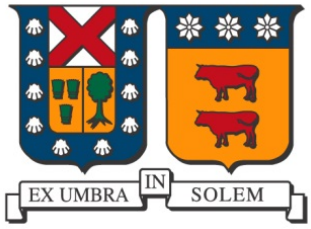
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“PLAN DE CALIDAD PARA DRACON”

ROBERTO SEBASTIÁN DÍAZ CALDERÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

Profesor Guía: Marcelo Visconti
Profesor Correferente: Pedro Francisco Godoy Barrera

Octubre - 2019

RESUMEN

Resumen— Dracon es una aplicación para smartphone que unifica servicios de distintos bancos bajo una misma interfaz. La aplicación tiene como objetivo ser una alternativa a los medios de pago convencionales, entregando la misma seguridad y conveniencia de otros medios de pago digitales.

De este modo, se tendría una manera de manejar el dinero digital, independiente de aparatos externos y accesible tanto para negocios pequeños como para gente común que quisiera realizar transferencias entre conocidos, o vendedores independientes.

El siguiente plan de calidad busca establecer una metodología que permita al equipo de desarrollo lograr un estándar de calidad adecuado que entregue seguridad al usuario, comodidad en el uso de la aplicación, y que finalmente logre convencerlo de utilizarla en comparación a otras alternativas.

ABSTRACT

Abstract— Dracon is a smartphone application that unifies services from different banks under the same interface. The app's objective is to be an alternative to the conventional payment methods, delivering the same security and convenience provided by other digital means of payment.

That way, we could have a way of administrating digital money independent from external devices, and affordable both for small businesses and common people that would like to make transfers between acquaintances, or to independent sellers.

The following quality plan aims to establish a methodology that allows the developer team to reach adequate quality standards that provide assurance to the user, comfort when using the app, and to finally convince them of choosing it over the alternatives.

GLOSARIO

- DI: Departamento de Informática.
- FESW: Feria de Software
- Framework: Conjunto estandarizado de conceptos, herramientas y prácticas que sirve para resolver un tipo de problemática, y es reutilizable entre distintos proyectos que compartan contextos similares.
- HH: Horas Hombre
- Hardware: Componentes físicos de una computadora que en conjunto permiten su funcionamiento.
- Peer Programming: Técnica de desarrollo ágil de Software en la cual los desarrolladores trabajan en pares, en donde uno escribe código, y el otro analiza el código escrito. Ambos roles son intercambiables, y la técnica promueve un plan estratégico para el desarrollo.
- POS: Point Of Sale (Punto de venta)
- UTFSM: Universidad Técnica Federico Santa María.

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN

De acuerdo al sitio de la **Organización Internacional para la Estandarización** (International Organization for Standardisation, ó ISO), se define un plan de calidad como un *documento que especifica qué procedimientos y recursos asociados deben aplicarse, quién debe aplicarlos y cuándo deben aplicarse a un proyecto, producto, proceso o contrato específico*[1]. En el siguiente documento se plantea el plan de calidad para un proyecto basado en la metodología ágil SCRUM, proporcionando al equipo desarrollador una pauta sobre la cual trabajar, y que busca finalmente optimizar la eficiencia de éste en base a conceptos como la cooperación y la comunicación.

1.1. Propósito

Para comercializar la aplicación *Dracon*, tomando como referencia la versión presentada en la XXVI Feria del Software (2018) de la Universidad Técnica Federico Santa María, es necesario definir los estándares de calidad que respalden su viabilidad para el uso cotidiano.

Por lo tanto, en el documento se establecerán dichos estándares de calidad, y las pautas y procedimientos a seguir por el equipo de desarrollo para cumplirlos.

1.2. Alcance

El alcance del presente plan de calidad es definir las metodologías adoptadas a lo largo de las distintas etapas de desarrollo del proyecto, que van desde su concebimiento hasta su puesta en marcha y futura mantención de la aplicación.

Se definirán además los potenciales riesgos presentes en el proceso de desarrollo, junto a medidas de mitigación para cada uno.

Por último, el enfoque de este plan de calidad será respecto a los requerimientos no funcionales del proyecto, puesto que estos permitirán finalmente entregar una experiencia positiva de uso de la aplicación para el usuario final.

[1] **Sistemas de Gestión de Calidad - Fundamentos y Vocabulario.**
`https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-3:v1:es:term:3.7.5`

REQUERIMIENTOS NO FUNCIONALES

Los requerimientos no funcionales corresponden a características generales o restricciones de la aplicación, asociadas a su rendimiento y calidad.

2.1. Definición de Atributos de Calidad

Los atributos de calidad de Software son el conjunto de factores que definen el desempeño del producto final.
En el contexto de *Dracon*, la aplicación debe ser fácil de usar, y su uso cotidiano no debe verse entorpecido por un mal entendimiento de la interfaz o fallas de conexión, que son puntos vitales para su funcionamiento correcto. Además, debe ser seguro debido a la delicadeza de los datos manejados, y capaz de funcionar en la mayor cantidad de dispositivos móviles para alcanzar un nivel de uso relevante para entrar al mercado. Esto último implica que se debe velar por satisfacer en primer lugar las necesidades de los SmartPhones más antiguos dentro de un rango determinado de popularidad de uso, considerando tanto Sistema Operativo como el *hardware* sobre el que operan dichos sistemas.

Basado en los modelos de calidad propuestos por McCall[ProfessionalQA, 2016] y Boehm[ProfessionalQA, 2019], y considerando además las características de la aplicación móvil, se considera que el rendimiento de *Dracon* debe ser medido mediante los siguientes criterios:

1. **Rendimiento -** Tiempo de respuesta que toma realizar una tarea o acción.
2. **Seguridad -** Existencia de controles administrativos que limitan el libre acceso a la información de la aplicación.
3. **Fiabilidad -** Correctitud y consistencia en el funcionamiento de las distintas funcionalidades del software.
4. **Interfaz -** Intermediario entre el usuario y las funcionalidades de la aplicación. Debe ser atractiva, simple e intuitiva.
5. **Usabilidad -** Simpleza de la aplicación y cantidad de pasos que debe recorrer el usuario desde una pantalla para acceder a la función deseada.
6. **Escalabilidad -** Capacidad del software para crecer e incorporar nuevas funcionalidades, manteniendo la calidad de sus otros servicios.
7. **Portabilidad -** Capacidad de la aplicación para ejecutarse en distintos equipos y distintos sistemas operativos.

Las razones detrás de cada una y cómo deben ser medidas se plantean en el punto a continuación.

2.2. Objetivos Cuantificables

Es importante que la calidad de los requerimientos no funcionales sea objetiva, por lo que estos se definen en base a atributos cuantificables, los cuales conforman los criterios a utilizar para la evaluación de cada requerimiento.

2.2.1. Rendimiento

Objetivos cuantificables

Dado que Dracon es una aplicación móvil planteada como una alternativa de pago, se espera que las transacciones realizadas sean lo suficientemente rápidas para ameritar su uso en el contexto cotidiano, simulando la velocidad que toma realizar un pago con tarjeta en el mejor caso, y no demorando más que un pago con dinero físico.

1. Rápidez de carga de detalles de cuentas involucradas.
2. Rápidez de transacciones.

Criterios de evaluación

Criterio	Medición
Rápidez de carga de detalles de cuentas.	Al momento de realizar una transacción entre dos cuentas, el tiempo de carga de los detalles de las cuentas involucradas en la operación debe ser < 0.5 segundos.
Rapidez de transacciones.	Tiempo transcurrido entre la confirmación de la transacción por parte del usuario y la respuesta asociada a esta por parte de la aplicación < 2 segundos.

Tabla 1: Criterios de rendimiento. Cada criterio tiene un tiempo asociado a cuanto se espera que demore la acción respectiva.

2.2.2. Seguridad

Objetivos cuantificables

Dracon trabaja con información sensible para el usuario, por lo que se debe tener una protección robusta de estos, evitando así accesos no autorizados y/o la filtración de estos, por lo cual se requiere la menor exposición posible de dichos datos, y una clave maestra que sirva tanto para el acceso a la aplicación como para acciones comprometedoras como las transferencias de dinero.

1. Contraseña maestra segura
2. Limitar cantidad de intentos de acceso
3. Cifrado de datos
4. Manejo local de información

Criterios de evaluación

Criterio	Medición
Contraseña maestra segura.	La contraseña maestra de la aplicación consiste en una combinación de 6 a 8 números.
Limitar cantidad de intentos de acceso.	Al ingresar la contraseña erróneamente más de 10 veces consecutivas, se recurre a un borrado de seguridad de los datos guardados.
Cifrado de datos.	La aplicación debe contar con un algoritmo de cifrado para los datos asociados a la aplicación, incluyendo la contraseña.
Manejo local de información.	La contraseña maestra y las credenciales bancarias guardados por el usuario deben existir únicamente en su dispositivo.

Tabla 2: Criterios de seguridad. Estos cirterios cubren todos los aspectos asociados a la seguridad de datos locales, mientras que las APIs de los bancos proporcionan la seguridad en línea.

2.2.3. Fiabilidad

La aplicación debe ser consistente en los resultados que entrega, ya que su uso es idealmente cotidiano, y las inconsistencias en el producto llevan al descontento y a un eventual abandono de la aplicación.

Objetivos cuantificables

1. Notificar al usuario cuando se esté realizando una operación en línea.
2. Notificar resultado de una transacción monetaria a usuarios involucrados.
3. Entregar feedback de cambios realizados en distintas funcionalidades de la aplicación.
4. Monitorear estado de servidores bancarios.
5. Alertar a usuario de fallas en los servicios.
6. Notificar al usuario cuando se presenten errores al realizar transacciones.

