Adecuación de la norma ISO/IEC 29110-5.1.1 a la metodología TDD

Castañeda Marrero, Gabriela[1]

[1] Universidad de Belgrano, Buenos Aires, Argentina

gab.cmarrero@gmail.com

Abstract En los años 90 surgió una metodología de desarrollo que planteaba invertir el orden tradicional de un desarrollo de software. En esencia, proponía desarrollar primero las pruebas y utilizarlas como base y guía para la programación del sistema. Esta metodología se conoce como Desarrollo dirigido por pruebas ó Test Driven Development (TDD).

En este trabajo se construirá un sistema de software mobile (aplicación y web), utilizando esta metodología. Se seguirán las directrices de la norma ISO/IEC 29110-5.1.1 que regula el desarrollo de software para pequeñas organizaciones. El trabajo contempla adaptaciones en la norma para adecuarse a la metodología TDD utilizada en el desarrollo.

La metodología basada en pruebas asegura calidad, y un código más limpio y robusto en comparación a las metodologías tradicionales. Para obtener una medida objetiva de calidad se evaluará el sistema resultante con la norma MyFEPS /QSAT.

Se pretende comprobar la factibilidad de la utilización de la metodología TDD en el desarrollo de una aplicación Android y un sitio web PHP utilizando la norma ISO/IEC 29110-5.1.1 en el proceso de desarrollo. Se analizará el impacto que tiene la aplicación de estas metodologías en el tiempo de desarrollo y la calidad del código resultante.

Keywords: Test Driven Development, ISO/IEC 29110, Ingeniería de Software, Aplicación mobil

1. Introducción

1.1. Planteamiento del Problema

Se busca aplicar la metodología de desarrollo dirigida por pruebas (de ahora en adelante, TDD) al desarrollo de un sistema de administración de franquicias, y al mismo tiempo validar el cumplimiento de la norma ISO/IEC 29110-5-1-1[1].

Se trata de validar la combinación de la metodología TDD con esta norma de la ISO y de IEC, destinada a pequeñas entidades que desarrollan software, en un proyecto de software mobile, en plataforma Android y web. A la fecha no se ha encontrado ninguna publicación referida al uso de estos dos estándares.

Además de validara la calidad del producto final utilizando el *framework* MyFEPS, desarrollado por la universidad de Belgrano [2].

Se pretende responder las siguientes preguntas:

- ¿Es factible aplicar la metodología TDD siguiendo las buenas prácticas establecidas en las normas ISO para el ciclo de vida del software?
- ¿Es factible / recomendable la utilización de TDD en proyectos pequeños?
- ¿Al aplicar TDD al desarrollo de una aplicación, se logra una mejora sustancial en los tiempos del proceso de desarrollo y en la calidad del producto final?
- ¿Es medible / cuantificable el nivel de mejora obtenido en el producto de software?

El sistema a desarrollar es una plataforma para controlar y administrar franquicias, tanto desde el punto de vista del franquiciante (otorgante de la franquicia) como del franquiciado (receptor de la franquicia). El objetivo del sistema es facilitar la interacción entre el franquiciante y sus franquiciados. Consta de dos aplicaciones: un sitio web (PHP) de administración y una aplicación móvil (Android) para controlar pedidos, entregas, comunicaciones.

Desde el sitio web, el franquiciante debe poder crear, modificar y borrar usuarios, configurar productos, restricciones, notificaciones y comunicarse con sus franquiciados. Además, debe poder personalizar ambas aplicaciones para darle el *Look and Feel* de la empresa de que se trate.

El usuario franquiciado debe ver la app móvil personalizada, debe ver los productos disponibles, debe poder recibir las notificaciones del administrador y comunicarse con este, debe poder hacer pedidos de los productos hasta una hora especificada por el administrador o libremente. Debe enviársele al franquiciante una notificación con los pedidos de sus franquiciados debidamente identificados.

1.2. Objetivos

1.2.1. Objetivo principal

El objetivo principal de este trabajo es el desarrollo de una aplicación (App) utilizando la metodología TDD mientras se mantiene la conformidad

con la norma ISO/IEC 29110-5.1.1, y las características de calidad deseadas por las partes interesadas (utilizando MyFEPS). El sistema desarrollado resolverá, además, un problema existente en la gestión de franquicias y la distribución de productos a las mismas.

1.2.2. Objetivos específicos

Objetivos específicos:

- Investigación de la metodología de desarrollo dirigido por pruebas (TDD).
- Investigación de la norma 29110-5.1.1 para pequeñas entidades que desarrollan software.
- Adecuación de la norma ISO/IEC 29110-5.1.1 para la metodología TDD
- Aplicación práctica de TDD en una aplicación mobile de un Sistema de gestión de franquicias, personalizable respecto de su *Look and Feel*.
- Resultados objetivos en cuanto a la calidad del sistema obtenido, en base a los resultados obtenidos en la evaluación de software con MyFEPS.
- Medir la calidad del software obtenido, siendo este un proyecto de pequeña escala.

1.3. Justificación

Existen muchos artículos en los cuales TDD es mostrada como una gran metodología para los proyectos de desarrollo, brindando calidad, robustez y limpieza al código generado. Se expresa en ellos también que usando esta metodología los errores en manos del usuario se reducen significativamente [3].

Sin embargo, es una metodología que a pesar de aportar todos estos beneficios aún no se ha esparcido por todo el mundo del desarrollo, sino que sigue siendo considerada como algo “nuevo”. No ha sido hasta la actualidad que varias empresas han comenzado a utilizarla como base de sus proyectos (Ejemplo: Etermax) [4].

Pero los artículos indican que TDD tiene una curva alta de aprendizaje cuando un desarrollador comienza a adentrarse en ella. En el artículo *The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis* [5], donde se estudian los efectos de usar TDD, visualiza el gran impacto que tiene sobre la productividad y calidad en proyectos industriales de gran escala, pero advierte que en proyectos de envergadura menor esta calidad no se ve reflejada.

Desde el punto de vista del ciclo de vida de desarrollo, la metodología no abarca el análisis y diseño del sistema, es por eso que en este trabajo se implementará la norma ISO/IEC 29110-5.1.1 para entidades pequeñas, adaptándola para el uso de TDD en todo el ciclo de vida del software.

De esta manera este trabajo se basará en comprobar la factibilidad de la aplicación de la metodología TDD al desarrollo de una aplicación Android y un sitio web PHP, proyecto de pequeña envergadura, utilizando y adaptando la norma ISO/IEC 29110-5.1.1 para el proceso de desarrollo ágil con TDD.

1.4. Alcance y Limitaciones

Se expondrá la metodología de desarrollo basado en pruebas dándole un marco teórico al proyecto.

Se desarrollará un sistema consistente en una aplicación móvil y un sitio web que servirán como muestra de la utilización de dicha metodología.

Se documentará cada etapa del ciclo de desarrollo del sistema según lo especificado en la norma ISO/IEC 29110-5.1.1 que establece buenas prácticas en el perfil de entrada, en el ciclo de vida del software para equipos de desarrollo pequeños.

Se analizarán los éxitos y las dificultades que se presenten durante el proceso de desarrollo. Se utilizará la metodología MyFEPS / QSAT [6] para evaluar el sistema resultante y tener una medida objetiva de la calidad del mismo. Finalmente se concluirá cómo influye la aplicación de TDD / ISO en los tiempos del proceso y en la calidad del producto obtenido.

El presente proyecto no pretende realizar un análisis teórico exhaustivo de las normas ISO ni de la metodología TDD, sino servir como ejemplo práctico de la aplicación de ambas.

El sistema de Gestión *Franquia* consta de:

- Una app móvil para Android que gestionará la interacción entre los franquiciados y el franquiciante.
- Un sitio web (PHP) para la administración de usuarios, permisos de acceso, productos, mensajes, y la personalización de la empresa.

La app no incluirá la gestión de la administración de la empresa franquiciante.
La app incluirá la gestión de los franquiciados:

- Recibir las comunicaciones del administrador y responder a estas.
- Ver los productos disponibles.
- Hacer pedidos de los productos hasta la hora especificada por el administrador o libremente si no hay restricciones de horario.
- Ver el estado de sus pedidos, aceptación y horario de entrega estimado.

La web no incluirá la gestión de los franquiciados.
La web incluirá la gestión de los franquiciantes:

- Especificar la empresa franquiciante y personalizar el ambiente de trabajo para darle el *Look and Feel* deseado. Esta personalización se aplicará a ambas aplicaciones.
- Crear, modificar y eliminar usuarios franquiciados.
- Crear, modificar y eliminar usuarios productos.
- Establecer restricciones de horarios para la recepción de pedidos si así lo desea.
- Crear alarmas / notificaciones para enviar a sus franquiciados a través de la app móvil y recibir respuestas u otras comunicaciones de estos.
- Recibir notificaciones de los pedidos de sus franquiciados debidamente identificados y aceptar o rechazar los mismos.

2. Marco teórico

2.1. Fundamentos de TDD

TDD es una metodología de desarrollo que cambia el paradigma comúnmente utilizado en el desarrollo de software. Estamos acostumbrados a que durante el ciclo de vida de desarrollo existan dos grandes momentos: programación y pruebas, en ese orden. Es decir, luego de tener una funcionalidad programada se realizan las pruebas pertinentes. TDD invierte este paradigma: primero se realiza la prueba y luego se programa la funcionalidad [7].

En su libro "*Test Driven Development By Example*" [3], Kent Beck asegura que este cambio en la metodología incrementa la calidad del desarrollo y genera un código más limpio.

Es importante destacar que TDD no es una manera de *testear* sino una metodología para el desarrollo. Esta se basa en que las pruebas unitarias ayuden a desarrollar la funcionalidad requerida, de esta manera el desarrollador evita el "sobre pensar" mientras desarrolla, solo se programa lo que se requiera para pasar las pruebas.

Comúnmente cuando se programa tendemos a pensar en el futuro problema que puede enfrentar la función, pero de esta manera podemos perder de vista cual era el fin primario de la función. TDD mantiene la premisa de realizar una prueba para cada cosa, por ejemplo, con una función de división, la función primaria es dividir por lo cual deberemos tener un test que pruebe la división. ¿Pero qué pasaría si se le pasa como divisor un cero? Para ese caso realizaremos otro test: un test a la vez. [8][3].

A pesar de que la metodología vio la luz en los 90 no tuvo una gran difusión y no es hasta la actualidad que ha resurgido con fuerza. Esto se debe a que los beneficios de su uso no son fácilmente visibles. Incluso se ha comprobado que se requiere un cambio en la estructura del pensamiento, es decir que la curva de aprendizaje al principio es elevada [3] [9].

En el artículo *The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis* [5] donde se estudian los efectos de usar TDD sobre un desarrollo, se destaca que parte del éxito de la metodología se basa en las habilidades de programación, *testing*, diseño, refactorización y pensamiento en TDD del programador.

Además, visualiza el gran impacto que tiene sobre la productividad y calidad en proyectos industriales de gran escala, pero advierte que en proyectos de envergadura menor esta calidad no se ve reflejada. Gabriel Lowe reafirma esto << "*Existe cierta evidencia que sugiere que TDD mejora la calidad externa, y aunque los resultados de los experimentos controlados en su mayoría son no concluyentes, los estudios de uso industrial y piloto le favorecen fuertemente (...)"* >> [10]

El ciclo de vida de TDD está compuesto por tres fases:

1. **Codificar la prueba** (También conocida como fase roja, *RED*)
   En este paso se piensa en qué se debe desarrollar y se crea una lista de requerimientos. A partir de esta lista se van codificando las pruebas para cada funcionalidad. Se le llama roja porque al codificar la prueba y correrla, esta fallará, pues aún no existe la funcionalidad.
2. **Codificar la funcionalidad** (También conocida como fase verde, *GREEN*)

En este paso se debe pensar en cómo hacer que la prueba pase, sin importar como se logra esto. Se pueden cometer todo tipo de "pecados informáticos" en este momento. Nos podemos tomar cualquier licencia, incluso devolver una constante, todo está permitido siempre que se logre que la prueba sea exitosa.

3. **Refactorizar**
   En este paso se debe pensar en cómo optimizar la funcionalidad desarrollada en el paso anterior. Se optimiza el código, se corrigen los errores, se eliminan posibles duplicaciones, etc.

Estas fases se repiten cíclicamente, tomando una funcionalidad de la lista a la vez, mientras queden funcionalidades por desarrollar.

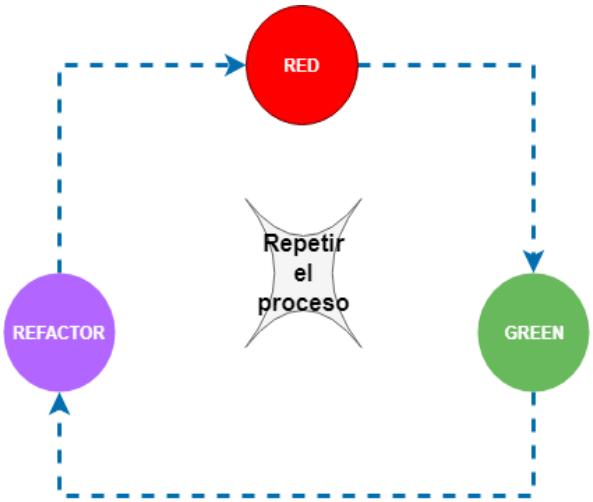

Figura 1: Fases de TDD [Imagen propia]

TDD propone una metodología incremental de desarrollo, donde no se programa nada que no tenga antes su correspondiente prueba.

De acuerdo a Kent Beck [3] hay tres estrategias para implementar una prueba en TDD:

- **Fake It:** Retornar una constante de manera que la prueba sea verde e ir cambiándole a la implementación real progresivamente.
- **Obvious Implementation:** Implementar lo obvio, si una función es una suma, sabemos exactamente que programar, no hacen falta pequeños pasos para llegar a la implementación final.
- **Triangulate:** Solo implementar la generalización de la prueba cuando tenemos dos o más ejemplos. Si tenemos dos ejemplos podemos obtener el tercero (generalización). Se utiliza cuando no sabe exactamente cómo seguir. Tener en cuenta que luego de generalizar hay que eliminar las duplicaciones.

En resumen, los pasos de TDD:

1. Escribir la prueba.
2. Compilar el código de la prueba.
3. Hacer que la prueba corra y ver que falla.
4. Implementar lo que haga falta para que la prueba sea exitosa.
5. Correr la prueba y ver que es exitosa.
6. Refactorizar el código para darle claridad y eliminar duplicaciones.
7. Repetir desde el principio.

