Universidad ORT Uruguay

Facultad de Ingeniería

Identificación de Deuda Técnica en instancias de reflexión de proyectos ágiles

Entregado como requisito para la obtención del título de Master en Ingeniería

María Cecilia Nacimento Ferrer - 116276

Tutor: Santiago Matalonga

2016

Declaración de autoría

Yo, Cecilia Nacimento, declaro que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizaba la Master en Ingeniería;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue contribuido por otros, y qué fue contribuido por mi;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

María Cecilia Nacimento Ferrer

15-02-2016

Agradecimientos

Realizar una Maestría fue todo un desafío para mi persona, llegar a esta instancia final, que significa escribir la tesis con la que se culmina una nueva etapa de estudiante involucra a muchas personas. Varias de ellas me han acompañado durante todo el ciclo de la maestría y he podido compartir bellas experiencias, otras han sido soporte fundamental durante estos años a nivel personal y laboral, a todas ellas desde ya muchas gracias por su dedicación y sobre todo por su paciencia y comprensión.

En primera instancia quiero realizar un agradecimiento muy especial a mis padres Adela y Sergio, quienes siempre nos inculcan seguir creciendo tanto a nivel personal como profesional. Para estudiar siempre hay recursos. A mis 3 hermanas, en especial a Carolina quien ha sido mi correctora en ortografía y redacción, a Adela quien siempre me incentiva a seguir y tener confianza en mí. A mis sobrinos, a mi cuñado y a mi abuela que aunque hoy ya no esté presente vivió este proceso, en tu honor abue.

A mis amigos, a mis compañeros y tutores de clínica, a mis compañeros de taller. A Leo, mi psicólogo fundamental a la hora de confiar en mi capacidad, gracias por ayudarme en este camino tan especial.

A nivel laboral agradecerle a Mónica, que nunca tuvo inconveniente en permitirme tomarme días por estudios o para las conferencias que asistí a presentar mis trabajos, a todos mis compañeros de oficina que soportaron mi proceso de maestría en estos años.

Por último, debo agradecer a quienes me acompañaron en este proceso en la Universidad ORT Uruguay. Agradecer muy especialmente a mi tutor Santiago quien fue mi guía durante todos los años de trabajo de maestría. Gracias Santi, por todo el soporte que me diste para que llega hasta aquí, el conocimiento que me transferiste y por sobre todo la paciencia con que me has tratado. Gracias a Gastón, docente a quien admiro, y Alejandro gran compañero de equipo, por el trabajo realizado en conjunto Mecanismo de medición de agilidad. A Jean Carlos, por abrirme las puertas para poder realizar un trabajo en conjunto el cual me permitió llegar a presentar un artículo en Agile Brasil como única extranjera participante. Gracias a la Universidad ORT Uruguay por brindarme el apoyo para poder participar en las conferencias que asistí y representar a la misma. Tato gracias por el apoyo y rápida respuesta a cada una de las consultas, dudas que surgían durante estos años de maestría, Daniel al igual que Tato siempre danto respuesta a los problemas surgidos a nivel académico. Agradecer a cada uno de mis docentes y compañeros que sin

ellos no sería posible esta instancia. En Martin y Gastón gracias al cuerpo docente y a través de Lu y Alejandro agradecer a cada uno de mis compañeros de equipo y cursos.

Reconocimientos

Las asistencias y estadías con el fin de participar en los congresos y conferencias en las cuales se presentaron los trabajos fueron financiadas por Universidad ORT Uruguay.

Abstract

La metáfora de Deuda Técnica surge como concepto en los ’90, para la comunidad de ingeniería de software profesional como de investigación es un término de interés. Existe gran cantidad de material a nivel de la literatura sobre Deuda Técnica, sin embargo su inserción a nivel de la industria del software no es significante. Se halla poca evidencia de su aplicación, a pesar de considerar que es útil para explicar fenómenos que ocurren durante la producción de software.

El objetivo del presente trabajo es estudiar la Identificación de Deuda Técnica en instancias de reflexión de proyectos ágiles de desarrollo de software. Con el fin de resolver el objetivo planteado se instancia en primer lugar un mecanismo de medición que es capaz de evaluar a un proyecto según una escala que representa el Grado de Agilidad. El concepto de Grado de Agilidad alude a cuanto es el apego del proceso de desarrollo de software a la propuesta contenida en él Manifiesto Ágil. Finalmente, para dar respuesta al objetivo se aplicó una metodología de investigación cualitativa sobre los registros de retrospectivas de un proyecto real de desarrollo de software con el fin de identificar instancias de Deuda Técnica. Los resultados obtenidos muestran que es posible identificar constructos asociados con la metáfora. Es así que, el presente trabajo contribuye al cuerpo de conocimiento empírico de Deuda Técnica.

Palabras Claves: Deuda Técnica, proyectos ágiles, instancias de reflexión.

Índice

Índice de tablas

Índice de Imágenes

Índice de Gráficas

1 Introducción

La metáfora se manifiesta como concepto por los años ’90, fue introducido por Cunningham [1] durante la conferencia OOPSLA (O*bject Oriented Programming, Systems, Languages & Applications*). Se sostiene que identificar tempranamente la existencia de la Deuda Técnica en los proyectos proporciona en primer nivel tener conciencia de su existencia, en un segundo nivel permite incluirla a la hora de toma de decisiones teniendo la posibilidad de decidir asumir dicha Deuda Técnica o realizar acciones para mitigarla o eliminarla. Según Cunningham, la Deuda Técnica surge si, “*durante la realización de un proyecto de software si existen elementos que se eligen no realizar y que al no hacerlos provocan que se paralicé el desarrollo futuro del proyecto, estos elementos conforman la Deuda Técnica*”.

La Deuda Técnica surge vinculada a deuda de código, sin embargo, hoy día a la metáfora se la considera en todo el ciclo de vida de un proyecto. A pesar de que existe gran cantidad de material a nivel de la literatura, no se ha logrado la inserción de la metáfora a nivel de la industria del software, según Shull [2] existe poca evidencia de su aplicación aunque se considera que sería útil para explicar varios de los fenómenos de Ingeniería de Software que ocurren.

El objetivo de la presente tesis es la Identificación de Deuda Técnica en instancias de reflexión de proyectos ágiles de desarrollo de software. El concepto Deuda Técnica es relevante en el planteo del objetivo del trabajo, a continuación se presenta una breve introducción. La metáfora Deuda Técnica es un concepto que se utiliza con el fin de explicar los problemas que suceden en los proyectos cuando para poder cumplir con requisitos del mismo relegamos una dimensión de la gestión de proyectos de software en beneficio de otra. Por ejemplo, para culminar en la fecha acordada un proyecto, se reducen los tiempos dedicados al aseguramiento de la calidad. Cunningham en los ’90 introdujo el concepto de la metáfora, la definió en términos de inmadurez de código [1]. Martin Fowler encuentra similitud con la deuda financiera en la cual se debe pagar intereses [3]. Mientras Guo, Seaman, sostienen que se hacen compensaciones durante el desarrollo de software y esto implica luego la generación de Deuda Técnica [4]. El conocimiento empírico sobre Deuda Técnica se amplió y hoy se habla de Deuda Técnica a nivel de código, diseño, arquitectura, pruebas; ver sección 3.3 .

Los proyectos de desarrollo que utilizan metodologías ágiles son una alternativa real al desarrollo de software. El 40% de los proyectos de desarrollo de software se identifican con una metodología ágil según la encuesta [5]. Agilidad proviene del Manifiesto Ágil, en el cual se reconocieron valores y prácticas que serán características de las

metodologías ágiles, las que mejoraban las prácticas habituales del desarrollo de software. Cockburn [6] expresa que el desarrollo ágil de software es un movimiento dentro de la disciplina de la Ingeniería de Software, el mismo presenta características que lo distancian de lo propuesto por las metodologías tradicionales las cuales son orientadas a la planificación. Las principales características del paradigma de desarrollo ágil son: destacar el programador como productor de conocimiento, la interacción con sus pares, la capacidad de comunicación, y la adaptación al cambio.

La práctica de reflexiones en proyectos es anterior a las metodologías ágiles, el esfuerzo por incorporar prácticas de reflexión surge por trabajos que intentan implementarlas primero en metodologías tradicionales [7]. Entre las prácticas que se proponen en el Manifiesto Ágil [8], se encuentra la necesidad de "*reflexionar a intervalos regulares, sobre el proceso de desarrollo utilizado para mejorarlo continuamente*".

En primera instancia para lograr el objetivo de tesis se debe determinar cuándo un proyecto de desarrollo es – o puede considerarse – "ágil". El propósito es medir la agilidad de cada proyecto de desarrollo de software independientemente de la metodología que expresen seguir o aplicar. Investigando a nivel de la literatura se encuentran propuestas de metodologías de medición de agilidad los que miden la agilidad basándose principalmente en las metodologías que usan los proyectos o comparan proyectos que aplican distintas metodologías para identificar cuál de estas es más ágil, ver sección 3.2.1. De los resultados obtenidos ninguno cubre la necesidad planteada, es por esto que para dar respuesta al problema se propone un mecanismo de medición del grado de agilidad de los proyectos de desarrollo de software, cuyo objetivo es medir el grado de agilidad de proyectos de similares características (como por ejemplo, la duración, la cantidad de integrantes, el sector) que presentan independientemente de la metodología utilizada en los proyectos. El mecanismo propuesto se basa en los valores y principios del Manifiesto Ágil. Este utiliza como método de recolección de datos la encuesta, los datos que ingresan en la encuesta son procesados para obtener el grado de agilidad que es el resultado del mecanismo propuesto [9]. El Concepto de Grado de Agilidad alude a cuanto es el apego del proceso de desarrollo de software a la propuesta contenida en él Manifiesto Ágil [8].

Finalmente para incorporar los conceptos (Deuda Técnica, proyectos ágiles e instancias de reflexión) y dar respuesta al objetivo se aplicó un método de investigación cualitativo (teoría justificada [10]) con el fin de identificar instancias de Deuda Técnica. Los resultados obtenidos en el proceso se publicaron en [11].

Los resultados de este trabajo de tesis muestran que es posible identificar constructos asociados con la metáfora [11]. Por lo cual, el presente trabajo contribuye al cuerpo de conocimiento empírico de Deuda Técnica, mediante la identificación de factores de Deuda Técnica en un proyecto de desarrollo de software real.

La estructura del presente trabajo de tesis es la siguiente, en el capítulo 2 se presentan los Objetivos del Trabajo, en este capítulo se describen las Contribuciones realizadas, como ser el Mecanismo de medición de agilidad y la Identificación de Deuda Técnica en proyecto ágil de desarrollo de software. Asimismo lo conforman las aportaciones metodológicas, como lo son el concepto de grado de agilidad y las modificaciones al proceso de Teoría Justificada; y las publicaciones asociadas tanto en conferencias internacionales así como nacionales. Por ultimo compone este capítulo la organización del documento. En el tercer capítulo se expone el estado del arte de las áreas de conocimientos, las mismas son desarrollo ágil de software en la cual se presentan los mecanismos de medición identificación en la literatura y las instancias de reflexión en las metodologías ágiles de desarrollo. Por otra parte conforma este capítulo una sección que referencia a Deuda Técnica. Al capítulo 4 lo compone lo referente a los métodos de investigación, los mismos se estructuran en métodos cuantitativos – Encuesta y la teoría justificada. Luego en el capítulo 5, se presenta el Planteamiento del trabajo, se definen los objetivos y las metodologías de la investigación. El sexto capítulo presenta el Proceso de diseño del mecanismo de medición de agilidad, el mismo se estructura de la siguiente manera, primero se presenta el diseño de las preguntas del mecanismo de recopilación de datos, luego se expone la valoración de las respuestas y por último se realiza la validación del mecanismo. El capítulo 7 corresponde a la experimentación, en el mismo se presentan los resultados de las distintas instancias de ejecución del mecanismo de medición de agilidad, también se presentan los resultados correspondientes a la identificación de Deuda Técnica en instancias de reflexión de proyectos ágiles. Las conclusiones del trabajo y las futuras líneas de investigación se presentan como cierre del trabajo en el capítulo 8.

2 Objetivo del Trabajo

El objetivo de la presente tesis es la Identificación de Deuda Técnica en instancias de reflexión de Proyectos Ágiles de Desarrollo de Software.

Para identificar estos factores se necesitan poder identificar si un proyecto de desarrollo está siguiendo alguno de los Ciclos de vida identificados con las metodologías ágiles. La disparidad existente en los proyectos de desarrollo observados, llevo a la necesidad de definir una escala que permita agrupar estos proyectos de forma de poder compararlos.

2.1 Contribuciones

A continuación se listan las contribuciones realizadas por este trabajo.

2.1.1 Mecanismo de medición del grado de agilidad

El mecanismo de medición del Grado de Agilidad de los proyectos de desarrollo de software propuesto surge de la necesidad real de poder comparar dos o más proyectos de desarrollo de software. Mediante la utilización del mecanismo propuesto se logró identificar el grado de agilidad que posee el proyecto del cual se utilizan los informes de retrospectiva [9].

2.1.2 Identificación de Deuda Técnica en proyecto ágil de desarrollo de software

Con el fin de identificar la existencia de factores de Deuda Técnica en un proyecto de desarrollo ágil de software se utiliza la teoría justificada, la cual se aplica a los informes de retrospectiva del proyecto para localizar frases que se puedan mapear a los factores de Deuda Técnica identificados en un artículo previamente realizado en la línea de investigación [12].

2.1.3 Aportaciones metodológicas

2.1.3.1 Concepto de Grado de Agilidad

Cuando se hace referencia al Concepto de Grado de Agilidad lo que se quiere expresar es cuanto apego del proceso de producción de software a la propuesta contenida en el Manifi

esto Ágil [8]. Esto quiere decir que no se detiene el mecanismo propuesto en medir si una metodología de desarrollo ágil es más ágil que otra sino que permite comparar proyectos independientemente de la metodología de desarrollo que utilice. De esta forma se podrán agrupar proyectos cuyo grado de agilidad sean similares permitiendo poder comparar sus resultados.

2.1.3.2 Modificación al proceso de Teoría Justificada

Es relevante en la investigación de este trabajo determinar si es posible la vinculación de factores de Deuda Técnica [13] en los reportes de reuniones de retrospectivas de proyectos ágiles. Para ello seleccionamos la técnica de teoría justificada [10], la misma fue adaptada para su utilización. Se aplicó con las dos iteraciones propuestas teniendo en cuenta para la segunda iteración la definición de una categoría central, luego de la segunda iteración se obtienen los resultados.

2.1.4 Publicaciones asociadas

Durante el desarrollo de la tesis se han publicado los siguientes trabajos;

2.1.4.1 En conferencias internacionales o regionales:

1. Nacimento, C., Matalonga, S., Adorjan, A., Mousqués, G.: Propuesta de Mecanismo de Medición de Agilidad de Proyectos de Desarrollo. In: XVIII Ibero-American Conference on Software Engineering. pp. 109–122. UCSP, Lima-Peru (2015) [9].
2. Nacimento, C., Matalonga, S., Hauck, J.C.R.: Identificación de Factores de Deuda Técnica en Instancias de Reflexión de un Proyecto Ágil. CLEI 2014 (2014) [11].
3. Nacimento, C., Matalonga, S., Rossa-Hauck, J.C.: *Identificando Fatores de Débito Técnico em Retrospectivas de Projetos Ágeils*. 2014 Workshop Brazilero en metodos ageils (2014) [14].

2.1.4.2 En conferencia nacional:

1. Matalonga, S., Villar, A., Nacimento, C.: Deuda Técnica: ¿Hasta dónde podemos llevar la metáfora?. Universidad La Salle, Arequipa (2013). Perú.

2.2 Estructura del documento

El presente documento de tesis se estructura de la siguiente forma, en primer lugar se desarrolla el estado de arte de las áreas de conocimientos bases para la tesis. Como punto de partida se realiza un acercamiento a las Metodologías de Desarrollo de Software Ágiles. Dentro de este capítulo se presenta algunas de las metodologías de desarrollo de software ágiles que se utilizan en nuestros días, se hace referencia a los mecanismos de medición existentes en la literatura, para por último explicar las instancias de reflexión en proyectos ágiles.

En segunda instancia nos referimos a la metáfora Deuda Técnica, explicando su significado y su estado del arte en nuestros días. Un punto relevante dentro de este capítulo es la medición de Deuda Técnica.

En el siguiente capítulo se presentan los métodos de investigación que han sido utilizados durante el desarrollo del trabajo. En principio nos referimos al método cuantitativo que se seleccionó para trabajar, la Encuesta. En segunda instancia se presenta la Teoría Justificada [10], metodología seleccionada para utilizar en el proceso de identificación de factores de Deuda Técnica.

3 Estado del Arte: Áreas de conocimiento

La presente tesis consta de dos secciones claramente diferenciadas, en primer lugar se hará referencia a Metodologías de Desarrollo de Software Ágiles. En segunda instancia se presentará la metáfora Deuda Técnica. La razón por la cual ambas secciones son base para la tesis es porque el objetivo de la investigación apunta a poder realizar una vinculación entre ambas. Se busca poder utilizar la metáfora Deuda Técnica en instancia de reflexión en proyectos de desarrollo ágil.

Este capítulo por tanto, se divide en dos secciones, en primer lugar se presentará un acercamiento al estado del arte de metodologías de desarrollo identificando las metodologías de desarrollo ágil haciendo foco en instancias de reflexión de dichas metodologías. Luego presentaremos el estado del arte de la Metáfora Deuda Técnica, en este caso el foco estará situado en lo referente a la medición de las mismas.

