación del framework deje establezca documentación suficiente dentro del mismo framework como para que otros puedan continuar su trabajo.

- Expectativas del equipo con respecto a las pruebas

Los responsables del negocio suelen ver a las pruebas automatizadas como una implementación muy técnica, cuando no tiene que ser necesariamente así. Las pruebas deben reflejar los requisitos solicitados, por ello, también deberían estar escritas en un lenguaje lo suficientemente humano como para que el Product Owner las vea y las pueda entender. Si las pruebas son materializadas en lenguaje común para todos, entonces:

- Se transforman automáticamente en documentación
- Demuestran que la aplicación funciona puesto que son ejecutables. Y con este último punto se refuerza uno de los valores del Manifiesto Ágil [1]: “Software funcionando sobre documentación exhaustiva”.

- Alcance de las pruebas

El framework a construir debe ser siempre capaz de aceptar en su estructura la implementación de pruebas a nivel de integración, a nivel web y a nivel de aplicaciones móviles. La parte más compleja es el soporte a aplicaciones móviles, pero no por ello se debe dejar de lado la implementación web puesto que no se sabe si en el futuro, la aplicación requerirá soporte web.

> *Lineamientos para el desarrollo de un framework de pruebas automatizadas*

Por lo general, las aplicaciones móviles tienen un ecosistema con 03 frentes: la aplicación móvil en sí, el desarrollo de los servicios a consumir de su propio sistema backend y, un frente inesperado pero de muy probable construcción futura, que es un soporte vía web. Por ello, se deben tomar diferentes consideraciones por cada una.

>> *Al abordar pruebas de integración*

- **Ambientes de ejecución**

Si las pruebas a ejecutar tienen como objetivo ambientes locales, de desarrollo, de pruebas o de certificación, entonces estos deben tomarse en cuenta al momento de establecer valores de configuraciones en las pruebas.

- **Precondiciones a establecer en pruebas**

Las precondiciones que una prueba de integración podría necesitar simular serían a nivel de llamadas a servicios de soporte, llamadas a servicios pertenecientes a flujos de prueba contruidos con anterioridad o a datos que necesitan ser configurados vía base de datos.

- **Sistemas objetivo de la prueba**

Uno más sistemas backend cuyos servicios puedan ser usados por el aplicativo móvil.

>> *Al abordar pruebas de aplicaciones web*

- **Ambientes de ejecución**

Las pruebas a nivel web pueden ser ejecutadas en diferentes tipos de navegadores tales como Chrome, Firefox o Internet Explorer. También debe considerarse dependiendo del aplicativo, que ésta funcione sólo para algunas versiones de dichos navegadores.

- **Precondiciones a establecer en pruebas**

Las precondiciones que las pruebas de aplicaciones web podrían requerir se pueden simular usando como base acciones ya programadas para las pruebas de integración.

- **Sistemas objetivo de la prueba**

Una o más aplicaciones con interfaz de usuario de tipo web.

>> *Al abordar pruebas de aplicaciones móviles*

- **Plataformas:** es decir, ¿bajo qué plataformas se ejecutará el aplicativo? Dependiendo de si ésta corre en Android y/o iOS se deben hacer configuraciones y se deben ejecutar en ambientes especiales.

- **Versiones de sistema operativo y adecuaciones propias de fabricantes a dispositivos:** Cada versión de los sistemas operativos establecen nuevas funcionalidades conforme pasa el tiempo, pero no todos los dispositivos tienen disponibles las actualizaciones, por lo que corresponde dar soporte en las aplicaciones a una gran gama de versiones. Adicionalmente, cada fabricante

de dispositivos inteligentes hace adecuaciones especiales a los sistemas operativos base (Android), lo que puede suscitar incidencias inesperadas.

- **Tipos de aplicaciones objetivo:** Aplicaciones nativas o híbridas. Esto es necesario debido a que el mapeo de objetos diferente mucho una de otra. Mapear los elementos de una aplicación se asemeja a hacerlo con una aplicación web, por otro lado, hacerlo para aplicaciones nativas implica conocer la estructura interna de cómo se construye la interfaz gráfica del aplicativo nativo.
- **Versión de la librería que da soporte a pruebas en móviles:** cada versión nueva soluciona incidencias reportadas en versiones antiguas de la librería y puede presentar nuevas, que a su vez pueden convertirse en riesgos para la construcción de pruebas dentro del sprint.
- **Precondiciones a establecer en las pruebas:** por lo general, estas precondiciones se crean usando pruebas de integración o al ejecutar acciones específicas hacia la base de datos.
- **Ambientes de ejecución de pruebas móviles:** esto responde a la pregunta ¿en donde se van a ejecutar las pruebas? Su ejecución puede ser local y si es así, se puede hacer en dispositivos reales o emuladores. Pero si se desea ejecutar en un ambiente externo, se debería considerar su ejecución en servicios de emuladores en la nube.

> *Lineamientos para lidiar con servicios externos usados por el aplicativo*

La estrategia seguida en el caso de la aplicación de transferencias monetarias fue la de establecer un mock que simulara las respuestas de los servicios bancarios, de forma que en ambiente de desarrollo y de pruebas se permitiera validar que la comunicación con el componente externo fuera la correcta y que el desarrollo de los servicios backend propios del aplicativo móvil si estuviesen funcionando como se esperab.