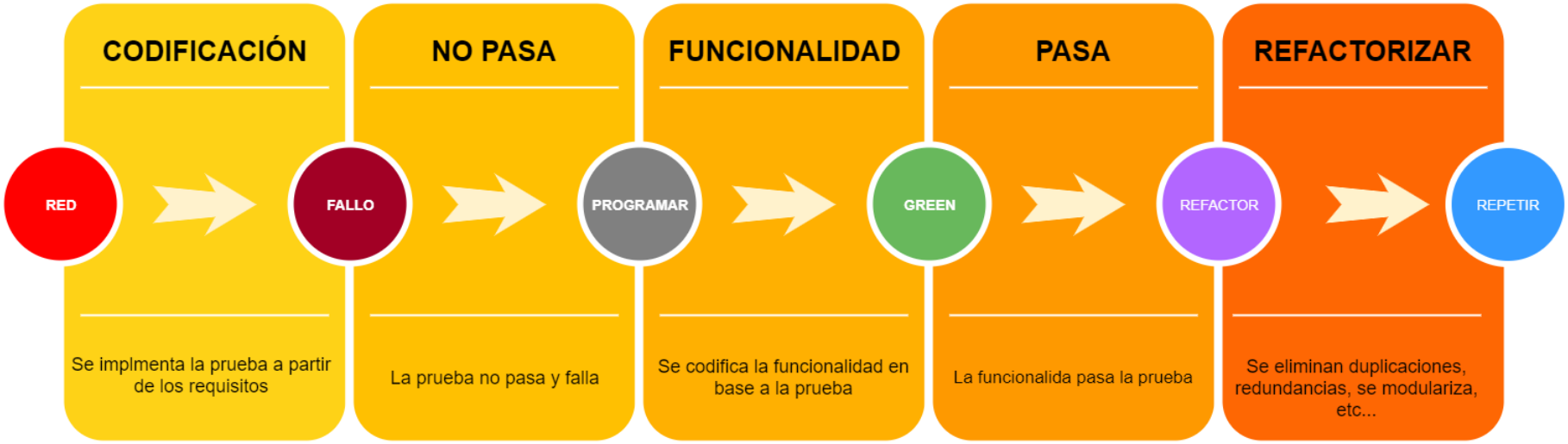

Figura 2: Proceso de TDD [Imagen propia]

2.1.1.Ejemplo de TDD

Ejemplo simple basado en el libro "*Test Driven Development By Example*" [3] de cómo funciona TDD.

Se necesita programar una calculadora de suma y resta.
Como paso inicial se genera la lista con las funcionalidades necesarias:

1. Una función que realice la operación Suma
2. Una función que realice la operación Resta

Tomamos la primera funcionalidad de la lista, en este caso la función que realiza la operación Suma y pensamos en cómo debería funcionar, recibiendo dos números y devolviendo su suma: sumar (1, 1), debería devolver 2.

Ante todo, creamos una prueba para comprobar el resultado de la función:

```
assertEquals(2, sumar(1, 1) );
```

Figura 3: Ejemplo TDD

Estamos utilizando el método assertEquals de JUnit, que compara dos valores y devuelve true cuando estos son iguales.

Esta prueba va a fallar porque no existe una función sumar, de hecho, ni siquiera va a compilar. Hasta aquí tenemos la fase roja, en la que definimos la prueba y esta falla.

Entonces creamos la función sumar y hacemos que devuelva lo que la prueba espera, simplemente un 2

```
public void int sumar (int a, int b) {
        return 2;
}
```

Figura 4: Función aplicando "Fake It"

Con esta implementación nos aseguramos de que la prueba pase, aunque hayamos hecho trampa, pues no es una implementación real de una suma, pero estamos en la fase verde y todo esté permitido para lograr una prueba exitosa.

Ahora vamos a la fase de Refactorizar, eliminamos la duplicación, sustituimos el 2 de la función por la suma de los dos parámetros:

```
public void int sumar (int a, int b) {
        return a + b;
}
```

Figura 5: Función para aplicar TDD

Finalmente se obtiene una prueba en verde con la implementación correcta. Ahora podemos proceder con la siguiente funcionalidad de la lista de requerimientos.

Por supuesto, con una funcionalidad tan simple, podríamos haber creado la función desde el comienzo, pero elegimos un código muy sencillo para concentrarnos en la secuencia de TDD.

2.2. Fundamentos de la ISO/IEC 29110

Hasta este momento se ha expuesto la metodología TDD en el desarrollo únicamente; es decir, la metodología no abarca el análisis y diseño del sistema, es por esto que este trabajo se implementará considerando las buenas prácticas de la norma ISO/IEC 29110-5-1-1 para entidades de desarrollo pequeñas, adaptándola para el uso de TDD en todo el ciclo de vida del software.

La serie de normas ISO/IEC 29110 [1] es la primera serie internacional para la mejora de procesos del software y buenas prácticas aplicados para organizaciones pequeñas, llamadas por sus siglas en ingles VSE, *very small entities*.

Las normas para procesos de software como la ISO/IEC 12207 están pensadas para grandes organizaciones. Una empresa pequeña donde una persona cumple varios roles no puede seguir las buenas prácticas como se espera de la norma. Es así como la norma 29110-5.1.1 permite a estas entidades pequeñas evolucionar en el proceso de mejora y calidad.

La norma tiene como objetivo definir modelos de procesos más sencillos para facilitar su implementación. Por ejemplo, simplificar puntos de la norma 12207 omitiendo pasos para que sea accesible para las VSE.

2.2.1.Perfiles de la ISO/IEC 29110

Las organizaciones que implementan esta serie de normas pueden certificarse de acuerdo a tres perfiles de procesos que son:

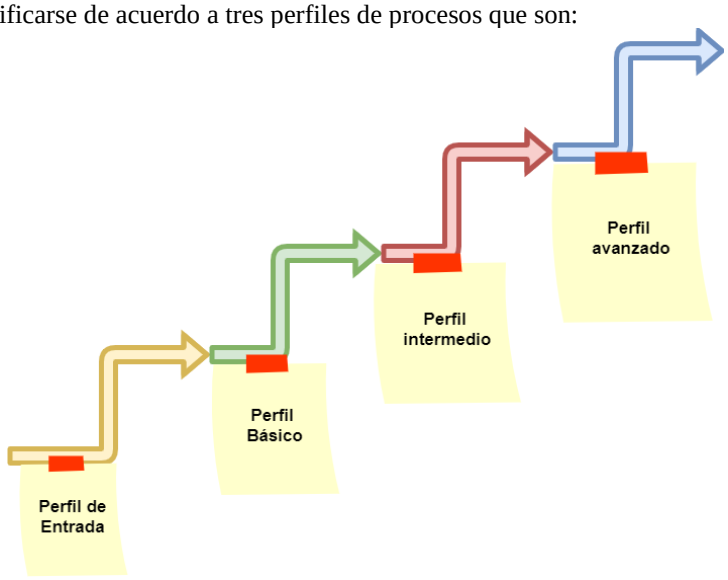

Figura 6: Perfiles de la ISO/IEC 29110 [Imagen propia]

1. **Perfil de entrada:** dirigido a VSE que están comenzando (menos de tres años de haber sido creada) y llevan a cabo un solo proyecto donde están involucradas menos de seis personas por mes.
2. **Perfil básico:** dirigido a VSE que tienen un solo proyecto que será llevado a cabo por un solo grupo de trabajo.
3. **Perfil intermedio:** dirigido a VSE que tienen más de un proyecto en paralelo que será llevado a cabo por más de un grupo de trabajo.
4. **Perfil avanzado:** dirigido a VSE que desean crecer competitivamente.

2.2.2. Serie de normas de ISO/IEC 29110

La serie de normas 29110, desde el punto de vista de sus destinatarios, contempla principalmente las siguientes normas:

- ISO/IEC TR 29110-1:2016:
  Systems and software engineering — Lifecycle profiles for Very Small Entities (VSEs) – Part 1: Overview. [11]
- ISO/IEC TR 29110-3-1:2015:
  Systems and software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 3-1: Assessment guide. [12]
- ISO/IEC 29110-3-3:2016:
  Systems and software engineering — Lifecycle profiles for Very Small Enterprises (VSEs) — Part 3-3: Certification requirements for conformity assessments of VSE profiles using process assessment and maturity models. [13]
- ISO/IEC 29110-4-1:2011:
  Software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 4-1: Profile specifications: Generic profile group. [14]
- ISO/IEC TR 29110-5-1-1:2012:
  Software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 5-1-1: Management and engineering guide: Generic profile group: Entry profile. [15]
- ISO/IEC TR 29110-5-1-2:2011:
  Lifecycle profiles for Very Small Entities (VSEs) — Part 5-1-2: Management and engineering guide: Generic profile group: Basic profile. [16]
- ISO/IEC TR 29110-5-2-1:2016:
  Systems and software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 5-2-1: Organizational management guidelines. [17]

En este trabajo se utilizará el perfil de entrada que corresponde a la ISO/IEC 29110-5.1.1

El **perfil de entrada** consta de dos procesos:

- Gestión del Proyecto
- Implementación del Software

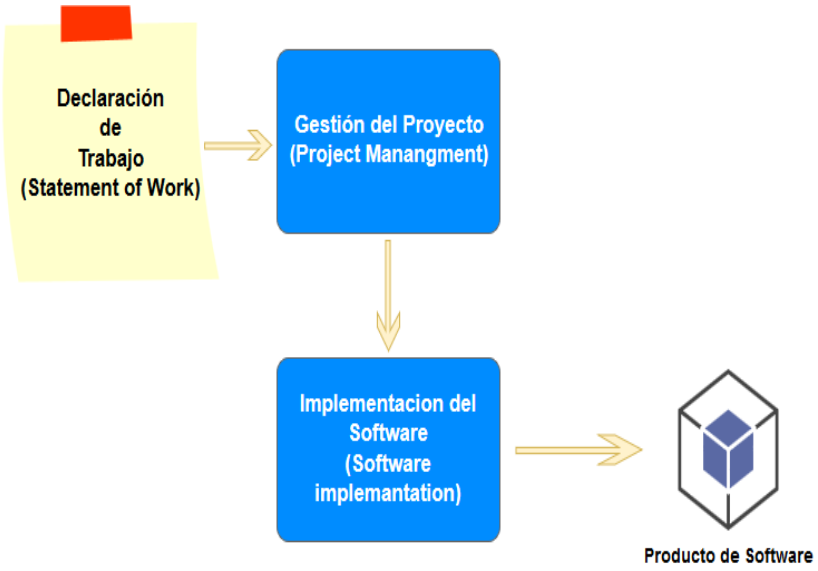

Figura 7: Etapas de la ISO/IEC 29110-5.1.1 [Imagen propia]

2.2.3.Declaración de Trabajo

Siguiendo los pasos que indica el sitio "UpTo25" [18], una plataforma paga de ayuda para la implementación de la norma ISO/IEC 29110, todo comienza con la Declaración de Trabajo (SoW, por sus siglas en inglés). El SoW es uno de los documentos más importantes de proyecto, debido a que contiene todos los lineamientos del mismo y a que es el documento de entrada para el uso de la norma ISO/IEC 29110-5.1.1.

Este documento define todos los aspectos del proyecto [19]; actividades, entregables, fechas límites, planificación, recursos, propósito, etc. El SoW es de gran ayuda sobre todo para el director del proyecto debido a que provee una estructura desde la cual se puede planificar y construir el proyecto.

El SoW debe especificar, como regla general, los siguientes aspectos, cada uno de los cuales debe estar bien detallado para lograr un proyecto exitoso:

1. **Introducción**. ¿Qué se hará en el proyecto? ¿Quiénes están involucrados?
2. **Propósito**. ¿Por qué se está haciendo el proyecto?
3. **Alcance**. ¿Qué hay que hacer? ¿Qué hardware/software es necesario?
4. **Ubicación** del equipo de trabajo del proyecto.
5. **Tareas**. Definir las tareas a partir del alcance con una mayor especificación de que es lo que se necesita hacer para obtener los entregables.
6. **Hitos**. Definir fechas de comienzo y cierre del proyecto esperados. Definir plazos de realización de tareas y su finalización.
7. **Entregables**. Listar los entregables del proyecto y detallarlos.

8. **Programa / Planificación**. Programa de fechas de todos los procesos que deben realizarse durante del proyecto, ya sean las entregas, pruebas, implementación, desarrollo, etc.
9. **Estándares y pruebas**. ¿Hay algún estándar que el proyecto deba seguir?
10. **Resultado**. ¿Cuál es resultado esperado por los interesados?
11. **Requisitos**. ¿Qué otros recursos son necesarios para completar el proyecto?
12. **Pagos**. ¿Cuál es la lista de costos del proyecto?
13. **Cierre**. ¿Qué se aceptará como entregable terminado?

2.3. Fundamentos de MyFEPS

2.3.1. Evaluar calidad

La calidad está definida por el *grado en el que un conjunto de características cumple con los requisitos*. [20]

Como se presentó antes en este trabajo, varios artículos afirman del incremento de la calidad de código y del software al utilizar la metodología dirigida por pruebas. Pero, a su vez, existen artículos que afirman, basados en estudios, que solo agrega calidad visible en proyectos de gran escala.

Es por eso que este trabajo medirá la calidad del software siendo este un proyecto de pequeña escala. Además de utilizar nuevas tecnologías como lo es el desarrollo mobile que hoy en día está en constante cambio y crecimiento. Algunas empresas orientadas al desarrollo mobile han comenzado a utilizar la metodología TDD en sus desarrollos, como Etermax [4].

Franquia será evaluada con el método de evaluación Ágil del *Framework* MyFEPS [6], utilizando el modelo de calidad de producto QSAT [21] [22], para verificar la factibilidad de utilizar la metodología TDD en pequeños proyectos mobile y determinar si el costo de la curva de aprendizaje al principio del proyecto es compensable con los resultados en cuanto a calidad del software.

2.3.2. Framework MyFEPS / QSAT

MyFEPS, según sus siglas; Metodologías y *Framework* para la Evaluación de Productos de Software, es un marco de trabajo que agrupa una seria de procesos y subprocesos para la evaluación de software, guías, plantillas y un modelo de calidad de productos software (QSAT), aunque el método de evaluación MyFEPS permite usar otros modelos [23].

2.3.3.Modelo QSAT

QSAT [21] [22], según sus siglas; Quality Model Sorgen, Angeleri & Titiosky, es un modelo de calidad de productos software, elaborado con el propósito de facilitar la evaluación de productos de software, siendo un modelo de propósito general, para ser aplicado a todo tipo de productos software, y en la mayor cantidad de contextos.