3.1 Desarrollo ágil de Software

Las metodologías ágiles de desarrollo toman su propuesta de los valores del manifiesto ágil [8]. El manifiesto fue firmado en el año 2001, es en él que se encuentran el conjunto de premisas, que en principio deberían estar presentes en las metodologías que dicen seguir los valores y principios definidos en el mismo. Si bien el manifiesto muestra un contexto amplio, distintos autores destacan aspectos relevantes de las metodologías.

Según describe Agile Alliance [15], a finales de la década de los ‘90 varias metodologías comenzaron a llamar cada vez más la atención. Cada una tenía una diferente combinación de viejas y nuevas ideas, sin embargo todas ellas destacaban la estrecha colaboración entre el equipo de programadores y los expertos en negocios; la necesidad de comunicación cara a cara (como más eficiente que la documentación escrita); las entregas frecuentes y los equipos compactos y auto-organizados.

Cockburn [16] y Highsmith [17] mencionan que la idea predominante en el desarrollo ágil, se presenta cuando un equipo puede responder más efectivamente a los cambios, reduciendo los costos de intercambiar información y los tiempos de toma de decisiones. Los autores señalan que los equipos ágiles focalizan la competencia individual como factor crítico en el éxito de un proyecto. Los procesos ágiles a su vez, están diseñados para capitalizar en cada individuo y en cada equipo fortalezas únicas. En este contexto, los equipos ágiles se caracterizan por ser auto-organizados y con intensa colaboración a través de las fronteras organizacionales.

Boehm y Turner [18] definen agilidad, en el contexto de aplicar la memoria y la historia para ajustarse a nuevos entornos, reaccionar y adaptarse, tomar ventajas de oportunidades inesperadas y actualizar las bases de experiencias para el futuro.

Por su parte, Kruchten [19] define agilidad como la capacidad de una organización para reaccionar ante los cambios en su entorno más rápido que la velocidad en que se producen dichos cambios.

Una vez analizadas las diferentes definiciones encontradas en la literatura del área, se puede concluir que no existe una definición rigurosa, completa y consensuada del término agilidad en la literatura. Sin embargo, la mayoría de los autores desde sus distintos puntos de vista coinciden en los conceptos de agilidad propuestos por el manifiesto ágil. Como ser, lo mencionado por Cockburn [16] y Highsmith [17] "responder más efectivamente a los cambios", se puede mapear con "las entregas frecuentes" propuestas por Agile Alliance [15], también Boehm y Turner [18]refieren a dicha virtud en "reaccionar y adaptarse". Coinciden a su vez en la importancia de los equipos, lo justifican las siguientes frases, "la competencia individual como factor crítico del éxito de los proyectos" según proponen Cockburn [16] y Highsmith [17], para Agile Alliance [15] "los equipos compactos y auto-organizados"

3.2 Metodologías Ágiles de Desarrollo

Los proyectos de desarrollo de software que utilizan metodologías ágiles se presentan en nuestros días como una alternativa real al desarrollo de software. Como lo indica la encuesta presentada por Begel y Nagappan [5], aproximadamente el 40% de los proyectos de desarrollo de software que se realizan se identifican con una metodología de desarrollo ágil.

Debido a lo expuesto anteriormente, es que al día de hoy se puede constatar la existencia de diversas metodologías ágiles de desarrollo de software. A continuación, se presentarán las metodologías y practicas ágiles más utilizadas en proyectos de desarrollo de software, según lo que expresa el resultado de la encuesta *State of Agile Survey* presentada en 2015 por *VersionOne* [20].

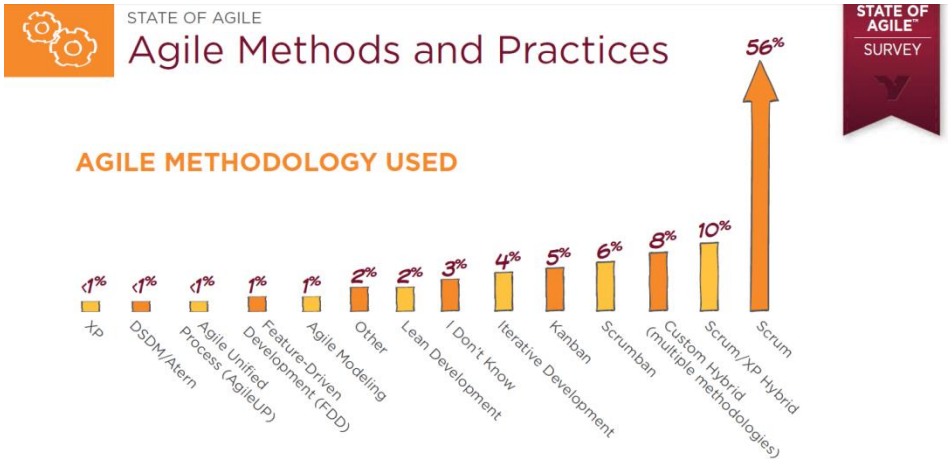


Imagen 1. Grafica Adopción de Metodologías de Desarrollo Ágil según *VersionOne* [20]

- ***Scrum*:** esta metodología es indicada para proyectos con alto ratio de cambio de requerimientos, su principal característica es la definición de lo que denominan *sprints* – se denomina a cada una de las iteraciones que componen el proceso, una característica de los mismos es la duración fija y que en general tiene un máximo de 30 días. El resultado obtenido en cada uno de los *sprints* es un incremento ejecutable que se muestra al cliente. Otra característica destacable de los *sprints* son las reuniones diarias que se llevan a cabo a lo largo del proyecto. Dichas reuniones no requieren de una duración extensa por el contrario el máximo tiempo dedicado a la misma es de 15 minutos, en las mismas participa el equipo de desarrollo y su objetivo son la coordinación e integración del producto a entregar [21], [22].

- ***Scrum/XP Hybrid***: la presente metodología surge a partir de la combinación de características o prácticas propias de la metodología *Scrum* [21], [22] con las propuestas por la metodología *Extreme Programming* (XP) [23]. No se especifica que características son las que se combina, sino que es abierto a las combinaciones que surjan de la mezcla de ambas metodologías.

- ***Custom Hybrid*:** la presente metodología implica la combinación de prácticas y características propias de las distintas metodologías de desarrollo ágiles existentes con el fin de lograr la adaptabilidad a proyectos de diferentes escalas (pequeños, medianos y grandes) [24], [25].

- ***Scrumban*:** la presente metodología nace de combinar *Scrum* [21], [22] y *Kanban* [26], por ello es que se compone de las mejores prácticas y reglas que caracterizan a ambos métodos. Por un lado, utiliza la naturaleza de *Scrum* de

ser una metodología Ágil y por otro, fomenta equipos para mejorar constantemente sus procesos, es de esta manera que se logra cumplir el objetivo de *Kanban* de mejora continua. Ladas [27] compara el enfoque ágil de iteraciones de tamaño constante y el enfoque *Just-In-Time* del sistema *Kanban*. Subraya el hecho de que el tamaño de la iteración en los métodos ágiles es uno de los parámetros que deben ser definidos. El cambio a un sistema *Kanban* en realidad puede hacer que el conjunto de concepto iterativo sea superfluo. El libro finaliza de hecho, en proponer un enfoque mucho más similar a *Kanban* que *Scrum*, conservando las buenas prácticas de *Scrum*.

- ***Kanban*:** *Kanban Software Development* se basa en la metodología de fabricación industrial que posee el mismo nombre. Su objetivo es gestionar de forma general como se van completando las tareas, pero además se ha utilizado en la gestión de proyectos de desarrollo de software. Se basa en el desarrollo incremental, propone dividir el trabajo en partes. Se destaca por la incorporación de técnicas visuales las cuales permiten identificar rápidamente en qué situación se encuentra cada tarea. La metodología no maneja fase sino que maneja flujo de actividades [26].

- ***Lean Development*:** Este método ofrece un marco teórico sólido y basado en la experiencia, para las prácticas ágiles de gestión. El término desarrollo de software *Lean* tiene origen en un libro propuesto por Mary Poppendieck y Tom Poppendieck. En el libro se presentan los principios tradicionales del método *Lean* (Sistema de Producción de Toyota) a los cuales se le realizaron modificaciones. El desarrollo Lean puede resumirse en siete principios básicos: Eliminar los desperdicios, Ampliar el aprendizaje, Decidir lo más tarde posible, Reaccionar tan rápido como sea posible, Crear la integridad, Empoderar el equipo y Véase todo el conjunto [28].

- ***Agile Modeling*:** es una metodología basada en una colección de prácticas, guías de valores y principios para ejecutar por los profesionales en el día a día. *Agile Modeling* no es un proceso, en otras palabras no define un procedimiento detallado a seguir sino que indica la forma de cómo ser efectivo a la hora de modelar. Los valores de *agile modeling* son: comunicación, simplicidad, comentarios, coraje, humildad [29].

- ***Feature Driven Development (FDD)*:** en este caso se define un proceso iterativo, el cual consta de iteraciones cortas con duración de dos semanas como máximo. El ciclo de vida propuesto se compone de cinco pasos: Desarrollo de un modelo global, Construcción de una lista de funcionalidades,

Planeación por funcionalidad, Diseño por funcionalidad y Construcción por funcionalidad [30].

- ***Agile Unified***: es una versión simple de *Rational Unified Process (RUP)*, la cual describe un enfoque simple y fácil de entender para aplicar en el desarrollo de aplicaciones de software utilizando técnicas y conceptos ágiles manteniéndose fiel a la propuesta de RUP. El proceso se compone de cuatro fases: comienzo, elaboración, construcción y transición. En cada fase existen disciplinas que se deben realizar: modelar, implementar, probar, desplegar, gestión de la configuración, gestión de proyecto y entorno [31].

- ***Dynamic Systems Development Method (DSDM)*:** el método cumple con las características generales de definir un proceso iterativo e incremental. Propone cinco etapas de desarrollo: Estudio de Viabilidad, Estudio del Negocio, Modelado Funcional, Diseño y Construcción, e Implementación. La iteración se produce en las tres últimas etapas, sin embargo prevé retro-alimentación en todas [32].

- ***Extreme Programming (XP)*:** Define un proceso iterativo e incremental con pruebas unitarias continuas y entregas frecuentes. El cliente o un representante del cliente son integrados al equipo de desarrollo. Recomienda que el desarrollo de las funciones del producto sea realizado por dos personas en el mismo puesto – programación por pares. Antes de incorporar nueva funcionalidad, se deben corregir todos los defectos encontrados. Constantemente, se llevan a cabo pruebas de regresión a fin de detectar los posibles errores [23].

Las respuestas, “no Se” y “desarrollo iterativo”, ejemplifican el problema a la hora de identificar si una metodología es ágil o no lo es. Tanto la encuesta de *VersionOne* [20], como los resultados de Begel y Nagappan [5] son sensibles al grado de entendimiento y capacitación de los respondientes.

3.2.1 Mecanismos de medición identificados en la literatura

Recientes investigaciones se han centrado en el problema de lo que implica ser ágil. Diversos autores, como por ejemplo, Boehm and Turner [18], Pikkarainen [33] y Leffingwell [34] se preocupan por definir un *framework* que permita establecer una medición de agilidad. Entre ellos no existe un consenso en cuanto a la definición, ni en la valuación de las dimensiones. Pero el esquema general se basa en la interpretación que ser ágil, no es una caracterización binaria, sino un rango de valores que puede ser medido a través de escalas ordinales. Por tanto, las propuestas tratan en general de establecer

estas escalas para poder establecer una medida de que tan ágil es un proyecto o proceso de desarrollo de software.

Boehm and Turner [18] describen un *framework* cuyo objetivo es poder determinar el grado de agilidad/disciplina requerido por un proyecto en base a las siguientes dimensiones: criticidad, experiencia del personal, dinamismo del entorno, cultura organizacional y tamaño del equipo.

Leffingwell [34] sugiere un modelo de autoevaluación de la agilidad. El objetivo de este modelo de evaluación es ayudar a los equipos a identificar cuáles son los puntos que permiten mejorar el proceso. El mismo se basa en dimensiones ordinales de 5 valores. En dichas dimensiones están modelados conceptos del proceso como por ejemplo: la existencia del *product owner*, la gestión de liberaciones, la gestión de la iteración, la integración del equipo, las pruebas y prácticas de desarrollo. Según el autor, el objetivo de esta evaluación no es dar un grado de agilidad, ni comparar procesos de desarrollo ágil, sino ayudar a los equipos a identificar puntos para mejorar el proceso.

Mendez [35], proponen un *framework* de evaluación cuantitativo para las metodologías ágiles de desarrollo. El objetivo del mecanismo es permitir la evaluación de cuanto las metodologías ágiles satisfacen los principios básicos que se definen en el Manifiesto Ágil [8]. El *framework* se aplicó a *Scrum* y *eXtreme Programming* (XP).

Qumer, et. al. [36] por su parte presenta un marco de evaluación denominado 4-DAT. El objetivo de este marco es proporcionar un mecanismo de medición de agilidad de cualquier metodología cuantitativamente, en un nivel específico en un proceso y utilizando prácticas específicas. El método basa su análisis en 4 perspectivas o dimensiones. Estas son: alcance del método (tamaño del proyecto, tamaño del equipo, estilo de desarrollo, estilo de código, entorno tecnológico, entorno físico, cultura de negocio, mecanismo de abstracción), características de agilidad (flexibilidad, velocidad, aprendizaje, capacidad de respuesta, *leanness*), valores de agilidad (individuos e iteraciones sobre procesos y herramientas, software funcionando sobre documentación, colaboración del cliente sobre negociación de contrato, responder al cambio sobre seguir un plan, mantener el costo efectivo del proceso, mantener el proceso ágil), características del proceso de software (proceso de desarrollo, proceso de gerenciamiento del proyecto, proceso de control de configuración de software/proceso de soporte, proceso de gerenciamiento de procesos).

Kruchten [19] presenta una definición del concepto de agilidad basado en dimensiones individuales que se mapean a los valores del manifiesto, concluyendo que la agilidad no debería estar definida en términos de prácticas, sino en la habilidad de la organización a reaccionar a los cambios del entorno.

Luego de presentar los métodos propuestos por los distintos autores, se puede notar que para su totalidad se enfocan a medir metodologías o realizar comparaciones entre proyectos que utilizan distintas metodologías ágiles con el fin de determinar cuál de ellos presenta mayor agilidad. Es por esto, que en este trabajo de tesis se propone un mecanismo que sea capaz de comparar proyectos de forma independiente a la metodología de desarrollo que se aplique. Se propone medir el grado de agilidad que presentan los proyectos evaluados, tal es el caso que el mismo puede utilizarse en cualquier proyecto, incluso en los que no tienen claro que tipo de metodología de desarrollo utilizan o que mezclan distintos tipos de metodologías situación común hoy día en los proyectos de desarrollo de software que combinan las mejores prácticas de las distintas metodologías, [20].

3.2.2 Instancias de Reflexión en metodologías ágiles de desarrollo

Como mencionamos en Metodologías Ágiles de Desarrollo agilidad proviene de la firma del manifiesto ágil [8], en esta instancia los firmantes reconocieron valores y prácticas que a su criterio mejoraban las prácticas habituales de desarrollo de software. Entre estas prácticas se encuentra la necesidad de "*reflexionar a intervalos regulares, sobre el proceso de desarrollo utilizado para así mejorarlo continuamente*".

La práctica de reflexiones en proyectos es anterior a las metodologías ágiles, y el esfuerzo por incorporar prácticas de reflexión en metodologías ágiles ha venido por trabajos que primero han intentado implementar procesos de reflexión utilizados en metodologías tradicionales [7].

En los últimos años ha habido más atención hacia la aplicación de técnicas que mejoren los resultados de las instancias de reflexión. En [37], los autores presentan una aplicación del uso de los 5 factores de personalidad para la autogestión de equipo de desarrollo. Es interesante destacar que las recomendaciones de aplicación para la conducción de instancias de reflexión son las mismas que [7], cuando este último sigue una metodología distinta de investigación.

Otros métodos para la realización de actividades de reflexión son propuestos por [38] y [39]. Sin embargo, a nuestro leal saber y entender únicamente en el trabajo de [40], se vincula el concepto de Deuda Técnica con el de actividades de reflexión. Sin embargo en este último artículo, a la experiencia se centra en el proyecto tradicional de mantenimiento y no se menciona el uso de prácticas ágiles de desarrollo.

En este trabajo se presenta la experiencia de identificar aspectos relacionados con la metáfora de Deuda Técnica en el contexto de las actividades de reflexión de un proyecto de desarrollo que utiliza metodologías ágiles.

3.3 Deuda Técnica

Deuda Técnica es una metáfora que se utiliza como forma de explicar los problemas que ocurren en los proyectos cuando para poder cumplir algunos requisitos del proyecto relegamos una dimensión de la gestión de proyectos de software en beneficio de otra. Un ejemplo que explica lo expresado anteriormente es que para finalizar en la fecha acordada un proyecto, no sé efectúen actividades de aseguramiento de la calidad. Dicha metáfora fue introducida por Cunningham en 1992, la cual definió en términos de inmadurez de código, "*Durante la realización de un proyecto de software si existen elementos que se elige no realizar y que al no hacerlos provocan que se paralice el desarrollo futuro del proyecto. Estos elementos conforman la Deuda Técnica*"[1].

Se presentan nuevas definiciones de reconocidos autores involucrados en la investigación de la metáfora. Jann Thomas aplica la metáfora para describir situaciones en que el equipo de desarrollo acepta sacrificar una dimensión, como puede ser calidad de software con el fin de optimizar otra dimensión como por ejemplo culminar características prioritarias del sistema en fecha [41].

Martin Fowler propone que la Deuda Técnica es similar a la deuda financiera en la cual se debe pagar intereses, los mismos se representan en esfuerzo extra que se debe hacer en un futuro desarrollo por no haber seleccionado un diseño limpio [3].