Llevar a cabo la estrategia representó en un primer momento capturar todas las posibles respuestas por cada caso de prueba, de forma que la simulación fuese lo más cercana a la realidad posible.

El uso de esta estrategia no dejó de lado las pruebas en ambiente certificación de los servicios externos, simplemente, permitió ahorrar tiempo en configuraciones y evitó que los trámites de acceso al ambiente se convirtiesen en una traba durante el sprint de trabajo.

Recomendaciones para el desarrollo de pruebas automatizadas

> Sobre el desarrollo de pruebas de integración

Una vez que ya se tiene construida la base técnica, es decir, el framework de pruebas automatizadas, entonces se puede proceder con el desarrollo mismo de las pruebas.

Para el caso de las pruebas de integración se tuvo que trabajar de cerca con los desarrolladores backend para que ellos pudieran explicar qué servicios se iban a implementar, qué particularidades se iban a encontrar, cuál era el contrato acordado con los desarrolladores frontend de forma que las incidencias reportadas por no enviar ni recibir la información correcta fueran casi nulas.

Con esta información preliminar, es posible empezar la construcción de las pruebas y esperar a tener una versión anticipada del desarrollo del servicio backend.

Cuando se tiene la versión preliminar de la construcción del servicio backend y se ejecutan las pruebas de integración puede ocurrir que el resultado retorne error y no se sepa si es error de la prueba bajo desarrollo por el ingeniero de calidad o que el servicio backend en construcción es el que presenta un error. En casos como este, se recomienda primero, revisar minuciosamente la implementación de la prueba y buscar la razón del error; cuando se determine que no es error del desarrollo de la prueba, entonces ir y conversar con el desarrollador backend para determinar la causa de la incidencia.

Realizar este ejercicio fue provechoso para los desarrolladores backend y el ingenieros de calidad a cargo, puesto que se podía obtener retroalimentación constante sobre la funcionalidad en construcción. Como valor agregado, una

vez que la construcción de la prueba era finalizada, los desarrolladores backend podían ejecutar las pruebas contra su propia implementación haciéndolos más productivos.

> *Sobre el desarrollo de pruebas end-to-end (pruebas para móviles y web)*

Para el caso de las pruebas end-to-end, se tuvo que trabajar de cerca con los diseñadores y los desarrolladores frontend-mobile. Fue necesario trabajar con los diseñadores debido a que ellos proveen los diseños finales que la aplicación debe mostrar y con esta información, era posible establecer las pruebas y validar las características más importantes.

Mientras los desarrolladores frontend-mobile realizaban la construcción de la funcionalidad a nivel visual, el ingeniero de calidad a cargo realizaba el desarrollo de la prueba sin considerar el mapeo de los objetos visuales.

Cuando los ingenieros frontend-mobile podían liberar una versión preliminar de su desarrollo, entonces se procedía a realizar el mapeo de los elementos por cada plataforma. Si la plataforma es web, la identificación de elementos es sencilla puesto que es la misma sin importar el navegador; sin embargo, se debe recordar que en el caso de los móviles, cada plataforma móvil (Android y iOS) tiene una forma muy distinta de identificar sus elementos de interfaz de usuario.

Cada prueba de aplicativo móvil construida debía ejecutarse para cada plataforma, puesto que en más de una ocasión fue posible ver que la funcionalidad pasaba la prueba con éxito en iOS pero no Android y viceversa. En casos como este, se procedía a conversar con los desarrolladores frontend para reportar la incidencia.

Estas pruebas son muy útiles cuando se trata de ejecutar una regresión en diferentes dispositivos móviles, ya que si bien se ejecutan siempre pruebas manuales para validar interacciones y evaluar la fluidez de la navegación, no es posible realizar pruebas exhaustivas en toda la combinatoria de dispositivos y sistemas operativos. Los responsables de negocio son quienes tienen la responsabilidad de establecer el alcance de los dispositivos de prueba en los cuáles se deben probar y, en base a su selección, se ejecutan las pruebas automatizadas y se priorizan las incidencias con respecto a la criticidad evaluada por el responsable de calidad del aplicativo. El mayor valor agregado que este tipo de pruebas aporta es cuando se está muy cerca a las fechas de entrega y de pase a producción, puesto que nos permiten saber el status del aplicativo en minutos y en el peor de los casos, en horas.

3.2.5 Contribuciones obtenidas en su aplicación en el caso contexto

El caso del presente informe de experiencia profesional es el contexto en donde se desarrolló el Framework Ágil para realizar pruebas de software orientado a aplicaciones móviles

El Framework Ágil para realizar pruebas de software orientado a aplicaciones móviles es un compendio de todos los aprendizajes obtenidos durante la realización del proyecto y cuyas prácticas fueron primero, sugeridas y al obtenerse buenos resultados fueron aplicadas como regla a lo largo del tiempo del proyecto.

A continuación se listan las contribuciones obtenidas al llevar a cabo las diferentes acciones establecidas por el framework en cada etapa del proyecto:

En el Sprint Inicial o Sprint 0

Se realizaron hasta varias pruebas de concepto con diversas tecnologías, entre ellas *Robot Framework*, *Appium*, usando como base diferentes lenguajes de programación como *Java*, *Python*.