2.3.4.Esquema QSAT

Contexto
Características Fundamentales
Sub – características
... (Puede haber de 0 a varios niveles de Subcaracterísticas)
Atributos En Uso: métricas en Uso
Atributos Externos: métricas Externas
Atributos Internos: métricas Internas [2]

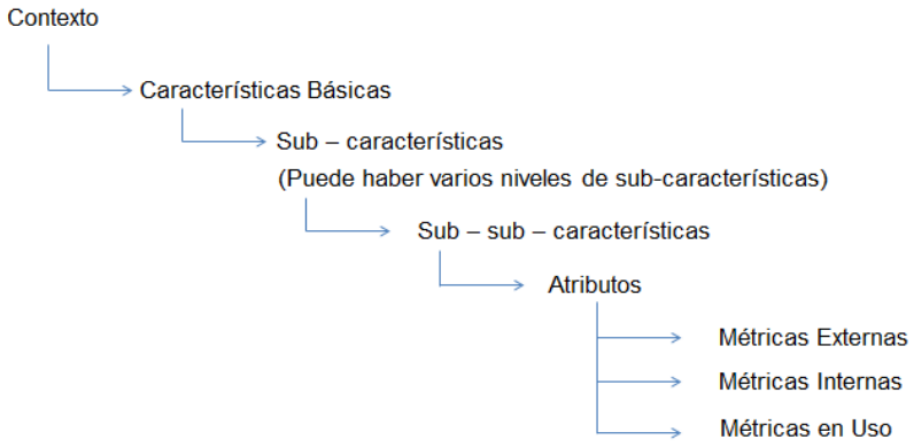

Figura 8: Modelo QSAT [48]

El software puede ser evaluado en diferentes contextos, como por ejemplo las normas ISO/IEC utilizan por un lado el contexto interno/externo y por otro lado el contexto del software en el uso [24] [25]. Luego de analizado del contexto, se analizan las características a evaluar.

Cada característica está compuesta por sub-características y estas en sub-sub-características según lo requiera la evaluación de software.

Este proyecto seguirá las guías provistas por la Universidad de Belgrano para la evaluación según el modelo QSAT.

Las características, sub características y guías serán desarrolladas siguiendo los lineamientos del *Framework* MyFEPS, en el capítulo cuatro de este trabajo, en la sección 4 de desarrollo de la evaluación del software.

2.4. Sistema Mobile

Antes de continuar avanzando es necesario explicar el concepto de sistema mobile, que es parte fundamental de este trabajo.

Cuando hablamos de un sistema mobile nos referimos tanto a la parte del sistema incluida en la aplicación como a la parte web. La aplicación y la web están completamente relacionadas debido a los servicios que provee la web y la aplicación utiliza para su funcionamiento.

Se utiliza una arquitectura de n capas: Las aplicaciones se dividen en cliente y servidor: lo que se descarga en los celulares son aplicaciones de *frontend* que se conectan mediante *apis* (servicios web) a servidores donde se encuentra la lógica de negocio. Así mismo la lógica de negocio se conecta mediante otra interfaz (*EntityFramework*, *ADO*, etc.) al servidor donde se encuentra la base de datos.

A su vez, la aplicación se conecta a servidores de Google (*Google Services*).

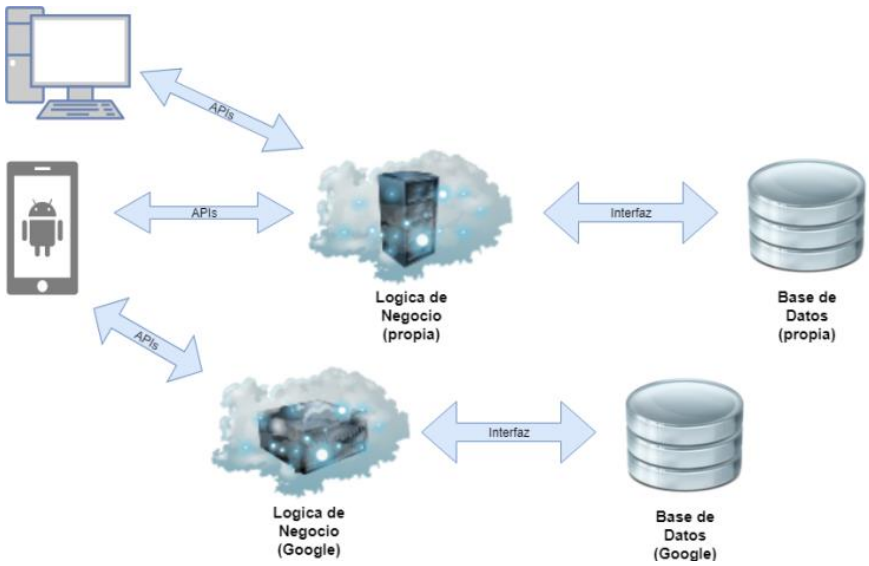

Figura 9: Arquitectura n-capas [Imagen propia]

En la aplicación móvil se utiliza la arquitectura MVP:

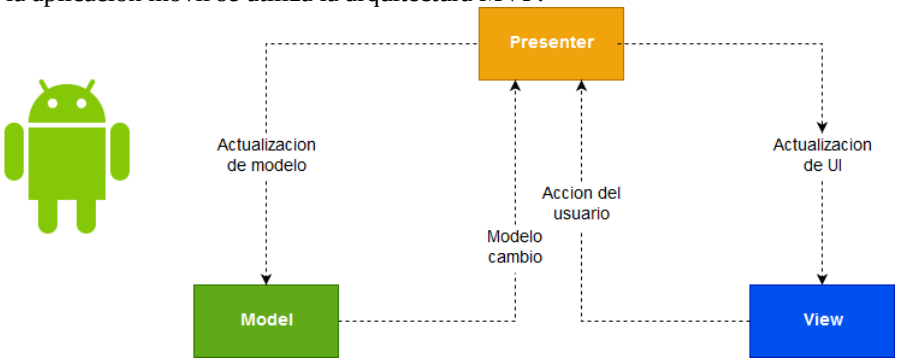

Figura 10: Arquitectura MVP [Imagen propia]

Mientras que en los servicios web tanto del sitio como en los servicios API utilizados por la app, se utiliza la arquitectura MVC:

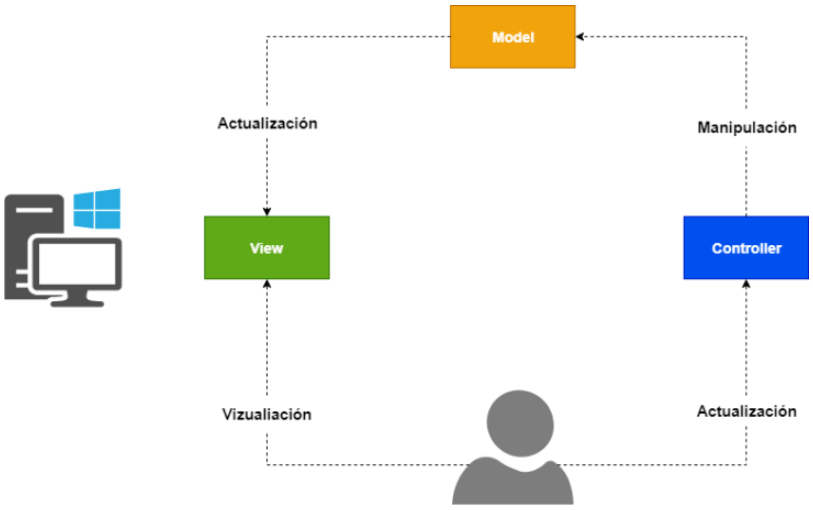

Figura 11: Arquitectura MVC [Imagen propia]

2.5. Metodologías Ágiles

En el desarrollo de este trabajo se aplicarán las metodologías de desarrollo ágiles, es por eso que se realizará un breve acercamiento a ellas y a una en particular, SCRUM.

Las metodologías agiles de desarrollo son aquellas que pueden adaptarse fácilmente ante los cambios del entorno, brindando flexibilidad en cuanto a las respuestas a esos cambios. La metodología ágil tiene un enfoque iterativo frente al ciclo de vida del desarrollo, también prevalecen las personas sobre la documentación [26] [27].

Hay muchas metodologías que se desprenden las ágiles, la que abarcaremos en este trabajo es SCRUM.

2.5.1. Metodologías SCRUM

SCRUM es una metodología ágil con enfoque incremental donde el proyecto de desarrollo es dividido en varias etapas bien definidas en cuanto a las funcionalidades, el tiempo y las responsabilidades.

Se utiliza sobre todo en proyectos donde los requisitos no están bien especificados o pueden ser muy cambiantes [28].

El proyecto se divide en lo que SCRUM llama *sprints* estos son pequeños periodos de tiempo, de no más de dos semanas, donde se realiza todo el ciclo de vida del software sobre un conjunto predefinido de funcionalidades.

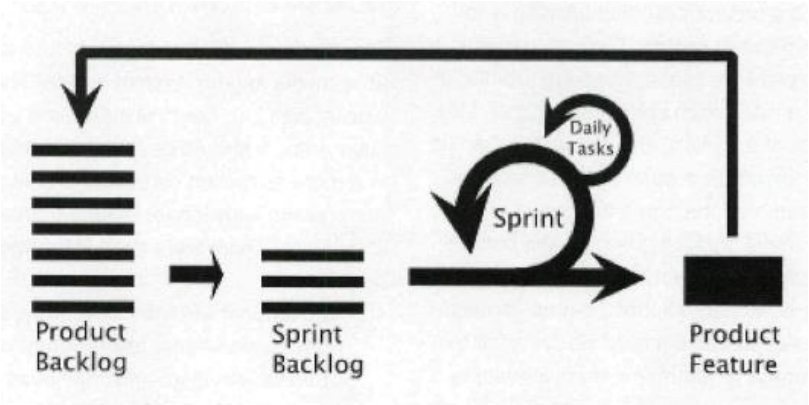

Figura 12: Proceso SCRUM [54]

En la figura 12 se muestra el ciclo de SCRUM. El *product backlog* es el repositorio de todas las funcionalidades del proyecto, del se elegirán las funcionalidades que se harán en el *sprint*. Ese conjunto será llamado *sprint backlog.*

A partir del momento que se obtiene el *sprint backlog* comienza el *sprint*, es un proceso iterativo donde cada día se realizan las preguntas más importantes de SCRUM:

1. **¿Qué se hizo ayer?**
2. **¿Qué hacer hoy?**
3. **¿Hubo problemas?**

Con esas preguntas se mantiene control sobre el desarrollo evitando retrasos innecesarios y promoviendo la cooperación entre las personas involucradas.

Por último, se obtiene un *feedback* del *sprint*, aceptado el desarrollo final y coordinando futuras mejoras en el proceso.

2.6. Fundamentos de *User Experience*

Parte de los requisitos de este proyecto de software es la personalización del mismo en consecuencia a la organización a la que pertenezca cada usuario final. Esto no es más ni menos que la utilización de *User Experiencie* en el diseño.

User Experiencie, conocido también como UX y en español “Experiencia de usuario”, es la interacción del usuario con todos los aspectos del software [29]. Es decir, que comprende y engloba tanto el aspecto visual, el contenido, la funcionalidad y las emociones que puedan derivar del uso.

Entonces UX se utiliza para que la percepción del usuario al usar el sistema sea óptima en todos los factores antes mencionados.

En el artículo *L&D, Meet UX Design* [30] se menciona la importancia que tiene para los usuarios finales comprender rápidamente un sistema y que el mismo los haga sentir cómodos en el uso, por lo tanto, la curva de aprendizaje debe ser rápida e intuitiva, además de brindar un espacio visualmente entendible.

Está técnica nace del campo de la Interacción computador-humano, mejor conocida como *human–computer interaction* (HCI) [31]. Está se ha encargado de

estudiar enfoques en métodos y técnicas para el diseño orientado al usuario. En los últimos años este enfoque ha hecho que UX tome más fuerza con un mayor entendimiento de las emociones humanas y la percepción en el uso interactivo de los sistemas.

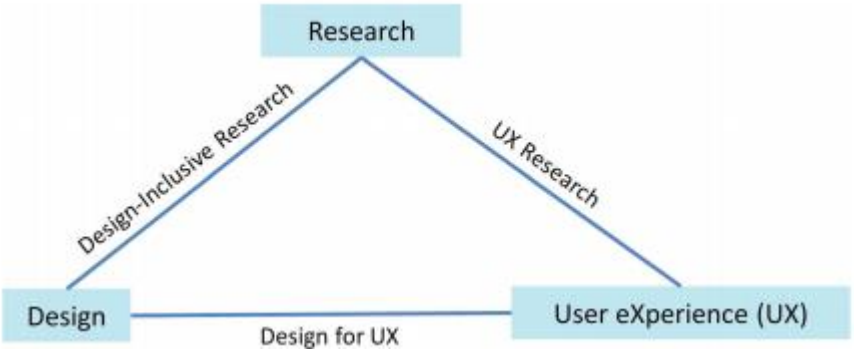

Figura 13: Diseño UX [31]

Es así como este proyecto implementa UX, de manera acotada, incorporando un módulo de personalización del entorno, donde el usuario final en vez de interactuar con un sistema de gestión plano, lo hará con uno en el cual se puede identificar. El sistema permite personalizar el *Look and Feel* del sistema por parte de la organización para que sus usuarios se sientan parte del mismo.

3. Desarrollo del Problema

3.1. Análisis del Problema

La metodología de desarrollo dirigida por las pruebas basa su práctica justamente en el desarrollo del código del software en cuestión. No así del ciclo de vida completo de un sistema donde se abarca desde el análisis hasta la implementación.

Al comienzo de este trabajo se plantearon los objetivos de llevar a la práctica la metodología TDD en el desarrollo de un sistema mobile, estudiando a su vez la conformidad con la norma, ISO/IEC 29110-5.1.1, que es una guía que garantía la calidad durante el desarrollo del software y la gestión del proyecto en equipos de desarrollo muy pequeños. También se planteó el desafío de validar la calidad del software desarrollado utilizando el marco MyFEPS.

Como resultado, al utilizarse como guía para el desarrollo la norma ISO/IEC 29110-5.1.1 se detectó la necesidad de adecuar esta norma al proceso TDD.

3.2. Desarrollo del proceso de aplicación de la metodología TDD

3.2.1. Generalidades

Las primeras aplicaciones de la metodología dirigida por pruebas tienen una curva de aprendizaje elevada. Al no ser la forma en que la aprendimos a programar, es costoso cambiar la manera de pensar / proceder al desarrollar software.

También, las pruebas codificadas no son un hábito en el común de los desarrolladores hoy en día, de hecho, es una práctica bastante ajena a los que trabajamos en esa área.

Es por esta razón que el primer acercamiento a la metodología no es tarea fácil. Es un objetivo de este tranajo presentar el primer acercamiento práctico a TDD aplicado a un sistema mobile.

Como se explicó anteriormente un sistema mobile consta de varias capas, es por esto que la presentación práctica estará dividida en la aplicación de la metodología a la app Android, a los servicios web, y a la web. Los servicios web y la web pertenecen al mismo entorno por lo tanto serán tratado como un todo en una sola sección.

Para cada parte del sistema se utiliza un *framework* de pruebas unitarias distinto, adecuado para cada entorno.

3.2.2. Implementación en PHP (web y servicios)

El sistema web y los servicios de la app fueron desarrollados en PHP, es así que en esta sección se explicará porque se eligió PHPUnit como

framework para realizar las pruebas del *backend* en este proyecto, además se detallará la aplicación del mismo con ejemplos propios del desarrollo de *Franquia*.