Criterios de evaluación

Criterio	Medición
Notificar al usuario cuando se esté realizando una operación en línea.	La aplicación debe mostrar una alerta con una animación de peso < 200 [kB], y tamaño $< 50x50$ [px], además de el texto ”Cargando” mientras se procese cualquier acción que requiera conexión a internet. De no estar conectado el dispositivo a la red, debe notificar al usuario que no se encuentra conectado.
Notificar resultado de una transferencia de dinero a los usuarios involucrados.	La aplicación debe notificar el éxito de la operación tanto al usuario emisor de la transferencia como al receptor de esta.
Entregar feedback de cambios realizados en distintas funcionalidades de la aplicación.	Tras realizar alguna acción de tipo CRUD para los **contactos**, **grupos** o **eventos**, la aplicación debe mostrar el resultado de dicha acción tal que: ▪ Al agregar o eliminar un elemento, la aplicación debe volver a la vista *index* con la lista de los otros elementos del mismo tipo. ▪ Al editar un elemento, la aplicaicón debe volver a la vista detallada de dicho elemento, con los campos que fueron editados ya actualizados.
Monitorear estado de servidores bancarios.	La aplicación debe contar con una vista en la que se muestre el estado actual de cada servicio bancario vinculado a la aplicación, diferenciando los estados: disponible, intermitente, y caído, con una explicación de cada estado que no supere los 75 carácteres.
Alertar a usuario de fallas en los servicios.	Aplicación debe mostrar alerta en caso de que algún servicio presente estado intermitente o caído, que redirija a la vista de monitoreo al interactuar con ella. Se debe bloquear la opción de realizar transacciones asociadas a servicios caídos.

Notificar al usuario cuando se presenten errores al realizar transacciones.	La aplicación debe notificar al usuario cuando falla una transacción, e informar en caso de que haya cambiado el estado de disponibilidad del servicio. Debido a que la aplicación funciona como puente entre distintos servicios, cualquier reversión de dineros involucrados es realizada por los servicios de los bancos.

Tabla 3: Criterios de fiabilidad. Estos cirterios aseguran que el usuario tenga una experiencia *correcta* y consistente.

2.2.4. Interfaz

La interfaz debe ser lo más sencilla posible, tanto en cantidad de pasos requeridos para acceder a las funcionalidades de la aplicación como en el estilo del texto y los colores utilizados, los cuales en conjunto promueven una imagen seria y profesional de la aplicación.

Objetivos cuantificables

1. El diseño de los elementos y texto de la aplicación deben seguir una paleta limitada de colores.
2. El texto de la aplicación debe ser fácil de leer.
3. Cuadros de texto deben ser concisos.

Criterios de evaluación

Criterio	Medición
El diseño de los elementos y texto de la aplicación deben seguir una paleta limitada de colores.	Los colores de la aplicación (headers, footers, íconos, etc) deben seguir una paleta de entre 3 a 5 colores, y contener el color del logo de *Dracon* (valor hexadecimal: #622569). Para colores asociados a texto, debe existir una paleta de 2 a 3 colores que contenga al color blanco (valor hexadecimal: #FFFFFF).
El texto de la aplicación debe ser fácil de leer.	Todo texto debe ser legible sobre el color de fondo, por lo que su color debe destacar tanto para personas sin problemas a la vista, como para quienes sufren algún tipo de daltonismo o impedimento visual. Para esto, el *ratio de contraste* entre el color de fondo y el color de texto debe ser como mínimo igual a $[7:1]$[2].
Cuadros de texto deben ser concisos.	Las alertas mostradas por la aplicación no deben superar los 150 carácteres para comunicar la información deseada.

Tabla 4: Criterios de interfaz. Estos criterios le dan a la aplicación una imagen profesional y llamativa.

2.2.5. Usabilidad

La aplicación debe ser fácil de comprender y manejar, entregando así una experiencia de uso eficaz, eficiente y satisfactoria. Debido a que este punto se enfoca en la experiencia de usuario, se requiere realizar pruebas de usabilidad sobre grupos participantes que sean representativos del público objetivo.

[2]Ratio recomendado por las *Web Content Accessibility Guidelines*.

Objetivos cuantificables

1. El usuario debe recorrer la menor cantidad de vistas posibles para llegar a una funcionalidad.
2. Las funcionalidades deben ser fáciles de ubicar.
3. Debe existir la posibilidad de importar y exportar contactos.
4. Primera interacción con la aplicación debe ser intuitiva para el usuario.
5. Campos de ingreso de datos en formularios deben restringir carácteres de acuerdo al tipo asociado.
6. Campos de ingreso de datos en formularios deben arrojar errores no invasivos al momento de ingresar un carácter no permitido o no cumplir el formato.

Criterios de evaluación

Criterio	Medición
El usuario debe realizar la menor cantidad de clics para llegar a una funcionalidad.	Tomando la pantalla de inicio de la aplicación como referencia, la cantidad de vistas requeridas para acceder a cualquier funcionalidad debe ser ≤ 3.
Las funcionalidades deben ser fáciles de ubicar.	La cantidad de sub-menús requeridos para llegar a cualquier funcionalidad debe ser ≤ 2.
Debe existir la posibilidad de importar y exportar contactos.	Aplicación debe proveer una herramienta para exportar información de una cuenta bancaria propia, para que otro usuario pueda importar dicha información de contacto con tan solo un clic desde la lista de contactos.
Primera interacción con la aplicación debe ser intuitiva para el usuario.	El flujo de la primera interacción del usuario con la funcionalidad principal de la aplicación (Vincular cuenta bancaria -> Agregar Contacto -> Realizar transferencia de dinero) debe ser intuitivo para al menos un 99 % de los usuarios de prueba, demorando $\leq 5[minutos]$ en ejecutar la secuencia completa, y $\leq 45[segundos]$ en ejecutar la secuencia de pago.

Campos de ingreso de datos en formularios deben restringir carácteres de acuerdo al tipo asociado.	En campos de formularios, se debe restringir el ingreso de carácteres de acuerdo al formato asociado al campo (Ej: si es un campo de correo, se debe cumplir el formato de correo)
Campos de ingreso de datos en formularios deben arrojar errores no invasivos al momento de ingresar un carácter no permitido o no cumplir el formato.	Si usuario ingresa un carácter no permitido, o si ingresa una secuencia de carácteres que no cumple un formato (como por ejemplo RUT o correo), campo se debe marcar de color rojo, y arrojar una burbuja de texto con ancho igual al campo, en donde se indique en menos de 50 carácteres el origen del error.

Tabla 5: Criterios de usabilidad. Mientras más accesible sea la aplicación, mayor será la retención de usuario nuevos.

2.2.6. Escalabilidad

La aplicación en concepto es escalable debido a que los datos se manejan localmente. Aún así, existen registros que deben procesarse en el servidor de la aplicación, por lo que es necesario asegurar que este sea capaz de mantenerse eficiente en los momentos con mayor tránsito de información, que normalmente serían períodos como los fines de semana en la tarde-noche, momentos en los que se realizan la mayor cantidad de salidas a locales de comida y otros tipos de entretenimiento.

Objetivos cuantificables

1. La aplicación debe soportar la alta demanda de transferencias simultáneas en períodos con mayor carga.
2. La aplicación debe soportar la carga producida por acciones relacionadas a creación de grupos y eventoss.

Criterios de evaluación

Criterio	Medición
La aplicación debe soportar la alta demanda de transferencias simultáneas en períodos con mayor carga.	Los servidores dedicados a manejar las transferencias deben soportar ≥ 25000 transacciones simultáneas en períodos de alta demanda. En períodos valle, se deben soportar ≥ 10000 transferencias simultáneas.
La aplicación debe soportar la carga producida por acciones relacionadas a creación de grupos y eventos.	Los servidores dedicados a manejar las notificaciones producto del manejo de grupos y eventos deben soportar $\geq$ 15000 acciones simultáneas.

Tabla 6: Criterios de escalabilidad. La aplicación debe soportar períodos de alta demanda y un potencial crecimiento sin perder eficiencia en su funcionamiento.

2.2.7. Portabilidad

Como la aplicación está pensada para el uso cotidiano, es necesario que esta sea compatible con la mayoría de los usuarios de smartphone, lo que correspondería a usuarios de Android[3], y que la aplicación en sí no sea muy demandante en cuanto a memoria utilizada.

Objetivos cuantificables

1. La aplicación debe correr en dispositivos con las versiones más populares de Android.
2. La aplicación base no debe usar mucha memoria.

[3]IAB Trends (9 de Agosto, 2016). "Chile lidera el uso de smartphones en Latinomaérica con 7,9 millones de usuarios". Extraído el 12 de Febrero del 2019, de https://iabtrends.cl/2016/08/09/chile-lidera-el-uso-de-smartphones-en-latinomaerica-con-7-9-millones-de-usuarios/

Criterios de evaluación

Criterio	Medición
La aplicación debe correr en dispositivos con las versiones más populares de Android.	La aplicación debe ser compatible con sistemas operativos a partir de la versión 4.4 de Android, correspondiente a *KitKat*.
La aplicación base no debe usar mucha memoria.	El peso del APK de la aplicación no debe superar los 25 MB.

Tabla 7: Criterios de portabilidad. Estos criterios definen el grado de accesibilidad para los distintos usuarios de smartphone.

MODELO DE DESARROLO

3.1. Elección del Modelo

Ya que *Dracon* es una aplicación compuesta por funcionalidades independientes entre ellas, el modelo de desarrollo adecuado es el iterativo incremental[Jummp, 2011] utilizando el framework Scrum[Schwaber y Sutherland, 2011]. Este framework permite dividir el proyecto en distintas entregas o *sprints*, las cuales contendrán una cantidad de actividades y/o funcionalidades a implementar. Luego, dentro de cada entrega existen reportes periódicos sobre el avance que cada desarrollador ha logrado a lo largo de dicho período.

Se prefiere este modelo debido a las siguientes ventajas que presenta:

- **Feedback en etapas tempranas de producción:** Debido a que los sprints aseguran una versión funcional del producto desde la primera iteración, esta se puede presentar tempranamente al cliente, obteniendo un feedback inicial que corrobore los elementos que estén bien implementados, y que identifique los que hay que redefinir o implementar de manera distinta.
- **Testing modular y eficiente:** Ya que el producto se desarrolla en base a la implementación de funcionalidades independientes entre sí, se simplifican las pruebas que se realicen sobre las funciones implementadas en cada Sprint, lo que lleva a que sea más simple la identificación de fallas o casos no cubiertos anteriormente. Además, gracias a las pruebas de integración, se corrobora que los componentes pre-existentes del producto y los elementos nuevos funcionen correctamente en conjunto.

- **Trabajo paralelo:** Debido a que se implementan funciones normalmente independientes entre sí, la distribución de trabajo se hace más clara y sencilla, además de que se evita la dependencia entre los distintos equipos de trabajo.
- **Control sobre el avance del desarrollo:** Los reportes diarios del avance de cada miembro del equipo permite a los integrantes y jefe de proyecto saber el estado de cada uno, además de sus complicaciones. Esto le da al equipo la oportunidad de abordar problemas en el desarrollo de la aplicación a medida que estos aparecen, reduciendo así períodos muertos de producción.