Como resultado de las pruebas de concepto y del uso de los criterios de elección de herramientas base, se seleccionaron 05 tecnologías base:

- ***Ruby***: como lenguaje de programación, debido a que permite un fácil desarrollo aminorando tiempos de configuración en el proyecto.
- ***Cucumber***: para permitir expresar y ejecutar las pruebas en un lenguaje entendible por los expertos de negocio. También permite obtener también

reportes en diferentes formatos (.html, .json, .xml, etc) sobre el status de las pruebas.

- ***Appium***: para el manejo de la interfaz de usuario en Android y iOS.
- ***Gema rest-api***: *gema* es el nombre genérico que Ruby usa para denominar a dependencias o librerías. La *gema rest-api* permite el manejo de solicitudes y respuestas de servicios API.
- ***Gema mysql***: permite el manejo de la base de datos, por si en algún momento el framework requiriera establecer precondiciones por base de datos.
- ***Selenium WebDriver***: para el manejo de la interfaz de usuario en aplicaciones web

Con estas tecnologías se armó un framework de pruebas automatizadas, la cual permitía originalmente la construcción de pruebas de integración y de pruebas en aplicaciones móviles lo suficientemente estable como para soportar aplicaciones nativas o híbridas. Era necesario proveer este nivel de estabilidad puesto que en el Sprint 0 aún no se había tomado la decisión de qué tecnología usar.

Históricamente antes de cada Sprint de trabajo

Se llevaron a cabo una gran cantidad de sprints, donde en cada uno de ellos se obtuvieron diferentes aprendizajes. Llegado a cierto sprint de madurez, alrededor del sprint número 16, fue cuando las buenas prácticas para definición de historias de usuario empezaron a seguirse como lineamientos establecidos.

Es crítico notar en este punto que las buenas prácticas para la definición de historias de usuario hace hincapié en que los diseños se encuentren adjuntos a la historia y esto se debe a que bajo la metodología seguida, las historias de usuarios deben ser

comprender todos los aspectos de una funcionalidad y siendo el proyecto una aplicación móvil es mandatorio analizar su aspecto visual

En los sprints donde se tomó por idea dejar de lado las buenas prácticas determinadas por el framework (debido a que miembros del equipo en alguna reunión de retrospectiva hacían que las reuniones de backlog grooming se extendieran algunas horas), se tuvo un número mayor de historias de usuario sin terminar o llegando al plazo límite de entrega.

Al inicio, fue complicado realizar el diseño de casos de prueba en medio de una reunión pero con el pasar de los sprints, resultó más sencillo puesto que se convirtió en una costumbre y a la vez, fue fácil interiorizar los criterios establecidos para detectar oportunidades de automatización.

Históricamente durante cada Sprint de trabajo

Como equipo se establecía la visibilidad del progreso haciendo uso del tablero de trabajo. Apenas se iniciaba con el progreso de una historia, se comenzaba a dialogar con los desarrolladores a cargo de la construcción de la funcionalidad y con ello se podía iniciar el desarrollo de las pruebas automatizadas a nivel de backend.

En el caso de las pruebas a nivel de aplicativo móvil era posible implementar 02 de las 03 capas que requiere la implementación de la prueba:

- La primera capa es la de definición del requisito a validar en lenguaje de negocio.
- La segunda capa es la capa intermedia que permite la comunicación entre la definición del requisito y la implementación en código que permite la manipulación de los controles a nivel de interfaz gráfica.

- La tercera capa es la capa más técnica, en la que se establece el mapeo de objetos y de las múltiples acciones que se pueden hacer en cada vista.

Este último punto es aquel que no se conoce con certeza porque depende mucho de la estructura con la que la aplicación es construida.

Luego de que las implementaciones preliminares se volviesen accesibles al ingeniero de calidad, entonces se volvía posible la implementación complementaria de la prueba y su posterior ejecución.

Si la ejecución de prueba presentaba un defecto, se pide el apoyo de los desarrolladores para determinar si la causa raíz es un error a nivel de desarrollo de script de pruebas o si es un defecto.

Históricamente al finalizar cada Sprint de trabajo

Como parte de Scrum, se sostuvieron reuniones de retrospectiva en donde se evaluaba el progreso del equipo. En estas reuniones se obtenía el sentir de cada integrante y por lo que estas sesiones se convirtieron en la fuente de muchas lecciones aprendidas.

Inmediatamente después, se llevaron a cabo reuniones de demostración del aplicativo con la finalidad de que todos los miembros del equipo conozcan el estado del aplicativo. Por lo general, este tipo de reuniones eran conducidas por el ingeniero de calidad a cargo, ya que él tiene gran parte del conocimiento de negocio y base técnica para poder explicar a los responsables de negocio qué impedimentos se tuvieron si por alguna razón alguna historia no fue concluida.

A lo largo de la vida del proyecto

El framework de pruebas automatizadas, al igual que los proyectos de desarrollo, evoluciona a medida que el tiempo transcurre y que aparecen necesidades en cobertura de pruebas (ligada a nuevas funcionalidades a construir por el equipo).

En la Figura 04 se muestra la arquitectura del framework para pruebas automatizadas en donde se puede apreciar cómo el framework interactúa de forma directa con el aplicativo backend, la base de datos del producto a ofrecer a usuarios finales, el aplicativo móvil, módulo de servicios de soporte a qa (creado y mantenido por los desarrolladores backend), con el aplicativo web y con el mock que simula el comportamiento de los servicios bancarios y/o servicios externos en general.