3.2.2.1. Introducción a PHPUnit

Para este entorno se eligió PHPUnit como *framework* de pruebas unitarias debido a la popularidad de esta herramienta y la cantidad de documentación disponible [32].

Si se busca con las palabras "pruebas unitarias" + PHP en Google, es difícil encontrar un artículo que no esté referido a PHPUnit. Este es, sin dudas, el *framework* de pruebas unitarias más utilizado por los desarrolladores de PHP.

Este *framework* ha sido fácil de instalar y se pudieron separar las pruebas del código principal, de esta manera no se agregan archivos de *testing* al código fuente principal, manteniendo la estructura limpia y organizada y sin aumentar el tamaño de la solución final.

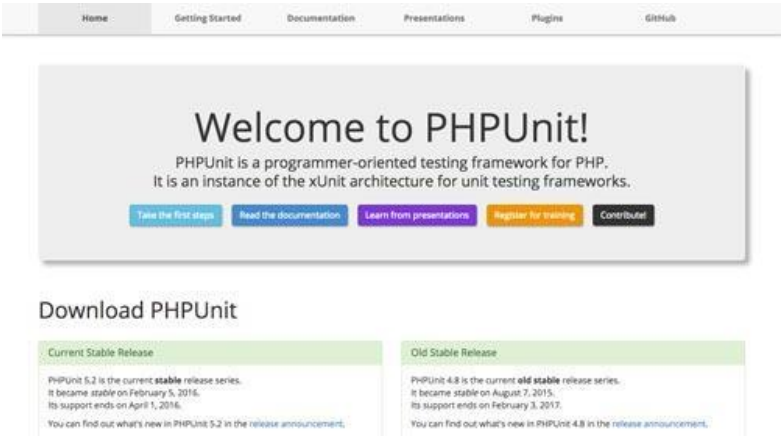

Figura 14: Sitio PHP [32]

PHPUnit se puede descargar de manera gratuita y en su página web se puede encontrar toda la documentación necesaria para empezar a utilizarla [33].

3.2.2.2. Aplicación de PHPUnit

La instalación de PHPUnit es muy sencilla, a través de composer.

composer require --dev phpunit/phpunit ^5.7

Composer es un administrator de dependencias que permite la descarga e instalación de paquetes para PHP en nuestras computadoras [34].

Para utilizar PHPUnit creamos una clase estática que extiende la clase TestCase del *framework*:

```
use PHPUnit\Framework\TestCase

final class FranquiaTest extends TestCase
{
// casos de prueba
}
```

Figura 15: Ejemplo de PHPUnit (1)

Dentro de esta clase se programan los casos de prueba de PHPUnit. El nombre de la función debe comenzar con el prefijo "test" y el resto del nombre identifica el objetivo de la prueba. Si este nombre se escribe en CamelCase, cuando se muestre la ejecución de la prueba, las palabras aparecerán separadas para una mejor legibilidad: testLoginWithGoodCredentials [35] [36].

Un caso de prueba normalmente se escribe intentando realizar la acción a probar. Y luego se chequean los resultados esperados mediante las funciones *assert* provistas por PHPUnit. Con estas funciones podemos probar, cosas como que una variable sea o no sea nula, que tenga un valor determinado, que sea de un tipo determinado, y otras muchas condiciones.

Por ejemplo, para probar el login al sistema con credenciales correctas, intentamos realizar el logueo con un usuario y contraseña conocidos:

```
$user = Franquia::login("admin", "admin");
```

Figura 16: Ejemplo de PHPUnit (2)

Sabiendo que esta función de login debe devolver un objeto Usuario, con los datos del usuario logueado, realizamos varios chequeos (con funciones assert):

El objeto devuelto no puede ser nulo:

```
$this->assertNotNull($user);
```

Figura 17: Ejemplo de PHPUnit (3)

El objeto devuelto debe ser una instancia de la clase Usuario:

```
$this->assertInstanceOf('Usuario', $user);
```

Figura 18: Ejemplo de PHPUnit (4)

La propiedad 'id', del objeto Usuario devuelto, debe ser igual a "admin":

```
$this->assertEquals($user->get('id'), "admin");
```

Figura 19: Ejemplo de PHPUnit (5)

La prueba completa quedaría como sigue:

```
    public function
testLoginWithGoodCredentials()
    {
        $user = Franquia::login("admin",
"admin");
        $this->assertNotNull($user);
        $this->assertInstanceOf('Usuario', $user);
        $this->assertEquals($user->get('id'),
"admin");
    }
```

Figura 20: Ejemplo de PHPUnit (6)

Al ejecutar esta prueba, si todas las condiciones chequeadas son exitosas, la prueba aparecerá marcada con una [x], mostrando el título de la prueba con las palabras del título CamelCase separadas por espacios:

Figura 21: PHPUnit en consola

Una prueba complementaria a la anterior, sería el intento de logueo con credenciales erróneas, se intenta loguearse con una contraseña incorrecta, y se espera que el resultado sea un objeto nulo, de modo que no se permita el logueo del usuario con esas credenciales:

```
    public function
testLoginWithWrongCredentials()
    {
        $user = Franquia::login("admin", "xyz");
        $this->assertNull($user);
    }
```

Figura 22: Prueba complementaria en PHPUnit

Es posible ejecutar pruebas que dependan de pruebas anteriores. Para esto, una prueba puede devolver un objeto que será pasado como parámetro a la prueba dependiente, la cual debe especificar de cual prueba depende:

```
    public function testGetPedidos() {
        … // devuelve un idPedido que podrá
utilizarse en      otras pruebas
        return $pedido->get('idPedido');
    }
    /**
     * @depends testGetPedidos
     */
    function testGetPedido($idPedido) {
        … // recibe un idPedido que se obtuvo en la
prueba testGetPedidos
    }
```

Figura 23: Dependencia de pruebas en PHPUnit

Veamos una prueba un poco más compleja, la que chequea la obtención de los productos que puede pedir un usuario franquiciado.

Codificamos las pruebas que establecen los resultados esperados:

La función getProductos debe devolver un objeto no nulo, que sea del tipo arreglo, que este arreglo contenga al menos un elemento, que a su vez este elemento no sea nulo y que sea una instancia de la clase Producto:

```
    public function testGetProductos()
    {
        $productos =
Franquia::getProductos("fdo1");
        $this->assertNotNull($productos);
        $this-
>assertInternalType('array',$productos);
        $this->assertGreaterThan(0,
count($productos));
        $producto = $productos[0];
        $this->assertNotNull($producto);
        $this->assertInstanceOf('Producto',
$producto);
    }
```

Figura 24: Función getProductos en PHPUnit

De este mismo modo creamos el conjunto de funciones de prueba para los servicios de la aplicación y para las funciones del sitio web.

Se continúa la codificación de las pruebas y de las funciones chequeadas hasta que no haya fallos en la ejecución de los casos de prueba.

A continuación, mostramos el resultado de dicha ejecución:

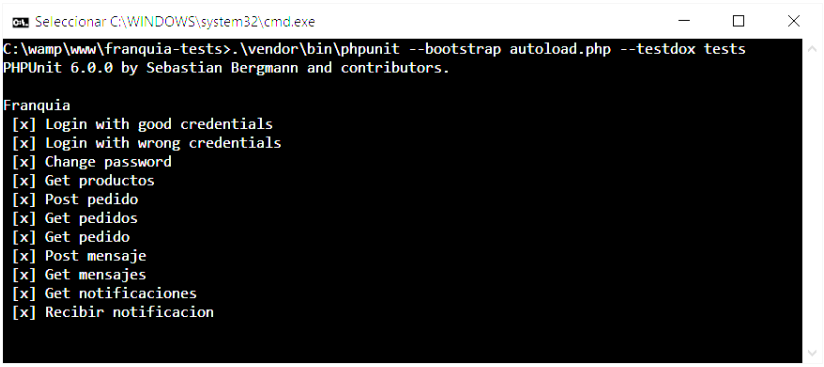

Figura 25: Muestra de ejecución de casos de prueba de PHPUnit [Imagen propia]

3.2.3. Implementación en Android (app)

En esta sección se presentarán la gran variedad de *frameworks* disponibles para realizar pruebas en Android, todos ellos no se pueden utilizar al mismo tiempo porque causaría problemas de compatibilidades. Aquí se explicará cual es la mejor combinación a utilizar para alcanzar los objetivos de TDD en el proyecto.

3.2.3.1. Tipos de pruebas

Al utilizar Android Studio (JAVA) los proyectos se crean por default con dos entornos de *testing* [37] [38] [39], estos son:

- Pruebas instrumentadas
- Pruebas de unidad local

3.2.3.1.1. Pruebas instrumentadas

Ubicación: *module-name/src/androidTest/java/.*

Son pruebas que se ejecutan en un dispositivo o emulador de hardware, tienen acceso a las API *Instrumentation* [40], y permiten acceder a información como el *Context* de la app. Se utilizan cuando se escriben pruebas de IU funcionales e integradoras para automatizar la interacción de usuarios, o cuando las pruebas tengan dependencias de Android que los objetos ficticios que no puedan contemplar.

El *Context* es una instancia de alguna de las clases visuales de Android (actividades) necesaria para ejecutar funciones desde fuera de la actividad [41].

Debido a que las pruebas instrumentadas se compilan en

un APK (por separado del APK de la app), deben tener su propio archivo AndroidManifest.xml; pero el *Gradle* automáticamente genera este archivo durante la compilación para que no se vea en el conjunto de fuentes del proyecto.

Se pueden agregar un *manifest* propio de ser necesario a fin de especificar un valor diferente para `minSdkVersion` o registrar receptores de ejecución solo para las pruebas. Cuando se compila, el *Gradle* combina varios archivos de *manifest* en uno único.

Las pruebas instrumentadas pueden correr con diferentes *frameworks* de *testing* como pueden ser AndroidJUnit4 o RobolectricTestRunner.

Android incluye su propio *framework* de pruebas, pero no hay mucha documentación al respecto y su uso no está muy extendido. Existen muchas bibliotecas que realizan las pruebas sobre este *framework*, por lo tanto, es complicado dar con la información correcta para su utilización. La curva de aprendizaje es muy elevada.

Sin embargo, existen tutoriales de los cuales se puede entender el funcionamiento de las pruebas en esta plataforma, pero estos no son oficiales de Google [42] [43] [44]. En este proyecto se utilizó una biblioteca brindada por un usuario en la plataforma GitHub para el testeo de la programación reactiva utilizada [17].

AndroidJUnit4 es el *framework* más utilizado debido a que es una extensión de JUnit de Java, en cambio RobolectricTestRunner es específico para Android y solucionar problemas de testeo de UI.

Las recomendaciones son utilizar una combinación de JUnit con Mockito y Expresso (*Android support*) [46] para testear de manera integral la app debido a que la utilización de Robolectric implica también la implementación de las librerías androidx que generan conflicto con las librerías com.android.support.test utilizadas por default.

Debido a su estabilidad, en este trabajo utilizamos la combinación de las bibliotecas Mockito y Expresso para la implementación de pruebas en Android.

Mockito provee, además, la biblioteca Dexmaker, que se utiliza en el *testing* de RxJava/RxAndroid, Se utiliza una librería diferenciada debido al carácter asincrónico de la programación reactiva. Dexmaker maneja la parte asincrónica sin que tengamos que preocuparnos por ello.

Por lo tanto, las herramientas a utilizar en el *testing* en la aplicación son:

- JUnit:

- 'junit:junit:4.12'
- Mockito:
- 'org.mockito:mockito-core:1.10.19'
- 'com.google.dexmaker:dexmaker:1.2'
- 'com.google.dexmaker:dexmaker-mockito:1.2'
- Android support:
- 'com.android.support.test:runner:1.0.2'
- 'com.android.support.test:rules:1.0.2'
- 'com.android.support.test.espresso:espresso-core:3.0.2'

Ejemplo:

```
@RunWith(AndroidJUnit4.class)
public class ExampleInstrumentedTest {
  @Test
  public void useAppContext() {
    Context appContext =
      InstrumentationRegistry.getTargetContext();
    assertEquals( "com.tesina.ub.Franquiaapp",
                appContext.getPackageName());
  }
}
```

Figura 26: Función instrumentada con JUnit

3.2.3.1.2. Pruebas de unidad local

Ubicación: *module-name/src/test/java/.*

Son pruebas que se ejecutan en la máquina virtual Java (JVM) local de la computadora donde corre la aplicación.

Se utilizan cuando se quiere minimizar el tiempo de ejecución de ellas y se puedan simular las dependencias del marco de Android o no se necesiten.

En el tiempo de ejecución, estas pruebas se ejecutan en una versión modificad de android.jar en la que se quitan todos los modificadores finales. Esto te permite usar bibliotecas de simulación populares, como Mockito.

Ejemplo:

```
public class ExampleUnitTest {
    @Test
    public void addition_isCorrect() {
        assertEquals(4, 2 + 2);
    }
}
```

Figura 27: Ejemplo de prueba de unidad local en JUnit

3.2.4. Conclusiones acerca de la implementación

El *framework* de pruebas depende del lenguaje utilizado. En nuestro caso utilizamos JUnit para Android y PHPUnit para PHP.

Independientemente del *framework* de que se trate, es necesario simular las condiciones (contexto) en que se ejecutará la función a ser probada en el sistema. Se deben crear todas las premisas que la función necesite para que la prueba se desarrolle en las condiciones reales de ejecución.

En el caso de Android / JUnit, debido a que una aplicación móvil es en un 95% visual y de interacción con el usuario, la mayoría de las pruebas realizadas necesitan el Context.
Es por esta razón que la mayoría de las pruebas en Android se realizan en el entorno de pruebas instrumentadas, en el cual se puede simular el Context (creando un *mock context* o *instrument context*).

Las pruebas de unidad local solo se utilizarán para aquellas funciones que no necesiten la instancia Context.

En combinación con jUnit, se utilizaron las bibliotecas Mockito y Expresso para la implementación de pruebas en Android.

Durante el proceso de desarrollo de este proyecto, encontramos que las pruebas en Android parecen no tener mucho valor, dado que una app tiene más interfaces de usuario que procesamiento de información. Es complicado y de relativamente poca utilidad realizar pruebas sobre las interfaces visuales.

El proceso de *debugging*, búsqueda y corrección de errores de jUnit en Android es muy engorroso. El debugger no hace una buena trazabilidad de los errores en las pruebas, lo cual no contribuye a la agilidad del proceso.

Todo lo contrario, pasa en PHP, donde utilizamos el *framework* PHPUnit.

El uso de PHPUnit es bastante intuitivo. La instalación y configuración están bien documentadas y es un proceso relativamente sencillo.
Las pruebas son legibles y contemplan una amplia gama de posibilidades.

A diferencia de Android, se pueden manejar en un proyecto separado, lo cual permite mantener la solución final limpia sin incluir todo el *framework* de pruebas.

Lo más importante es decidir qué pruebas realizar, para no probar trivialidades ni hacerlas demasiado complejas.