3.2. Equipo Scrum

En un equipo que esté empleando Scrum, se designan roles con el fin de optimizar la flexibilidad, creatividad y productividad de este, manteniendo un sistema de trabajo autosuficiente. Estos roles son Product Owner, equipo de Desarrollo y Scrum Master:

3.2.1. Product Owner

Su rol es optimizar el desempeño del equipo mediante una buena administración de los requerimientos a implementar. Es responsable de que el equipo de desarrollo entienda con claridad qué es lo que va a desarrollar, tanto a nivel de establecer el próximo objetivo como también definir los detalles de este.

3.2.2. Scrum Master

El Scrum Master se encarga de hacer respetar los eventos y herramientas de Scrum, promoviendo una buena implementación del framework en el contexto empírico. Actúa además como un soporte para las interacciones entre la organización, el equipo de desarrollo y el Product Owner.

Tiene la responsabilidad de apoyar al Product Owner en la administración de los requerimientos para maximizar el valor del producto y mantener la agilidad del framework.

Su rol con el equipo de desarrollo consiste en tutelar al equipo para obtener un buen desempeño, ya sea facilitando los eventos Scrum cuando se requieran, enseñando al equipo buenas metodologías de autosuficiencia y funcionalidad cruzada, y asegurándose de que no existan elementos impidiendo el progreso del equipo.

3.2.3. Equipo de Desarrollo

El equipo de desarrollo consiste en los integrantes que implementan el producto, los cuales tienen control sobre cómo dividen su trabajo, y de cómo implementar las soluciones requeridas. Posee una estructura horizontal al no considerar sub-equipos, y responder de manera grupal ante el Scrum Master y el Product Owner.

El equipo de desarrollo debe ser tener una cantidad de integrantes manejable y con distintas habilidades que permitan una buena complementación dentro del equipo.

3.3. Definición de Actividades

El desarrollo del producto se divide en las siguientes etapas, compuestas por la interacción inicial con el cliente, los eventos de Scrum, y la posterior mantención del producto:

1. Escritura de Historias de Usuario
2. Planificación del Sprint (Sprint Planning)
3. Reunión Diaria de Scrum (Daily Scrum Meeting)
4. Revisión del Sprint (Sprint Review)
5. Retrospectiva del Sprint (Sprint Retrospective)
6. Mantención

Los elementos 2-5 se repetirán por cada entrega hasta que el producto se encuentre finalizado.

Además de los eventos Scrum, el framework cuenta con los siguientes artefactos:

- **Product Backlog:** es una lista de todo lo que se conoce que debe realizarse para el producto final, y deriva de la toma de requerimientos anterior. Está sujeto a cambios a medida que se avanza en el proyecto, reflejando qué requiere el producto para ser competitivo, útil y exitoso. Estos cambios se traducen en cambiar el orden de elementos a implementar, y agregar detalles o estimaciones a los requerimientos. Está a cargo del Product Owner.
- **Sprint Backlog:** consiste en una lista de objetivos a completar dentro de un sprint, sirviendo a grandes rasgos para definir qué funcionalidad se implementará para el siguiente *Increment*. Detalla qué actividades deben realizarse, cuales ya están hechas, y las que aún están pendientes, transparentando así el trabajo que debe realizar el equipo de desarrollo. Está a cargo del equipo de desarrollo.

- **Increment:** es la suma de todos los objetivos logrados en el Sprint Backlog, más el valor agregado de los Increments anteriores. Un increment debe presentar una versión funcional del producto. El equipo Scrum debe estar de acuerdo en que el trabajo realizado fue suficiente, consolidando dicho increment antes de avanazar hacia nuevos objetivos.

3.3.1. Escritura de Historias de Usuario

En la escritura de historias de usuario, se registran las necesidades del cliente respecto al funcionamiento del producto. Ya que las historias de usuario son la guía bajo la cual trabaja el equipo de desarrollo ágil, es necesario que el cliente defina claramente lo que desea respecto a aspectos funcionales, operacionales, técnicos y de interfaz.

El Product Owner debe llegar a un acuerdo con el cliente respecto a la factibilidad e importancia de los requerimientos, procurando también que sean consistentes con el resto. Luego, estas historias le sirven como puente para la comunicación con el equipo de desarrollo; con quienes, en conjunto, debe refinar las historias y responder: ¿qué se requiere?, ¿quién se beneficia de la historia?, y ¿cómo se beneficia?.

Finalmente, el Scrum Master se encarga de monitorear el proceso, facilitando la comunicación entre ambas entidades del equipo Scrum, y asegurando que las historias sean concisas y claras.

3.3.2. Planificación del Sprint (Sprint Planning)

En este evento se define lo que se entregará al final del sprint, y cómo se llevará a cabo el trabajo. Para esto se toman elementos del Product Backlog para llevarlos al Sprint Backlog, y se define una meta del sprint que le dice al equipo por qué se desarrolla este Increment, y cuanta flexibilidad existe en cuanto a lo que se presentará al final del Sprint.

3.3.3. Reunión diaria de Scrum (Daily Scrum Meeting)

Reunión diaria en la cual los integrantes del equipo de desarrollo reportan al resto qué realizaron en el día, y planifican que se hará el día siguiente. Su objetivo es evaluar el progreso del Sprint y mejorar la comunicación dentro del equipo.

3.3.4. Revisión del Sprint (Sprint Review)

Se realiza al final del sprint, y tiene como función analizar el Increment y adaptar el Product Backlog. El equipo Scrum se reúne con *stakeholders* para recibir feedback del sprint realizado y discutir maneras para optimizar el valor del producto. En esta reunión se discuten tanto aspectos relacionados al desarrollo en sí, como al futuro del producto, información que se utiliza para la planificación del siguiente Sprint.

3.3.5. Retrospectiva del Sprint (Sprint Retrospective)

Este evento ocurre entre la evaluación del Sprint y la planificación del Sprint siguiente. Se discuten los puntos positivos y negativos del Sprint para identificar qué aspectos se pueden mejorar, y los integrantes del equipo se comprometen para llevar esa mejora a cambio de acuerdo a lo discutido.

3.3.6. Mantenimiento

Una vez finalizado el producto, y que este se encuentre en operación, existirán períodos en que el equipo de desarrollo deberá arreglar eventuales errores que surjan en producción, o de futuras actualizaciones de la aplicación, adaptando la aplicación a un contexto que se encuentra en constante cambio.[Pressman, 2010]

3.4. Definición de Productos de Trabajo

3.4.1. Plan de Proyecto

El plan de proyecto consiste en el preámbulo del producto a realizar: en él se define qué se hará en el proyecto, y el modelo a utilizar para efectuarlo.

El **Plan de Proyecto** adaptado al modelo escogido considera una planificación centrada en los eventos que conforman un Sprint. Con esto se busca optimizar el trabajo del equipo, procurando que las estimaciones de costo y tiempo iniciales se apeguen lo más posible al contexto empírico.

Estas estimaciones además deben estar acompañadas por la distribución de trabajo entre los integrantes, en lo posible aprovechando las habilidades individuales que ellos posean. Por último, debe quedar definida una manera de medir el trabajo realizado.

El **Plan de Proyecto** debe seguir la siguiente estructura:

1. **INTRODUCCIÓN**
    1.1. Objetivo
    1.2. Alcance
2. **ESTIMACIÓN DEL ESFUERZO Y DURACIÓN**
    2.1. Estimación del esfuerzo realizado
    2.2. Estimación de la duración del desarrollo
3. **EQUIPO DE DESARROLLO DEL PROYECTO**

    Se identifican las habilidades y conocimientos de los miembros que componen el equipo, y el rol que cumplen en el desarrollo a grandes rasgos.
4. **PRODUCT BACKLOG**

    Debido a que el proyecto se divide en iteraciones, se elabora un Product Backlog con las estimaciones iniciales en torno a todas las actividades identificadas en un principio (dando flexibilidad para agregar nuevos detalles a medida que se avanza en las entregas), definiendo:

    4.1. Actividades a realizar
    4.2. Historias de Usuario a desarrollar
    4.3. Herramientas requeridas
5. **SPRINT BACKLOG**

    Cada subdivisión de actividades del product Backlog consiste en un Sprint Backlog, para el cual se definen:

    5.1. Actividades a realizar para la entrega
    5.2. Historias de Usuario a desarrollar para la entrega
    5.3. Estimación del tiempo requerido para completar actividades e historias de usuario
6. **ESTRATEGIA DE SEGUIMIENTO**
7. **INTERFAZ EXTERNA DEL PROYECTO**
8. **IDENTIFICACIÓN Y ANÁLISIS DE RIESGOS**

3.4.2. Plan de Riesgos

Para evitar el fracaso del proyecto debido a eventualidades que lo afecten negativamente, se requiere un **Plan de Riesgos** en el cual se identifiquen las mayores amenazas al proyecto, definiendo para cada estrategias de prevención y mitigación, y un plan de contingencia.

Luego, el documento del **Plan de Riesgos** debe contener:

1. **INTRODUCCIÓN**
   1.1. Objetivo
   1.2. Alcance
2. **IDENTIFICACIÓN DEL RIESGO**

   Evaluación del impacto de los riesgos, agrupados en categorías que permitan una distinción clara de estos.
3. **PROYECCIÓN DEL RIESGO**

   Estimación de la probabilidad de que ocurra cada riesgo, y del alcance e impacto de sus consecuencias en el proyecto.
4. **PRIORIZACIÓN DE RIESGOS**

   De acuerdo a un criterio definido previamente, en el cual se consideren tanto el impacto como la probabilidad del riesgo, se deben priorizar los que presenten mayor peligro para el proyecto.
5. **GESTIÓN DEL RIESGO**

   Estrategias para prevenir y mitigar los riesgos, además de los planes de contingencia para éstos.

3.4.3. Plan de Pruebas

Uno de los principales factores de éxito de un proyecto es la calidad de sus funcionalidades. El **Plan de Pruebas** deberá considerar el funcionamiento correcto de cada funcionalidad, que la estén considerados todos los tipos de *input*, tanto esperado como no esperado, y procurar que se cumplan los estándares de aceptación acordados por el equipo.