Guo, Seaman, et al., por su parte sostienen que los proyectos de mantenimiento de software son a menudo ajustados por el presupuesto, fechas y recursos. Para hacer frente a esos ajustes, se hacen compensaciones durante todo el ciclo de desarrollo del software. Cuando esas compensaciones permiten a los equipos del proyecto hacer frente a las expectativas de la dirección y del cliente en el corto plazo, el compromiso resulta en

costo adicional posterior en el proceso de mantenimiento. Este fenómeno es conocido como "Deuda Técnica" [42].

En los últimos años se produjo un incremento en el conocimiento empírico acerca de la influencia y utilidad de la metáfora Deuda Técnica en Ingeniería de software. Es a partir de aquí que se comienza a ampliar el espectro de fenómenos de la Ingeniería de software los cuales pueden explicarse a través de la metáfora Deuda Técnica. En el artículo presentado por Villar [13], este clasifica el alcance de Deuda Técnica de Código, Arquitectura, Diseño, Pruebas, etc. La Deuda Técnica de Código, fue a la primera Deuda Técnica a la cual se hace referencia, es a este tipo de deuda que refiere Cunningham [1] cuando introduce el termino en el 92. El proceso de Ingeniería de Software sobre el cual en estos días existe mayor información y estudios realizados es sobre la Deuda Técnica generada a nivel de código Villar [13]. Cuando se refieren a Deuda Técnica de código en la mayoría de los casos se hace hincapié en la forma en que pobremente se escribe el código. Deuda Técnica seria en este caso, la diferencia existente entre el óptimo de cómo escribir código y cómo realmente se escribió el mismo. También existen la deuda de arquitectura y la deuda de diseño, muchas veces no se focaliza o no se prevé el mejor diseño y arquitectura para que el sistema pueda crecer a futuro sin necesidad de hacer un rediseño o en caso de la arquitectura que la misma permita el crecimiento sin necesidad de cambiar la arquitectura propuesta en principio. Ejemplos de estos se encuentran en: los artículos referentes a deuda de código [42], [43], [41], [44] de diseño y arquitectura [45] [46] [47], de pruebas [48][49].

Además de tener impacto en dimensiones del desarrollo de software, también existen otras facetas sobre las que impacta la Deuda Técnica, la cual genera una deuda financiera. Se impactan dimensiones como por ejemplo, la moral del equipo, la productividad, la calidad y el riesgo del proyecto.

En resumen, la metáfora de Deuda Técnica es útil porque es capaz de explicar fenómenos que ocurren durante la producción de software. A la industria del software le ha resultado útil porque permite, mediante la utilización de términos de uso cotidiano, la comunicación de problemas. A la academia le resulta útil, porque la Deuda Técnica es representativa con el espacio actual de conocimiento de la Ingeniería del Software, en donde es posible seguir modas sin que estas estén soportadas por teorías. Este estado presenta espacio para el aporte de conocimiento empírico a los fenómenos de producción de software que pueden ser descritos con la metáfora [2].

4 Métodos de investigación y planteamiento

Este capítulo resume los métodos de investigación utilizados. En primera instancia se utiliza el método cuantitativo encuesta. Este método es utilizado en el mecanismo de medición de agilidad que se plantea. A través de la encuesta se obtienen las respuestas de los distintos integrantes del equipo respecto a las consideraciones que cada uno de ellos posee del proyecto de desarrollo del cual forman parte. En segunda instancia se ha utilizado en la tesis la Teoría Justificada, en especial el método *open coding* de la misma. Mediante su utilización se permite analizar cuáles de los factores vinculados a Deuda Técnica se pueden identificar en las instancias de reflexión del proyecto seleccionado en la presente investigación.

4.1 Métodos cuantitativos. Encuesta

El método cuantitativo Encuesta fue seleccionado como método para la recolección de la información de entrada para el mecanismo de medición de agilidad que se propone en el presente trabajo.

La idea básica de la metodología de la encuesta es recoger información de un grupo de personas mediante el muestreo de individuos de una población grande. Ejemplos de las encuestas se encuentran en la vida cotidiana en varias situaciones, las encuestas sociales, las encuestas electorales, etc. En el caso del presente trabajo se debe aplicar una encuesta a nivel de ingeniería de software, la metodología para esta disciplina es similar a las realizadas en otras disciplinas [50].

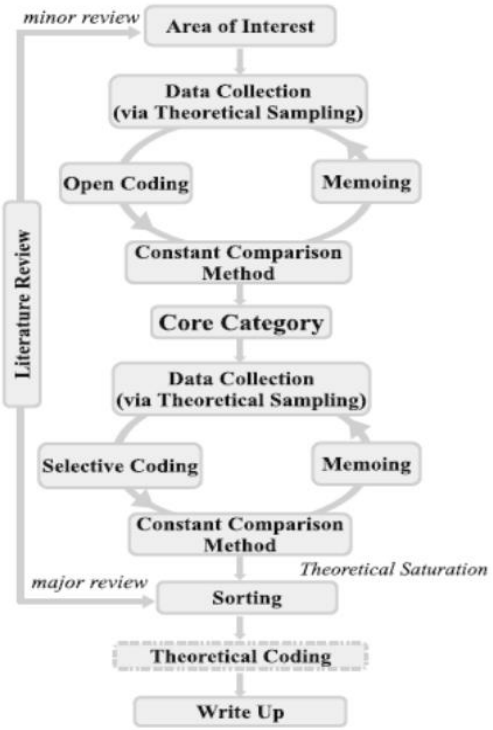
Para utilizar la encuesta como método de recabar datos se debe en primer lugar definir las preguntas de investigación sobre la base de la necesidad de investigación. Las preguntas deben tener las siguientes características, ser abiertas y responder al interés del investigador. El segundo paso a realizar es definir la población objetivo y definir el tamaño de la muestra que se va a utilizar.

Una vez que la población de la muestra elegida ha respondido a las preguntas en el instrumento, es que se puede comenzar el análisis los datos los cuales brindaran respuesta a la necesidad planteada, también son útiles para sacar conclusiones basadas en el análisis realizado a los mismos. Luego de obtenidos y analizados los datos se está en condiciones de confeccionar el informe.

4.2 Teoría Justificada

La Teoría Justificada fue la metodología y la técnica de codificación que se seleccionó para utilizar en la investigación. En esta sección se describe la metodología y la técnica de codificación utilizada. En la Imagen 2, se puede observar gráficamente el proceso que propone Hoda [10] como proceso general de Teoría Justificada para su aplicación en Ingeniería de software.

Imagen 2- Proceso Teoría Justificada propuesto por Hoda [10]



A continuación se presenta una breve descripción que refiere a cada una de las partes que componen el proceso propuesto:

Area of Interest and the Research Question: el método de Teoría Justificada tiene como objetivo generar teoría, es por esto que recomienda abstenerse de generar una pregunta de investigación con antelación, esto para evitar influencias de la literatura existente. El investigador en lugar de formularse una pregunta de investigación debe identificar un área de interés general.

Literature Review: se recomienda que el investigador debe comenzar con la recogida periódica de datos, codificación y análisis sin ningún problema preconcebido, un capítulo o métodos de una extensa revisión de bibliografía en la misma área sustantiva.

Data Collection (via Theoretical Sampling): muestreo teórico se denomina en Teoría Justificada a la recolección de datos. El muestreo teórico es el proceso de recopilación de datos para la generación de teoría mediante el cual el analista recoge de forma conjunta, los códigos, y analiza sus datos y decide qué datos se recogen a continuación y donde se encuentran los mismos, con el fin de desarrollar su teoría a medida que emerge.

- ***Recruiting participants***: encontrar a los participantes puede ser difícil, un gran desafío. El muestreo teórico es un proceso continuo que ayuda a decidir qué datos recopilar próximo sobre la base de la teoría emergente.

- ***Interviews and Observations:*** el dictamen en la teoría justificada es que "todo es datos" y, como tal varias fuentes de datos pueden ser utilizadas. Los datos cualitativos obtenidos a través de entrevistas y observaciones, son la forma más popular.

Data Analysis: se puede comenzar el análisis de datos llamado codificación en el de Teoría Justificada tan pronto como se han recogido algunos datos. Hay dos tipos de códigos que se producen como resultado de análisis o codificación de datos: códigos sustantivos y códigos teóricos. Los códigos sustantivos son "las categorías y las propiedades de la teoría que emerge de imágenes y conceptualmente del área sustantiva en fase de investigación. Por el contrario, los códigos teóricos "los códigos sustantivos se relacionan entre sí como un modelo, interrelacionado conjunto, multivariado de hipótesis en la contabilización de la resolución de la principal preocupación".

- ***Open Coding:*** técnica que sirve para generar conocimiento o abstracciones a partir de una fuente de datos. Típicamente se utilizan transcripciones de entrevistas o documentos.

- ***Constant Comparison Method***: Los códigos que deriven de cada entrevista se comparan contra los códigos de la misma entrevista, y con los de otras entrevistas y observaciones. Esto es constante en el método Teoría Justificada, la comparación que se utilizó de nuevo para agrupar estos códigos y producir un nivel de abstracción más alto, llamado conceptos en el proceso.

- ***Core Category:*** *open coding* se caracteriza por la aparición de una categoría *Core* al final. La categoría central capta la principal preocupación de los participantes que se convierte en el problema de investigación.

  Algunos criterios para la elección de una categoría son: debe ser el centro en relación con otras categorías y sus propiedades, o debe aparecer con frecuencia en los datos, o se relaciona de manera significativa y fácilmente con otras categorías.

- ***Selective Coding:*** luego que se establece la categoría de núcleo, el investigador utiliza la codificación selectiva, un proceso que implica la codificación selectiva para la variable de núcleo mediante la delimitación de la codificación de "sólo aquellas variables que se relacionan con la variable de núcleo de manera suficientemente significativas como para producir una teoría parsimoniosa. La categoría central orienta aún más la recopilación de datos, análisis y muestreo teórico.

Memoing: es el proceso de escribir memos teóricos a lo largo del proceso de Teoría Justificada. Memos son "notas teóricas sobre los datos y las conexiones conceptuales entre las categorías". Memos tienden a ser las ideas fluyen libremente del investigador sobre los códigos y sus relaciones.

Sorting: cuando la recopilación de datos se encuentra casi terminada y la codificación está casi saturada, se puede comenzar a clasificar conceptualmente los memos teóricos. La clasificación es un paso fundamental y no se puede perder. La ventaja de la clasificación es que "pone los datos de fractura de nuevo juntos". El investigador debe tener en cuenta que ellos necesitan para ordenar las ideas y no de datos. La clasificación debe hacerse en un nivel conceptual que resulta en un esbozo de la teoría en términos de cómo las diferentes categorías se relacionan con el núcleo-categoría.

Theoretical Coding: se define como la propiedad de la codificación y el análisis comparativo constante que produce la relación conceptual entre las categorías y sus propiedades a medida que surjan. La misma implica conceptualizar cómo las categorías se relacionan entre sí como una hipótesis para ser integrados en una teoría.

Write–up: el último paso propuesto en el proceso es escribir la teoría, que sigue el teórico generado como resultado de la clasificación y codificación teórica [10].

4.3 Planteamiento

El objetivo de la investigación es la Identificación de Deuda Técnica en instancias de reflexión de Proyectos Ágiles de Desarrollo de Software

Para lograr el objetivo planteado, existió la necesidad de descomponer el mismo en partes. En primera instancia se necesitaba obtener los factores de Deuda Técnica que existen para poder luego identificarlos en las instancias de reflexión, estos surgen de un trabajo realizado previamente dentro del equipo de investigación [12].

Sobre la base de que se debía obtener instancias de reflexión de un proyecto de desarrollo ágil es que surge la necesidad de poder medir que grado de agilidad tenían los proyectos candidatos a ser entradas del presente trabajo. Es a partir de esta realidad que surge la necesidad de contar con un mecanismo de medición de grado de agilidad. Se propone un mecanismo de medición del grado de agilidad, el cual permite comparar instancias de proyectos independientes de las metodologías que utilicen [9].

Una vez que se finaliza el mecanismo de medición de agilidad se ejecuta la validación de este, es a partir de los resultados obtenidos (ver capitulo, 6.1.3) en esta etapa que se selecciona el proyecto del cual se utilizaran las instancias de reflexión.

En el momento que se cuanta con el proyecto a evaluar, se determina como metodología de análisis a la Teoría Justificada, descripta en la sección 4.2. Realizada la evaluación de la metodología se propone adaptarla a la necesidad del trabajo, la adaptación realizada se describe en la sección 7.

5 Proceso de Diseño del Mecanismo de Medición de Agilidad

Como se expresó en la sección 4.3, el mecanismo de medición de agilidad se propone como solución a la necesidad de identificar el grado de agilidad de los proyectos de desarrollo de software, es por este motivo que el objetivo del mecanismo es medir el Grado de Agilidad de un proyecto. Por lo expresado anteriormente, es claro que el elemento de interés es el proyecto y no la metodología que se indique que se sigue en cada proyecto. El objetivo al que se quiere arribar finalmente con el mecanismo planteado es lograr comparar proyectos, de modo que permita agregar resultados de diferentes proyectos de desarrollo. Los contenidos expuestos en este capítulo fueron publicados en Nacimento [9].

El propósito de la creación del mecanismo de medición es que a través del mismo se permita ordenar los proyectos de desarrollo de software según la definición (establecida y descrita en este trabajo) del Grado de Agilidad. El asunto principal para la construcción del mecanismo de medición es la conceptualización de que "ser ágil" no es una pregunta binaria, sino que la agilidad puede darse en un rango de valores los que probablemente estén condicionados por el entorno en que el proyecto se encuentra posicionado. Por lo cual, corresponde considerar que debería ser posible comparar la agilidad de dos proyectos de desarrollo de software teniendo en cuenta el grado de apego a los valores y principios propuestos en el manifiesto ágil [8] (ver Anexo 1).

Debido a lo explicado anteriormente, es que el mecanismo de medición de agilidad propuesto instaura una escala ordinal que posibilita ordenar proyectos de desarrollo a través del grado de agilidad de los mismos. El grado de agilidad se obtiene como resultado de aplicar el mecanismo de medición propuesto a los distintos proyectos de desarrollo.

Continuando con lo referente a la construcción del mecanismo, se puede apreciar que la mayoría de los mecanismos de medición citados en la sección 3.2.1 al igual que el construido en este trabajo se encuentran orientados a encuestas, las cuales son contestadas con frecuencia por individuos que se encuentran involucrados en entornos ágiles. En el presente trabajo se seleccionó como sistema de recolección de datos una encuesta, al igual que en los trabajos referenciados en 3.2.1. Sin embargo, nuestra metodología realiza un aporte dado que la misma se realizará a todos los integrantes del proyecto sin tener en cuenta los conocimientos o el uso de las metodologías ágiles que los distintos integrantes puedan tener previamente.

De lo expresado anteriormente en este capítulo, se puede resumir e infiere lo siguiente:

1. El mecanismo de medición de agilidad planteado, toma la definición de agilidad que se desprende del grado de alineación que posee un proyecto respecto a los valores y principios que se proponen en el manifiesto ágil [8].

2. A diferencia de otros mecanismos los cuales se enfocan en evaluar o medir las metodologías que los proyectos dicen utilizar, el mecanismo de medición de agilidad que se propone utilizar estará orientado a comparar instancias de proyectos. Lo que significa fundamentalmente que no es el objetivo final de este trabajo comparar definiciones teóricas de diferentes metodologías ágiles, sino que el propósito es comparar como dos o más proyectos implementan sus procesos de desarrollo teniendo en cuenta las restricciones del contexto donde se implementa cada proyecto.

3. El último punto del resumen refiere al método de recolección de datos seleccionado en este trabajo, el mismo se basa en encuestas a los miembros de los proyectos a ser medidos.

Él mecanismo de medición de agilidad que se plantea en el presente trabajo, se compone de dos partes principales: en primera instancia un método de recopilación de datos (Encuesta) y como segunda parte, lo que compone el trabajo es la fórmula para el cálculo de grado de agilidad a partir de los datos obtenidos en la encuesta (Métrica de Agilidad).

En el Anexo 1, se pueden apreciar en la tabla los principios del Manifiesto Ágil [8] los cuales se inspiran en los valores que se proponen en el mismo. A partir de los principios se pueden identificar los procesos ágiles, los mismos son características típicas de los procesos ágiles las cuales permiten diferenciar un proceso ágil de un proceso tradicional.

5.1 Diseño de las preguntas del mecanismo de recopilación de datos

Para asegurar que el mecanismo de medición efectivamente estuviese alineado con los preceptos del Manifiesto Ágil [8], cada pregunta que compone el mecanismo de medición de agilidad propuesto se encuentra mapeada a un valor o principio del manifiesto. Un ejemplo, es la siguiente pregunta PR20: “Considera que el equipo logró el involucramiento del usuario/cliente, logrando que respondiera consultas, planificara iteraciones y colaborara en la especificación de los requerimientos y pruebas.” En el Anexo 1 se encuentra el detalle.

Cada pregunta de la encuesta del mecanismo será valuada en una escala Likert con la siguiente interpretación.

1. **Nunca**, corresponde a que no se hace ningún tipo de actividad al respecto.

2. **Pocas veces**, cuando se realiza en un rango del 1% al 39%.

3. **Neutro**, cuando se ejecuta entre un 40 y 60 %.

4. **Muchas veces**, cuando se hace entre un rango del 61% al 98%.

5. **Siempre**, cuando se realiza más de un 98% de las veces.