Hemos encontrado que la realización de las pruebas influye en la estructura del proyecto final pues se deben aislar y hacer públicas las funciones a ser probadas, lo cual no sería necesario en otras condiciones.

En nuestro caso, creamos una clase pública estática con las funciones que deseamos incluir en las pruebas.

Por la propia naturaleza del contenido de nuestro proyecto PHP, que incluye el sitio web y el *backend* de la aplicación, pudimos realizar pruebas que se centraran en el procesamiento de datos más que en aspectos visuales y de interfaz.

Estas pruebas desarrolladas en PHP pudieron ser utilizadas efectivamente en la creación de las funciones del *backend* y en ellas pudimos comprobar la utilidad y potencia del desarrollo basado en pruebas (TDD).

En resumen, comprobamos que el desarrollo utilizando TDD tiene mucha más utilidad para el código de *backend* que para el *frontend*, salvo ciertos casos en que se haga algún procesamiento de información en la interfaz.

3.3. Desarrollo del proceso de adecuación de la norma ISO/IEC 29110-5.1.1 a la metodología TDD

3.3.1. Generalidades

La norma ISO/IEC 29110-5.1.1 define dos grandes procesos, estos son: **Gestión del Proyecto** desde ahora PM e **Implementación de Software** desde ahora SI. [47] [48]

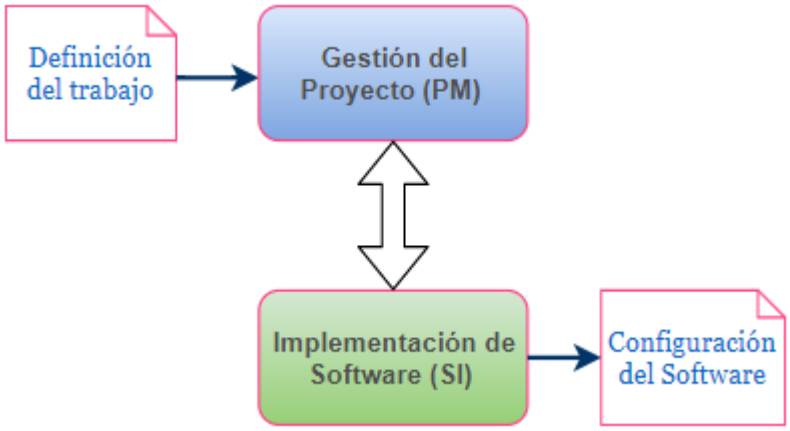

Figura 28: Flujo de la norma ISO/IEC 29110-5.1.1 [18]

El proceso PM lleva a cabo el seguimiento de las tareas de gestión e implementación del software para cumplir de manera eficiente los objetivos del proyecto con los tiempos y recursos esperados. Es decir, se encarga de la planificación, seguimiento y cierres del proyecto.

Por otro lado, el proceso SI se encarga del desarrollo del software en sí mismo. Es decir, del análisis, diseño, construcción, integración y pruebas del software resultante siguiendo los objetivos pautados en el PM.

Estos dos procesos irán en la mayor parte del tiempo interrelacionados alimentándose uno del otro. Mientras PM se encarga de todo el referente a la gestión el SI se encarga de todo el desarrollo del software.

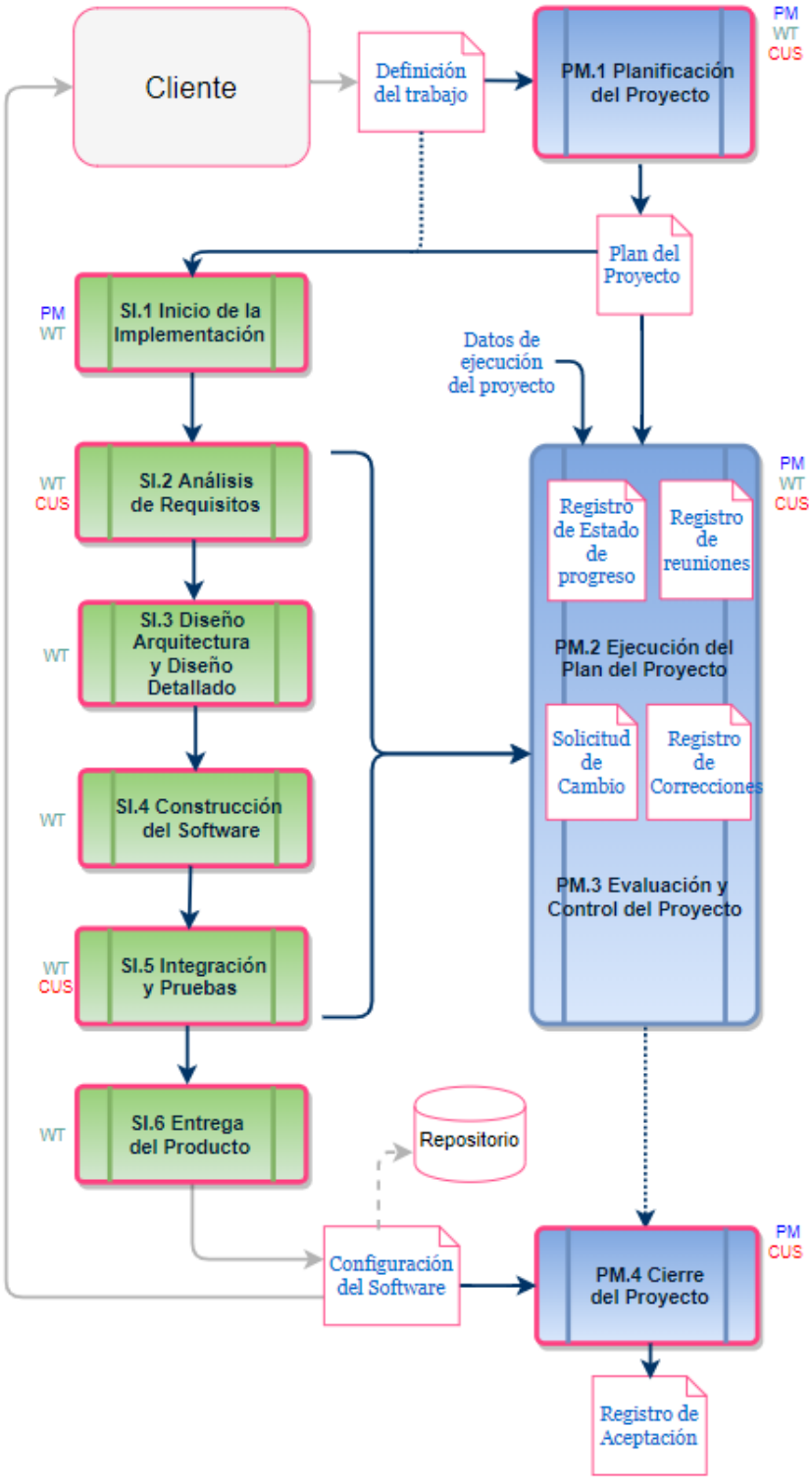

Figura 29: Relación PM y SI [18]

3.3.1.1. Gestión del Proyecto

El PM cuenta con cuatro sub procesos dentro de la gestión los que a su vez contienen tareas a realizar [13] [18].

Los subprocesos realizados para el perfil de entrada son:

1. PM1. Planificación del Proyecto
2. PM2. Ejecución del Plan del Proyecto
3. PM3. Evaluación y Control del Proyecto
4. PM4. Cierre del Proyecto

La PLANIFICACIÓN DEL PROYECTO1 es alimentada por la definición del trabajo anteriormente explicada. Es la única actividad dentro del proceso que se realiza una única vez, el resultado de las tareas es el Plan del Proyecto.

El PLAN DEL PROYECTO detalla la planificación necesaria para marcar las pautas de gestión del proyecto y como se ejecutarán las actividades para asegurar la finalización exitosa cumpliendo con los estándares de recursos, calidad y tiempo establecidos con el cliente.

En este trabajo el PLAN DE PROYECTO consta de un solo documento que establece la descripción general del proyecto y el ALCANCE como aspectos generales.

Por otro lado, se establece el modelo ciclo de vida del proyecto, que en este proyecto se eligió **Modelo de Ciclo de Vida Iterativo e Incremental** [26]**.**

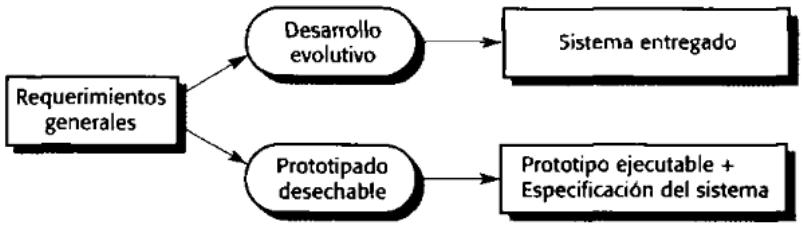

Figura 30: Ciclo de Vida Iterativo e Incremental [26]

Esto quiere decir que se generarán iteraciones con entregables funcionales en cada una de ellas, incrementando funcionalidades en cada iteración.

La utilización de este modelo en un desarrollo con TDD genera una mejora debido a que se mantiene el código limpio sin repeticiones, el proyecto se divide en entregables pequeños, por lo que se mantiene una visión clara de lo que debe hacerse. TDD

1 Todos los artefactos de la norma 25110-5.1.1 serán mostrados en mayúsculas para su fácil identificación

aporta la misma visión, pero a nivel código, evitando duplicaciones y re trabajos innecesarios.

Además, se establecen las limitaciones del proyecto, la gestión de control de repositorio para el versionado del sistema; en este caso se utilizó GitLab [49].

También se establece el método de desarrollo, para este sistema se eligió la técnica WBS (*Work Breakdown Structure* - Descomposición de Paquetes de Trabajo) [50] [51] que organiza el trabajo en un formato jerárquico donde el proyecto se descompone en procesos y actividades con una duración estimada. Se utilizará este método para la división de las actividades y consecuente agrupación de ellas en *sprints* utilizando una metodología basada en SCRUM para estos entregables.

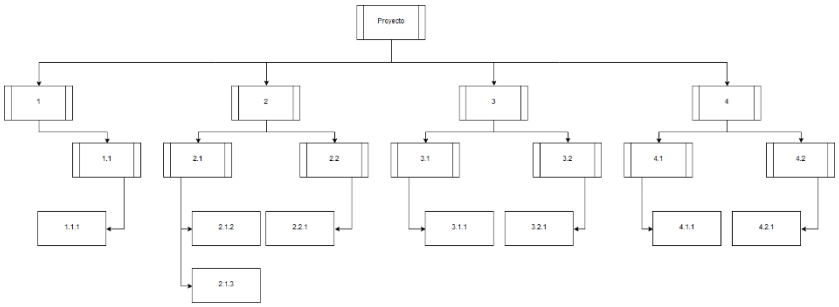

Figura 31: WBS [Imagen propia]

Se establecen los recursos humanos y de tiempo utilizando un GANTT (solo utilizado para especificar los sprints), y se agrupan las actividades en *sprints* con los alcances de cada uno de ellos y su finalización en el tiempo. El sistema que abarca este trabajo fue dividido en seis *sprints*.

Por último, se especifican los riesgos del proyecto y se ponderan.

Las actividades de EJECUCIÓN DEL PLAN DEL PROYECTO, EVALUACIÓN y CONTROL DEL PROYECTO y CIERRE DEL PROYECTO se realizan a lo largo de todo el proyecto, de cada *sprint* a escala de cada alcance de los *sprints*.

3.3.1.2. Implementación del Software

El SI lleva a cabo el desarrollo del software en su ciclo de vida, con las actividades de ANÁLISIS, DISEÑO, DESARROLLO, INTEGRACIÓN y PRUEBAS del sistema.

Cuenta con una serie de subprocesos que completan el ciclo de vida del software, se apoya en los documentos de DEFINICIÓN DE TRABAJO y PLAN DE PROYECTO.

Debido al modelo especificado en el PLAN DE PROYECTO y de la partición del sistema en seis *sprints*, el SI se realizará completamente para cada *sprint*.

Los subprocesos son:

1. SI1. Inicio de la Implementación de Software
2. SI2. Análisis de Requisitos de Software
3. SI3. Diseño de Arquitectura y Diseño Detallado del Software
4. SI4. Construcción del Software
5. SI5. Integración y Pruebas del Software
6. SI6. Entrega del Producto

La actividad de Inicio de la Implementación del Software asegura que el Equipo de Trabajo se comprometa con el PLAN DEL PROYECTO establecido en la actividad de PLANIFICACIÓN DE PROYECTOS.

Para realizar la tarea de entender el proyecto en este caso se utilizó la herramienta de mapa conceptual [52], según la recomendación por la Dra. Graciela Hadad [53]. Los mapas conceptuales ayudan a entender la construcción del conocimiento de manera esquemática. Se puede observar el problema a un grado de abstracción donde se visualizan todos los actores y las acciones involucradas en el problema.

De esta manera nos acercamos al problema y se pudo verificar junto al interesado el correcto entendimiento del problema y su acercamiento a la solución.

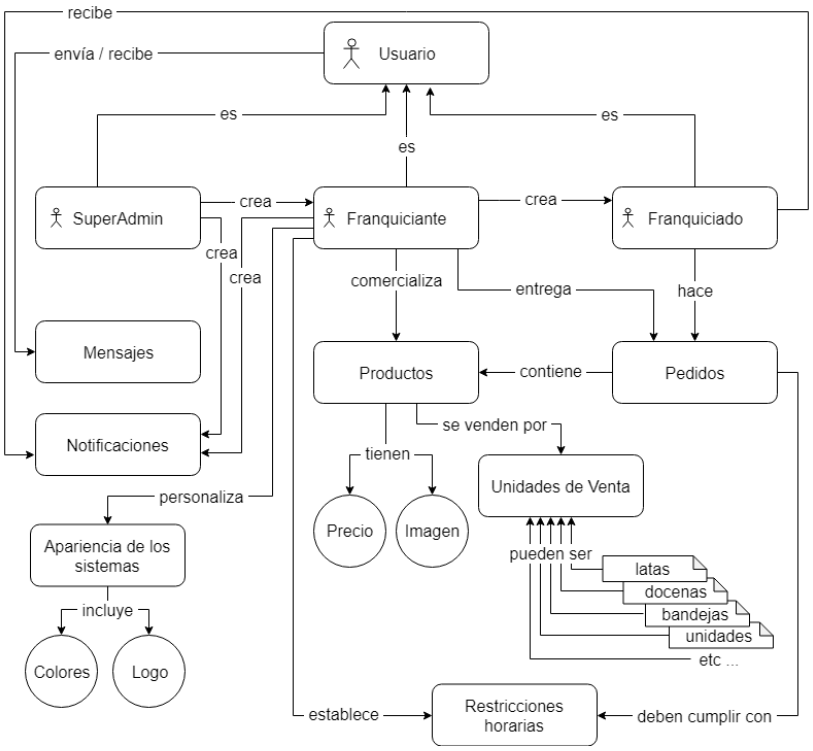

Figura 32: Mapa conceptual [Imagen propia]

Ejemplo de la organización de un *sprint* con los subprocesos de la implementación del software.