Por lo tanto, el documento debe establecer los elementos que se probarán, junto a las pruebas a realizar, las herramientas que se utilizarán en ello, y los criterios de aceptación de los elementos evaluados.

El **Plan de Pruebas** tendrá entonces el siguiente contenido:

1. **INTRODUCCIÓN**
    1.1. Objetivo
    1.2. Alcance
2. **DEFINICIÓN DE ELEMENTOS A PROBAR**
3. **DEFINICIÓN DE ESTRATEGIA A UTILIZAR**
   Se definen los criterios de aceptación, las técnicas de pruebas y herramientas a utilizar.
4. **ASIGNACIÓN DE MIEMBROS QUE REALIZARÁN ACTIVIDADES DE PRUEBA**

3.4.4. Plan de Aseguramiento de Calidad (Q&A)

1. **INTRODUCCIÓN**
2. **GESTIÓN SQA**
   Para cada Sprint se define:
    2.1. Miembros del equipo a cargo de actividades de SQA.
    2.2. Atributos prioritarios del Increment acordado.
    2.3. Actividades a realizar para medir atributos anteriores, y en qué momento se deben ejecutar.
    2.4. Resultados esperados.
    2.5. Resultados obtenidos
    2.6. Reuniones a realizar para discutir los resultados obtenidos.

3.4.5. Plan de Gestión de Configuración (SCM)

El **Plan de Gestión de Configuración** tiene como fin garantizar la integridad del proyecto a lo largo del ciclo de vida del software mediante el control de los cambios realizados en cualquier documento o artefacto asociado al producto.

1. **INTRODUCCIÓN**
    1.1. Objetivo
    1.2. Alcance
    1.3. Definiciones
    1.4. Resumen
2. **IDENTIFICACIÓN DE ÍTEMS DE CONFIGURACIÓN**
   Definición de la *baseline*[4].
3. **IDENTIFICACIÓN DE LA CONFIGURACIÓN**
    3.1. Control de Cambios
    3.2. Control de Documentos
4. **ESTADO DE LA CONFIGURACIÓN**
5. **HERRAMIENTAS SCM**
6. **CONJUNTO DE REGISTROS**

3.4.6. Manual de Usuario

El **Manual de Usuario** es un documento que funciona como guía para el usuario final del sistema, en donde se detalla a grandes rasgos las distintas funcionalidades de la aplicación, a modo de *tour*.

El manual debe contener también una sección de preguntas frecuentes que considere tanto las dudas básicas sobre el uso de la aplicación como preguntas más específicas sobre las funcionalidades más importantes. Por último, debe existir una sección de contacto que sirva para obtener feedback de los usuarios que utilicen constantemente la aplicación, permitiendo así una mejor interacción con estos, lo que servirá para guiar futuros cambios a implementar.

[4]Documento que detalla las características de un producto en algún tiempo específico, sobre las cuales se implementan cambios.[BSSC, 1995]

1. **PREFACIO**
    1.1. Introducción.
    1.2. Cómo usar el manual
2. **ÍNDICE**
3. **DESCRIPCIÓN GENERAL DEL SISTEMA**
4. **FUNCIONES DE LA APLICACIÓN**

    Se explican las distintas vistas de la aplicación, y para cada una se detalla qué hace cada componente interactivo dentro de esta.
5. **PREGUNTAS FREQUENTES**
6. **CONTACTO**

3.5. Hitos de Desarrollo

Trabajar con hitos le da al jefe de proyecto herramientas para hacer seguimiento y evaluación continua del proyecto, dándole mayor control sobre el flujo de trabajo. Para esto, se deben establecer criterios objetivos para medir el éxito de dichos hitos.

3.5.1. Por evento de Scrum

Evento	Descripción
Sprint Planning	Debe considerar flexibilidad en el tiempo de entrega, evitando retrasos mayores a 24 horas del plazo original. Además, debe cumplirse el 95 % de lo acordado para el Increment del Sprint.
Daily Scrum Meeting	La hora de la reunión debe ser constante, y en un momento en que todo el equipo se encuentre disponible. No debe durar más de 15 minutos. Los integrantes deben reportar las actividades realizadas en el día, y se registran observaciones respecto a cómo proceder el día siguiente.
Sprint Review	Las funciones desarrolladas en el Sprint deben cumplir con un 95 % de lo acordado para el Increment. Además, el producto debe encontrarse en un estado funcional, y tener la aprobación del *Product Owner* y del cliente.
Retrospectiva del Sprint	Se deben documentar las decisiones tomadas respecto a los cambios que se implementarán dentro del equipo para el próximo Sprint, comentando los puntos altos y bajos del Sprint actual. La reunión no debe durar más de 3 horas.
Sprint	Es el evento que contiene todo el trabajo realizado en los 4 eventos anteriores durante una entrega.

Tabla 8: Hitos de desarrollo por eventos de Scrum.

3.5.2. Por artefactos de trabajo

Evento	Descripción
Product Backlog	Debe registrar todas las funcionalidades base del producto, así como también todos los requerimientos asociados a cada una. Ambos elementos deben estar definidos de manera clara, precisa, fácil de entender, y que satisfagan las necesidades del cliente. Para cada requerimiento y funcionalidad deberá actualizarse su estado actual, estimación de dificultad y prioridad según se estime conveniente al final de cada Sprint.
Sprint Backlog	Para cada elemento del Product Backlog a implementar en el Sprint, deben quedar definidos todos los detalles conocidos, y se deben establecer las actividades que deben realizarse para llevarlo a cabo.

Tabla 9: Hitos de desarrollo por artefactos de trabajo.

3.5.3. Por producto de trabajo

Evento	Descripción
Plan de Proyecto	Debe identificar los roles de cada integrante, las herramientas a utilizar, y cumplir con los requisitos previamente mencionados para el Product Backlog.
Plan de Riesgo	Debe existir un plan de mitigación y un plan de contingencia para cada riesgo identificado.

Plan de Pruebas	Debe contener como mínimo pruebas de: ▪ **Aceptación:** el flujo de acciones del caso de uso asociado a la función se ajusta a lo requerido por el cliente. ▪ **Unitarias:** • **Funcionalidad:** la aplicación no presenta defectos ni errores la ejecución de sus funciones. • **Rendimiento:** los tiempos de espera y/o carga se ajustan a los criterios de aceptación. • **Compatibilidad:** la aplicación corre en dispositivos incluídos dentro del rango de versiones acordado. ▪ **Integración:** Componentes pre-existentes de la aplicación funcionan correctamente después de haber integrado la nueva funcionalidad. ▪ **Interfaz:** la interfaz de las nuevas funcionalidades implementadas es entendible para nuevos usuarios.
Plan de Aseguramiento de Calidad (SQA)	Debe considerar todos los puntos acordados para asegurar la calidad del software.
Plan de Gestión de Configuración (SCM)	Debe documentar todas las actividades relacionadas con el desarrollo de sistema. (incluye código fuente, ejecutables, modelos, requisitos, pruebas, etc)[pae, 2001]
Manual de Usuario	Debe tener un máximo de 20 páginas con contenido instructivo. Cualquier consulta al manual debe poder ser respondidad en menos de 3 minutos.

Tabla 10: Hitos de desarrollo por producto de trabajo

GESTIÓN DE CALIDAD

La gestión de calidad de un proyecto corresponde al conjunto de responsabilidades, actividades y procesos de la organización que determinan objetivos y criterios de calidad, con el fin de que el producto final satisfaga de manera adecuada las necesidades del cliente. Para el plan de proyecto presentado en este documento, dichas cualidades son cubiertas por los analistas QA, quienes se enfocan en asegurar un estándar de calidad para el producto y los métodos de trabajo involucrados en su construcción.

4.1. Equipo de Trabajo

4.1.1. Composición del equipo

En un equipo que sigue el modelo Scrum, los roles de Product Owner y Scrum Master siguen una jerarquía vertical, mientras que el equipo de desarrollo es horizontal entre todos sus integrantes, independiente de sus habilidades individuales y la naturaleza de su trabajo. Luego, el organigrama resultante puede ser visto en la **Figura 1**:

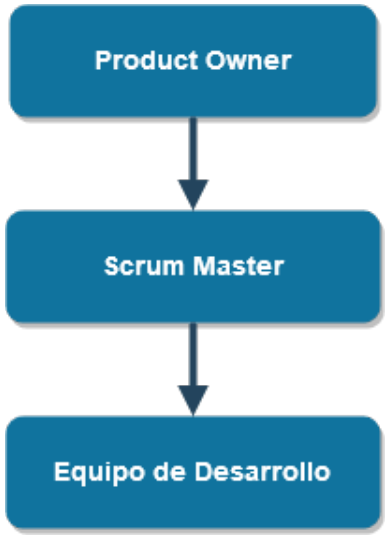


Figura 1: Organigrama de un equipo Scrum.

Las responsabilidades del **Product Owner** y **Scrum Master** fueron definidas en las secciones 3.2.1 y 3.2.2 respectivamente. El **equipo de desarrollo**, por otro lado, se compone de desarrolladores con conocimientos en distintas áreas. Este aspecto le entrega autonomía al ser sus integrantes quienes deciden como dividir sus tareas al comienzo de cada Sprint,

entre las que se encuentran también las actividades de SQA y SCM. Los detalles de dichas actividades están definidas más adelante, en la sección 4.3. Otros tipos de desarrolladores y sus respondabilidades son:

- **Desarrollador Back-End**
  - Diseñar e implementar la lógica detrás de las funciones de la aplicación.
  - Estudiar constantemente los lenguajes y frameworks utilizados en el desarrollo de la aplicación.
  - Configurar servidores y servicios externos vinculados a la aplicación.
- **Desarrollador Front-End**
  - Estudiar las herramientas nativas del sistema sobre el que se va a trabajar.
  - Estudiar las teconologías de diseño web a utilizar.
  - Implementar la interfaz siguiendo la visión de los diseñadores.
  - Mantener un estilo constante entre las distintas vistas de la aplicación.
- **Diseñador UX-UI**
  - Investigar al público objetivo de la aplicación.
  - Generar un esquema objetivo para evaluar experiencia del cliente.
  - Generar wireframes para las vistas de la aplicación
  - Generar diseño de la aplicación a partir de wireframes previos.
  - Corroborar interfaz de las vistas ya implementas.