Arquitectura del Framework de pruebas automatizar

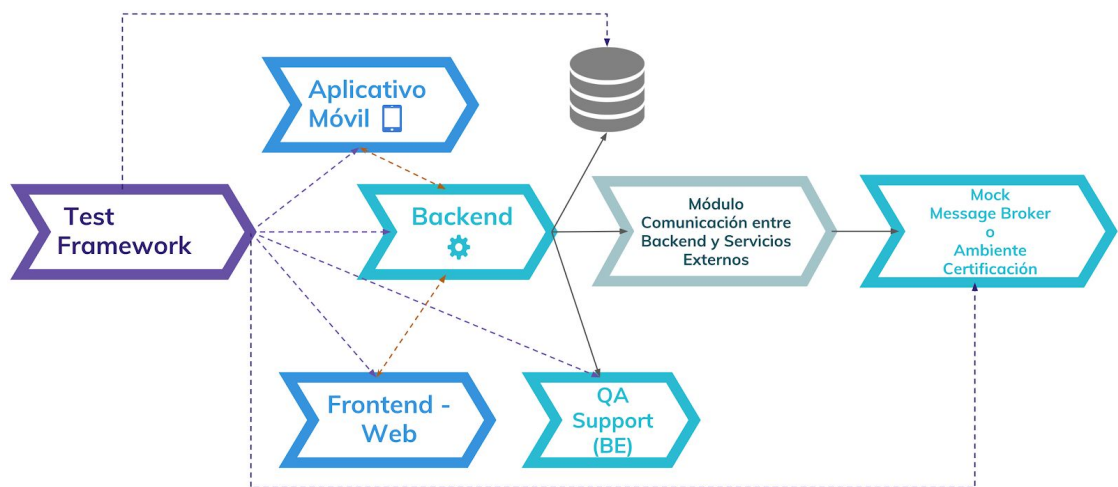


Figura 05: Arquitectura del framework de pruebas automatizadas para la aplicación móvil de transferencias bancarias. Fuente propia.

La estructura de las pruebas a nivel de móviles se dividió en 03 capas y tuvo como base *Cucumber* y *Appium* en *Ruby*:

- La primera que es la capa de negocio. En la Figura 05 se muestra un ejemplo desarrollado por el autor del presente informe sobre la definición a nivel de negocio que se debe hacer de un requisito.



```
16 lines (13 sloc) | 490 Bytes
Raw Blame History
1 @run_android
2 Feature: As a user
3   I want the play 2048 because its fun
4
5 Scenario: Create New Game
6   Given I have read the new updates of the game
7   When I want to start a new game
8   Then I should be able to confirm I want a new game
9   And I should be able to see new game board
10
```

Figura 06: Ejemplo de una definición de pruebas basada en requisitos de negocio.

Fuente propia.

- La segunda es la capa intermedia de comunicación entre la definición de pruebas en lenguaje de negocio y la definición de la vista a nivel de construcción del aplicativo. En la Figura 06 se muestra un ejemplo relacionado a la Figura 05.

```
37 lines (28 sloc) | 846 Bytes
Raw Blame History
1 Given(/^I have read the new updates of the games$/) do
2   on(GameNewsScreen).being_shown?.should be true
3   on(GameNewsScreen).dismiss_change_log
4   on(GameBoard).being_shown?.should be true
5 end
6
7 When(/^I want to start a new game$/) do
8   on(GameBoard).choose_start_new_game
9 end
10
11 Then(/^I should be able to confirm I want a new game$/) do
12   on(GameBoard).confirm_new_game
13 end
14
15 And(/^I should be able to see (?:new|the) game board$/) do
16   on(GameBoard).being_shown?.should be true
17 end
```

Figura 07: Ejemplo del desarrollo de la capa de comunicación entre los requisitos de negocio y la definición de las vistas del aplicativo móvil. Fuente propia.

- La tercera capa es la parte más técnica de la implementación puesto que refiere a la definición de las vistas web o móviles, o a la definición de API de forma que se permita su manipulación.

Si se implementaran pruebas de integración, se usaría la gema *rest-api* y si las pruebas fueran web, entonces se mapearían elementos y controles web con el uso de *Selenium WebDriver*.