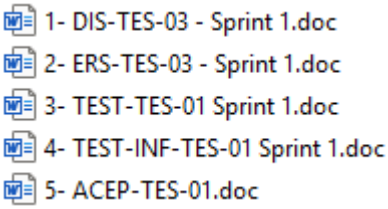

Figura 33: Sprint Uno [Imagen propia]

3.3.1.3. Utilización de Plantillas

Todas las plantillas utilizadas en este proyecto para la implementación de la norma ISO/IEC 29110-5.1.1 fueron proporcionadas por la web paga UpTo25.

3.3.2. Adecuación de la norma a la metodología TDD

Ya explicada la implementación de la norma ISO/IEC 29110-5.1.1 se puede observar que sigue un ciclo de vida tradicional: análisis, diseño, desarrollo, integración y pruebas.

Dado que el perfil AGILE2 de esta norma está en estudio incipiente como Nueva Propuesta de estudio en la ISO, se ha trabajado con la versión disponible que se asemeja más el modelo en Cascada, pero, por las modalidades de TDD, se le ha dado un foco Iterativo que ha requerido un ajuste a la norma utilizada como referencia.

Debido a la utilización de la metodología TDD se reestructuró la norma para adecuarla a las necesidades del proyecto. Primero que nada, se debe mencionar los pasos de la metodología TDD respecto al desarrollo:

1. **Elegir un requisito:** Se elige un requisito a realizar de una lista previamente confeccionada.
2. **Escribir una prueba:** Se codifica la prueba para ese requisito. De esta manera el desarrollador debe tomar el lugar del usuario desde el punto de vista de las interfaces.
3. **Verificar que la prueba falla:** Dado que el requisito no ha sido programado la prueba debe fallar indudablemente.
4. **Escribir la implementación:** Se codifica el requisito de acuerdo a la prueba, es fundamental desarrollar lo mínimo indispensable para que supere la prueba, de esta manera se mantiene el código limpio.
5. **Ejecutar las pruebas automatizadas:** Corres las pruebas.
6. **Eliminación de duplicación:** La refactorización se realiza para evitar funciones repetidas, los cambios se deben realizar de a poco para que las pruebas sigan funcionando.
7. **Actualización de la lista de requisitos:** Se actualiza la lista de requisitos, esto es eliminado el requisito programado y otros requisitos que hayan sido encontrados mientras se realizaba la iteración.

Pero la metodología TDD siempre habla del desarrollo y no de la documentación alrededor de ella, es por eso que la norma ISO/IEC 29110-5.1.1 se ha adecuado de la siguiente manera.

El proceso SI ha sido el afectado en la modificación de la norma. Los subprocesos se realizaron en el siguiente orden:

1. ANÁLISIS DE REQUISITOS
2. DISEÑO DEL SOFTWARE
3. CASOS Y PROCEDIMIENTOS DE PRUEBAS
4. CONSTRUCCIÓN DEL SOFTWARE
5. INFORME DE PRUEBAS
6. ACEPTACIÓN DEL *SPRINT*

2 ISO/IEC NP 29110-4-4 [ISO/IEC NP 29110-4-4] Systems and software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 4-4: Agile software development — Profile specifications — Generic profile

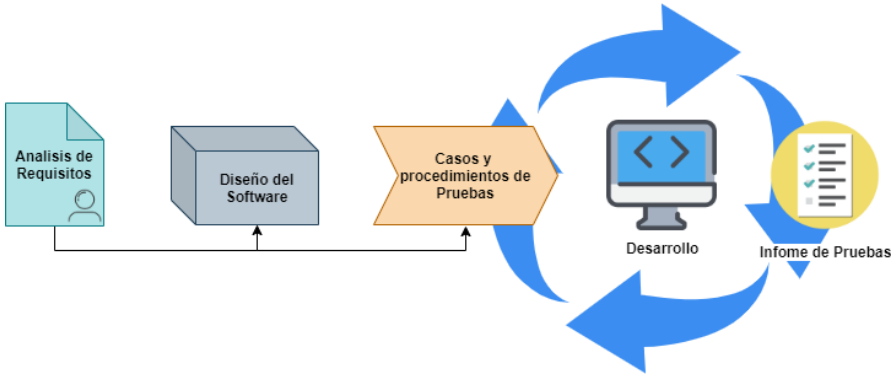

Figura 34: Modificación de la norma [Imagen propia]

De esta manera se respeta el ciclo de vida TDD, luego de detallar el DISEÑO de los requisitos del *sprint*, se realiza el informe de las pruebas basados en los casos de uso del *sprint*.

Al culminar el desarrollo se realizan las pruebas y son plasmadas en el INFORME DE PRUEBAS. De esta manera se genera un proceso iterativo debido a que las pruebas al ser realizadas se encuentran los errores y se vuelve al desarrollo y a realizar las pruebas nuevamente.

Las PRUEBAS se basan en los casos y procedimientos de pruebas, estos dos documentos van de la mano junto con el desarrollo.

Al ser aceptado el documento de INFORME DE PRUEBAS con éxito, se realiza la ACEPTACIÓN de *sprint* en curso con el cliente.

3.3.3. Conclusiones del proceso de adecuación de la norma ISO/IEC 29110-5.1.1

A lo largo del desarrollo del proyecto se llevó a cabo un sistema de gestión adecuando a la norma a la metodología dirigida por pruebas. El uso de la norma organizo y dirigió el proyecto siguiendo la metodología iterativa, de esta manera se obtuvo un ciclo de seis iteraciones para llevar a la culminación exitosa del proceso. Se obtuvieron todos los componentes necesarios para completar la documentación de todo el ciclo de vida correctamente.

Se cumplieron de manera exitosa los tiempos establecidos por el cliente para los seis entregables, donde se obtuvieron seis versiones funcionales para el cliente, donde gracias a la norma se pudo hacer la trazabilidad de pedidos de cambios y los documentos correspondientes.

Como conclusión se obtuvo un modelo iterativo donde se obtiene del Plan del Proyecto los elementos para generar el documento del Análisis de Requisitos para el *sprint* en cuestión, a partir de ese documento se realiza el documento de Diseño para los requisitos del *sprint*.

A partir de los casos de uso obtenidos en diseño se deben realizar los casos y procedimientos de pruebas, este es el momento en el que la norma ha sido reestructurada en conveniencia del uso de TDD, se deben realizar

los casos de pruebas poniéndose en el lugar de los usuarios y que están esperando de la funcionalidad.

A partir de los casos de pruebas y en función de cumplirlos se desarrolla la construcción del software donde de la misma manera a nivel codificación se utiliza la metodología dirigida por pruebas al nivel de programación, como fue expuesto anteriormente en la sección correspondiente.

Luego al realizar las pruebas se obtiene el informe de pruebas y si estas no han sido pasadas con éxito se vuelve al ciclo, cuando son aprobadas se genera el documento de aceptación donde se involucra al cliente y en este momento pueden aparecer solicitudes de cambio para próximos *sprints* o el producto funcional es aprobado.

Consiguientemente se pasa al próximo *sprint* re comenzando el ciclo iterativo de la norma ISO/IEC 29110 basado en la metodología TDD.

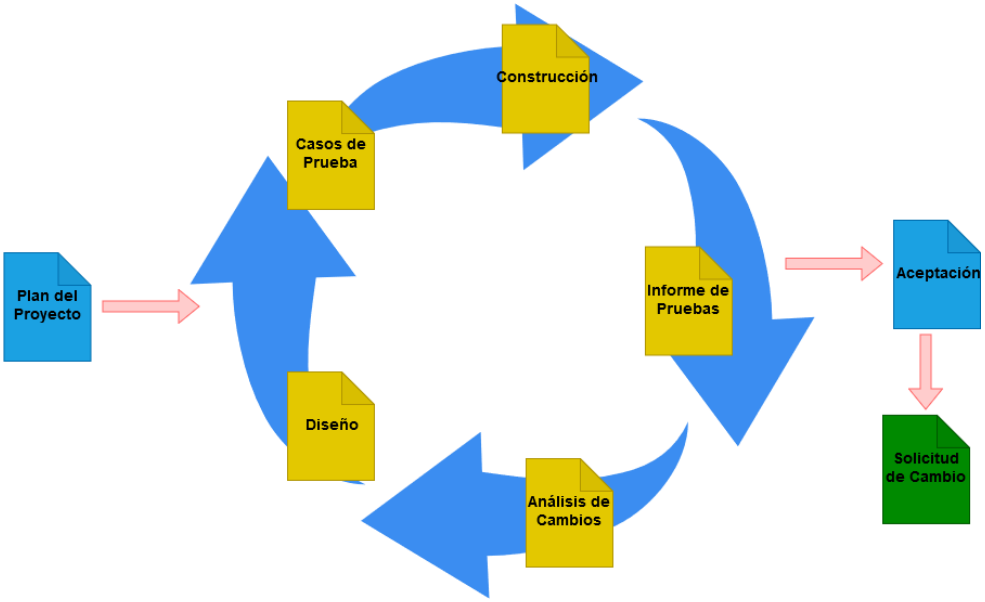

Figura 35: Ciclo iterativo de la norma basada en TDD [Imagen propia]

Como conclusión los subprocesos SI2, SI3, SI4 y SI5 (SI2. Análisis de Requisitos de Software - SI3. Diseño de Arquitectura y Diseño Detallado del Software - SI4.Construcción del Software - SI5.Integración y Pruebas del Software) del proceso Implementación del Software se realizan de manera iterativa para cada *sprint* y no en cascada como indica la norma. Además, los procesos SI4 y SI5 se relacionarán de tal manera que sus actividades quedan intercaladas en la práctica.

El subproceso PM4. Cierre del Proyecto se confecciona por cada iteración del proyecto, de esta manera se obtienen un documento de aceptación por cada reléase realizado.

3.4. Proceso de re trabajo en el desarrollo del sistema *Franquia*

3.4.1. Re trabajo a lo largo del desarrollo

Para comprobar que la utilización de la metodología de desarrollo dirigida por pruebas evita el re trabajo y contribuye a mantener el código limpio, es decir, sin duplicaciones, a medida que se desarrollaron los sistemas mobile y web se llevó un registro en Excel para mantener información sobre el tema.

Como se mencionó antes, el sistema fue dividido en seis *sprints* funcionales y cada uno de ellos en módulos a desarrollar. Toda la información pertinente a la división y los módulos se encuentra en el anexo de este trabajo.

Por lo tanto, el documento Excel fue divido por *sprints* y a su vez por módulos, especificando la fecha en que se trabajó en el módulo y si requirió re trabajo o no. Esto es para comprobar la eficacia de TDD en temas de optimización de tiempo y robustez del código obtenido.

También se especifica la causa del re trabajo si lo hubiera. Estas causas pueden ser en primer lugar diferentes problemas funcionales, como errores en el flujo de negocio, de guardado, de obtención de datos, errores que causen la ruptura del sistema inesperadamente, cálculos erróneos, etc. Por otro lado, distinguimos errores visuales, estos no afectan al usuario en su experiencia con el sistema ni causan el cierre inesperado. Son, más bien, errores de adaptabilidad a la pantalla o de cómo podrían mostrarse mejor las pantallas al usuario.

Se entiende que TDD busca evitar los errores funcionales y brindar un sistema robusto frente a ellos, por lo tanto, serán estos los considerados como críticos mientras los visuales serán considerados triviales.

A continuación, se presentarán los resultados para cada *sprint* obtenido a través de todo el desarrollo del sistema.

Sprint	**Modulo**	**Inicio de *Sprint***	**Cierre *Sprint***	**Inicio de Trabajo**	**Fin de Trabajo**	**Re trabajo**	**Causa**
Sprint 1	Recuperar contraseña	25-abr	15-may	10-may	10-may	No	-
Sprint 1	Cambiar contraseña	25-abr	15-may	10-may	10-may	No	-
Sprint 1	Cambiar contraseña generada	25-abr	15-may	10-may	11-may	No	-
Sprint 1	Interfaces de usuario	25-abr	15-may	10-may	11-may	No	-
Sprint 1	Acceso al sistema	25-abr	15-may	10-may	12-may	No	-
Sprint 1	Interfaces de errores	25-abr	15-may	13-may	15-may	No	-
Sprint 1	Estilo	25-abr	15-may	14-may	14-may	No	-
Sprint 1	Cerrar el sistema	25-abr	15-may	13-may	13-may	No	-
Sprint	**Modulo**	**Inicio de *Sprint***	**Cierre *Sprint***	**Inicio de Trabajo**	**Fin de Trabajo**	**Re trabajo**	**Causa**
Sprint 2	Listado de productos	16-may	6-jun	1-jun	2-jun	No	-
Sprint 2	Selección de productos	16-may	6-jun	1-jun	2-jun	No	-
Sprint 2	Valorización de pedido	16-may	6-jun	3-jun	3-jun	No	-
Sprint 2	Listado de pedidos pendientes	16-may	6-jun	3-jun	3-jun	No	-
Sprint 2	Listado de estados de pedidos	16-may	6-jun	3-jun	3-jun	No	-
Sprint 2	Listado de pedidos entregados	16-may	6-jun	3-jun	3-jun	No	-
Sprint 2	Creación de pedidos	16-may	6-jun	3-jun	4-jun	No	-
Sprint 2	Verificación de restricciones	16-may	6-jun	5-jun	5-jun	No	-
Sprint 2	Ver detalle de pedidos	16-may	6-jun	3-jun	6-jun	No	-
Sprint	**Modulo**	**Inicio de *Sprint***	**Cierre *Sprint***	**Inicio de Trabajo**	**Fin de Trabajo**	**Re trabajo**	**Causa**
Sprint 3	Comunicación con el franquiciante	7-jun	27-jun	25-jun	27-jun	Si	Visual
Sprint 3	Envío de mensajes	7-jun	27-jun	25-jun	27-jun	No	-
Sprint 3	Listado de mensajes	7-jun	27-jun	26-jun	26-jun	No	-
Sprint 3	Recibimiento de notificaciones	7-jun	27-jun	26-jun	26-jun	No	-