4.1.2. QA en proyectos ágiles

En un entorno ágil, el proyecto se compone de actividades asíncronas que se desarrollan en el orden en que se requieran; no tienen un orden definido, salvo excepciones (como el desarrollo del prototipo, el cual igualmente cabe dentro de la misma definición). Como el equipo debe estar constantemente trabajando en conjunto, es necesario que exista unidad y buena comunicación, en base a las cuales se genere retroalimentación mutua entre las distintas subdivisiones existentes.[Scrum.org, nd] Esto último aplica especialmente al equipo de QA, quienes deberán adaptarse para cubrir distintos roles que ayuden a la evaluación de calidad del software.

QA puede ayudar al *Product Owner* tanto a armar casos de aceptación como a captar requerimientos de clientes, por lo que sirven además como un puente entre este y el equipo de desarrollo. Con esto se mantienen al tanto de los detalles de los requerimientos del proyecto, y son capaces de informar de mejor manera al equipo de desarrollo. Además,

esta función de nexo y el trabajo cercano con el equipo de desarrollo les permiten entregar *feedback* rápido, sin necesidad de esperar a que las funcionalidades estén ya desarrolladas de antemano, previniendo así ciclos innecesarios de desarrollo-prueba, y de arreglo de defectos en el software.

Pueden también estimar complejidad de historias de usuario junto al equipo de desarrollo, apoyando en la tarea de identificar flujos de uso realistas, en vez de enfocarse únicamente en ”caminos felices”. Esto influye en que se pueda informar de manera más precisa el tiempo que toma implementar nuevas funcionalidades, resultando en una mejor distribución de los recursos tanto humanos como materiales distponibles.[Hasija, 2012]

Aparte de lo mencionado anteriormente, las pruebas del sistema a realizar por QA aumentan en dificultad a medida que se completen Sprints, pues además de las nuevas funcionalidades, se debe probar que las anteriormente implementadas sigan entregando resultados correctos. Para esto, existe la posibilidad de automatizar pruebas de regresión, es decir, de funcionalidades previas a la inclusión de una nueva, evitando la realización manual de estas, ahorrando horas hombre, y evitando potenciales errores al realizar pruebas repetitivas y monótonas.

4.2. Infraestructura

Ya que la aplicación considera compatibilidad en un rango amplio de dispositivos móviles, se hace necesario contar con una cantidad de *smartphones* de gama media-baja que asegure compatibilidad con los equipos más antiguos. Como existen emuladores de dispositivos móviles, solo se requerirán 2 aparatos para probar de manera física las aplicaciones con los requerimientos mínimos detallados en la **Tabla 11**.

Elemento	Detalle
RAM	2 GB.
Almacenamiento	2 GB libres.
Conexión a Internet	3G o H.
Cámara	Dispositivo debe poseer una cámara para la lectura de códigos QR.

Tabla 11: Requerimientos mínimos relevantes para correr la aplicación cómodamente. Características no listadas en la tabla (como procesador) no requieren una especificación detallada.

Utilizando la herramienta *Android Studio Emulator* como referencia, se listan los requerimientos mínimos para una buena ejecución del entorno de pruebas en la **Tabla 12**.

Elemento	Detalle
Sistema Operativo	▪ **Windows:** Microsoft® Windows® 7/8/10 (32- o 64-bit) ▪ **Mac:** Mac® OS X® 10.10 (Yosemite) o mayor, hasta 10.13 (macOS High Sierra) ▪ **Linux:** GNOME o KDE. Distribución de 64-bits capaz de ejecutar aplicaciones de 32-bits. Librería GNU C (glibc) 2.19 o mayor.
RAM	8 GB.
Almacenamiento	4 GB libres.
Video RAM	2 GB.

Tabla 12: Requerimientos mínimos recomendables para ejecutar la herramienta de manera eficiente.

4.3. Actividades de SQA

La definición del oficio de un analista de SQA es, a grandes rasgos, la ”prevención de fallas mediante inspección y pruebas del proceso”. Esto se extiende a definir objetivos calidad del software, y la evaluación del producto respecto a dichas metas. En el contexto ágil del proyecto, es posible realizar paralelamente tareas de SQA[Visconti, 2000][HelpingTesters, 2016] mientras el equipo continúa desarrollando otras funcinalidades. Dichas tareas deben ser realizadas cada vez que se produzca algún cambio en el software, y consisten en:

4.3.1. Formulación de plan de gestión de calidad

El equipo de SQA debe crear un plan de gestión de calidad en el cual se identifiquen atributos de calidad del producto, además de registrar el avance del proyecto basado en los resultados obtenidos en evaluaciones realizadas en las distintas etapas de desarollo.

4.3.2. Realizar reseñas técnicas

Se realiza una reunión con expertos técnicos, quienes apoyan la definición de métricas que definen la calidad de los requisitos y el diseño del prototipo. Esto permite encontrar errores tempranamente, evitando que se propaguen a etapas futuras.

4.3.3. Aplicar estrategia de pruebas

Se diseña y aplica la estrategia de pruebas, la cual debe abarcar pruebas unitarias, de integración y a nivel de sistema. Estas pruebas pueden o no ser realizadas en conjunto al equipo de desarrollo (como en el caso de pruebas de funcionalidad[5] y de sistema[6]).

4.3.4. Imponer adherencia al proceso

Se compara el estado actual del producto con los estándares de calidad acordados en la etapa de gestión del proyecto, reflejando así los requerimientos acordados. Se verifica además que se cumplan los procedimientos de prueba establecidos en la documentación.

4.3.5. Controlar el cambio

Formaliza las peticiones de cambio, evaluando la calidad y naturaleza de estas y su impacto en el proyecto (por ej. en la estimación de costos y el alcance del producto, entre otros). El mecanismo asociado a esta actividad se implementa en las etapas de diseño y mantención.

4.3.6. Realizar auditorías

Un auditor encargado de esta actividad genera un registro de cómo se están desarrollando los procedimientos acordados en la documentación, y de los resultados de los casos de prueba ejecutados sobre las funcionalidades, probando tanto una ejecución correcta como que estas se encuentren alineadas con las reglas de negocio. En caso de hayar errores o defectos, el equipo debe decidir como reaccionar ante estos.

[5]Pruebas de caja negra en las que se corrobora que las funciones implementadas cumplan con los requerimientos funcionales asociados

[6]Pruebas de caja negra al sistema completo que compone el producto, corroborando que cumpla con los requerimientos asociados

4.3.7. Manejar registros y reportes

La documentación generada en el proyecto a partir de eventos Scrum, historias de usuario, artefactos asociados al framework, etc., debe ser reportada y archivada para promover un buen flujo de información relevante al proceso de SQA, manteniendo la línea de concisión y claridad que caracteriza al desarrollo ágil.

ANÁLISIS DE RIESGOS

En la **Tabla 13** se encuentran listados los riesgos identificados para el proyecto, con su categoría respectiva a la cual pertenecen. Las categorías se dividen en:

- **Externo:** asociado a fuerzas ajenas como el mercado y regulaciones de gobierno, entre otras.
- **Operacional:** asociado a la correcta implementación y funcionamiento de los procesos.
- **Organizacional:** asociado a la planificación del equipo, tanto para estimación de tiempos, asignación de trabajos y presupuestos, etc.
- **Técnico:** asociado a la usabilidad y conocimiento de las tecnologías involucradas en el proyecto.

La importancia de cada uno será analizada en la siguiente sección.

5.1. Identificación de Riesgos

ID	Riesgo	Categoría
1	Aparición en el mercado de productos similares que hacen competencia directa a la aplicación.	Externo
2	Servidores no son capaces de aguantar carga de usuarios.	Técnico
3	Realizar una mala estimación de HH para una entrega.	Organizacional
4	Incumplimiento de eventos y artefactos asociadas al proceso de desarrollo ágil.	Organizacional
5	Desconocimiento de las tecnologías con las que se trabaja.	Técnico
6	Pérdida de interés del público en el producto.	Externo
7	Mala definición de historias de usuario.	Operacional
8	Existencia de vulnerabilidades de seguridad en datos de usuarios.	Técnico
9	Falta de comunicación entre miembros del equipo.	Operacional
10	Cambio en código de APIs de transacciones bancarias.	Técnico

Tabla 13: Principales riesgos que pueden afectar el correcto funcionamiento del proyecto.

5.2. Estimación de Riesgos

Para clasificar los riesgos, se utilizan los atributos de **impacto** y **probabilidad**, los cuales tienen una puntuación asociada a cada uno de sus niveles para una evaluación objetiva en su grado de importancia.

- **Impacto:** el grado de amenaza que presenta el riesgo para el proyecto.
  - **[1.0] Bajo:** No representa mayor peligro para el desarrollo del proyecto.
  - **[2.0] Medio:** No compromete el proyecto, pero puede generar complicaciones en el desarrollo de éste, o dentro del equipo.
  - **[3.0] Alto:** Si no se mitiga a tiempo puede resultar en complicaciones graves dentro del proyecto que lleven al fracaso de este.

- **Probabilidad:** qué tan posible es encontrar el riesgo descrito.
  - **[1.0] Muy Baja:** $< 10\,\%$
  - **[1.5] Baja:** $10\,\% - 24\,\%$
  - **[2.0] Media:** $25\,\% - 49\,\%$
  - **[2.5] Media alta:** $50\,\% - 74\,\%$
  - **[3.0] Alta:** $\geq 75\,\%$

Luego, sumando los puntajes se obtendrán valores en el rango de 2 a 6, que indicarán el grado de importancia del riesgo; mientras mayor sea el valor, más importante es el riesgo. En la **Tabla 14** se observan los riesgos previamente identificados, en orden descendiente de acuerdo a su nivel de importancia.

ID	Riesgo	Impacto	Probabilidad	Nivel de importancia
1	Aparición en el mercado de productos similares que hacen competencia directa a la aplicación.	3.0	2.0	5.0
6	Pérdida de interés del público en el producto.	3.0	2.0	5.0
3	Realizar una mala estimación de HH para una entrega.	2.0	2.5	4.5
4	Incumplimiento de eventos y artefactos asociadas al proceso de desarrollo ágil.	2.0	2.5	4.5
5	Desconocimiento de las tecnologías/herramientas con las que se trabaja.	2.0	2.0	4.0
8	Existencia de vulnerabilidad(es) de seguridad en datos de usuarios.	2.0	2.0	4.0
9	Falta de comunicación entre mimebros del equipo.	2.0	2.0	4.0
2	Servidores no son capaces de aguantar la carga de usuarios.	2.0	1.5	3.5
7	Mala definición de historias de usuario.	2.0	1.5	3.5
10	Cambio en código de APIs de transacciones bancarias.	1.0	2.5	3.5

Tabla 14: Riesgos del proyecto. El nivel de importancia depende del contexto, y no siempre significa catástrofe inmediata.