En la Figura 07 se muestra la definición de una vista de aplicativo móvil en donde se mapean los elementos y controles presentes de forma que a nivel de prueba se pueda simular el comportamiento humano. El soporte para el manejo de controles en móviles se hizo mediante el uso de *Appium*.

```

47 lines (34 sloc) | 1.73 KB
Raw Blame History

1 require 'appium_lib'
2
3 class GameBoard < BaseMobileScreen
4
5   Undo_action_xpath = "//android.view.View[@content-desc='Undo the current move?']"
6   Start_game_action_xpath = "//android.view.View[@content-desc='Start a new game?']"
7   Tiles_xpath = "(//android.view.View[@content-desc='Undo']/following-sibling:android.view.View//android.view.View)[2]//*[con
8
9   element(:board_desc_title, :xpath, "//android.view.View[@content-desc='Join the numbers and get to the 2048 tile!']")
10
11  element(:new_game_btn, :xpath, "//android.view.View[@content-desc='New Game']")
12  element(:new_game_confirmation_desc, :xpath, Start_game_action_xpath)
13  element(:new_game_confirmation_btn, :xpath, "#{Start_game_action_xpath}/following-sibling:android.view.View//*[content-desc
14  element(:new_game_cancel_btn, :xpath, "#{Start_game_action_xpath}/following-sibling:android.view.View//*[content-desc='Canc
15
16  element(:undo_btn, :xpath, "//android.view.View[@content-desc='Undo']")
17  element(:undo_confirmation_desc, :xpath, Undo_action_xpath)
18  element(:undo_confirmation_btn, :xpath, "#{Undo_action_xpath}/following-sibling:android.view.View//*[content-desc='Undo']")
19  element(:undo_cancel_btn, :xpath, "#{Undo_action_xpath}/following-sibling:android.view.View//*[content-desc='Cancel']")
20
21  elements(:game_tiles, :xpath, "#{Tiles_xpath}")
22
23  def being_shown?
24    board_desc_title.enabled?
25  end
26
27  def choose_start_new_game
28    new_game_btn.click
29  end
30
31  def confirm_new_game
32    new_game_confirmation_btn.click
33  end
34
35  def choose_undo
36    undo_btn.click
37  end
38
39  def confirm_undo_action
40    undo_confirmation_btn.click
41  end
42
43  def move_tile
44    @appium.swipe start_x: 0, start_y: 0, end_x: 100, end_y: 100, duration: 0.8
45  end
46
47  end

```

Figura 08: Definición de los elementos y controles de una vista de aplicativo móvil.

Fuente propia.

Por último, cuando se lidia con un proyecto de desarrollo evolutivo, también se debe considerar la evolución del framework de pruebas automatizadas y el mantenimiento y actualización de las pruebas.

3.3 EVALUACIÓN

3.3.1 Evaluación económica

Al decidir realizar el desarrollo y aplicación del framework, se consideraron los costos y beneficios que se obtendrían:

Costos directos

Inversión en el salario del ingeniero de calidad especialista a cargo, en el tiempo que requería en capacitarse así mismo e investigar diferentes alternativas para escoger la mejor según contexto del proyecto. También se incluyó la compra de una laptop Mac con 16GB de RAM (esto se requería para poder automatizar aplicativos iOS) y las licencias para entornos integrados de desarrollo.

Costos de mantenimiento

Pago mensual en la plataforma cloud de emuladores móviles, inversión en el contrato del ingeniero de calidad a cargo, mantenimiento del equipo Mac.

Costos de oportunidad

Debido a que los profesionales que dominan esta tecnología son escasos, el costo de transmitir conocimiento a otros miembros del equipo es alto en tiempos.

Beneficios

Los beneficios obtenidos al aplicar las prácticas establecidas en el Framework Ágil para realizar Pruebas Software orientado a Aplicaciones Móviles fueron los siguientes:

- Requisitos ambiguos casi nulos.
- Ciclos de retesting no mayores a 04 por cada funcionalidad implementada.
- Cero rollbacks en pases a producción causado por incidencias graves detectados en el aplicativo móvil y sus servicios empleados.

- Bajo esfuerzo en pruebas manuales. Del 100% de casos de pruebas, sólo el 20% o menos se ejecutan de forma manual, donde este porcentaje está compuesto por casos particulares y validación visual del acabado a nivel de interfaz de usuario.
- Las pruebas integración ejecutan casos hasta 35x veces más rápido de lo que hubiera tardado un tester manual.
- Las pruebas end-to-end ejecutan casos hasta 4.5x veces más rápido de lo que hubiera tardado hacer este ejercicio de forma manual.
- Teniendo pruebas automatizadas ya construidas, se puede realizar la ejecución en diversos ambientes y dispositivos con diferentes características en horas no laborables, evitando que el personal experto en pruebas deba realizar trabajos fuera de hora y ahorrando costos por horas extras.

CAPÍTULO IV. REFLEXIÓN CRÍTICA DE LA EXPERIENCIA

La participación del autor del presente informe con respecto al Framework Ágil para realizar Pruebas de Software orientado a Aplicaciones Móviles fue en su creación e implementación en el proyecto, ya que ella era la responsable por el control de calidad.

El equipo responsable por el desarrollo del aplicativo estuvo conformado por 03 desarrolladores Backend, 03 desarrolladores Frontend - Mobile, un Arquitecto, un ingeniero DevOps y el autor de este informe, como Ingeniero de Calidad Automatizador de Pruebas.

Los esfuerzos de :

- establecer buenas prácticas,
- decidir qué estrategias de pruebas seguir según la naturaleza del proyecto,
- mostrar a los responsables de negocio cuáles son los beneficios de implementar pruebas automatizadas para que puedan destinar tiempo para su desarrollo y de
- desarrollar las pruebas automatizadas a nivel de integración y end-to-end que permitan tener una buena cobertura de los flujos de la aplicación

tomaron más de un año de trabajo, en el que se interactuó con todos y cada uno de los integrantes del equipo y su apoyo desde su perspectiva de desarrollo fue muy valiosa.

El resultado final fue un éxito gracias a la colaboración y constancia de todo el equipo y de los responsables de negocio, ya que sin su apoyo, este esfuerzo no hubiera podido materializarse.

Es importante recalcar que al establecer un framework, parte fundamental es tomar cada acción y ejecutarla como parte de una rutina, con ello, se logra crear una cultura de trabajo grupal orientada a la calidad.