Tabla 1: Re trabajo en los sprints 1, 2 y 3

Sprint	**Modulo**	**Inicio de *Sprint***	**Cierre *Sprint***	**Inicio de Trabajo**	**Fin de Trabajo**	**Re trabajo**	**Causa**
Sprint 4	Creación de franquiciantes	28-jun	22-jul	19-jul	19-jul	No	-
Sprint 4	Modificación de franquiciantes	28-jun	22-jul	19-jul	19-jul	No	-
Sprint 4	Eliminación de franquiciantes	28-jun	22-jul	19-jul	19-jul	No	-
Sprint 4	Listado de franquiciantes	28-jun	22-jul	19-jul	19-jul	No	-
Sprint 4	Selección de franquiciantes	28-jun	22-jul	19-jul	19-jul	No	-
Sprint 4	Creación de franquiciados	28-jun	22-jul	21-jul	21-jul	No	-
Sprint 4	Modificación de franquiciados	28-jun	22-jul	21-jul	21-jul	No	-
Sprint 4	Eliminación de franquiciados	28-jun	22-jul	21-jul	21-jul	No	-
Sprint 4	Listado de franquiciados	28-jun	22-jul	21-jul	21-jul	No	-
Sprint 4	Selección de franquiciados	28-jun	22-jul	21-jul	21-jul	No	-
Sprint 4	Recuperar contraseña	28-jun	22-jul	20-jul	20-jul	No	-
Sprint 4	Cambiar contraseña	28-jun	22-jul	20-jul	20-jul	No	-
Sprint 4	Generar contraseñas	28-jun	22-jul	20-jul	20-jul	No	-
Sprint 4	Enviar contraseñas	28-jun	22-jul	20-jul	20-jul	No	-
Sprint 4	Cambiar contraseña generada	28-jun	22-jul	20-jul	20-jul	No	-
Sprint 4	Interfaces de usuario	28-jun	22-jul	19-jul	22-jul	No	-
Sprint 4	Acceso al sistema	28-jun	22-jul	21-jul	21-jul	No	-
Sprint 4	Interfaces de errores	28-jun	22-jul	19-jul	22-jul	No	-
Sprint 4	Estilo	28-jun	22-jul	19-jul	22-jul	No	-
Sprint 4	Cerrar el sistema	28-jun	22-jul	21-jul	21-jul	No	-

Tabla 2: Re trabajo en el sprint 4

Sprint	**Modulo**	**Inicio de *Sprint***	**Cierre *Sprint***	**Inicio de Trabajo**	**Fin de Trabajo**	**Re trabajo**	**Causa**
Sprint 5	Listado de productos	23-jul	9-ago	4-ago	4-ago	No	-
Sprint 5	Selección de productos	23-jul	9-ago	4-ago	4-ago	No	-
Sprint 5	Creación de productos	23-jul	9-ago	5-ago	5-ago	No	-
Sprint 5	Modificación de productos	23-jul	9-ago	5-ago	5-ago	No	-
Sprint 5	Eliminación de productos	23-jul	9-ago	5-ago	5-ago	No	-
Sprint 5	Listado de pedidos pendientes	23-jul	9-ago	8-ago	8-ago	No	-
Sprint 5	Selección de pedido	23-jul	9-ago	8-ago	8-ago	No	-
Sprint 5	Indicar pedido recibido	23-jul	9-ago	6-jul	6-jul	No	-
Sprint 5	Indicar pedido en producción	23-jul	9-ago	6-jul	6-jul	No	-
Sprint 5	Indicar pedido enviado	23-jul	9-ago	6-jul	6-jul	No	-
Sprint 5	Indicar pedido entregado	23-jul	9-ago	6-jul	6-jul	No	-
Sprint 5	Listado de pedidos entregados	23-jul	9-ago	6-jul	6-jul	No	-
Sprint 5	Ver detalle de pedidos	23-jul	9-ago	9-ago	9-ago	No	-

Tabla 3: Re trabajo en el sprint 5

Sprint	**Modulo**	**Inicio de *Sprint***	**Cierre *Sprint***	**Inicio de Trabajo**	**Fin de Trabajo**	**Re trabajo**	**Causa**
Sprint 6	Definición de logotipo	12-ago	4-sep	1-sep	1-sep	No	-
Sprint 6	Definición de colores	12-ago	4-sep	1-sep	1-sep	No	-
Sprint 6	Gestión de personalización	12-ago	4-sep	1-sep	1-sep	No	-
Sprint 6	Configuración de restricción	12-ago	4-sep	3-sep	3-sep	No	-
Sprint 6	Selección de restricción	12-ago	4-sep	3-sep	3-sep	No	-
Sprint 6	Creación de restricción	12-ago	4-sep	3-sep	3-sep	No	-
Sprint 6	Modificación de restricción	12-ago	4-sep	3-sep	3-sep	No	-
Sprint 6	Eliminación de restricción	12-ago	4-sep	3-sep	3-sep	No	-
Sprint 6	Validación de restricción	12-ago	4-sep	3-sep	3-sep	No	-
Sprint 6	Listado de canales de comunicación abiertos	12-ago	4-sep	2-sep	2-sep	No	-
Sprint 6	Selección de canal de comunicación	12-ago	4-sep	2-sep	2-sep	No	-
Sprint 6	Envío de mensajes	12-ago	4-sep	30-ago	31-ago	No	-
Sprint 6	Listado de mensajes	12-ago	4-sep	30-ago	31-ago	No	-
Sprint 6	Listado de notificaciones	12-ago	4-sep	4-sep	4-sep	No	-
Sprint 6	Creación de notificaciones	12-ago	4-sep	4-sep	4-sep	Si	Visual
Sprint 6	Envío de notificaciones	12-ago	4-sep	4-sep	4-sep	Si	Visual

Tabla 4: Re trabajo en el sprint 5

3.4.2. Conclusiones de re trabajo en el desarrollo del sistema

Como se puede observar en las tablas anteriores el sistema no requirió de re trabajo con errores críticos, incluso los errores triviales fueron pocos y ocurrieron en los módulos que requerían más interacción visual como lo fue el de comunicación, donde se creó un chat en tiempo real para el usuario.

Podemos concluir que no se necesitó re trabajo en el sistema, cada módulo pasó de manera satisfactoria las pruebas sin errores que interrumpieran el correcto funcionamiento ni la interacción del usuario del mismo.

Es así que podemos confirmar la utilidad de la metodología dirigida por pruebas en el desarrollo técnico del sistema, evitando el re trabajo y agregando robustez.

Por otro lado, haber hecho uso de la norma ISO/IEC 29110-5.1.1 adaptada a las necesidades del proyecto fue de gran ayuda a la hora del desarrollo. Debido a la confección de los casos y procedimientos de pruebas en una etapa anterior al desarrollo del *sprint*, se obtuvo una visión mucho más completa de los errores que debían prevenirse por el mal uso del sistema y del flujo de negocio que debía seguir el sistema.

Es de esta manera que se evitaron errores en la programación de cada *sprint* y fueron aceptados en su totalidad en la primera iteración de cada uno de ellos.

Por un lado, la norma brindó un sostén desde la documentación y robustez en las etapas desarrolladas y por otro TDD, en la programación, un código limpio y libre de errores que impidieran el correcto funcionamiento del sistema.

Aunque, como se mencionó anteriormente, TDD puede tener una curva elevada de aprendizaje, donde se deben entender los principios de las pruebas, como utilizarlas e incluso comprender el manejo de las bibliotecas de programación, el tiempo luego es recuperado evitando correcciones de errores y re trabajo.

4. Evaluación

4.1. Aplicación de MyFEPS / QSAT a la evaluación de la calidad del sistema desarrollado

4.1.1. Generalidades

Como se ha manifestado anteriormente el sistema obtenido durante este proyecto será evaluado a través del *framework* MyFEPS / QSAT. El objetivo de esta evaluación es establecer la calidad del producto obtenido a través de la implementación de la metodología de desarrollo dirigida por pruebas y basada en una norma internacional que guía en el proceso de desarrollo de software con foco en la calidad. MyFEPS será el instrumento para evaluar si ambas técnicas han sido adecuadas.

Este modelo facilita la evaluación de los productos debido a que tiene un propósito general siendo fácilmente adaptable a cada producto de software y a cada contexto donde este se desarrolle [54].

Se evaluarán las características más importantes para el proyecto según los *stakeholders*, esto se logró mediante la realización de cuestionarios de características, sub características y subsubcaracterísticas.

La documentación de MyFEPS / QSAT declara que para obtener las características fundamentales elegidas sean las correctas se deben cumplir dos condiciones: [54]

1. No se podrá derivar de otra característica.
2. Se podrá explicar usando uno de los siguientes esquemas (lo definido entre corchetes [] es opcional):

    1. Cumplir con los requisitos de <Característica Fundamental> significa que el sistema <provee X> (o <tiene X>) [en el contexto de <C>] [de modo que <M>]. Evaluable en términos de <T>

        ó

    2. Evaluar la/el <Característica Fundamental> de un sistema es evaluar en qué medida [en el contexto de <C>], el sistema <hace X> (o <tiene X>) [de modo que <M>]. Evaluable en términos de <T>

Por otro lado, el diseño de la evaluación con MyFEPS / QSAT propone un proceso ágil de evaluación, mediante un procedimiento general de evaluación ágil [55].

4.1.1.1. Evaluación ágil

Un proceso de evaluación ágil es aquel que cumple con los principios básicos del manifiesto ágil [27]. MyFEPS / QSAT adaptó los lineamientos del manifiesto para la evaluación ágil de la siguiente manera [55]:

- **Los individuos y su interacción** frente al problema de la Evaluación, por encima de los procesos, los estándares y las herramientas.
- Los aspectos del **producto Evaluado**, frente a la documentación exhaustiva y/o los modelos sumamente estructurados.
- La **colaboración con el cliente**, en las definiciones por encima de la negociación contractual.
- Una rápida **respuesta a la Evaluación**, por encima del seguimiento de un plan.

De esta manera se evitan los altos costos del proceso, la cantidad de recursos humanos necesarios y el tiempo prolongado que se debe implementar en el proceso.

Así es como se tiene un pequeño grupo de evaluadores que generan la documentación mínima e indispensable para el proceso de evaluación y la rápida respuesta ante el cambio en lo planificado.

No obstante, el proceso de evaluación ágil de MyFEPS / QSAT mantiene compatibilidad con la norma ISO/IEC 25040.

4.1.2.Pasos de la Evaluación ágil

El proceso de evaluación ágil cuenta con once pasos a realizar, estos son [55]:

1. **Establecer el propósito de la prueba**

   El objetivo general de una evaluación de calidad es el aseguramiento de que el producto de software brinda la calidad requeridas por los usuarios y que satisface las necesidades explicitas e implícitas de los mismos. Por lo tanto, el propósito puede ser aceptar o rechazar el producto, comparar el producto con la competencia, decidir módulos de mejora o sustitución, etcétera…

2. **Identificar el producto a evaluar**

   Se debe identificar el producto completo o parte a evaluar. En el caso de este proyecto se tiene un producto ya desarrollado por lo tanto se quiere estimar la calidad del producto final. Se deben documentar todos los aspectos conocidos del producto a evaluar.

3. **Identificar los requerimientos de calidad**

   Se deben identificar los *stakeholders* del producto, especificar los requerimientos de calidad del software siguiendo un

modelo de calidad existente. En este caso el modelo utilizado es MyFEPS / QSAT. Por último, se deben definir las características, sub características, subsubcaracterísticas y los atributos y métricas para ellas.

4. **Definir recursos y equipo para la evaluación**

   Se definen los recursos necesarios para la evaluación, estos pueden ser de infraestructura, tecnología, personas, herramientas tecnológicas como software.

5. **Establecer ponderación**

   Se establecen los criterios de aprobación para cada métrica elegida anteriormente. Esto se realiza en conjunto con los *stakeholders* y el equipo de evaluación.

6. **Preparar recursos de infraestructura**

   Se debe realizar un plan de acción teniendo en cuenta la disponibilidad de todos los recursos. Los recursos deben estar preparados para ser usados según marque el calendario realizado.

7. **Elaborar casos de prueba**

   En consecuencia, con las métricas y características se realizan los casos de prueba, se define el ambiente para las actividades, los métodos para realizar las pruebas y definir un cronograma de realización de evaluación.

   En el caso de este proyecto, los casos de prueba y realización de los mismos fueron realizados en el ciclo de vida de desarrollo, debido a la utilización de la metodología dirigida por pruebas y de la adecuación de la norma ISO/IEC 29110-5.1.1 de la misma. El modelo de evaluación avala que los casos de pruebas hayan sido realizados en otro momento y por otro grupo de evaluación.

8. **Obtener y adecuar recursos para la evaluación**

   Preparación para las pruebas.

9. **Realizar las pruebas**

   Se obtienen el total de pruebas erróneas y la identificación de los errores.

10. **Medir los atributos requeridos por las métricas**

    Se calcular las métricas en base a los resultados de las pruebas obtenidas. Se realiza una tabla con las características, sub características y subsubcaracterísticas. Se recomienda el uso de un Excel para realizar las mediciones.

11. **Analizar y concluir evaluación**

    Se realiza el análisis de los resultados obtenidos y se obtienen las conclusiones de los mismos.

Se debe dejar documentado únicamente de manera obligatoria las Planillas de cálculo G: Grado de Calidad de la Evaluación. Esta es proporcionada por el modelo.

Es objetivo de este trabajo dejar documentadas las conclusiones de los resultados obtenidos de la evaluación para concluir el grado de calidad de un proyecto de software dirigido por prueba.

Los resultados serán expuestos en la próxima sección.

5. Resultados

5.1. Presentación y evaluación de los resultados

5.1.1. Resultados de la evaluación de calidad

En este capítulo se presentarán los resultados obtenidos de la evaluación realizada al sistema con el *framework* MyFEPS / QSAT. En el anexo de este trabajo se encontrarán los documentos obtenidos, como los formularios realizados al cliente de características y sub características. También así, las métricas utilizadas para la evaluación de cada característica y sub característica.

La evaluación fue realizada a la aplicación Android y a la web como un sistema único y no como dos entidades separadas, es así que el valor final de calidad de la evaluación corresponde a ambos.

A continuación, se presenta la planilla de cálculo G de la evaluación, siendo el único documento obligatorio para la metodología utilizada.

					GCSC = SUM(PAT(i) * GCAT(i)) / SUM(PAT(i))	GCCB = SUM(PSC(i) * GCSC(i)) / SUM(PSC(i))		
			PSC	GCAT	GCSSC	GC		
Básica CB	**Sub**	**peso C**	**peso SC**	**Grado de Calidad: Es la Medición del Atributo**	**Grado de Calidad de la SC**	**Grado de Calidad de la CB**	**Grado de CALIDAD Del Producto**	
							0,97	
Efectividad		**1,00**				**1,00**		
	Cobertura de las funcionalidades útiles		1,00	1,00	1,00			
	Ausencia de funcionalidades inútiles		1,00	1,00	1,00			
Eficiencia		**1,00**				**1,00**		
	En la Interfaz del Usuario		1,00	1,00	1,00			
	En los tiempos de respuesta		1,00	1,00	1,00			
Manejo de Fallas		**1,00**				**0,88**		
	Previniendo: Causadas por el software		1,00	1,00	1,00			
	Previniendo: Causadas por mal uso		1,00	1,00	1,00			
Mantenibilidad		**1,00**				**0,97**		
	Eficiencia para corregir errores		1,00	0,95	0,95			
	Eficiencia para ampliar y mejorar		1,00	1,00	1,00			
Satisfacción objetiva de los usuarios		**1,00**				**0,97**		
	En el Acceso a las funciones		1,00	0,95	0,95			

	En la Comprensión de las salidas del sistema		1,00	0,98	0,98			
	Satisfacción total		1,00	0,97	0,97			
Testeabilidad		**1,00**				**1,00**		
	Madurez en a la cobertura manual: *testing*		1,00	1,00	1,00			
	Madurez en a la cobertura manual: artefactos		1,00	1,00	1,00			
	Madurez en a la cobertura manual: entorno de uso		1,00	1,00	1,00			
	Madurez en a la cobertura Automática: *testing*		1,00	1,00	1,00			
	Madurez en a la cobertura Automática: artefactos		1,00	1,00	1,00			
	Madurez en a la cobertura Automática: entorno de uso		1,00	1,00	1,00			
			Media	0,98				

Tabla 5: Evaluación MyFEPS

Se puede apreciar que en la gran mayoría de las métricas el sistema ha cumplido con los valores esperados por el cliente. Es así que todas las métricas han alcanzado valores mayores al estandarizado como aceptado por el método de evaluación, este siendo: 0.70.