A continuación, se presentan las preguntas que se diseñaron para conformar la encuesta. Dentro de las cuales, se incorporaron preguntas que son conceptualmente opuestas para el contexto de los principios y valores ágiles (estas se encuentran identificadas por un asterisco en el campo ID de la Tabla 5-1).

Tabla 5-1. Preguntas del mecanismo de medición de agilidad.

ID	Pregunta	ID	Pregunta
PR1	Se definieron claramente los roles, responsabilidades, conocimientos técnicos e interacciones entre miembros del equipo de trabajo.	PR14*	Se realizaron las mismas actividades en cada iteración.
PR2	Existió comunicación fluida en el equipo.	PR15	Se generaron entregables con testing satisfactorio e integrado con el resto de las funciones al finalizar cada iteración.
PR3	Existió comunicación Cara a Cara.	PR16	Se realizaron revisiones de a pares o inspecciones.
PR4	El usuario trabajó conjuntamente con el equipo.	PR17	Se estableció un plan de entrega o cronograma de revisiones modificable.
PR5	Existió motivación en el equipo.	PR18	Se priorizaron las necesidades de los entregables frente al grado de apego a los estándares de documentación.

ID	Pregunta	ID	Pregunta
PR6	Se autoorganizó el equipo.	PR19*	Se requirió documentación para comenzar a implementar la funcionalidad incluida en cada iteración.
PR7	Se reflexionó sobre la forma de que el equipo fuera más efectivo y ajustó la conducta en consecuencia dentro de cada intervalo.	PR20	Considera que el equipo logró el involucramiento del usuario/cliente, logrando que respondiera consultas, planificara iteraciones y colaborara en la especificación de los requerimientos y pruebas.
PR8	Se priorizaron las tareas.	PR21	Se priorizó la satisfacción del cliente.
PR9	Los entregables fueron incrementales.	PR22*	Se definió un contrato para la totalidad del producto.
PR10	Se realizaron las entregas con frecuencia constante (frecuencia varía: 1 semana a 2 meses).	PR23*	Se previeron cláusulas para cambios adicionales en el contrato.
PR11	Se involucró el usuario en el proceso.	PR24	Se permitió introducir cambios en cada una de las iteraciones.
PR12	Se generaron pruebas automáticas.	PR25	Se permitieron cambios durante cada iteración.
PR13	Se realizaron pruebas unitarias.	PR26	Se definió un plan detallado que incluyó recursos, fechas y responsables.

5.2 Valoración de las respuestas (Métrica de Agilidad)

En concordancia con lo expresado en la sección 5.1, las preguntas que componen la métrica se encuentran asociadas al manifiesto, cada una de ellas implica una valoración. El método utilizado para el cálculo de la métrica es el promedio ponderado de los valores recibidos en cada respuesta. Vale recordar que el mecanismo se componen por ciertas preguntas que refieren al concepto opuesto, en estos casos el valor que se utiliza en el cálculo es el valor absoluto de la diferencia entre el valor de la respuesta con respecto a la escala. El resultado final de la métrica genera un valor de una escala ordinal.

Para un mejor entendimiento de lo propuesto se brinda un ejemplo del calculo que se efectúa en la métrica para una pregunta marcada con *, la cual es una pregunta por

opuesto. En este caso, si se ponderó en 4 la pregunta, se considera el valor absoluto de (5 - 4 = 1) siendo (5) el valor máximo de la escala, obteniendo como resultado valor absoluto en 1, luego el valor obtenido se debe sumar al total y promediar. La fórmula utilizada en la métrica es la que se expone a continuación:

$$\text{Métrica} = \frac{\sum_{i=1,j=1}^{i=4,j=n1} P_{ij} + \sum_{i=1,j=1}^{i=4,j=n2} |Esc - P_{i@j}|}{n1+n2}$$

Pij refiere a una pregunta de la encuesta, i alude al valor correspondiente al punto Vi del manifiesto ágil y j al índice correlativo de la pregunta. Mientras que Pi@j describe a la pregunta que se opone al concepto Vi del manifiesto ágil. La ponderación de una pregunta Pi@j deberá ser opuesta a una Pij.

5.3 Validación del mecanismo

Una vez implementado el mecanismo de medición de agilidad fue sometido a tres instancias de validación. La validación fue realizada en tres entornos diferentes, dos validaciones se realizaron en el ámbito académico, mientras que la última instancia se realizó a nivel de la industria. En cada instancia en que se ejecutó el mecanismo, se verificó la facilidad de los participantes para completar la encuesta, y la capacidad del algoritmo de determinar los tipos de proyectos. En el presente capitulo se describen los diferentes ámbitos de validación, mientras que en el siguiente capítulo se presentarán los resultados obtenidos en cada evaluación. En la Universidad ORT Uruguay se llevaron a cabo las dos primeras instancias de validación, en dichas instancias participaron grupos de estudiantes de la universidad. En la primera etapa de validación, participaron estudiantes de grado y postgrado de la asignatura Desarrollo Ágil de Software. Para la segunda etapa de aplicación del mecanismo los participantes fueron estudiantes que realizaban sus proyectos de fin de carrera de Licenciatura e Ingeniería en Sistemas. En última instancia los participantes eran integrantes de proyectos de desarrollo de empresas de software uruguayas.

Los alumnos de la asignatura Desarrollo de Software Ágil de la Universidad ORT Uruguay fueron los primeros en participar en la evaluación. La asignatura forma parte de las electivas que se proponen a los alumnos de las carreras de Ingeniería de Software y Licenciatura en Sistemas. Además, dicha asignatura puede ser seleccionada por alumnos

de post-grado, en cuyo caso tendrán una carga horaria ampliada. En la asignatura, a todos los alumnos se les planteo realizar un trabajo obligatorio que consistía en un proyecto de desarrollo con iteraciones de duración fija, con cambios de requerimientos introducidos por los docentes. A pesar de no ser requisito de los docentes, el método de desarrollo que siguieron la mayoría de los equipos de trabajo estaba basado en *Scrum* [22]. El trabajo propuesto se ejecutó a lo largo de un mes y se realizaron cuatro iteraciones por parte de los distintos grupos. Cada uno de ellos se encontraba conformado por dos alumnos. A cada integrante del equipo se le administró el mecanismo de medición. Los mismos completaron la encuesta en base a la experiencia vivida durante el mes de trabajo.

Luego de realizada la primera experiencia de ejecución, el mecanismo fue sometido nuevamente a una validación, en esta ocasión se ejecutó en los proyectos finales de las carreras de Licenciatura en Sistemas e Ingeniería en Sistemas de la Universidad ORT Uruguay. Para los proyectos de fin de carrera se busca simular entornos reales de desarrollo, los cuales se enmarcan en el contexto del laboratorio ORT Software Factory (ORTSF) [51]. Los proyectos que se realizan dentro del contexto de ORTSF deben contar con un cliente real, dicho cliente es quien presenta el problema a ser resuelto por parte de los proyectos. El laboratorio ORTSF es el que se encarga de gestionar la selección de estos clientes, los cuales en su mayoría son representantes de empresas productoras de software uruguayas. Existen también, casos de proyectos que son emprendimientos comerciales de los alumnos, para los mismos se busca el compromiso de un cliente final real que valide los requerimientos. Los proyectos se realizan por equipos integrados entre 3 y 5 alumnos los cuales son acompañados durante su ciclo de vida por un tutor. A los alumnos se los evalúa según su capacidad de generar software con calidad de producción, por su habilidad de diseñar el proceso de desarrollo que solucione el problema que se les asigna y genere el software requerido. Por las razones expuestas anteriormente, los investigadores no tienen control del proceso de desarrollo seguido por los equipos a la hora de realizar los proyectos. Son los alumnos quienes tienen la potestad de decidir que metodología es la mejor a la hora de solucionar el problema del cliente, si es una ágil o una tradicional. El entorno descripto hace que los proyectos desarrollados dentro del laboratorio de ORTSF sean buenos ambientes semi-controlados para la evaluación de experiencias antes de que sean probadas en la industria con desarrolladores profesionales. Asociado a los objetivos de esta investigación al comparar el resultado del mecanismo de medición en entornos ágiles y tradicionales. Además, en esta instancia se administró la encuesta a los tutores de cada proyecto de forma de poder comparar la respuesta de sus grupos con la del tutor.

Finalmente, con los resultados obtenidos, se realizó una instancia de validación a nivel de la industria, para ello se cursó una invitación a empresas de desarrollo de software

para que respondan a la encuesta. De esta forma, se obtuvieron respuestas de tres proyectos los cuales pertenecen a distintas empresas involucradas en el desarrollo de software en la industria uruguaya. Anteriormente a que se aplicará la encuesta en cada empresa, se realizó una visita que permitió evaluar el ambiente, dicha evaluación proporciono los medios para poder realizar una comparación cualitativa de los resultados obtenidos luego.

A continuación aportaremos información de cada una de las empresas participantes manteniendo el anonimato de las mismas por lo cual de ahora en adelante se denominaran empresa 1, empresa 2 y empresa 3. La primera empresa en participar respondiendo la encuesta, la empresa 1, se dedica al desarrollo de software en el sector financiero-bancario. El proyecto que fue seleccionado para evaluar en dicha empresa, realizo su desarrollo bajo una metodología ágil basada en *Scrum*, la duración del proyecto evaluado fue de 6 meses durante los cuales se ejecutaron 15 *Sprints*. La encuesta fue respondida por la totalidad de los participantes involucrados en el proyecto. La segunda empresa que participo de la evaluación, la empresa 2, es una empresa que se dedica al desarrollo de software a medida, dicha empresa basa sus proyectos en una metodología basada en *Scrum*. En el caso del proyecto participante, el mismo tuvo una duración de 5 meses en los cuales se llevaron a cabo un total de 13 *Sprints*. En dicho proyecto, al igual que en el anterior se obtuvieron respuestas de la totalidad de los integrantes del equipo del mismo. La última empresa participante, la empresa 3, es una empresa del sector financiero-bancario que tiene su propia fábrica de software. El proyecto evaluado por tanto corresponde a un desarrollo interno de dicha institución. En este último proyecto evaluado a diferencia de los anteriores, se encontró que el mismo no utiliza una metodología bien definida pero sin embargo presenta ciertas características – como ser la presencia del cliente final, la escasa documentación – que pueden interpretarse como proveniente de las metodologías ágiles. La duración del proyecto fue de 8 meses, en dicho lapso de tiempo se realizó la entrega total del mismo.

Los resultados de las validaciones que se llevaron a cabo se encuentran publicados en la sección 6.1 Resultados de la medición de agilidad.

6 Experimentación

6.1 Resultados de la medición de agilidad

En el presente capítulo del trabajo se presentan los resultados obtenidos durante la ejecución de las validaciones realizadas al mecanismo de medición de agilidad, tal cual se mencionó en el capítulo de la Validación del mecanismo bajo el subtítulo Instancias de validación del mecanismo de medición de agilidad. Los resultados se expondrán en el mismo orden que se presentaron las validaciones en el capítulo anterior. Los contenidos de este capítulo se encuentran publicados en el artículo perteneciente a Nacimento [9].

6.1.1 Resultados de la evaluación con alumnos

De acuerdo a lo expresado anteriormente, la primera validación del mecanismo de medición de agilidad se llevó a cabo en el segundo semestre del año 2012. Dicha validación fue realizada por los alumnos de la asignatura Desarrollo Ágil de Software, luego que dichos alumnos realizaran un proyecto en el que se utilizó una metodología de desarrollo ágil.

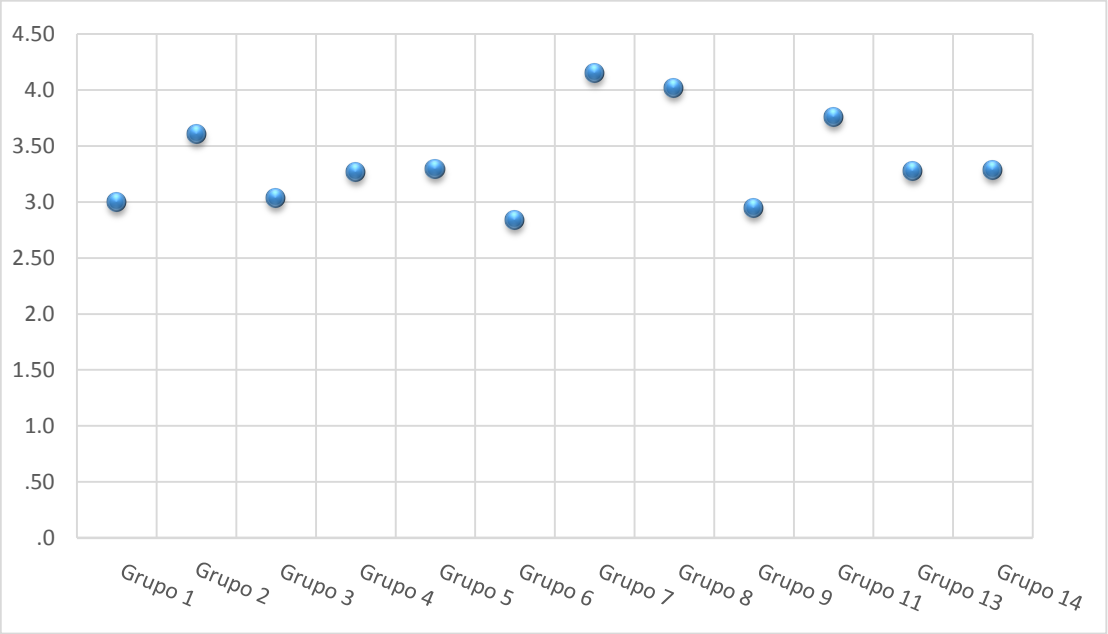
En la Tabla 6-1, se presentan los datos correspondientes a la primera evaluación del mecanismo. Cada celda representa el promedio de las respuestas de los dos integrantes que componían cada uno de los grupos.

Tabla 6-1. Resultados de evaluación de alumnos de desarrollo ágil de software.

Valores	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5	Grupo 6	Grupo 7	Grupo 8	Grupo 9	Grupo 11	Grupo 13	Grupo 14
Individuos e interacciones sobre procesos y herramientas	3.4	3.7	3.2	2.9	3.0	2.9	4.1	3.6	3.4	3.2	3.2	3.3
Software funcionando sobre documentación extensiva	2.6	3.5	2.8	2.9	2.5	2.9	4.2	3.8	3.1	3.2	2.6	2.5
Colaboración con el cliente sobre negociación contractual	3.5	3.9	3.1	4.1	3.6	3.4	4.3	4.4	2.5	4.0	3.5	4.0
Respuesta ante el cambio sobre seguir un plan	2.5	3.3	3.0	3.2	4.0	2.7	4.0	4.3	2.8	4.7	3.8	3.3
Agilidad Total	**3.0**	**3.6**	**3.0**	**3.3**	**3.3**	**2.8**	**4.2**	**4.0**	**2.9**	**3.7**	**3.3**	**3.3**

Si se realiza un análisis la gráfica obtenida, ver Gráfica 1, se puede observar que los resultados de la mayor parte de los grupos de la materia logran un puntaje que supera o iguala el valor medio de la escala diseñada, su valor se encuentra por encima de 3. Teniendo en cuenta que el contexto en el que se desarrollaron estos proyectos favorecía los Ciclos de Vida asociados a las metodologías ágiles, era esperable obtener resultados que se ubicaran dentro de los valores más altos de la escala.

Gráfica 1. Trabajos de asignatura desarrollo ágil de software.



6.1.2 Resultados de la evaluación con proyectos de grado

La segunda evaluación se realizó con los proyectos de fin de carrera de Ingeniería de Sistemas y Licenciatura en Sistemas, los proyectos seleccionados fueron los que comenzaron en los meses de Marzo y Agosto del año 2013.

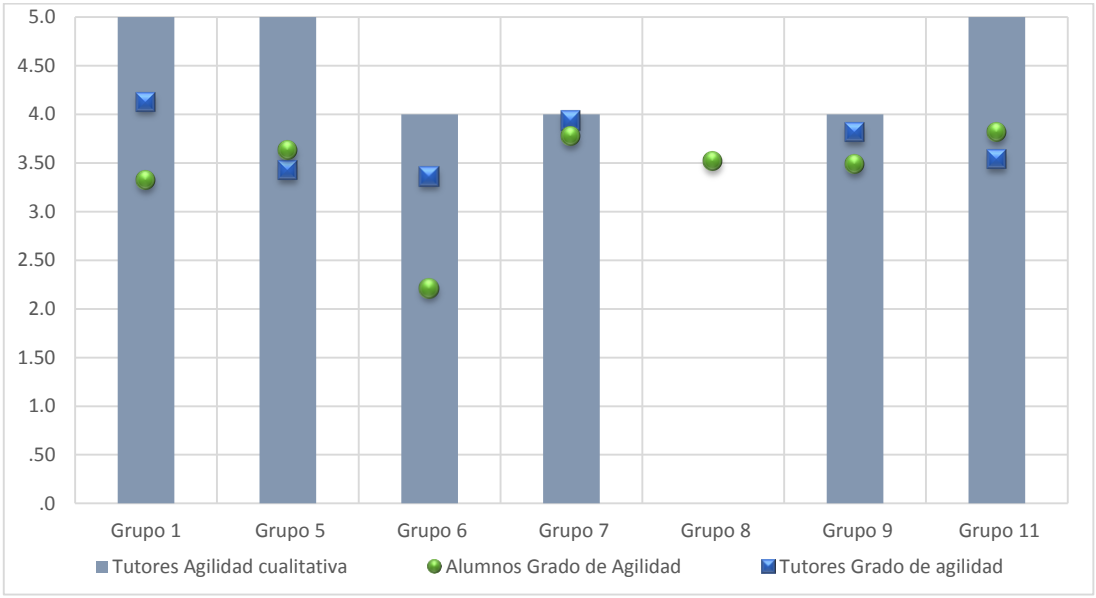
En esta evaluación participaron los tutores y los alumnos de los proyectos seleccionados. El mecanismo debió ser completado por el tutor, en cuyo caso contaba con una pregunta extra cuyo fin era determinar la perspectiva del tutor respecto a si su equipo se manejó dentro del contexto de las metodologías ágiles de desarrollo de software o no. La pregunta brinda una evaluación independiente de los resultados obtenidos mediante la aplicación del mecanismo, propia del tutor. Los tutores brindaron una opinión de experto en la instanciación del proceso de evaluación.