Luego, según la evaluación observada en la **Tabla 14**, se tiene que el riesgo más iportante para el proyecto es el correspondiente a la ID 1: ”Aparición en el mercado de productos similares que hacen competencia a la aplicación”, debido a que la suma entre su nota de *Impacto* y *Probabilidad* es mayor al resto.

5.3. Mitigación de Riesgos

A continuación se presentan hojas de control de riesgo para cada uno de los riesgos identificados en la sección anterior. Estas hojas presentan los detalles de cada uno, su ***Timeframe***, que consiste en el período de tiempo en que el riesgo influirá en el proyecto en caso de volverse realidad; el **Contexto** del riesgo, en donde se detalla su efecto negativo sobre el proyecto, y las estrategias de **mitigación** y de **contingencia** para cada uno.

Hoja de Control de Riesgos		
ID: 1		**Proyecto:** Dracon
Orden de Prioridad: 1	**Declaración del Riesgo:**	
Categoría: Externo	Aparición en el mercado de productos similares que hacen competencia directa a la aplicación.	
Probabilidad: Media (2.0)	**Impacto:** Alto (3.0)	**Timeframe:** Inmediato
Contexto: La aparición de una nueva app con funcionalidades similares o idénticas a Dracon provoca competencia y posible sustituto.		
Estrategia de mitigacion: Investigar a la competencia actual del mercado en etapas tempranas del proyecto. Esto implica conocer sus funciones tangibles, servicios ofrecidos, público objetivo, precios, cualidades innovadoras, etc; de modo que se puedan elaborar estrategias de diferenciación en torno a esta información.		
Estrategia de contigencia: La estrategia de contingencia debe estar definida como parte de la estrategia de mitigación. Cursos de acción a seguir incluyen: reforzar estrategias de diferenciación de marca (en torno al público objetivo) y funcionalidad (en base a las funciones base de la aplicación), buscar nuevos nichos de mercado, y mejorar la relación con los clientes. Estas alternativas están sujetas al estado de la industria.		

Hoja de Control de Riesgos		
ID: 6		**Proyecto:** Dracon
Orden de Prioridad: 2	**Declaración del Riesgo:**	
Categoría: Externo	Pérdida de interés del público en el producto.	
Probabilidad: Media (2.0)	**Impacto:** Alto (3.0)	**Timeframe:** Medio Plazo

Contexto:
Si el público pierde el interés en la aplicación, la base usuaria se reducirá hasta que potencialmente la aplicación deje de ser rentable y se deba cancelar soporte futuro.

Estrategia de mitigacion:
En la etapa de planificación se deben definir las ventajas competitivas de la aplicación y el cómo se diferenciará de sus competidores, aspectos que pueden seguir redefiniéndose a lo largo de su desarrollo.

Estrategia de contigencia:
Estudiar a las aplicaciones sustitutas y la posible razón del desuso del producto. Luego, con la información reunida, realizar reuniones informativas con el equipo, identificar oportunidades de crecimiento para establecer prioridades en el enfoque de desarrollo, y las necesidades del cliente que aún no han sido cubiertas. El esfuerzo producido después del estudio debe ser en pos de entregar un servicio distintivo, por lo que la calidad es importante para destacar ante la competencia.

Hoja de Control de Riesgos		
ID: 3		**Proyecto:** Dracon
Orden de Prioridad: 3	**Declaración del Riesgo:**	
Categoría: Organizacional	Realizar una mala estimación de HH para una entrega.	
Probabilidad: Alta (2.5)	**Impacto:** Medio (2.0)	**Timeframe:** Medio Plazo

Contexto:
Realizar una mala estimación del tiempo involucrados en los entregables lleva a retrasos inesperados, compensación de este retraso que involucra costos tanto en HH como potencialmente en dinero, y a menor tiempo y/o flexibilidad en entregas futuras.

Estrategia de mitigación:
Al momento de estimar esfuerzos, dar un período flexible que amortigüe retrasos y que al mismo tiempo no influya mayormente en el resto del proyecto.

Estrategia de contigencia:
Compensar la mala estimación de HH en el tiempo restante del Sprint, o en el Sprint siguiente. El esfuerzo adicional debe cubrir el avance hasta asegurar que la entrega será cumplida, o al menos que se llegue a un estándar aceptable; evitando que el esfuerzo restante sea muy disruptivo para el siguiente Sprint.
Una vez terminado el período de desarrollo, se deben discutir y analizar los elementos mal estimados en las etapas de revisión y retrospectiva del Sprint, lo que servirá como retroalimentación al equipo para incorporar tareas que hayan quedado pendientes, y para recalcular costos afectados. El análisis debe incluir qué elementos afectaron a la ejecución del Sprint, y el grado en que lo afectaron para volver a definir estimaciones siguientes que sean necesarias.

Hoja de Control de Riesgos		
ID: 4		**Proyecto:** Dracon
Orden de Prioridad: 4	**Declaración del Riesgo:**	
Categoría: Organizacional	Incumplimiento de eventos y artefactos asociadas al proceso de desarrollo ágil.	
Probabilidad: Media Alta (2.5)	**Impacto:** Medio (2.0)	**Timeframe:** Medio Plazo

Contexto:
El no realizar ni respetar los eventos y/o artefactos de Scrum contradice su objetivo al ignorar la coordinación requerida para que el framework funcione de manera óptima.

Estrategia de mitigacion:
Scrum Master debe estar constantemente monitoreando que se respeten las reuniones, eventos y artefactos asociados al framework.

Estrategia de contigencia:
Retomar las actividades y artefactos que hayan sido descuidados. Scrum Master debe aprovechar la próxima instancia en la que se reúna con el equipo para comunicar las faltas al proceso, y las acciones a tomar en caso de que se requiera compensar los eventos mal implementados o ignorados, y/o los artefactos desactualizados.

Hoja de Control de Riesgos		
ID: 5		**Proyecto:** Dracon
Orden de Prioridad: 5	**Declaración del Riesgo:**	
Categoría: Técnico	Desconocimiento de las tecnologías/herramientas con las que se trabaja.	
Probabilidad: Media (2.0)	**Impacto:** Medio (2.0)	**Timeframe:** Medio Plazo

Contexto:
No conocer las herramientas y/o tecnologías con las que se trabaja significa que el equipo tendrá menos herramientas para solucionar conflictos y deberá invertir tiempo adicional en el estudio de estas.

Estrategia de mitigacion:
Elegir tecnologías con una curva de aprendizaje que se adapte al tiempo de desarrollo, o bien elegir las que el equipo conozca de antemano y sirvan para el desarrollo del producto.

Estrategia de contigencia:
Considerar mayor flexibilidad en los tiempos de entrega que consideren el estudio de las tecnologías utilizadas, y utilizar técnicas que promuevan el aprendizaje colaborativo, como *peer progamming*.

Hoja de Control de Riesgos		
ID: 8		**Proyecto:** Dracon
Orden de Prioridad: 6	**Declaración del Riesgo:**	
Categoría: Técnico	Existencia de vulnerabilidad(es) de seguridad en datos de usuarios.	
Probabilidad: Media (2.0)	**Impacto:** Medio (2.0)	**Timeframe:** Inmediato

Contexto:
Se considera este riesgo en el período de desarrollo de la aplicación. La falta de métodos de seguridad para resguardar los datos de usuario pueden llevar al fracaso de la aplicación, y problemas legales.

Estrategia de mitigacion:
Establecer un estándar que cubra los problemas de seguridad más conocidos para aplicaciones móviles, evitando escatimar en medidas adicionales mientras se evita entorpecer la experiencia del usuario. Algunas de estas medidas incluyen encriptar contraseñas, encriptar datos sensibles que se envíen por la red, e implementar un sistema de bloqueo/borrado de datos después de una secuencia de intentos de acceso erróneos, entre otros.

Estrategia de contigencia:
Implementar nuevas barreras de seguridad o trabajar sobre aspectos de la aplicación asociadas a la vulnerabilidad.

Hoja de Control de Riesgos		
ID: 9	**Proyecto:** Dracon	
Orden de Prioridad: 7	**Declaración del Riesgo:**	
Categoría: Operacional	Falta de comunicación entre miembros del equipo.	
Probabilidad: Media (2.0)	**Impacto:** Medio (2.0)	**Timeframe:** Medio Plazo

Contexto:
Una mala sinergia dentro del equipo se traduce en un entorpecimiento del proceso y falta de coordinación, provocando el efecto opuesto al de una metodología ágil. Esto se puede traducir en diferencias al entender aspectos del proyecto, conflictos interpersonales, y traspaso de información defectuosa, entre otros.

Estrategia de mitigacion:
Fomentar la comunicación cara a cara dentro del equipo, incluyendo incluso a los integrantes remotos en reuniones, promover actividades enfocadas a reforzar habilidades blandas de los integrantes a través de desarrollo de vínculos de confianza mutua y que excedan el contexto laboral.

Estrategia de contigencia:
Dirigir reuniones para asegurar que el equipo completo tenga el mismo nivel de entendimiento de los temas discutidos, y se comprometa en los mismos términos que el resto. Asignar a integrantes que corrijan la información escrita que deba ser traspasada a otro integrante. Solucionar problemas individuales cara a cara, y en privado; evitando que estos escalen

Hoja de Control de Riesgos		
ID: 2	**Proyecto:** Dracon	
Orden de Prioridad: 8	**Declaración del Riesgo:**	
Categoría: Técnico	Servidores no son capaces de aguantar la carga de usuarios.	
Probabilidad: Baja (1.5)	**Impacto:** Medio (2.0)	**Timeframe:** Medio Plazo

Contexto:
Si la carga de las transacciones es muy demandante para los servidores, se producirá lentitud en el servicio, empeorando la experiencia del usuario.

Estrategia de mitigacion:
Estudiar el volumen de de transacciones bancarias online en el proceso de definición de requisitos, y definir métricas que aseguren un servicio consistente para dicha cantidad.