CAPÍTULO V. CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

El objetivo principal de este proyecto era el de establecer un framework ágil que sirva como base para la realización de pruebas de software en aplicaciones móviles similares. Actualmente la aplicación en la que se aplicó el framework se encuentra en producción y con la calidad esperada por el cliente. Las conclusiones obtenidas son las siguientes:

1. Establecer el proceso de trabajo y la aplicación de buenas prácticas establecidas por el framework para la definición de historias de usuario es positivo puesto que se permite iniciar las actividades de pruebas desde la concepción de los requisitos funcionales, aclarar las expectativas de los involucrados, solicitar cambios en el diseño de la interfaz de usuario y a nivel de pruebas minimiza la cantidad de ciclos de retesting.
2. Las herramientas de soporte desarrolladas por los responsables de control de calidad con enfoque a pruebas permiten la fácil construcción de pruebas automatizadas; mientras que las herramientas con enfoque a evitar dependencias en el desarrollo, tales como los mocks, permiten no sólo eliminar futuros bloqueos en pruebas, sino también bloqueos presentes a los programadores cuando ellos deseen probar sus desarrollo progresivamente.
3. Se estableció la forma de trabajo que los ingenieros de calidad debían seguir con respecto al diseño y construcción de pruebas con el fin de agregar el mayor valor posible al negocio y minimizar los tiempos de ejecución de prueba al máximo. Esta forma
4. La automatización de pruebas es una actividad que no debe ser tomada como opcional, porque reduce dramáticamente los tiempos de ejecución de pruebas de aplicaciones móviles (las pruebas end-to-end son 4.5x veces más veloces y las

pruebas de integración son hasta 35x veces más rápidas) y permite incrementar la cobertura de pruebas sobre múltiples dispositivos de prueba.

5. El esfuerzo hecho en el presente trabajo muestra un caso de éxito puesto que lo narrado se puso en práctica sobre una aplicación móvil de transferencias bancarias con soporte en los sistemas operativos Android y iOS que cumple con las expectativas del cliente y se encuentra actualmente en producción.

5.2 RECOMENDACIONES

1. Sensibilizar a todo el equipo de trabajo sobre la importancia del control de calidad, con el fin de obtener la colaboración y retroalimentación de todos.
2. Las reuniones en las que se definen las historias de usuarios son en extremo importantes para los responsables de la calidad del aplicativo y no deben ser despriorizadas por ninguna razón, ya que este es el primer momento que tienen para poder iniciar con el diseño de casos de prueba y la identificación de posibilidades de automatización dentro del sprint.
3. Se recomienda seguir las buenas prácticas propuestas por el framework ágil para pruebas de software, ya que, en los sprints donde no se siguieron dichos lineamientos (por situaciones ajenas al equipo de trabajo), las entregas del sprint se vieron afectadas con historias a medio completar, o entregando a los responsables de control de calidad hasta horas antes de la reunión de Sprint Review.
4. Este framework de pruebas funcionales tiene la base suficiente como para permitir una extensión e incluir pruebas no funcionales, como por ejemplo, pruebas de stress.

5.3 FUENTES DE INFORMACIÓN

[1] Beck, Kent *et al*, *Manifesto for Agile Software Development*, 2001, página web:
<http://agilemanifesto.org/iso/es/manifesto.html>

[2] Krishnamurty, V., Scrum Alliance, *Agile User Stories*, September 2013. página web de artículos:
<https://www.scrumalliance.org/community/articles/2013/september/agile-user-stories>

[3] Schwaber , Ken & Sutherland, Jeff, *Scrum Guide*, November 2017. [versión electrónica].
<http://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf>

[4] Spadini, D., Aniche, M., Bruntink, M. and Bacchelli, A., *To Mock or Not To Mock? An Empirical Study on Mocking Practices*, en *14th International Conference on Mining Software Repositories*, 2017, pp. 402-412.

[5] *What is Agile Software Development?* June 2013. Agile Alliance, página web:
<https://www.agilealliance.org/agile101/>

[6] *What is DevOps.* 2017. Amazon Web Services, página web:
<https://aws.amazon.com/devops/what-is-devops/>

5.4 GLOSARIO

- **API**

En desarrollo de software, *API (application programming interface)* es el término que se usa para denominar al conjunto de definiciones de subrutinas, protocolos y herramientas para realizar acciones específicas. Su base es la definición de un contrato, una forma de comunicación, de forma que componentes externos puedan usarlo en base a la información que se le envía.

- **APLICATIVO HÍBRIDO**

Es un tipo de aplicación móvil cuya construcción está basada en tecnologías web que permiten manipular varias de las opciones nativas de los dispositivos móviles. Aunque no permiten controlar todas las opciones nativas, pueden ejecutarse en la mayoría de plataformas móviles, como Android y iOS.

- **APLICATIVO NATIVO**

Es un tipo de aplicación móvil cuya construcción se hace de forma específica para un sólo tipo de plataforma y permiten tener control de todas las opciones nativas de los dispositivos móviles. Es posible crear aplicaciones para diferentes plataformas, pero ello implica que se haga en proyectos diferentes con tecnologías diferentes.