Los resultados obtenidos en esta nueva evaluación se presentan en la Tabla 6-2, en la misma se exponen los resultados recabados a los equipos y su tutor. En la Gráfica 2, se visualizan los datos de los proyectos de los que respondieron dos o más integrantes del equipo, ya que es el mínimo para poder calcular el grado de agilidad. Se aprecian los resultados del tutor y del equipo (representados por un cuadrado y círculo). La barra graficada representa la respuesta a la pregunta cualitativa realizada a los tutores. Permitiendo de este modo comparar los resultados del mecanismo contra la pregunta cualitativa a los tutores.

Tabla 6-2. Resultados de evaluación con proyectos de fin de carrera.

Valores	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5	Grupo 6	Grupo 7	Grupo 8	Grupo 9	Grupo 10	Grupo 11	Grupo 12
Alumnos en el Grupo	7	5	3	4	3	4	3	4	3	3	5	3
Numero de respuestas	4	1	1	1	2	2	3	4	2	1	2	1
Individuos e interacciones sobre procesos y herramientas	3.4	3.4	3.1	4.0	3.4	2.5	3.9	3.5	3.4	3.4	3.4	3.1
Software funcionando sobre documentación extensiva	2.8	3.8	2.6	3.6	2.5	1.8	3.6	3.0	3.6	2.4	3.8	2.0
Colaboración con el cliente sobre negociación contractual	4.0	2.3	3.5	2.8	4.1	3.0	4.1	4.1	3.1	3.3	4.3	4.5
Respuesta ante el cambio sobre seguir un plan	3.1	2.3	4.3	3.3	4.5	1.6	3.6	3.6	3.8	4.3	3.8	3.3
Alumnos Grado de Agilidad	**3.3**	**2.9**	**3.4**	**3.4**	**3.6**	**2.2**	**3.8**	**3.5**	**3.5**	**3.3**	**3.8**	**3.2**
Tutores Grado de Agilidad	**4.1**	**3.2**	**3.6**	**3.6**	**3.4**	**3.4**	**3.9**	**-**	**3.8**	**3.6**	**3.5**	**3.5**
Tutores Agilidad Cualitativa	**5.0**	**1.0**	**3.0**	**4.0**	**5.0**	**4.0**	**4.0**	**-**	**4.0**	**5.0**	**5.0**	**5.0**

Gráfica 2. Medición de agilidad de proyectos de fin de carrera.



En la gráfica se puede observar, en primer lugar que el rango de valores obtenido es semejante al publicado en 6.1.1. Como se aprecia en la Gráfica 2, la mayoría de los resultados alcanzados se sitúan por encima del valor medio de la escala utilizada (3), tanto en el resultado obtenido de las respuestas de los equipos como los resultados de los tutores. Otro dato destacado que surge del análisis es que la valoración de los tutores y alumnos, es casi coincidente en un 72% de los casos evaluados (solo dos tienen un resultado con una desviación mayor a un punto).

Al examinar los resultados se nota que la pregunta de control realizada a los tutores de los proyectos sobre la agilidad percibida, en general es consistente con los resultados de medición. Existe una excepción en el grupo 6, en cuyo caso se realizó el análisis causal correspondiente, se estudió la documentación del proyecto donde sus autores describen que la metodología utilizada en el mismo era un hibrido entre tradicionales y ágiles [52].

6.1.3 Resultados de la evaluación con proyectos de la industria

Finalmente, se realiza la evaluación del método a nivel de la industria de desarrollo de software, la misma se realizó en dos etapas de acuerdo a la disponibilidad de las empresas participantes, la primera etapa se llevó a cabo en marzo de 2014 mientras que la segunda se ejecutó en diciembre del mismo año.

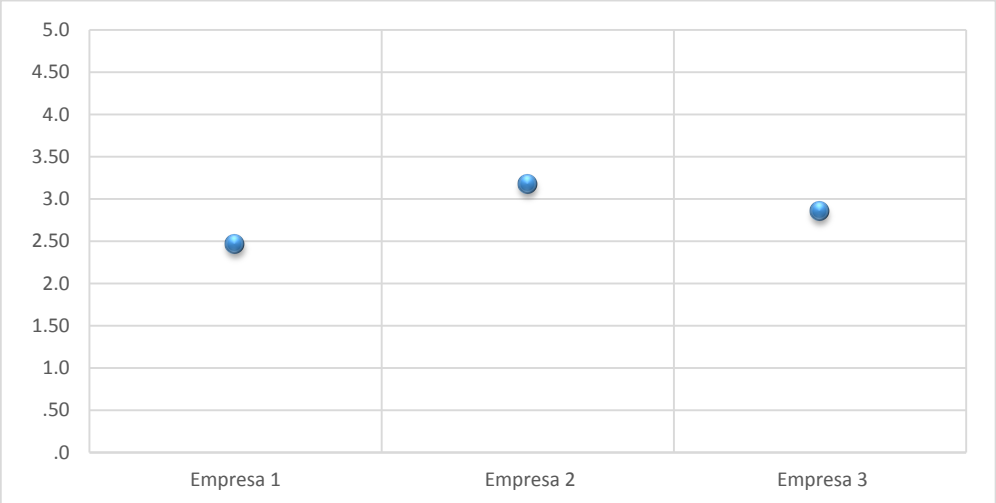
Repasando lo expresado en la sección 5.3, en esta etapa de validación participaron tres empresas de la industria de desarrollo de software. La empresa 1 y la empresa 3 pertenecen al sector financiero-bancario, mientras que la empresa 2 se dedica al desarrollo de software a medida. La empresa 1 y la empresa 2 utilizan una metodología basada en *Scrum*, en los proyectos que participaron en la evaluación del mecanismo de medición, entretanto que la empresa 3 no utiliza una metodología definida pero la forma en que se desarrolla en el proyecto participante presenta características de las metodologías ágiles.

A continuación se presentan los resultados alcanzados en estas dos intervenciones.

Tabla 6-3. Resultados de la evaluación de la industria del software.

Valores	Empresa 1	Empresa 2	Empresa 3
Número de respuestas	4	4	2
Individuos e interacciones sobre procesos y herramientas	3.2	3.9	3.0
Software funcionando sobre documentación extensiva	3.2	3.4	2.9
Colaboración con el cliente sobre negociación contractual	1.8	3.1	3.6
Respuesta ante el cambio sobre seguir un plan	1.7	2.3	1.9
Agilidad Total	**2.5**	**3.2**	**2.8**

Gráfica 3. Validación con Empresas.



Si se observan los resultados obtenidos mediante la aplicación del mecanismo de medición presentados en la

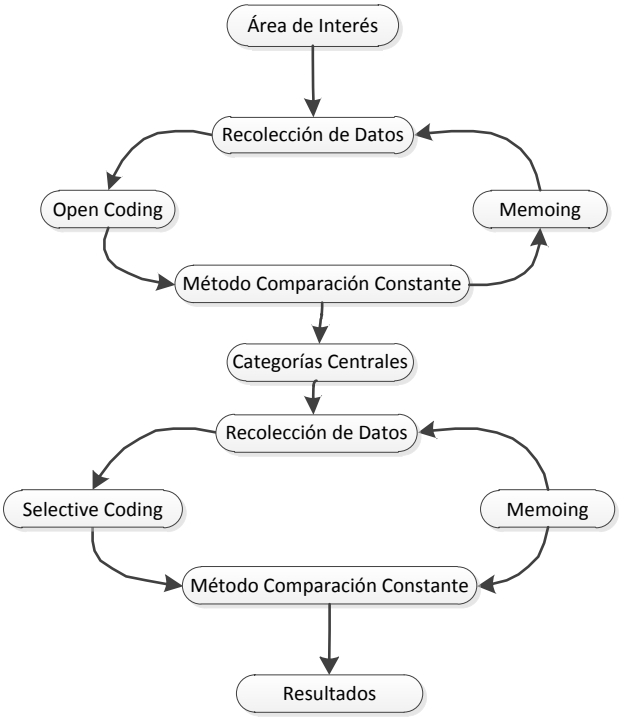
Gráfica 3, observamos que las tres empresas se clasifican entorno al valor medio de la escala (valor medio 3). Es interesante notar el resultado de la Empresa 3, la cual obtiene un Grado de Agilidad por encima de la media y netamente comparable con los resultados de las otras dos empresas que declaran utilizar metodologías de desarrollo ágiles. Si realizamos un análisis causal del resultado es importante notar que aunque no utilice una metodología ágil definida existen practicas propias de dichas metodologías que se aplican por lo cual el resultado obtenido es justificable.

7 Identificación de Deuda Técnica en Instancias de Reflexión de proyectos ágiles.

El objetivo de esta sección del trabajo es lograr identificar si en las instancias de retrospectivas de un proyecto de desarrollo ágil, existen constructos que pueden asociarse a la metáfora Deuda Técnica o a fenómenos que la literatura indica que pueden ser explicados con la metáfora de Deuda Técnica. Los resultados de esta sección fueron publicados en la Conferencia Latinoamericana en Informática Nacimento [11] y en Agile Brasil Nacimento [14]

El proceso de investigación que se ha utilizado es una adaptación del propuesto por Hoda [10] (ver sección, 4.2) para la aplicación de teoría justificada en Ingeniería de Software. En la Imagen 3, se visualiza el proceso de Teoría Justificada adaptado.

Imagen 3 - Proceso Teoría Justificada Adaptado



Recolección de Datos: la información utilizada como entrada para la realización de la codificación abierta (*Open Coding*) se obtiene de los informes de retrospectivas de un proyecto de desarrollo ágil perteneciente a la industria uruguaya de software. El proyecto

seleccionado aplicaba como metodología de desarrollo ágil *Scrum*. El equipo del proyecto para cada una de las 15 retrospectivas realizada generó un informe de dicha reunión. Es en estos informes que se registran a lo largo de todo el proyecto las problemáticas encontradas en el desarrollo de cada *sprint* junto con las soluciones propuestas por el equipo.

Codificación Abierta, Método de Comparación Constante y Memorándum: se realizaron tres iteraciones sobre los informes de retrospectivas. Durante la primera pasada se identificaron que tareas se podrían tomar como generadoras de Deuda Técnica en el proyecto. En la segunda revisión se chequearon las tareas contra los informes y las propias tareas identificadas en la pasada anterior. Fue así que se logró la depuración y la consolidación las tareas consiguiendo quedar con el conjunto de tareas relevantes para el estudio.

A continuación se presenta un ejemplo de las frases obtenidas del documento de retrospectiva del sprint número 7 del proyecto seleccionado para la investigación, las cuales luego se intentaran mapear a los factores de Deuda Técnica existentes. En el Anexo 2 se presenta la totalidad de frases encontradas a lo largo de los 15 sprints que componen el proyecto.

Tabla 7-1. Resultados de codificación abierta.

Sprint	Descripción de Temas y Acciones
7	Reservar un tiempo para hacer un análisis y revisar el código anterior para identificar y cubrir todos los casos
7	Si se detecta algo muy grande que exceda la estimación se separara lo nuevo y se escribirá una nueva historia para esa parte
7	Los bugs trabajados fueron ajustes de diseño y re trabajo que implico la incorporación de los mismos
7	Unificar criterios generales de forma que no se generen bugs similares en distintas funcionalidades
7	Dedicar tiempo para ver todo el proyecto de forma de encontrar criterios y realizar los ajustes generales
7	La cantidad de bugs reportados se explica porque se prueban varios Sprint a la vez
7	Se piensa que en futuros Sprint bajara y que los existentes se van a resolver mas rápido

Sprint	Descripción de Temas y Acciones
7	Reservar tiempo en la próxima planificación para resolver los bugs priorizando los que son parte de la primera liberación al cliente
7	Identificación y definición de estándares
7	Sacarle más jugo a las reuniones diarias con UI o arte
7	Comunicación más fluida entre el equipo y arte para minimizar las intervenciones del PM
7	Solicitar a arte apoyo en la diagramación de ciertas pantalla
7	Evaluar si vale la pena realizar ciertas reuniones o no
7	Definir objetivos para las reuniones y velar para que se cumpla al final

Categoría Núcleo: una vez que se identificaron las tareas relevantes de cada *sprint* se realizó el mapeo con los factores de Deuda Técnica que fueran identificados en un estudio llevado a cabo previamente [12], en esta etapa se analiza cada una de las tareas y se identifica cual o cuales de los factores pudo ser provocado por dicha tarea. Es a partir de las vinculaciones encontradas que se definen las categorías centrales, las cuales presentamos a continuación:

- Deuda Técnica aplicada a proyectos ágiles.
- Aplicar Deuda Técnica en decisiones de proyectos ágiles.
- Deuda Técnica en instancia de reflexión.
- Aplicar factores de costo de Deuda Técnica para instancias de reflexión en proyectos de desarrollo ágil.

Codificación Selectiva, Método de Comparación Constante y Memorándum: se define ampliamente como el estudio de la viabilidad de la aplicación de la metáfora de Deuda Técnica en instancias de reflexión de proyecto ágiles de desarrollo. Como podemos observar en la Tabla 7-2, se marca para cada una de las frases el o los factor es que asociamos a la misma.

Tabla 7-2. Resultado final de Teoría Justificada, Sprint 7.

Descripción de Temas y Acciones	**Factores**						
Reservar un tiempo para hacer un análisis y revisar el código anterior para identificar y cubrir todos los casos	Time to market	Refactoring	Deterioro del código /diseño	Pobre codificación en etapa de desarrollo	Deuda de Testing		
Si se detecta algo muy grande que exceda la estimación se separara lo nuevo y se escribirá una nueva historia para esa parte	Time to market	Pobre codificación en etapa de desarrollo	Calidad	Deuda de Testing			
Los bugs trabajados fueron ajustes de diseño y re trabajo que implico la incorporación de los mismos	Refactoring	Deterioro del código /diseño	Falta de tiempo / presupuesto	Añejamiento de Código	Falta de transparencia hacia otros stakeholders	Falta de conocimiento	Pasaje de defectos en un sprint (Defects carry over)
Unificar criterios generales de forma que no se generen bugs similares en distintas funcionalidades	Deuda de Testing	Falta de transparencia hacia otros stakeholders	Costos de implementación/re trabajo	Re trabajo	Falta de conocimiento	Deuda Funcional (funcionalidades prometidas no entregadas)	
Dedicar tiempo para ver todo el proyecto de forma de encontrar criterios y realizar los ajustes generales	Calidad	Deuda de Testing	Costos de implementación/re trabajo	Re trabajo			
La cantidad de bugs reportados se explica porque se prueban varios Sprint a la vez	Compromiso a corto y largo plazo (nivel código y diseño)	Documentación Obsoleta	Calidad	Deuda de Testing	Falta de transparencia hacia otros stakeholders	Falta de conocimiento	Deuda Funcional (funcionalidades prometidas no entregadas)
Se piensa que en futuros Sprint bajara y que los existentes se van a resolver más rápido	Compromiso a corto y largo plazo (nivel código y diseño)	Documentación Obsoleta	Calidad	Deuda de Testing	Pasaje de defectos en un sprint (Defects carry over)		
Reservar tiempo en la próxima planificación para resolver los bugs priorizando los que son parte de la primera liberación al cliente	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de Testing	Falta de transparencia hacia otros stakeholders	Falta de conocimiento	Pasaje de defectos en un sprint (Defects carry over)
Identificación y definición de estándares	Añejamiento de Código	Re trabajo					
Sacarle más jugo a las reuniones diarias con UI o arte	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*				
Comunicación más fluida entre el equipo y arte para minimizar las intervenciones del PM	Calidad	Deuda de *Testing*	*Like-to-like migration*				
Solicitar a arte apoyo en la diagramación de ciertas pantalla	Calidad	Deuda de *Testing*	*Like-to-like migration*				
Evaluar si vale la pena realizar ciertas reuniones o no	Calidad	Deuda de *Testing*					
Definir objetivos para las reuniones y velar para que se cumpla al final	Calidad	Deuda de *Testing*					

7.1.1 Factores de Deuda Técnica Identificados

En esta sección se presentan los resultados obtenidos del proceso de codificación y abstracción descripto anteriormente. En la Tabla 7-3, se presenta el resultado final del mapeo de los conceptos asociados a la metáfora con los factores de costo de Deuda Técnica.

Tabla 7-3. Factores, descripción y frecuencia.