Estrategia de contigencia:
Estudiar y estimar la cantidad de usuarios de la aplicación y el contexto en el que ocurre la caída del servicio para poder evaluar cuánto y si se debe invertir en más servidores.

Hoja de Control de Riesgos		
ID: 7		**Proyecto:** Dracon
Orden de Prioridad: 9	**Declaración del Riesgo:**	
Categoría: Operacional	Mala definición de historias de usuario.	
Probabilidad: Baja (1.5)	**Impacto:** Medio (2.0)	**Timeframe:** Medio Plazo

Contexto:
La mala definición de una historia de usuario lleva a que esta no se pueda descomponer en unidades que entreguen valor a la aplicación, derrochando recursos (principalmente HH) en funciones mal enfocadas o innecesarias.

Estrategia de mitigacion:
Ya que Dracon se plantea como una aplicación que nace desde un equipo independiente, este equipo debe definir correctamente cómo será el prototipo del producto a vender. Luego, al interactuar con los bancos, estos agregarán sus propios requerimientos a negociar, los cuales también deben ser abordados de la manera más íntegra posible.

Estrategia de contigencia:
En reuniones de planificación de Sprint, el equipo debe llegar a un concenso sobre qué se pide en cada historia, evitando que los errores deriven en una potencial entrega fallida. Si la historia es ambigua, se debe re-escribir considerando que la historia debe:

- Agregar valor a la aplicación.
- Estar bien redactada (el equipo debe entender la historia al leerla).
- Tener definido a quién va dirigida (Usuario final, rol en la aplicación, sistema, entre otros).
- Ser alcanzable dentro del período de desarrollo.
- Permitir flexibilidad en el desarrollo. (La historia no debe contener detalles técnicos de implementación, a menos que sean estrictamente necesarios)

Hoja de Control de Riesgos		
ID: 10		**Proyecto:** Dracon
Orden de Prioridad: 10	**Declaración del Riesgo:**	
Categoría: Técnico	Cambio en código de APIs de transacciones bancarias.	
Probabilidad: Media (2.0)	**Impacto:** Bajo (1.0)	**Timeframe:** Inmediato

Contexto:
Asumiendo que el proyecto consta con el apoyo de los bancos, cualquier cambio en la API de un banco impedirá realizar transacciones desde cuentas asociadas a dicho banco.

Estrategia de mitigacion:
Mantener contacto constante con los bancos, y mantener al tanto de los cambios a realizar para planificar con anticipación acciones de soporte.

Estrategia de contigencia:
Ponerse en contacto inmediatamente con el banco que presenta problemas/cambios. En caso de que se trate de un cambio en la API, realizar tareas de soporte inmediatas.

HERRAMIENTAS, TÉCNICAS Y REVISIONES

Los elementos mencionados en esta sección apoyan al equipo de SQA en la realización de sus actividades. Además, las herramientas, técnicas y revisiones deben ser compatibles con métodos de desarrollo ágiles.

6.1. Actividades de QA en SCRUM

La realización de pruebas en un proyecto ágil se deben ajustar a la velocidad de este, por lo que QA debe participar también en otras actividades del proyecto como:

- **Estimar la complejidad de las historias de usuarios -** Debido al rol del analista decalidad, su aporte en la estimación de historias puede aterrizar a los desarrolladores y formular estimaciones realistas que consideren posibilidades fuera del flujo normal de la historia discutida.
- **Escribir casos de prueba unitarios -** Junto a los desarrolladores, los analistas deben escribir casos de prueba unitarios y discutir criterios de aceptación, resultando en un desarrollo más claro.
- **Participar en el proceso de especificación de requerimientos -** Entender las historias de usuario descritas por el cliente ayuda a los analistas a realizar mejores casos de prueba que se ajusten a sus necesidades.
- **Automatizar pruebas -** En un contexto ágil, es necesario que las pruebas también lo sean, por lo que es necesario que las procesos repetitivos y/o manuales se encuentren optimizados lo mejor posible, permitiendo además entregar *feedback* inmediato de las nuevas funcionalidades.
- **Definir cuando una funcionalidad está lista -** Los analistas deben establecer una meta a alcanzar para que el equipo de desarrollo sepa cuando una funcionalidad, mediante una checklist de sub-objetivos a cumplir.

6.2. Registro y generación de informes

Cada reunión, evento y cambio en la versión del producto debe ser documentado, de modo de que el equipo pueda tener la información disponible lo antes posible y de manera transparente. Cada equipo es responsable de entregar su documentación al encargado de manejarla, y de asegurarse de que ésta sea lo más entendible y completa posible, de acuerdo al alcance relevante de cada documento.

6.3. Revisiones

Las revisiones tienen como función monitorear y mejorar los procesos asociados a Scrum, cerciorándose de que se estos se realicen como corresponde. Para esto, se separa el proceso de revisión en tres partes:

- **Desarrollar Plan de SQA -** Se planifican auditorías SQA periódicas, identificando los roles y responsabiliadades del equipo SQA, los productos a revisar por cada integrante, y la frecuencia de las auditorías. Las responsabilidades dentro del equipo SQA abarcan además de la revisión de productos de trabajo (dígase otros planes como de Recursos Humanos, de Estimaciones, de Pruebas, entre otros), la definición (y documentación) de métricas, y la coordinación con el resto del equipo para los eventos de Scrum.
  Todas estas actividades deben ser programadas considerando el estado de desarrollo del proyecto, además de estar documentadas junto al resto de los detalles establecidos dentro del plan de SQA.
- **Preparación -** Se definen y documentan las políticas y procedimientos a utilizar para la prevención de defectos, y se informa al equipo, realizando una sesión de entrenamiento en caso de ser necesario.
- **Revisión del proceso -** Se divide en revisiones individuales del Plan de Proyecto, Análisis de Requerimientos, Diseño de Pruebas, Revisión del producto antes de lanzamiento, y del cierre del proyecto. Las revisiones de producto, antes de la puesta en producción, se divide en las diferentes entregas, y la periodicidad que se estableció para estas.

6.4. Auditoría

Las auditorías son la inspección de registros, y de las actividades que las generan. En proyectos Scrum forman parte de la retroalimentación del proyecto, ayudando a identificar impedimentos y mitigar errores tempranamente y en paralelo a la entrega incremental de nuevas versiones del producto. Dentro de este contexto, se puede serparar el proceso de auditoría de un proyecto ágil en tres fases:

6.4.1. Investigación y Aplicación de Auditorías para Proyectos Ágiles

El auditor debe entender el proceso ágil y el proyecto en sí. Por lo tanto, éste debe cumplir las siguientes actividades, entre otras:

- Familiarizarse con las métricas de éxito.

- Identificar fortalezas individuales en el equipo Scrum.
- Asegurar que las herramientas de gestión del proyecto se ajusten a la agilidad de este.
- Validar que estrategia del proyecto se enuentre alineada con los objetivos .
- Relacionarse con los involucrados en el proyecto, desde los integrantes del equipo Scrum hasta los stakeholders.
- Involucrarse en eventos y reuniones del proyecto.

6.4.2. Análisis de Proyectos Ágiles

En esta etapa, el auditor se familiariza con el proyecto y los componenetes involucrados en su puesta en marcha mediante actividades como:

- Investigar a fondo las complicaciones que surjan en el proyecto para buscar sus posibles causas.
- Observar el desarrollo de los Sprint.
- Asistir a las reuniones asoaciadas a eventos Scrum.
- Revisar Backlog del producto y Backlog de Sprints.
- Revisar documentación de las historias de usuario.

6.4.3. Reportes y Recomendaciones

Con la información recopilada en las etapas anteriores, el auditor crea los reportes finales e incluye sus recomendaciones. Esto se traduce en actividades como:

- Documentar lo observado sobre artefactos del proyecto.
- Evaluar reuniones.
- Evaluar que requerimientos del cliente hayan sido entendidos y cumplidos.
- Evaluar la eficiencia en el uso de recursos humanos.
- Escribir recomendaciones que apunten a capitalizar potenciales oportunidades.
- Identificar falencias en el desempeño del proyecto, y evaluar medidas que permitan mejorarlo.

PRUEBAS

7.1. Estructura de las Pruebas

La realización de pruebas asegura que los requerimientos cumplan los estándares de calidad acordados con el cliente, y consisten en un un conjunto de actividades que ejecutan las funcionalidades del producto en distintos escenarios que abarcan los distintos y más probables *inputs* que puede recibir el sistema. Estas actividades arrojan resultados que se comparan con los esperados y/o propuestos. Como el objetivo de las pruebas es encontrar la mayor cantidad de defectos posibles, es necesario que éstas sean lo más exhaustivas posibles, para así encontrar errores y/o funcionamientos no deseados antes de la etapa de producción. Dicho esto, se pueden dividir las actividades de prueba en 4 etapas:

- Planificación
- Especificación
- Ejecución
- Análisis de Resultados

7.2. Planificación

Dado que el contexto del proyecto es el de una metodología ágil, las pruebas deben estar planificadas de tal manera que no retracen los Sprint ni las entregas. Es por esto que se deben priorizar las funcionalidades más críticas, dándoles a estas actividades de prueba más exhaustivas, mientras que a los no tan prioritarios se les pueden realizar pruebas de aceptación base, es decir, con una selección de los registros que más probablemente se procesen en las funcionalidades asociadas.

7.3. Especificación

Se deben definir el propósito y el procedimiento asociados a cada caso de prueba. Las actividades asociadas a dichas definiciones corresponden a:

- Identificar las funcionalidades más importantes
- Identificar aspectos que se deben probar en funcionalidades escogidas

- Definir casos de prueba asociados a aspectos a probar
- Definir procedimiento a seguir para el desarrollo de las pruebas, incluyendo qué funciones se van a probar y las variables de entrada asociadas a estas.
- Definir los requisitos de aceptación de las pruebas
- Definir rangos de tiempo requeridos para realización de pruebas
- Definir herramientas a utilizar para las pruebas
- Identificar aspectos automatizables del procedimiento de prueba
- Documentar casos de prueba considerando lo expuesto anteriormente

7.4. Pruebas base

Antes de realizar entregas a producción, es importante que se realicen pruebas base que aseguren el correcto funcionamiento del sistema en los aspectos más críticos del software. Ya que estas pruebas deben realizarse repetidas veces, es necesario automatizar aspectos de éstas que permitan mantener la agilidad del proyecto y no estancar las entregas en la etapa de *testing*. Las pruebas sugeridas para *Dracon* son:

7.4.1. Funcionamiento correcto de transferencias bancarias

La aplicación debe ser capaz de completar correctamente transferencias bancarias entre dos cuentas que están correctamente ingresadas, con un monto válido y verificando los datos de seguridad asociados a cada cuenta. Esta prueba incluye métodos de distintos bancos, sean estos tarjetas de coordenadas, códigos PIN, entre otros.