El grado de calidad de producto indica un valor de 0.97 en las características evaluadas.

Este valor es aceptable ya que el puntaje de todos los atributos a evaluar supera el valor mínimo de aprobación del mismo, como se ha mencionado antes. De esta forma, la evaluación para este software es de "Aprobado".

Se debe destacar que no se encontraron errores durante la realización de la prueba del sistema FRANQUIA.

5.2. Conclusiones

5.2.1. Conclusión final

A lo largo de este trabajo se han expuesto las bases teóricas y aplicaciones prácticas de la metodología dirigida por pruebas y la norma ISO/IEC 29110-5.1.1 al desarrollo de un sistema mobile y web. Este sistema ha sido evaluado para obtener el grado de calidad obtenido al haber utilizado la metodología y la norma.

La norma fue modificada para lograr su adecuación al proceso de TDD, y de esta manera adaptarla a la metodología, dado que esta última solo se basa en el desarrollo como tal del sistema.

Los *frameworks* para pruebas unitarias utilizados fueron JUnit y PHPUnit. Se pudo comprobar que las pruebas tienen mucho mayor valor en el *backend* (servicios y procesamiento) que en el *frontend* (interfaces visuales). Por tanto las pruebas en PHP fueron más útiles que en Android, dado que una app tiene más interfaces de usuario que procesamiento de información.

Las pruebas realizadas, sobre todo en PHP, ordenaron el proceso de desarrollo y contribuyeron significativamente a que el código resultante fuera limpio y centrado en lo estrictamente necesario, evitando el código superfluo.

Igualmente, como se puede observar en los registros de trabajo que se llevaron durante el desarrollo, el sistema no requirió re trabajo con errores críticos y muy poco en errores triviales. Los errores que se produjeron fueron pocos y ocurrieron en las interfaces visuales, justamente donde menos aplicable resultaba la metodología TDD.

A su vez, el uso de la norma ISO/IEC 29110-5.1.1 adaptada fue de gran ayuda a la hora del desarrollo, brindando una visión anticipada y completa de los errores que debían prevenirse y del flujo de negocio del sistema, evitando errores en la programación de cada *sprint* y facilitando la aceptación de estos en la primera iteración. La norma sirvió, además, como soporte de documentación de cada etapa desarrollada.

La evaluación de calidad del sistema resultante realizada con MyFEPS arrojó resultados satisfactorios en todos los aspectos evaluados. Hubo que modificar dos métricas del modelo de calidad QSAT para que los resultados dieran entre 0 y 1 que eran los valores de resultados posibles para todas las métricas y para medir el grado de calidad del producto en función de los objetivos de evaluación establecidos por los propios *stakeholders*.

Finalmente, se concluye que el uso de la metodología TDD y la norma ISO/IEC 29110-5.1.1 contribuyen a evitar el re trabajo, evitar errores, documentar efectivamente y lograr un código limpio y robusto, todo lo cual compensa cualquier demora producida por la curva de aprendizaje de dicha metodología, como muestra las tablas 1, 2, 3 y 4.

Glosario

1. ***Framework*:** Marco de trabajo estandarizado utilizado para el desarrollo de software.

2. **Stakeholder:** Persona interesada en el ciclo de vida de un software.

3. **Api:** Interfaz de programación de aplicaciones. Conjunto de protocolos y definiciones para la integración de aplicaciones.

4. ***Frontend*:** Es la parte del software a la que el usuario accede e interactúa.

5. ***Backend*:** Es la parte del software que brinda funcionalidad, pero el usuario no interactúa con ella.

6. **ISO:** International Organization for Standardization, es una federación mundial de organismos nacionales de normalización.

7. **IEC:** International Electrotechnical Commission, es una organización de normalización en los campos: eléctrico, electrónico y tecnologías relacionadas.

8. ***Look and Feel*:** Se refiere al aspecto del sistema de manera que el usuario final se sienta cómodo en el uso del mismo.

9. **MVC:** Model View Controller, es un estilo de arquitectura de software que separa la interfaz de usuario, de la lógica de negocio y de los datos en tres capas.

10. **MVP:** Model View Presenter, es una derivación de la arquitectura MVC utilizada mayormente para la construcción de interfaces de usuario.

11. **API:** Application programming interface, conjunto de rutinas para la comunicación de distintos sistemas de software.

12. ***Sprint*:** Iteración dentro de un proyecto donde se obtiene un sistema totalmente funcional.

Bibliografía

[1] Documento de la norma ISO/IEC 29110
Software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 5-1-1: Management and engineering guide: Generic profile group: Entry profile.

[2] Sitio MyFEPS / QSAT
https://sites.google.com/a/comunidad.ub.edu.ar/myfeps/qsat-1
Visitado: (29-09-2019 / 15h28)

[3] Kent Beck. (1 edition November 18, 2002). Test Driven Development by Example. Addison-Wesley Professional.

[4] Sitio web de la compañía Etermax
https://etermax.com/en/home/
Visitado: (3-10-2019 / 08h17)

[5] Yahya Rafique and Vojislav B. Misic, Senior Member, IEEE. (JUNE 2013). The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 39, NO. 6.

[6] Paula Angeleri, Rolando Titiosky, Abraham Dávila; 2019; Migrando MyFEPS a MyFEPS agile; Facultad de Ingeniería y Tecnología Informática, Universidad de Belgrano, Buenos Aires, Argentina. Departamento de Ingeniería, Pontificia Universidad Católica del Perú, Lima, Perú.

[7] Forrest Shull, Grigori Melnik, Burak Turhan, Lucas Layman, Madeline Diep, and Hakan Erdogmus. (2010). What Do We Know about Test-Driven Development? Fraunhofer Center for Experimental Software Engineering, Maryland.

[8] Eduardo Guerra. (2014). Designing a *Framework* with Test-Driven Development: A Journey. National Institute for Space Research, Brazil.

[9] Frank Buschmann. (2011). Tests: The Architect's Best Friend. IEEE SOFTWARE.

[10] Gabriel Lowe. (30-03-2015). Test-Driven Development as a Medical Treatment. Universidad de Oxford. cs.ox.uk.

[11] ISO/IEC TR 29110-1:2016
https://www.iso.org/standard/62711.html
Visitado: (30-04-2019 / 21h15)

[12] ISO/IEC TR 29110-3-1:2015
https://www.iso.org/standard/62713.html
Visitado: (30-04-2019 / 21h17)

[13] ISO/IEC 29110-3-3:2016
https://www.iso.org/standard/64781.html
Visitado: (30-04-2019 / 21h20)

[14] ISO/IEC 29110-4-1:2011
https://www.iso.org/standard/51154.html
Visitado: (30-04-2019 / 21h28)

[15] ISO/IEC TR 29110-5-1-1:2012
https://www.iso.org/standard/60389.html
Visitado: (30-04-2019 / 21h35)

[16] ISO/IEC TR 29110-5-1-2:2011
https://www.iso.org/standard/51153.html
Visitado: (30-04-2019 / 21h39)

[17] ISO/IEC TR 29110-5-2-1:2016
https://www.iso.org/standard/62741.html
Visitado: (30-04-2019 / 21h43)

[18] Sitio web dedicado a la norma ISO/IEC 29110
http://www.upto25.net/
Visitado: (30-04-2019 / 21h23)

[19] Statement of Work: Definition & Examples; Peter Landau
https://www.projectmanager.com/blog/statement-work-definition-examples
Visitado: (20-10-2019 / 14h00)

[20] Real Academia Española
https://dle.rae.es/?id=6nVpk8P|6nXVL1Z
Visitado: (25-09-2019 / 11h45)

[21] Amos Sorgen, Paula Angeleri. (2017). QSAT 0 – MyFEPS – Explicación.

[22] Paula M. Angeleri, Alejandro Oliveros, Amos Sorgen, Rolando Titiosky, Jaquelina Wuille Bille; 2014; Modelo de calidad de productos de software;
http://conaiisi.unsl.edu.ar/ProceedingsCoNaIISI2014.pdf
Visitado: (15-05-2019 / 21h58) Pag. 1043 a 1051

[23] Angeleri, P., Sorgen, A., Bidone, P., Fava, A., Grasso, W; 2014; Diseño y desarrollo de un *framework* metodológico e instrumental para asistir a la evaluación de software; Anales de las 43 JAIIO (Jornadas Argentinas de Informática); SADIO Sociedad Argentina de Informática; Pág. 9 a 13
http://43jaiio.sadio.org.ar/proceedings/JUI/836-2618-1-DR.pdf
Visitado: (20-12-2018 / 15h00)

[24] ISO/IEC 9126-1:2001 Software engineering — Product quality — Part 1: Quality model
https://www.iso.org/contents/data/standard/02/27/22749.html
Visitado: (30-04-2019 / 10h15)

[25] ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models
https://www.iso.org/standard/35733.html
Visitado: (30-04-2019 / 10h25)

[26] IAN SOMMERVILLE; Madrid. 2005; Séptima Edición; Ingeniería del software; Pág. 361 a 373

[27] Manifiesto ágil
https://agilemanifesto.org/iso/es/manifesto.html
Visitado: (19-10-2019 / 23h45)

[28] WILSON, DANIEL; BROWN, JOSHUA; BURKE, ADAM A; Noviembre 2013; LET'S Scrum! LEARNING DIGITAL MEDIA COLLABORATIVELY; Vol. 73 Issue 3, p16-22. 7p.

[29] The Definition of *User Experience* (UX)
https://www.nngroup.com/articles/definition-user-experience/
Visitado: (13-11-2019 / 14h00)

[30] DIRKSEN, JULIE; (Noviembre 2018), L&D, Meet UX Design, TD: Talent Development. Vol. 72 Issue 11, p61LT-64LT. 4p. 1 Color Photograph, 2 Diagrams.

[31] Vermeeren, Arnold P.O.S; Roto, Virpi; Väänänen, Kaisa; (Enero 2016), Design-inclusive UX research: design as a part of doing *User Experience* research, Behaviour & Information Technology. Vol. 35 Issue 1, p21-37. 17p. 2 Color Photographs, 2 Diagrams, 3 Charts.

[32] PHPUnit – The PHP Testing *Framework*
https://phpunit.de/
Visitado: (17-09-2019 / 19h00)

[33] Escribir pruebas con PHPUnit

https://phpunit.readthedocs.io/es/latest/writing-tests-for-phpunit.html
Visitado: (17-09-2019 / 19h07)

[34] Composer. A Dependency Manager for PHP
https://getcomposer.org/
Visitado: (18-03-2019 / 19h39)

[35] *Framework* PHP para automatizar pruebas
https://blog.aulaformativa.com/*framework*-php-automatizar-pruebas/
Visitado: (17-09-2019 / 19h19)

[36] *Frameworks* de Mock de Objetos para Pruebas Unitarias
https://folderit.net/itech/es/*frameworks*-de-mock-de-objetos-para-pruebas-unitarias-es/ Visitado: (17-09-2019 / 19h37)

[37] Sitio oficial de Android Studio *testing*
https://developer.android.com/studio/test
Visitado: (07-05-2019 / 11h55)

[38] Sitio oficinal de Robolectric
http://robolectric.org/
Visitado: (17-09-2019 / 20h25)

[39] Cómo compilar pruebas de unidades
https://developer.android.com/training/*testing*/unit-*testing*/local-unit-tests
Visitado: (17-09-2019 / 20h45)

[40] Sitio oficinal de Mockito
https://site.mockito.org/
Visitado: (17-09-2019 / 21h26)

[41] Cómo compilar pruebas de unidades
https://developer.android.com/training/*testing*/fundamentals
Visitado: (07-05-2019 / 21h49)

[42] All about TestRule — A Steroid Before/After
https://medium.com/@elye.project/all-about-testrule-a-steroid-before-after-a74ef421e3e5
Visitado: (07-05-2019 / 22h10)

[43] The basics of Unit and Instrumentation Testing on Android
https://medium.com/@ali.muzaffar/the-basics-of-unit-and-instrumentation-*testing*-on-android-7f3790e77bd
Visitado: (07-05-2019 / 22h36)

[44] Unit Testing RxJava Observables
https://labs.ribot.co.uk/unit-*testing*-rxjava-6e9540d4a329
Visitado: (13-05-2019 / 15h00)

[45] Librería para la implementación de pruebas unitarias en RxJava
https://gist.github.com/ivacf/874dcb476bfc97f4d555
Visitado: (13-05-2019 / 17h00)

[46] Sitio official de Expresso para Android
https://developer.android.com/training/*testing*/espresso
Visitado: (13-05-2019 / 15h25)

[47] Motoko Takeuchi*,†, Naohiko Kohtake, Seiko Shirasaka, Yumi Koishi and Kazunori Shioya. (3, March 2014). Report on an assessment experience based on ISO/IEC 29110. Journal of Software: Evolution and Process

[48] Professor Claude Y Laporte, P. Eng., Ph.D. La implementación de la norma ISO/IEC 29110 Guías de Gestión e Ingeniería para las organizaciones pequeñas. Project Editor of ISO/IEC 29110 Standards and Guides

[49] GitLab
https://about.gitlab.com/
Visitado: (05-05-2019 / 10h15)

[50] Bradley A. Malone PMP. (July 2012). Work Breakdown Structure. svconline.com

[51] Wu He. (2014). A *framework* of combining case-based reasoning with a work breakdown structure for estimating the cost of online course production projects. British Journal of Educational Technology.

[52] Arnobio Maya Betancourt, Nohora Díaz Garzón. (2002, 2da edición). Mapas conceptuales: elaboración y aplicación. Actualización pedagógica Colección actualización pedagógica Cooperativa Editorial Magisterio.

[53] Sebastián, Alberto; Hadad, Graciela Dora Susana; octubre 2015; Mejoras a un modelo léxico mediante mapas conceptuales; XXI Congreso Argentino de Ciencias de la Computación (Junín, 2015)

[54] Documento MyFEPS.
0 - MyFEPS - Explicación del QSAT v2017-09-01 v2.0

[55] Documento MyFEPS.
2 MyFEPS.Proceso de evaluación AGIL de productos de software v3 final