- **BACKEND**

En el mundo de desarrollo de software, éste es un término que se refiere a los sistemas por los que el usuario no accede directamente, que realizan funciones especializadas de procesamiento y que es el soporte de una aplicación frontend.

- **DAILY SCRUMS**

También, llamado Stand-ups o Dailym meetings, son un eventos de 15 minutos cada uno para el Equipo de Desarrollo. El Daily Scrum se lleva a cabo todos los días del Sprint. En él, el Equipo de Desarrollo planea trabajar durante las próximas 24 horas [3].

- **DEVOPS**

DevOps es la combinación de filosofías, prácticas y herramientas culturales que aumentan la capacidad de una organización para entregar aplicaciones y servicios a gran velocidad [6].

Bajo un modelo DevOps, los equipos de desarrollo y operaciones ya no están "aislados". A veces, estos dos equipos se fusionan en un único equipo donde los ingenieros trabajan en todo el ciclo de vida de la aplicación, desde desarrollo y prueba hasta implementación y operaciones, y desarrollan un rango de habilidades no limitadas a una sola función. Los equipos de seguridad y aseguramiento de la calidad también pueden integrarse más estrechamente con el desarrollo y las operaciones, y durante todo el ciclo de vida de la aplicación [6].

Estos equipos usan prácticas para automatizar procesos que históricamente han sido manuales y lentos. Usan una pila de tecnología y herramientas que les ayudan a operar y desarrollar aplicaciones de forma rápida y confiable. Estas herramientas también ayudan a los ingenieros a realizar tareas de manera independiente (por ejemplo, implementar código o infraestructura de aprovisionamiento) que normalmente requerirían ayuda de otros equipos, y esto aumenta aún más la velocidad de un equipo [6].

- **FRONTEND**

En el mundo de desarrollo de software, éste es un término que se refiere a los aplicativos con los que los usuarios finales interactúan directamente.

- **FINTECH**

Contracción de palabras de habla inglesa *finance* y *technology* usado para denominar a empresas de tecnología de la información que crean y/u ofrecen servicios relacionado al campo financiero. Por lo general, las Fintech son empresas startups expertas en nueva tecnología que buscan captar parte del mercado dominado por grandes empresas, que a menudo son poco innovadoras o lentas en adopción de tecnologías.

- **HISTORIA DE USUARIO**

Del inglés *User Story*, es una descripción breve y simple de una función contada desde la perspectiva de la persona que desea la nueva funcionalidad, generalmente un usuario o cliente del sistema. La esencia de la obtención de los requisitos de uso y centrados en el usuario es centrarse en lo que el usuario quiere tener, no en lo que el usuario quiere que el sistema haga [2].

- **OPEN SOURCE**

En el ámbito de la informática, Open Source, es el término usado para denominar a desarrollos de software cuyo código es de libre acceso por cualquier desarrollador y ello permite que ellos contribuyan a su desarrollo, solución de posibles errores y de mejorar la calidad de su código,. Nació con la filosofía de trabajo conjunto sobre los proyectos informáticos.

- **PRODUCT BACKLOG**

Es una lista ordenada de todo lo que se sabe que es necesario en el producto. Es la única fuente de requisitos para cualquier cambio que se realice en el producto. El desarrollo más temprano de él establece lo inicialmente conocido y evoluciona a medida que el producto y el ambiente en el que se use lo haga [3].

Enumera todas las características, funciones, requisitos, mejoras y soluciones que constituyen los cambios que se realizarán en el producto en futuras versiones. Los artículos del Backlog

del producto tienen los atributos de una descripción, orden, estimación y valor. Los requisitos nunca dejan de cambiar, por esta lista crece y decrece constantemente [3].

- **PRODUCT OWNER**

En español, el dueño del producto. En Scrum, es el rol responsable de maximizar el valor del producto resultante del trabajo del Equipo de Desarrollo. Es la única persona responsable de administrar el Product Backlog [3].

La gestión del Backlog incluye [3]:

- Expresar claramente las historias de usuario o tareas en general a realizar.
- Ordenar las historias de usuarios y tareas en general que se encuentran en el Product Backlog para alcanzar los objetivos.
- Optimizar el valor del trabajo que realiza el Equipo de Desarrollo.
- Asegurar que el equipo de desarrollo comprenda los ítems del Product Backlog.

- **PRUEBAS DE REGRESIÓN**

Es el conjunto de pruebas que se ejecuta sobre un componente previamente probado en donde se han introducido cambios y que tienen por objetivo asegurar que no se han agregado nuevos defectos con las modificaciones realizadas.

- **PRUEBA DE INTEGRACIÓN**

Prueba cuyo objetivo es el aseguramiento de la correcta comunicación entre componentes, es decir, valida las interacciones entre las diferentes partes de un sistema, tales como el sistema operativo, sistema de archivos, hardware, o interfaces con otros sistemas.

- **PRUEBA END-TO-END**

También conocida como pruebas de fin a fin, tiene como objetivo probar la aplicación final, aquella que va a ser ofrecida usuarios finales, ya que la misión del proyecto en sí es la satisfacción del usuario y sus requisitos.

- **RETESTING**

Es la acción de volver a ejecutar la prueba en una incidencia que ha sido identificada y corregida, con el fin de asegurar que dicho defecto ya no se presenta.