7.4.2. Operaciones de Creación/Lectura/Edición/Borrado deben funcionar correctamente para contactos, grupos y eventos

La creación, edición y borrado de los elementos mencionados debe reflejarse correctamente en su vista respectiva, asegurando además de que los objetos no contengan información incompatible con sus variables.

7.4.3. Códigos QR deben producir resultados correctos y rápidos

La lectura de los códigos QR debe entregar de manera fidedigna los datos asociados al código en sí. Además, el procesamiento de estos debe cumplir con el criterio de evaluación del tiempo de espera establecido en la **Tabla 1**.

7.4.4. Datos guardados en cuentas bancarias propias deben ser válidos

Se debe realizar una verificación de la correctitud de los datos ingresados al momento de asociar una cuenta propia o agregar un contacto con los sistemas bancarios, a modo de comprobar que la cuenta es real, evitando así la adición de cuentas ingresadas de manera errónea a la aplicación.

7.4.5. Rápidos tiempos de carga para operaciones en línea

Las operaciones que utilicen algún servicio web deben cumplir con los requerimientos asociados a tiempos de espera.

7.4.6. Mostrar estado de servicios bancarios

La aplicación debe estar sincronizada con el estado de los servicios bancarios, de modo que cualquier indisponibilidad que estos presenten se vea reflejada de manera visual en el software.

7.4.7. Borrado de Seguridad

El borrado de seguridad de los datos asociados a la aplicación debe ser absoluto, de modo que no quede rastro de la información que alguna vez manejó el programa.

7.5. Ejemplo de prueba base

A continuación se propone una estructura para el caso de prueba: **Funcionamiento correcto de transacciones bancarias**, con todos los componentes que la componene de acuerdo a evidencia práctica y a distintas fuentes:

[QA01] Funcionamiento correcto de transacciones bancarias

Descripción: Las transacciones bancarias deben ser ejecutadas con éxito, fallando únicamente cuando existan problemas de conexión o producto de errores humanos, como ingresar datos de manera errónea. Además, sus resultados deben ser notificados de manera inmediata a los usuarios involucrados.

Pre-Requisitos de la Prueba:

- Usuario que transfiere debe estar conectado a internet.
- Servicio bancario debe estar funcionando.
- Credenciales bancarias del usuario que transfiere deben estar correctamente ingresadas.
- Datos de la cuenta destino deben estar correctamente ingresados.

Validaciones:

- Realizar transacciones con credenciales incorrectas.
- Realizar transacciones a cuentas con datos incorrectos.
- Realizar transacciones sin conexión
- Realizar transacciones con datos y credenciales bien ingresados.

Criterios de Aceptación:

- Aplicación debe notificar a usuarios involucrados del movimiento realizado.
- Historial de movimientos se actualiza con información de la transacción.
- En caso de no haber conexión, aplicación debe informar la falta de conexión.
- En caso de error, aplicación debe informar al usuario que transfiere el origen de la falla.

El caso de prueba se compone de:

- **Identificador:** Clave única para cada caso de prueba, que facilita futuras referencias y una potencial clasificación.
- **Título:** Entrega una breve descripción de la prueba.
- **Descripción:** Detalla el contexto de la prueba, siguiendo el concepto del título.
- **Pre-Requisitos de la Prueba:** Elementos técnicos que componen el entorno de la prueba y la base sobre la que se ejecuta el procedimiento.
- **Validaciones:** Pasos a ejecutar para obtener resultados empíricos de las funcionalidad que se está probando.
- **Criterios de Aceptación:** Resultados ideales con los que se comparan los obtenidos en las pruebas de validación, y que establecen el criterio para aceptar o rechazar una funcionalidad.

7.6. Ejecución

La ejecución de las pruebas se realiza una vez que éstas se encuentren completamente definidas y se tenga una visión clara del objetivo de cada una. Dado la naturaleza ágil e incremental del proyecto, se sugieren las siguientes pruebas:

- **Pruebas unitarias:** Pruebas que se realizan a funcionalidades recién creadas.
- **Pruebas de integración:** Prueba grupal de múltiples módulos que incluyan y sean relevantes a nuevas funcionalidades. En caso de funciones críticas, identificar qué aspectos de las pruebas son atomatizables para poder optimizar el esfuerzo.
- **Pruebas de aceptación:** Pruebas que determinan si nuevas funcionalidades cumplen con requerimientos establecidos previamente.
- **Pruebas de Sistema:** Pruebas que determinan si el sistema completo se encuentra operacional y cumple con requerimientos. Dada la naturaleza del proyecto, este tipo de prueba se realiza previa a una nueva versión del producto, y es donde el rol de la automatización de pruebas base es crítico.

7.7. Análisis de Resultados

Una vez realizadas las pruebas, se identifican los defectos encontrados, sus causas, y la manera de resolverlos. Para esto se designa un equipo encargado que, con dicha información, se encargará de corregir el defecto y actualizar la funcionalidad con las correcciones necesarias.

CONCLUSIONES

En la época digital actual, existen numerosas y diversas maneras de mejorar procesos cotidianos, especialmente considerando el fácil acceso a computadoras portátiles como lo serían los dispositivos inteligentes, cuya calidad crece a una velocidad explosiva, y donde incluso las gamas más bajas de smartphones son considerablemente más potentes a equipos de la década pasada. *Dracon* nace dentro de este contexto, y con el desafío de llevar al usuario común una aplicación que le permita utilizar su dinero virtual desde la palma de su mano, en todas partes, y sin la necesidad de artefactos externos, asegurando paralelamente al usuario tanto comodidad de uso como seguridad en el manejo de sus datos.

Para este y todo proyecto en general, se hace necesario disponer de un plan de calidad, el cual asegurará que el equipo siga una estructura que lo mantenga enfocado hacia su meta, y sea capaz de entregar un producto que cumpla con estándares de calidad establecidos por un cliente que puede o no conocer el mundo informático, y con el cual solo la buena comunicación permitirá que el equipo satisfaga completamente sus necesidades. Después de todo, una buena planificación evita perder el tiempo en imprevistos.
En el contexto de la Feria de Software, un Plan de Calidad hubiese permitido dar al equipo una visión más objetiva de la aplicación a desarrollar, y del método de trabajo. Sin embargo, el proceso de la FESW se separa del agilismo debido a documentos ajenos al framework, que se traducen en esfuerzos contrarios al plan de calidad, por lo que si bien sigue teniendo valor para el desarrollo, convive con otras tareas de la asignatura que no agregan valor al producto *per se*.

La organización de un proyecto depende a grandes rasgos de los aspectos técnicos y de gestión, los cuales, al ser desglosados, dejan ver la cantidad de variables involucradas: desde los recursos más técnicos y de bajo nivel hasta los recursos humanos. En un proyecto Scrum, las relaciones interpersonales se valorizan tanto como los conocimientos técnicos, ya que la base del framework consiste en el trabajo rápido y eficaz, el cual no sería posible sin una buena comunicación interna. Además, todos los elementos de un proyecto ágil deben serlo, puesto que cualquier cuello de botella generado dentro del desarrollo producirá que se estanquen la entregas, cancelando el aspecto ágil del mismo.

El desarrollo de esta memoria me ha dado la posibilidad de sumergirme en la elaboración de un Plan de Calidad basado en el framework Scrum, lo que me ha permitido entender el núcleo del agilismo y compararlo con ejemplos prácticos; tanto en el trabajo realizado durante la Feria de Software, como en el ámbito laboral. Me ha dado a entender la importancia del Product Owner y Scrum Master, cuyos roles no dimensionaba en un principio; que el testing no queda relegado a proyectos tipo cascada, debido a su sinergia con la automatización y la gestión del proyecto; y que el agilismo promueve finalmente un mejor ambiente laboral, donde la colaboración y comunicación son clave, y cada integrante es un individuo y a la vez parte de un equipo que funciona en sintonía.

REFERENCIAS BIBLIOGRÁFICAS

[pae, 2001] (2001). Métrica v.3. https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html#.XHWHMbjQ_tQ.

[BSSC, 1995] BSSC (1995). *Guide to software configuration management*. European Space Agency.

[Hasija, 2012] Hasija, P. (2012). My experience as a qa in scrum. https://www.infoq.com/articles/experience-qa-scrum/. Accedido el 27 de Junio.

[HelpingTesters, 2016] HelpingTesters (2016). Software quality assurance activities. http://www.helpingtesters.com/software-quality-assurance-activities/.

[Jummp, 2011] Jummp (2011). Desarrollo de software. ciclo de vida iterativo incremental. https://jummp.wordpress.com/2011/03/31/desarrollo-de-software-ciclo-de-vida-iterativo-incremental/. Accedido el 14 de Febrero del 2019.

[Pressman, 2010] Pressman, R. (2010). *Ingeniería del Software. Un enfoque práctico 7ma Edición*. Mc Graw Hill.

[ProfessionalQA, 2016] ProfessionalQA (2016). Mccall software quality model. http://www.professionalqa.com/mc-call-software-quality-model. Accedido el 16 de Junio, 2019.

[ProfessionalQA, 2019] ProfessionalQA (2019). Boehm software quality model. http://www.professionalqa.com/boehm-software-quality-model. Accedido el 16 de Junio, 2019.

[Schwaber y Sutherland, 2011] Schwaber, K. y Sutherland, J. (2011). The scrum guide. *Scrum Alliance*, 21.

[Scrum.org, nd] Scrum.org ((n.d.)). What is a scrum development team? https://www.scrum.org/resources/what-is-a-scrum-development-team. Accedido el 26 de Junio.

[Visconti, 2000] Visconti, M. (2000). Herramientas específicas para sqa y scm. https://www.inf.utfsm.cl/~visconti/herramientas/. Accedido el 20 de Febrero, 2019.