Id. Factor	Descripción Factor de Deuda Técnica	Frecuencia
22	Deuda de *Testing*	104
19	Calidad	76
11	Documentación Obsoleta	42
4	*Time to market*	40
30	*Like-to-like migration*	26
35	Costo de Implementación	21
21	Añejamiento de Código	20
24	Falta de transparencia hacia otros *stakeholders*	20
1	Compromiso a corto y largo plazo (nivel código y diseño)	19
16	Poco esfuerzo y poca motivación	19
36	Re trabajo	17
5	*Refactoring*	15
10	Pobre codificación en etapa de desarrollo	14
49	Falta de conocimiento	12
6	Deterioro de la arquitectura	10
12	Mal control de versionado	8
54	Pasaje de defectos en un *sprint (Defects carry over)*	8
18	Obsolescencia tecnológica	7
33	Costos de implementación/re trabajo	7
55	Testear al final	6
58	Inexperiencia	6
39	Compromiso a corto y largo plazo (nivel arquitectura)	5
3	Bajo *refactoring* y rediseño	4

Id. Factor	Descripción Factor de Deuda Técnica	Frecuencia
17	Gerenciamiento de proyectos caótico	4
38	Compromiso a corto y largo plazo (Proceso)	4
7	Deterioro del código /diseño	3
9	Falta de tiempo / presupuesto	3
44	Cobertura de pruebas unitarias	3
48	Falta de mejora de procesos	3
56	Deuda Funcional (funcionalidades prometidas no entregadas)	3
43	Complejidad de código	2
46	Deuda de diseño	2
57	Código legado pobre	2
15	Cambios rápidos en tecnologías	1
27	Falta de fondos para realizar los cambios necesarios	1
28	Mala integración y manejo de liberaciones	1
32	Nuevas oportunidades de negocio	1
51	Energía	1
52	Corrección de defectos	1
60	Imprudencia	1

Mediante el trabajo de la aplicación de teoría justificada se mapearon los factores de Deuda Técnica con frases descriptas en los documentos de retrospectivas del proyecto estudiado.

Los factores identificados se muestran en la Tabla 7-3, en la cual se indica para cada factor con qué frecuencia se presentó el mismo durante el ciclo de vida del proyecto. Si se analizan los factores con más presencia se puede apreciar que los 8 primeros son típicamente aquellas dimensiones que se sacrifican comúnmente frente a otras para cumplir con tiempos o costos.

También se puede identificar que en los *sprints* centrales es donde más Deuda Técnica se puede identificar, mientras que en el inicio y fin del proyecto se identifican menos frases que tengan correlación con la misma. En la Tabla 7-4, se pueden ver los datos

correspondientes a cada sprint, la cantidad de factores que se detectaron y la frecuencia en que se presentaron.

Tabla 7-4. *Sprints*, cantidad de factores por *sprints* y frecuencia.

Sprint	Cantidad de Factores	Frecuencia
1	8	13
2	6	22
3	17	39
4	15	47
5	16	59
6	24	97
7	19	61
8	12	42
9	15	34
10	18	31
11	20	42
12	11	32
13	9	9
14	10	11
15	3	3

En el Anexo 3, se presentan para los 8 factores con mayor frecuencia el mapeo realizado. Se presentan los datos de los siguientes factores: Deuda de *Testing*, Calidad, Documentación obsoleta, *Time to market*, Migración de sistemas *legacy*, Costo de implementación, Añejamiento de código, Falta de transparencia hacia otros *stakeholders*.

De acuerdo a los resultados expuestos en el Anexo 3, (ver tablas de 9 a 16), se observa que de los registros de las retrospectivas de un proyecto de desarrollo de la industria de software es posible identificar conceptos que pueden ser asociados a la metáfora de Deuda Técnica. De esta forma, existe una confirmación más de que la metáfora Deuda Técnica es útil para explicar varios fenómenos que ocurren durante el desarrollo de software. Por otra parte, el proceso de mapeo sirve para confirmar, en un proyecto real

de desarrollo de software, la validez de los factores de costo de Deuda Técnica. Estos factores fueron identificados en la literatura académica mediante un proceso de revisión sistemática [12], y 40 de ellos fueron observados en las discusiones de retrospectivas del proyecto.

8 Conclusiones y futuras líneas de investigación

8.1 Conclusiones

El objetivo planteado para el presente trabajo se centraba en la identificación de deuda técnica en instancias de reflexión de proyectos ágiles de desarrollo de software. Para resolver el desafío planteado se presentaron 2 ítems relevantes a ser resueltos, en primera instancia identificar que apego al Manifiesto Ágil tenían los proyectos y por último identificar los factores de Deuda Técnica en las instancias de reflexión del proyecto seleccionado.

Los factores de Deuda Técnica a los cuales se hace referencia fueron identificados en la investigación realizada por Villar [12], en esta se identificaron 60 factores de Deuda Técnica en la literatura. Para dar respuesta al segundo problema planteado se crea un mecanismo de medición del grado de agilidad en proyectos de desarrollo, este se basa en los valores y principios del Manifiesto Ágil [8]. El mismo utiliza como método de recolección de datos la encuesta, en base a los datos obtenidos mediante una fórmula de cálculo se obtiene el grado de agilidad [9]. Cuando se habla de grado de Agilidad se hace referencia al apego del proceso de desarrollo de software a los valores y principios propuestos en él Manifiesto Ágil [8].

Con el fin de vincular los factores de Deuda Técnica con las instancias de reflexión en proyectos ágiles se utilizó la metodología de Teoría Justificada propuesta por Hoda [10], la metodología se aplicó en los informes de las instancias de reflexión. Como resultado se observa que en los registros de las instancias de reflexión de un proyecto de desarrollo de la industria de software es posible identificar conceptos que pueden ser asociados a la Deuda Técnica. De esta forma, existe una confirmación más de que la metáfora Deuda Técnica es útil para explicar varios fenómenos que ocurren durante el desarrollo de software.

Por otra parte, el proceso de mapeo sirve para confirmar, en un proyecto real de desarrollo de software, la validez de los factores de costo de Deuda Técnica. Estos factores fueron identificados en la literatura académica mediante un proceso de revisión sistemática [12], y 42 de ellos fueron observados en las discusiones de retrospectivas del proyecto.

8.2 Futuras líneas de investigación

Vista la confirmación empírica de la presencia de Deuda Técnica en proyectos ágiles de desarrollo, el siguiente paso en este proceso de investigación será la incorporación de la misma en la toma de decisiones de estos proyectos. Auguramos que los factores de Deuda Técnica, pueden ser tomados como entradas para mejorar el proceso de desarrollo de software de estos proyectos. En particular estamos interesados en mejorar las prácticas de ingeniería de software, que son, a nuestro criterio, las áreas más débiles que se observan en proyectos ágiles.

9 Bibliografía

[1] W. Cunningham, "The wycash portfolio management system." OOPSLA Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum), p. 29, 1992.

[2] F. Shull, D. Falessi, C. Seaman, M. Diep, and L. Layman, "Technical Debt: Showing the Way for Better Transfer of Empirical Results," *Perspectives on the Future of Software Engineering: Essays in Honor of Dieter Rombach*. pp. 179–190, 2013.

[3] M. Fowler, "Technical debt," 2009. [Online]. Available: http://martinfowler.com/bliki/TechnicalDebt.html.

[4] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. B. Da Silva, A. L. M. Santos, and C. Siebra, "Tracking technical debt — An exploratory case study," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 528–531.

[5] A. Begel and N. Nagappan, "Usage and perceptions of agile software development in an industrial context: An exploratory study," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 2007, pp. 255–264.

[6] A. Cockburn, *Agile software development*. Boston: Addison-Wesley, 2002.

[7] C. J. Stettina and W. Heijstek, "Five agile factors: Helping self-management to self-reflect," in *Systems, Software and Service Process Improvement*, Springer, 2011, pp. 84–96.

[8] "Manifesto agil." [Online]. Available: http://agilemanifesto.org/.

[9] C. Nacimento, S. Matalonga, A. Adorjan, and G. Mousqués, "Propuesta de Mecanismo de Medición de Agilidad de Proyectos de Desarrollo," *XVIII Ibero-American Conf. Softw. Eng. pp. 109–122. UCSP, Lima-Peru*, 2015.

[10] R. Hoda, J. Noble, and S. Marshall, "Developing a grounded theory to explain the practices of self-organizing Agile teams," *Empir. Softw. Eng.*, vol. 17, no. 6, pp. 609–639, 2012.

[11] C. Nacimento, S. Matalonga, J. Carlo, and R. Hauck, "Identifying Technical Debt Cost Factors in Reflection activities of an AgileProjects," *Conf. Latinoam. en Informática*, 2014.

[12] A. Villar and S. Matalonga, “Factors affecting Technical Debt Raw data from a Systematic literature map,” no. 12, p. 22, 2014.

[13] A. Villar and S. Matalonga, “Definiciones y tendencia de deuda técnica: Un mapeo sistemático de la literatura,” *Memorias la XVI Conf. Iberoam. Ing. Softw. CIbSE 2013*, pp. 33–46, 2013.

[14] C. Nacimento, J. Carlo, R. Hauck, and S. Matalonga, “Identificando Fatores de Débito Técnico em Retrospectivas de Projetos Ágeis,” *Work. Brazilero en Metod. ageils*, pp. 2–7, 2014.

[15] “Agile Alliance.” [Online]. Available: http://www.agilealliance.org/.

[16] A. Cockburn, “Crystal Clear a Human-powered Methodology for Small Teams.” 2004.

[17] J. Highsmith, “Agile project management creating innovative products.” 2004.

[18] R. Boehm, B and Turner, “Balancing agility and discipline A guide for the perplexed.” Addison-Wesley/Pearson Education, 2003.

[19] P. Kruchten, “Contextualizing agile software development,” *J. Softw. Evol. Process*, vol. 25, pp. 351–361, 2013.

[20] “Version one,” 2016. [Online]. Available: https://www.versionone.com/about/press-releases/versionone-opens-10th-annual-state-of-agile-survey/.

[21] K. Schwaber, *Agile Project Management with Scrum*. Sebastopol: O’Reilly Media, Inc, 2009.

[22] “Scrum Allience.” [Online]. Available: http://www.scrumaliance.org.

[23] K. Beck, *Extreme programming explained : embrace change*. Boston, MA: Addison-Wesley, 2005.

[24] G. Ahmad, T. R. Soomro, and M. N. Brohi, “XSR: Novel Hybrid Software Development Model (Integrating XP, Scrum & RUP),” *Int. J. Soft Comput. Eng. ISSN 2231-2307, Vol. Issue-3, April 2014*.

[25] S. Bashir and R. J. Qureshi, “Hybrid Software Development Approach for Small To Medium Scale Projects: Rup, Xp & Scrum,” *Sci.Int. (Lahore)*, vol. 24, no. 4. pp. 381–384, 2012.

[26] D. Anderson, *Agile management for software engineering : applying the theory of constraints for business results*. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2004.

[27] C. Ladas, *Scrumban and other essays on Kanban System for Lean Software develoment*. Saetle, WA: Modus Cooperandi Press, 2008.

[28] M. Poppendieck, *Lean software development : an agile toolkit*. Boston, Mass: Addison-Wesley, 2003.

[29] S. Ambler, *Agile modeling : effective practices for eXtreme programming and the unified process*. New York: J. Wiley, 2002.

[30] S. Palmer, *A practical guide to feature-driven development*. Upper Saddle River, NJ: Prentice Hall PTR, 2002.

[31] "Agile Unified Process." [Online]. Available: http://www.ambysoft.com/unifiedprocess/agileUP.html.

[32] J. Stapleton, *DSDM, dynamic systems development method : the method in practice*. Harlow, England Reading, Mass: Addison-Wesley, 1997.

[33] T. Pikkarainen, M and Huomo, "Agile Assesment Framework V1.0," 2005. [Online]. Available: http://www.agile-itea.org/public/deliverables/ITEA-AGILE-D4.1_v1.0.pdf.

[34] D. Leffingwell, "Scaling Software Agility." Addison-Wesley Professional, 2007.

[35] K. Mendez Calo, "Un framework para evaluación de metodologías Ágiles," 2009. [Online]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/21086/Documento_completo.pdf?sequence=1.

[36] A. Qumer and B. Henderson-Sellers, "An evaluation of the degree of agility in six agile methods and its applicability for method engineering," *Inf. Softw. Technol.*, vol. 50, pp. 280–295, 2008.

[37] D. Talby, O. Hazzan, Y. Dubinsky, and A. Keren, "Reflections on reflection in agile software development," in *Agile Conference, 2006*, 2006, p. 11 pp. –112.

[38] M. A. Ringstad, T. Dingsøyr, and N. B. Moe, "Agile process improvement: Diagnosis and planning to improve teamwork," in *Systems, Software and Service Process Improvement*, Springer, 2011, pp. 167–178.

[39] E. Bjarnason and B. Regnell, “Evidence-Based Timelines for Agile Project Retrospectives--A Method Proposal,” in *Agile Processes in Software Engineering and Extreme Programming*, Springer, 2012, pp. 177–184.

[40] C. Seaman, Y. Guo, C. Izurieta, Y. Cai, N. Zazworka, F. Shull, and A. Vetro, “Using technical debt data in decision making: Potential decision approaches,” in *Managing Technical Debt (MTD), 2012 Third International Workshop on*, 2012, pp. 45–48.

[41] J. Thomas, “Introducing Agile Development Practices from the Middle,” in *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*, 2008, pp. 401–407.

[42] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. B. Da Silva, A. L. M. Santos, and C. Siebra, “Tracking technical debt — An exploratory case study,” in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 528–531.

[43] N. Zazworka, R. O. Spínola, A. Vetro’, F. Shull, and C. Seaman, “A case study on effectively identifying technical debt,” in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering - EASE ’13*, 2013, p. 42.

[44] C. Seaman and Y. Guo, “Measuring and monitoring technical debt,” in *Advances in Computer Volume 82*, Elsevier, Ed. 2011, pp. 25–46.

[45] E. Rommes, A. Postma, and P. America, “Measuring Architecting Effort,” in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA’05)*, 2005, pp. 229–230.

[46] C. A. Siebra, G. S. Tonin, F. Q. B. Silva, G. Oliveira, A. L. C. Junior, C. G. Regina, and A. L. M. Santos, “Managing Technical Debt in Practice : An Industrial Report,” no. November 2006, pp. 247–250, 2007.

[47] B. Curtis, J. Sappidi, and J. Subramanyam, “Measuring the Structural Quality of Business Applications,” in *2011 AGILE Conference*, 2011, pp. 147–150.

[48] S. Muhammad, A. L. I. Shah, M. Torchiano, A. Vetro, and M. Morisio, “Exploratory testing as a source of testing technical debt,” *Autom. Informatics Dep. Politec. di Torino Corso Duca degli Abruzz. 24 10129 Torino, Italy*, 2013.

[49] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist, “Technical Debt in Test Automation,” in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 887–892.

[50] J. Linaker, S. M. Sulaman, M. Host, and R. M. de Mello, “Guidelines for Conducting Surveys in Software Engineering v.1.1,” vol. 08, 2015.

[51] “ORTSF.” [Online]. Available: http://fi.ort.edu.uy/ortsf.

[52] L. Camacho and E. Al, “VozDirecta, Sistema de Gestión de Quejas,” Facultad de Ingeniería Unviersidad ORT Uruguay, 2013.

10 Anexo 1 – Valores y principios del Manifiesto Ágil

En la Tabla 10-1, se puede observar la forma en que se vinculan los principios y valores del manifestó ágil con las preguntas que conforman la encuesta del mecanismo de medición agilidad.

Tabla 10-1. Valores y Principios del Manifiesto Ágil.

	ID	Descripción	ID	Descripción
VALORES	V1	Valorar al individuo y a las interacciones del equipo de desarrollo por encima del proceso y las herramientas.	V3	Valorar la colaboración con el cliente por sobre la negociación contractual.
	V2	Valorar el desarrollo de software que funcione por sobre una documentación exhaustiva.	V4	Valorar la respuesta al cambio por sobre el seguimiento de un plan.
PRINCIPIOS	P1	La prioridad es satisfacer al cliente mediante entregas tempranas y continuas de software que le aporte un valor.	P7	El software que funciona es la medida principal de progreso.
	P2	Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.	P8	Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
	P3	Entregar frecuentemente software que funcione en un período de tiempo de un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.	P9	La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
	P4	Los representantes del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.	P10	La simplicidad es esencial
	P5	Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.	P11	Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
	P6	El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.	P12	En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

Tabla 10-2. Preguntas que componen el mecanismo de medición de agilidad.

ID	Pregunta	Valor	ID	Pregunta	Valor
PR1	Se definieron claramente los roles, responsabilidades, conocimientos técnicos e interacciones entre miembros del equipo de trabajo	V1	PR14	Se realizaron las mismas actividades en cada iteración	V1*
PR2	Existió comunicación fluida en el equipo	V1	PR15	Se generaron entregables con testing satisfactorio e integrado con el resto de las funciones al finalizar cada iteración	V2
PR3	Existío comunicación Cara a Cara	V1	PR16	Se realizaron revisiones de a pares o inspecciones	V2
PR4	El usuario trabajó conjuntamente con el equipo	V1	PR17	Se estableció un plan de entrega o cronograma de revisiones modificable	V2
PR5	Existió motivación en el equipo	V1	PR18	Se priorizaron las necesidades de los entregables frente al grado de apego a los estándares de documentación	V2
PR6	Se autoorganizó el equipo	V1	PR19	Se requirió documentación para comenzar a implementar la funcionalidad incluida en cada iteración	V2*
PR7	Se reflexionó sobre la forma de que el equipo fuera más efectivo y ajustó la conducta en consecuencia dentro de cada intervalo	V1	PR20	Considera que el equipo logró el involucramiento del usuario/cliente, logrando que respondiera consultas, planificara iteraciones y colaborara en la especificación de los requerimientos y pruebas.	V3
PR8	Se priorizaron las tareas	V1	PR21	Se priorizó la satisfacción del cliente.	V3
PR9	Los entregables fueron incrementales	V1	PR22	Se definió un contrato para la totalidad del producto.	V3*

ID	Pregunta	Valor	ID	Pregunta	Valor
PR10	Se realizaron las entregas con frecuencia constante (frecuencia varia: 1 semana a 2 meses)	V1	PR23	Se previeron cláusulas para cambios adicionales en el contrato	V3*
PR11	Se involucró el usuario en el proceso	V1	PR24	Se permitió introducir cambios en cada una de las iteraciones	V4
PR12	Se generaron pruebas automáticas	V1	PR25	Se permitieron cambios durante cada iteración	V4
PR13	Se realizaron pruebas unitarias	V1	PR26	Se definió un plan detallado que incluyó recursos, fechas y responsables	V4*

11 Anexo 2 – Resultados de Codificación Abierta

Tabla 11-1. Resultados de codificación abierta.