- **SERVICIOS DE MÓVILES EN LA NUBE**

Algunos términos también usados son *Mobile Test Cloud* o *Device Farm*, refiere a servicios ofrecidos que permiten usar emuladores o dispositivos móviles reales en remoto, es decir, sin que el dispositivo se encuentre en el lugar físico del tester. Empresas que ofrecen estos servicios como Saucelabs, Perfecto Mobile, Amazon Web Services y demás.

- **SCRUM MASTER**

Scrum Master es un líder sirviente del equipo. El Scrum Master ayuda a aquellos ajenos al Scrum Team a comprender cuáles de sus interacciones con el Scrum Team son útiles y cuáles no. El Scrum Master ayuda a todos a cambiar estas interacciones para maximizar el valor creado por el equipo de Scrum [3].

Scrum Master apoya al Product Owner de varias maneras, que incluyen [3]:

- Asegurar que los objetivos, el alcance y el dominio del producto sean entendidos por todos en el equipo de Scrum de la mejor manera posible.
- Encontrar técnicas para una gestión eficaz de la acumulación de productos.
- Ayudar al equipo de Scrum a comprender la necesidad de elementos claros y concisos de productos atrasados.
- Comprender la planificación de productos en un entorno empírico.
- Asegurar que el Propietario del Producto sepa cómo organizar el Retraso del Producto para maximizar el valor.

El Scrum Master sirve al Equipo de Desarrollo de varias maneras, que incluyen [3]:

- Coaching del equipo de desarrollo en autoorganización y funcionalidad cruzada.
- Ayudando al equipo de desarrollo a crear productos de alto valor.

- Eliminar impedimentos para el progreso del Equipo de Desarrollo.
- Facilitar eventos Scrum según se solicite o necesite
- Coaching del equipo de desarrollo en entornos organizacionales en los que Scrum todavía no se ha adoptado ni entendido completamente.

El Scrum Master sirve a la organización de varias maneras, que incluyen [3]:

- Liderando y entrenando a la organización en su adopción de Scrum.
- Planear implementaciones de Scrum dentro de la organización.
- Ayudar a los empleados y partes interesadas a comprender y promulgar Scrum y el desarrollo de productos empíricos.
- Trabajar con otros Scrum Masters para aumentar la efectividad de la aplicación de Scrum en la organización.

● SPRINT

Es el núcleo de Scrum, un periodo de 1, 2 o 4 semanas durante el cual se crea un incremento de producto, utilizable y potencialmente liberable. Los sprints tienen duraciones consistentes a lo largo de un esfuerzo de desarrollo. Un nuevo Sprint comienza inmediatamente después de la conclusión del Sprint anterior [3].

Los sprints contienen y consisten en Sprint Planning, Daily Scrums, el trabajo de desarrollo, Sprint Review y Sprint Retrospective [3].

El alcance se puede aclarar y renegociar entre el propietario del producto y el equipo de desarrollo a medida que se aprende más [3].

Cada Sprint puede considerarse un proyecto con un horizonte no mayor a un mes. Al igual que los proyectos, Sprints se utilizan para lograr algo. Cada Sprint tiene una meta de lo que se construirá, un diseño y un plan flexible que guiará su construcción, el trabajo y el incremento resultante del producto [3].

Los Sprints permiten la previsibilidad al garantizar la inspección y la adaptación del progreso hacia un objetivo de Sprint al menos cada mes calendario. Los Sprints también limitan el riesgo a un mes calendario de costo [3].

- **SPRINT BACKLOG**

Es el conjunto de elementos del Product Backlog seleccionado para un Sprint, más un plan para entregar el siguiente incremento del producto y así, alcanzar el objetivo del Sprint [3]. Sprint Backlog es un pronóstico del Equipo de Desarrollo sobre qué funcionalidad será en el próximo incremento [3].

- **SPRINT PLANNING**

Es el plan de trabajo para el siguiente Sprint. Es creado por todo el equipo de Scrum. El Sprint Planning responde lo siguiente [3]:

1. ¿Qué se puede entregar en el Incremento resultante del próximo Sprint?
2. ¿Cómo se logrará el trabajo necesario para lograr el Incremento?

- **SPRINT RETROSPECTIVE**

En español, Retrospectiva de Sprint, es una oportunidad para que el equipo se inspeccione a sí mismo y cree un plan para que se implementen mejoras durante el próximo Sprint [3]. Ocurre después de la Revisión Sprint y antes de la próxima Planificación Sprint. Esta es como máximo una reunión de tres horas para Sprints de un mes. Para Sprints más cortos, el evento suele durar entre 45 minutos a 1 hora. El Scrum Master asegura que el evento tenga lugar y que los asistentes entiendan su propósito [3].

- **SPRINT REVIEW**

En español, Revisión de Sprint, es una reunión que se lleva a cabo al final del Sprint para inspeccionar el Incremento y adaptar el Retraso del Producto si es necesario [3]. Durante la Revisión de Sprint, el Equipo de Scrum y las partes interesadas colaboran sobre lo que se hizo en el Sprint. En función de eso y de cualquier cambio en la acumulación de productos

durante el Sprint, los asistentes colaboran en las siguientes acciones que podrían hacerse para optimizar el valor [3].

- **STARTUP**

Empresa o compañía emergente. Término usado para denominar a empresas que buscan emprender un nuevo negocio y aluden a ideas de negocio novedosas. Por lo general, su principal soporte es la tecnología.