#	Sprint	Descripción de Temas y Acciones
1	1	No se tenía la descomposición de tareas estimadas
2	1	Calcular el total de horas estimadas en tareas
3	1	Capacidad estimada del equipo para ese sprint
4	1	Impacto de atraso del equipo en áreas de soporte
5	2	Aplazamos la misma historia 2 veces
6	2	Separar capacidad en los próximos sprint
7	2	Estimación pesimista
8	2	Planificación dentro del 20% de capacidad estimada
9	2	Descomponer tareas de complejidad >=21
10	3	Historias no completadas en su totalidad dentro del sprint
11	3	Modificaran los criterios de aceptación
12	3	La historia vuelve al back log
13	3	Planificar y resolver los bugs en los próximos Sprints
14	3	Sobrecarga de trabajo en el líder técnico
15	3	LT no debería estar asignado al 100% de tareas de su capacidad
16	3	Registrar el tiempo dedicado a apoyo a otro miembro del equipo en la tarea en la cual realizo ese trabajo
17	3	Revisión de código y preparen algo para presentar en la retrospectiva, como puede ser unificar la forma en que está implementando algo
18	4	Definir dinámica con la cual se planificaran, asignaran y corregirán los bugs
19	4	Planificación solo dedicar un % de horas para solución de bugs
20	4	Como atacarlos: cada integrante lo maneja a su convenir
21	4	Aprueban y priorizan: el PM apruebe y priorice cada bug
22	4	Cada integrante se auto asigna los bugs
23	4	Los bugs que nadie tome serán asignados por el PM
24	4	Estabilización del ambiente de la empresa
25	4	Las mejoras / refactoreos se registraran como tareas en la historia del equipo a medida que se vayan viendo

#	Sprint	Descripción de Temas y Acciones
26	4	Avisar cuando quede estabilizado el ambiente del banco. No dejarlo para cuando este la UI porque se producirá una sobrecarga del área
27	4	Como se gestiona al faltar el SM
28	4	Seria bueno contar con las siguientes métricas en cada reunión de retrospectiva
29	5	Se definió el flujo de cambio de estados según lo que se había conversado en la retrospectiva pasada
30	5	Nos estamos quedando cortos en las estimaciones
31	5	Considerar el riesgo de las operaciones complejas al momento de estimar las tareas considerando el tiempo de investigación del código anterior
32	5	Cuidar la asignación de tareas y el riesgo que implican.
33	5	Las historias de este Sprint no fueron completadas óptimamente
34	5	A medida de que se vaya incorporando la UI, se ajustarán y redondearán las historias
35	5	Hay que considerar que las historias irán en aumento de complejidad, por lo que hay que mejorar la estimación.
36	5	Se agregará una tarea de validación de la funcionalidad contra el ambiente del banco.
37	5	Interacción con UI, se incluirán de forma de ayudar al equipo en la estimación de las historias
38	5	UI se incluirá en las reuniones de seguimiento
39	5	Es importante que QC pueda realizar los test cases a tiempo
40	5	QC tratará de ir un Sprint adelante con la realización de los test cases, sobre historias del back log
41	5	Test Cases aportan valor o es suficiente con el checklist y los criterios de aceptación, optimizar el tiempo de QC
42	5	Riesgos se repaso, riesgo 4 Exponer las vulnerabilidades en el sitio que afecten la información de los clientes, definir temas pendientes
43	5	Es importante que cada uno haga el seguimiento y actualice información respecto a los impedimentos que tienen asignados
44	5	Es importante que el equipo pueda alimentar la información de las historias de cada Sprint
45	6	Evaluar la lógica incluida en las vistas.

#	Sprint	Descripción de Temas y Acciones
46	6	Revisar que se estén respetando los estándares de desarrollo definidos por la empresa
47	6	Los métodos públicos deberían estar apropiadamente comentados
48	6	Evaluar valores hardcoded que convendría ser pasados a enumeradores
49	6	Revisar el uso de entidades repetidas
50	6	Reservar tiempo en el próximo sprint para que se ataque en parte los puntos anteriores
51	6	Identificar qué corresponde cambiar y definir quien hace cada cosa.
52	6	Lo mejor seria que cada uno ataque un punto completo
53	6	Al comenzar a integrar el diseño a una funcionalidad, no se contaba con el total de los insumos
54	6	No todas las funcionalidades quedan correctamente finalizadas y existe un re trabajo cuando se quiere utilizar un nuevo insumo ya integrado
55	6	Se crearan tareas asociadas a las mejoras y ajustes que hay que realizar en las funcionalidades a las cuales ya se les integro el diseño
56	6	Se tratara de reportar bugs genéricos de insumos o ajustes que falten para ser resueltos por PABLO (que rol tenia), para su reusó luego
57	6	Se incluirá a Pablo en las reuniones del próximo Sprint
58	6	Las propuestas de mejoras serán centralizadas en el Product Manager, será el quien las apruebe o no
59	6	No se debe perder el foco, respetar los bocetos que fueron validados por el cliente
60	6	Todas las estimaciones en horas estuvieron mucho mas altas respecto a la dedicación real que llevaron
61	6	La estimación en puntos de complejidad estuvo demasiado elevada
62	6	Esto se debió al miedo en quedar cortos en la estimación y que luego no fuera suficiente el tiempo para culminar con las historias del sprint
63	6	Falta de conocimiento respecto a lo que se enfrentarían al integrar el diseño
64	6	Falta de registro de la totalidad de horas
65	6	Preparar mejor la presentación para las reuniones
66	6	El equipo presenta y el único que tiene voz es el PM
67	6	El equipo toma notas y luego conversa o trasmite las mismas al PM al final de la reunión o en otra instancia

#	Sprint	Descripción de Temas y Acciones
68	6	Velar por no perder el foco en el objetivo de la reunión de cierre y evitar las interrupciones externas
69	7	Reservar un tiempo para hacer un análisis y revisar el código anterior para identificar y cubrir todos los casos
70	7	Si se detecta algo muy grande que exceda la estimación se separara lo nuevo y se escribirá una nueva historia para esa parte
71	7	Los bugs trabajados fueron ajustes de diseño y re trabajo que implico la incorporación de los mismos
72	7	Unificar criterios generales de forma que no se generen bugs similares en distintas funcionalidades
73	7	Dedicar tiempo para ver todo el proyecto de forma de encontrar criterios y realizar los ajustes generales
74	7	La cantidad de bugs reportados se explica porque se prueban varios Sprint a la vez
75	7	Se piensa que en futuros Sprint bajara y que los existentes se van a resolver mas rápido
76	7	Reservar tiempo en la próxima planificación para resolver los bugs priorizando los que son parte de la primera liberación al cliente
77	7	Identificación y definición de estándares
78	7	Sacarle mas jugo a las reuniones diarias con UI o arte
79	7	Comunicación mas fluida entre el equipo y arte para minimizar las intervenciones del PM
80	7	Solicitar a arte apoyo en la diagramación de ciertas pantalla
81	7	Evaluar si vale la pena realizar ciertas reuniones o no
82	7	Definir objetivos para las reuniones y velar para que se cumpla al final
83	8	Han quedado varias historias afuera de las comprometidas en la planificación
84	8	Contemplar el riesgo en las historias en la planificación tanto sea en la estimación como en las comprometidas a entregar
85	8	Tomar la experiencia de los Sprint anteriores para ajustar la estimación
86	8	Para las historias complejas, reservar una tarea de análisis para que se pueda identificar tempranamente si la estimación estuvo muy errada para ajustar la planificación
87	8	Redistribuir tareas de forma tal de que se pueda llegar a completar al menos una de las historias que no se llegaba al cierre del Sprint

#	Sprint	Descripción de Temas y Acciones
88	8	Velar por todo el grupo de forma de poder identificar y destrabar impedimentos para llevar adelante su tarea
89	8	QC propuso testing cruzado de forma de testear funcionalidades que QC no puede ver
90	8	Primer filtro para poder reducir la cantidad de bugs reportados por QC
91	9	Lecciones aprendidas (bugs, refactoreo y mejora de código), se definieron como acciones a llevar adelante durante todos los Sprint
92	9	No se completan todas las historias comprometidas para el Sprint
93	9	Nos estamos quedando cortos en las estimaciones
94	9	Obtener comparación de estimaciones históricas y lo que realmente llevo
95	9	Se trabajara sobre ello para mejorar las estimaciones
96	9	Tener en cuenta los imprevistos al momento de la planificación
97	9	Identificar y definir estándares de diseño
98	9	Testing cruzado en el sprint 10 para mejorar la calidad
99	9	Agregar una tarea a cada historia que indique realizar las pruebas cruzadas
100	10	Sprint demasiado corto
101	10	Incomodidad en el equipo por sprint a causa de fiestas y licencias
102	10	En caso parecido realizar un sprint de mayor duración o dedicarse a correcciones de bugs o refactoreo de código
103	10	Existieron mas solicitudes de desarrollos externos al proyecto de lo esperado
104	10	Verificar la tendencia en próximo sprint, si continua escalar a la gerencia.
105	10	Se mantiene una cantidad importante de bugs abiertos
106	10	Próximo sprint incorporar como tarea dentro de la historia el testing cruzado para bajar bugs reportados por QC
107	11	Testing cruzado la experiencia fue positiva. Se detectaron ajustes menores a realizar pero nada desde el lado de la codificación
108	11	Estimación y complejidad de las historias se esta mejorando
109	11	Cantidad de bugs abiertos daremos prioridad en el sprint 13
110	11	Identificar aquellos que se puedan asignar a Pablo de UI

#	Sprint	Descripción de Temas y Acciones
111	11	Respecto a Identificación y definición de estándares de diseño y generar el documento que sea input del equipo se definió que la prioridad es finalizar todas las funcionalidades
112	11	Ver todo con arte y MV, de allí identificar mejoras que serán aplicadas a todo el sitio
113	11	Estamos justos en el tiempo por lo cual hay que ajustar bien las planificaciones
114	11	Evaluar si es mejor proponer todos los temas y elegir solamente uno para atacar en el próximo Sprint
115	12	Este Sprint fue muy justo y presionado por la entrega de la tercera entrega al banco
116	12	El equipo dedico tiempo extra y cumplió con lo comprometido
117	12	Hay preocupación de poder cumplir con todo a tiempo y las próximas planificaciones estarán condicionadas por la información
118	12	Testing cruzado esta siendo útil como un primer filtro de detección de errores
119	12	Sigue creciendo la lista de bugs y en los próximos Sprint no habrá tiempo para resolver los mismos
120	12	Se atacaran los bugs una vez finalizado el desarrollo de las funcionalidades de la ultima liberación
121	12	En caso que haya un bug critico a resolver de inmediato notificar y asignar al equipo para ser resuelto en el Sprint en curso
122	13	Siguiente Sprint se debería resolver bugs. La forma de planificarlos es agruparlos y asignarlos a quien haya desarrollado para poder hacer una estimación del tiempo de resolución
123	13	Quedan varios temas para destrabar con el banco lo cual afecta definición de las historias pendientes y datos de pruebas
124	14	Corrección de bugs y temas de seguridad
125	14	Status general del proyecto y la proximidad con la fecha de finalización
126	15	En este Sprint no se pudo avanzar mucho debido a que la capacidad estuvo limitada por licencias y dedicaciones a otros proyectos

Tabla 11-2. Resultados de *Sprint* 1.

Descripción de Temas y Acciones	**Factores**			
No se tenía la descomposición de tareas estimadas	Pobre codificación en etapa de desarrollo	Calidad	Deuda de Testing	Energía
Calcular el total de horas estimadas en tareas	Deterioro de la arquitectura	Documentación Obsoleta	Deuda de Testing	
Capacidad estimada del equipo para ese sprint	Time to market	Documentación Obsoleta	Deuda de Testing	
Impacto de atraso del equipo en áreas de soporte	Cambios rápidos en tecnologías	Calidad	Deuda de Testing	

Tabla 11-3. Resultados de *Sprint* 2.

Descripción de Temas y Acciones	**Factores**				
Aplazamos la misma historia 2 veces	*Time to market*	Pobre codificación en etapa de desarrollo	Documentación Obsoleta	Poco esfuerzo y poca motivación	Deuda de *Testing*
Separar capacidad en los próximos sprint	*Time to market*	Pobre codificación en etapa de desarrollo	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*
Estimación pesimista	*Time to market*	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de Testing
Planificación dentro del 20% de capacidad estimada	*Time to market*	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*	
Descomponer tareas de complejidad >=21	*Time to market*	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*	

Tabla 11-4. Resultados de *Sprint* 3.

Descripción de Temas y Acciones	**Factores**						
Historias no completadas en su totalidad dentro del sprint	*Time to market*	Pobre codificación en etapa de desarrollo	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*	Inexperiencia
Modificaran los criterios de aceptación	Calidad	Deuda de *Testing*	Costos de implementación/re trabajo	*Like-to-like migration*			
La historia vuelve al back log	*Time to market*	Pobre codificación en etapa de desarrollo	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*	

Descripción de Temas y Acciones	**Factores**						
Planificar y resolver los bugs en los próximos Sprints	Compromiso a corto y largo plazo (nivel código y diseño)	*Refactoring*	Mal control de versionado	Deuda de *Testing*	Falta de transparencia hacia otros *stakeholders*	Costo de Implementación	Falta de conocimiento
Sobrecarga de trabajo en el líder técnico	*Time to market*	Documentación Obsoleta	Calidad	Deuda de *Testing*			
LT no debería estar asignado al 100% de tareas de su capacidad	*Time to market*	Documentación Obsoleta	Calidad	Deuda de *Testing*			
Registrar el tiempo dedicado a apoyo a otro miembro del equipo en la tarea en la cual realizo ese trabajo	Documentación Obsoleta	Calidad	Deuda de *Testing*				
Revisión de código y preparen algo para presentar en la retrospectiva, como puede ser unificar la forma en que está implementando algo	*Refactoring*	Obsolescencia tecnológica	Deuda de *Testing*	Re trabajo			

Tabla 11-5. Resultados de *Sprint* 4.

Descripción de Temas y Acciones	**Factores**				
Definir dinámica con la cual se planificarán, asignarán y corregirán los bugs	*Calidad*	*Deuda de Testing*	*Costo de Implementación*	*Falta de conocimiento*	
Planificación solo dedicar un % de horas para solución de bugs	*Documentación Obsoleta*	*Calidad*	*Deuda de Testing*	*Costo de Implementación*	*Falta de conocimiento*
Como atacarlos: cada integrante lo maneja a su convenir	*Time to market*	*Calidad*	*Deuda de Testing*	*Like-to-like migration*	
Aprueban y priorizan: el PM apruebe y priorice cada bug	*Calidad*	*Deuda de Testing*	*Falta de transparencia hacia otros stakeholders*		
Cada integrante se auto asigna los bugs	*Obsolescencia tecnológica*	*Calidad*	*Deuda de Testing*	*Falta de transparencia hacia otros stakeholders*	*Costo de Implementación*
Los bugs que nadie tome serán asignados por el PM	*Time to market*	*Calidad*	*Deuda de Testing*	*Falta de transparencia hacia otros stakeholders*	*Costo de Implementación*
Estabilización del ambiente de la empresa	*Calidad*	*Like-to-like migration*	*Nuevas oportunidades de negocio*	*Complejidad de código*	*Código legado pobre*
Las mejoras/refactoreos se registraran como tareas en la historia del equipo a medida que se vayan viendo	*Calidad*	*Deuda de Testing*	*Costo de Implementación*	*Compromiso a corto y largo plazo (Proceso)*	
Avisar cuando quede estabilizado el ambiente del banco. No dejarlo para cuando este la UI porque se producirá una sobrecarga del área	*Documentación Obsoleta*	*Calidad*	*Deuda de Testing*	*Like-to-like migration*	*Costo de Implementación*
Como se gestiona al faltar el SM	*Calidad*	*Deuda de Testing*			

Descripción de Temas y Acciones	Factores				
Sería bueno contar con las siguientes métricas en cada reunión de retrospectiva	*Calidad*	*Añejamiento de Código*	*Deuda de Testing*	*Like-to-like migration*	

Tabla 11-6. Resultados de *Sprint* 5.

Descripción de Temas y Acciones	Factores				
Se definió el flujo de cambio de estados según lo que se había conversado en la retrospectiva pasada	Calidad	Deuda de *Testing*	Costos de implementación/re trabajo		
Nos estamos quedando cortos en las estimaciones	*Time to market*	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*
Considerar el riesgo de las operaciones complejas al momento de estimar las tareas considerando el tiempo de investigación del código anterior	*Time to market*	Falta de tiempo / presupuesto	Documentación Obsoleta	Deuda de *Testing*	
Cuidar la asignación de tareas y el riesgo que implican.	*Time to market*	Calidad	Deuda de *Testing*		
Las historias de este Sprint no fueron completadas óptimamente	Documentación Obsoleta	Obsolescencia tecnológica	Calidad	Inexperiencia	
A medida de que se vaya incorporando la UI, se ajustarán y redondearán las historias	Documentación Obsoleta	Mal control de versionado	Deuda de *Testing*	*Like-to-like migration*	
Hay que considerar que las historias irán en aumento de complejidad, por lo que hay que mejorar la estimación.	*Time to market*	Pobre codificación en etapa de desarrollo	Documentación Obsoleta	Deuda de *Testing*	
Se agregará una tarea de validación de la funcionalidad contra el ambiente del banco.	Compromiso a corto y largo plazo (nivel código y diseño)	*Like-to-like migration*	57		
Interacción con UI, se incluirán de forma de ayudar al equipo en la estimación de las historias	*Time to market*	Documentación Obsoleta	Deuda de *Testing*	*Like-to-like migration*	
UI se incluirá en las reuniones de seguimiento	*Time to market*	Calidad	Deuda de *Testing*		
Es importante que QC pueda realizar los test cases a tiempo	Compromiso a corto y largo plazo (nivel código y diseño)	Documentación Obsoleta	Añejamiento de Código	Deuda de *Testing*	*Like-to-like migration*
QC tratará de ir un Sprint adelante con la realización de los test cases, sobre historias del back log	Compromiso a corto y largo plazo (nivel código y diseño)	Documentación Obsoleta	Deuda de *Testing*	*Like-to-like migration*	
Test Cases aportan valor o es suficiente con el checklist y los criterios de aceptación, optimizar el tiempo de QC	Compromiso a corto y largo plazo (nivel código y diseño)	Calidad	Añejamiento de Código	Deuda de *Testing*	
Riesgos se repaso, riesgo 4 Exponer las vulnerabilidades en el sitio que afecten la información de los clientes, definir temas pendientes	Calidad	Deuda de *Testing*	*Like-to-like migration*		
Es importante que cada uno haga el seguimiento y actualice información respecto a los impedimentos que tienen asignados	*Time to market*	Gerenciamiento de proyectos caótico	Deuda de *Testing*		

Descripción de Temas y Acciones	Factores				
Es importante que el equipo pueda alimentar la información de las historias de cada Sprint	*Time to market*	Gerenciamiento de proyectos caótico	Deuda de *Testing*		

Tabla 11-7. Resultados de *Sprint* 6.

Descripción de Temas y Acciones	Factores						
Evaluar la lógica incluida en las vistas.	Deterioro de la arquitectura	Mal control de versionado	Re trabajo	Testear al final			
Revisar que se estén respetando los estándares de desarrollo definidos por la empresa	Deterioro de la arquitectura	Re trabajo	Cobertura de pruebas unitarias	Testear al final			
Los métodos públicos deberían estar apropiadamente comentados	Deterioro de la arquitectura	Mal control de versionado	Re trabajo	Cobertura de pruebas unitarias	Testear al final		
Evaluar valores hardcoded que convendría ser pasados a enumeradores	Bajo *refactoring* y rediseño	Deterioro de la arquitectura	Mal control de versionado	Re trabajo	Cobertura de pruebas unitarias	Testear al final	
Revisar el uso de entidades repetidas	Bajo *refactoring* y rediseño	*Refactoring*	Deterioro de la arquitectura	Poco esfuerzo y poca motivación	Compromiso a corto y largo plazo (nivel arquitectura)		
Reservar tiempo en el próximo sprint para que se ataque en parte los puntos anteriores	*Refactoring*	Documentación Obsoleta	Deuda de *Testing*	Costo de Implementación			
Identificar qué corresponde cambiar y definir quien hace cada cosa.	Calidad	Deuda de *Testing*	Costo de Implementación				
Lo mejor sería que cada uno ataque un punto completo	Calidad	Deuda de *Testing*	Costo de Implementación				
Al comenzar a integrar el diseño a una funcionalidad, no se contaba con el total de los insumos	Calidad	Deuda de *Testing*	*Like-to-like migration*	Costo de Implementación			
No todas las funcionalidades quedan correctamente finalizadas y existe un re trabajo cuando se quiere utilizar un nuevo insumo ya integrado	Compromiso a corto y largo plazo (nivel código y diseño)	Deterioro de la arquitectura	Deuda de *Testing*	*Like-to-like migration*	Costo de Implementación	Compromiso a corto y largo plazo (Proceso)	Inexperiencia
Se crearan tareas asociadas a las mejoras y ajustes que hay que realizar en las funcionalidades a las cuales ya se les integro el diseño	*Refactoring*	Deuda de *Testing*	*Like-to-like migration*	Costo de Implementación	Compromiso a corto y largo plazo (Proceso)		

Descripción de Temas y Acciones	Factores						
Se tratara de reportar bugs genéricos de insumos o ajustes que falten para ser resueltos por PABLO (que rol tenia), para su reusó luego	*Refactoring*	Deuda de *Testing*	Costo de Implementación	Compromiso a corto y largo plazo (Proceso)			
Se incluirá a Pablo en las reuniones del próximo Sprint	*Time to market*	Documentación Obsoleta	Calidad	Deuda de *Testing*			
Las propuestas de mejoras serán centralizadas en el Product Manager, será el quien las apruebe o no	*Refactoring*	Calidad	Deuda de *Testing*	Costo de Implementación			
No se debe perder el foco, respetar los bocetos que fueron validados por el cliente	Añejamiento de Código	Re trabajo					
Todas las estimaciones en horas estuvieron mucho más altas respecto a la dedicación real que llevaron	*Time to market*	Pobre codificación en etapa de desarrollo	Calidad	Deuda de *Testing*			
La estimación en puntos de complejidad estuvo demasiado elevada	*Time to market*	Pobre codificación en etapa de desarrollo	Calidad	Deuda de *Testing*			
Esto se debió al miedo en quedar cortos en la estimación y que luego no fuera suficiente el tiempo para culminar con las historias del sprint	*Time to market*	Pobre codificación en etapa de desarrollo	Documentación Obsoleta	Calidad	Deuda de *Testing*		

Tabla 11-8. Resultados de *Sprint* 7.

Descripción de Temas y Acciones	Factores						
Reservar un tiempo para hacer un análisis y revisar el código anterior para identificar y cubrir todos los casos	*Time to market*	*Refactoring*	Deterioro del código /diseño	Pobre codificación en etapa de desarrollo	Deuda de *Testing*		
Si se detecta algo muy grande que exceda la estimación se separara lo nuevo y se escribirá una nueva historia para esa parte	*Time to market*	Pobre codificación en etapa de desarrollo	Calidad	Deuda de *Testing*			
Los bugs trabajados fueron ajustes de diseño y re trabajo que implico la incorporación de los mismos	*Refactoring*	Deterioro del código /diseño	Falta de tiempo / presupuesto	Añejamiento de Código	Falta de transparencia hacia otros *stakeholders*	Falta de conocimiento	Pasaje de defectos en un *sprint (Defects carry over)*
Unificar criterios generales de forma que no se generen bugs similares en distintas funcionalidades	Deuda de *Testing*	Falta de transparencia hacia otros *stakeholders*	Costos de implementación /re trabajo	Re trabajo	Falta de conocimiento	Deuda Funcional (funcionalidades prometidas no entregadas)	
Dedicar tiempo para ver todo el proyecto de forma de encontrar	Calidad	Deuda de *Testing*	Costos de implementación	Re trabajo			

Descripción de Temas y Acciones	Factores						
criterios y realizar los ajustes generales			/re trabajo				
La cantidad de bugs reportados se explica porque se prueban varios Sprint a la vez	Compromiso a corto y largo plazo (nivel código y diseño)	Documentación Obsoleta	Calidad	Deuda de *Testing*	Falta de transparencia hacia otros *stakeholders*	Falta de conocimiento	Deuda Funcional (funcionalidades prometidas no entregadas)
Se piensa que en futuros Sprint bajara y que los existentes se van a resolver mas rápido	Compromiso a corto y largo plazo (nivel código y diseño)	Documentación Obsoleta	Calidad	Deuda de *Testing*	Pasaje de defectos en un *sprint (Defects carry over)*		
Reservar tiempo en la próxima planificación para resolver los bugs priorizando los que son parte de la primera liberación al cliente	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*	Falta de transparencia hacia otros *stakeholders*	Falta de conocimiento	Pasaje de defectos en un *sprint (Defects carry over)*
Identificación y definición de estándares	Añejamiento de Código	Re trabajo					
Sacarle mas jugo a las reuniones diarias con UI o arte	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*				
Comunicación mas fluida entre el equipo y arte para minimizar las intervenciones del PM	Calidad	Deuda de *Testing*	*Like-to-like migration*				
Solicitar a arte apoyo en la diagramación de ciertas pantalla	Calidad	Deuda de *Testing*	*Like-to-like migration*				
Evaluar si vale la pena realizar ciertas reuniones o no	Calidad	Deuda de *Testing*					
Definir objetivos para las reuniones y velar para que se cumpla al final	Calidad	Deuda de *Testing*					

Tabla 11-9. Resultados de *Sprint* 8.

Descripción de Temas y Acciones	Factores						
Han quedado varias historias afuera de las comprometidas en la planificación	*Time to market*	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*	*Like-to-like migration*	Inexperiencia
Contemplar el riesgo en las historias en la planificación tanto sea en la estimación como en las comprometidas a entregar	*Time to market*	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*	*Like-to-like migration*	
Tomar la experiencia de los Sprint anteriores para ajustar la estimación	*Time to market*	Calidad	Deuda de *Testing*				
Para las historias complejas, reservar una tarea de análisis para que se pueda identificar tempranamente si la	*Time to market*	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*		

Descripción de Temas y Acciones	Factores						
estimación estuvo muy errada para ajustar la planificación							
Redistribuir tareas de forma tal de que se pueda llegar a completar al menos una de las historias que no se llegaba al cierre del Sprint	*Time to market*	Pobre codificación en etapa de desarrollo	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*	Inexperiencia
Velar por todo el grupo de forma de poder identificar y destrabar impedimentos para llevar adelante su tarea	*Time to market*	Pobre codificación en etapa de desarrollo	Calidad	Deuda de *Testing*			
QC propuso testing cruzado de forma de testear funcionalidades que QC no puede ver	Compromiso a corto y largo plazo (nivel código y diseño)	Documentación Obsoleta	Añejamiento de Código	Falta de transparencia hacia otros *stakeholders*	Deuda de diseño		
Primer filtro para poder reducir la cantidad de bugs reportados por QC	Compromiso a corto y largo plazo (nivel código y diseño)	Documentación Obsoleta	Añejamiento de Código	Falta de transparencia hacia otros *stakeholders*	Deuda de diseño		

Tabla 11-10. Resultados de *Sprint* 9.

Descripción de Temas y Acciones	Factores					
Lecciones aprendidas (bugs, refactoreo y mejora de código), se definieron como acciones a llevar adelante durante todos los Sprint	*Time to market*	Añejamiento de Código	Deuda de *Testing*			
No se completan todas las historias comprometidas para el Sprint	*Time to market*	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*	Inexperiencia
Nos estamos quedando cortos en las estimaciones	*Time to market*	Documentación Obsoleta	Calidad	Deuda de *Testing*		
Obtener comparación de estimaciones históricas y lo que realmente llevo	Calidad	Deuda de *Testing*	Costo de Implementación			
Se trabajara sobre ello para mejorar las estimaciones	Calidad	Deuda de *Testing*	Costo de Implementación			
Tener en cuenta los imprevistos al momento de la planificación	Calidad	Deuda de *Testing*				
Identificar y definir estándares de diseño	Bajo *refactoring* y rediseño	Falta de tiempo / presupuesto	Costos de implementación/re trabajo	Falta de mejora de procesos		
Testing cruzado en el sprint 10 para mejorar la calidad	Compromiso a corto y largo plazo (nivel código y diseño)	Añejamiento de Código	Falta de transparencia hacia otros *stakeholders*	Costos de implementación /re trabajo		
Agregar una tarea a cada historia que indique realizar las pruebas cruzadas	Compromiso a corto y largo plazo (nivel código y diseño)	Documentación Obsoleta	Calidad	Deuda de *Testing*		

Tabla 11-11. Resultados de *Sprint* 10.

Descripción de Temas y Acciones	**Factores**			
Sprint demasiado corto	*Time to market*	Documentación Obsoleta	Obsolescencia tecnológica	Deuda de *Testing*
Incomodidad en el equipo por sprint a causa de fiestas y licencias	Documentación Obsoleta	Obsolescencia tecnológica	Calidad	Deuda de *Testing*
En caso parecido realizar un sprint de mayor duración o dedicarse a correcciones de bugs o refactoreo de código	Calidad	Deuda de *Testing*		
Existieron más solicitudes de desarrollos externos al proyecto de lo esperado	Pobre codificación en etapa de desarrollo	Gerenciamiento de proyectos caótico	Falta de fondos para realizar los cambios necesarios	
Verificar la tendencia en próximo sprint, si continua escalar a la gerencia.	Gerenciamiento de proyectos caótico	Calidad	Deuda de *Testing*	

Tabla 11-12. Resultados de *Sprint* 11.

Descripción de Temas y Acciones	**Factores**						
Testing cruzado la experiencia fue positiva. Se detectaron ajustes menores a realizar pero nada desde el lado de la codificación	Compromiso a corto y largo plazo (nivel código y diseño)	Añejamiento de Código	Falta de transparencia hacia otros *stakeholders*	*Like-to-like migration*			
Estimación y complejidad de las historias se esta mejorando	*Time to market*	Poco esfuerzo y poca motivación	Calidad	Deuda de *Testing*	*Like-to-like migration*		
Cantidad de bugs abiertos daremos prioridad en el sprint 13	Compromiso a corto y largo plazo (nivel código y diseño)	*Refactoring*	Añejamiento de Código	Deuda de *Testing*	Falta de transparencia hacia otros *stakeholders*	Costo de Implementación	Falta de conocimiento
Identificar aquellos que se puedan asignar a Pablo de UI	Compromiso a corto y largo plazo (nivel código y diseño)	Calidad	Deuda de *Testing*	*Like-to-like migration*			
Respecto a Identificación y definición de estándares de diseño y generar el documento que sea input del equipo se definió que la prioridad es finalizar todas las funcionalidades	Deuda de *Testing*	Costos de implementación/re trabajo	Re trabajo	Compromiso a corto y largo plazo (nivel arquitectura)	Testear al final		
Ver todo con arte y MV, de allí identificar mejoras que serán aplicadas a todo el sitio	*Refactoring*	Deterioro del código /diseño	Añejamiento de Código	Deuda de *Testing*	*Like-to-like migration*		
Estamos justos en el tiempo por lo cual hay que ajustar bien las planificaciones	Documentación Obsoleta	Calidad	Deuda de *Testing*	*Like-to-like migration*	*Like-to-like migration*		
Evaluar si es mejor proponer todos los temas y elegir	*Time to market*	Calidad	Deuda de *Testing*	*Like-to-like migration*			

Descripción de Temas y Acciones	Factores						
solamente uno para atacar en el próximo Sprint							

Tabla 11-13. Resultados de *Sprint* 12.

Descripción de Temas y Acciones	Factores						
Este Sprint fue muy justo y presionado por la entrega de la tercera entrega al banco	Deterioro de la arquitectura	Documentación Obsoleta	Deuda de *Testing*	*Like-to-like migration*			
El equipo dedico tiempo extra y cumplió con lo comprometido	Deterioro de la arquitectura	Documentación Obsoleta	Deuda de *Testing*	*Like-to-like migration*			
Hay preocupación de poder cumplir con todo a tiempo y las próximas planificaciones estarán condicionadas por la información	Documentación Obsoleta	Añejamiento de Código	Deuda de *Testing*				
Testing cruzado está siendo útil como un primer filtro de detección de errores	Compromiso a corto y largo plazo (nivel código y diseño)	Añejamiento de Código	Falta de transparencia hacia otros *stakeholders*				
Sigue creciendo la lista de bugs y en los próximos Sprint no habrá tiempo para resolver los mismos	Compromiso a corto y largo plazo (nivel código y diseño)	Documentación Obsoleta	Poco esfuerzo y poca motivación	Calidad	Añejamiento de Código	Deuda de *Testing*	Falta de transparencia hacia otros *stakeholders*
Se atacaran los bugs una vez finalizado el desarrollo de las funcionalidades de la última liberación	*Refactoring*	Añejamiento de Código	Deuda de *Testing*	Falta de transparencia hacia otros *stakeholders*			
En caso que haya un bug critico a resolver de inmediato notificar y asignar al equipo para ser resuelto en el Sprint en curso	Compromiso a corto y largo plazo (nivel código y diseño)	Calidad	Añejamiento de Código	Deuda de *Testing*	Falta de transparencia hacia otros *stakeholders*	*Like-to-like migration*	

Tabla 11-14. Resultados de *Sprint* 13.

Descripción de Temas y Acciones	**Factores**				
Siguiente Sprint se deberían resolver bugs. La forma de planificarlos es agruparlos y asignarlos a quien haya desarrollado para poder hacer una estimación del tiempo de resolución	*Refactoring*	Mal control de versionado	Costo de Implementación	Corrección de defectos	Pasaje de defectos en un *sprint (Defects carry over)*
Quedan varios temas para destrabar con el banco lo cual afecta definición de las historias pendientes y datos de pruebas	*Time to market*	Calidad	Deuda de *Testing*	Re trabajo	

Tabla 11-15. Resultados de *Sprint* 14.

Descripción de Temas y Acciones	**Factores**							
Corrección de bugs y temas de seguridad	*Refactoring*	Mal control de versionado	Añejamiento de Código	Falta de transparencia hacia otros *stakeholders*	*Like-to-like migration*	Costo de Implementación	Falta de conocimiento	Pasaje de defectos en un *sprint (Defects carry over)*
Status general del proyecto y la proximidad con la fecha de finalización	Deterioro de la arquitectura	Poco esfuerzo y poca motivación	Falta de transparencia hacia otros *stakeholders*					

Tabla 11-16. Resultados de *Sprint* 15.

Descripción de Temas y Acciones	**Factores**		
En este Sprint no se pudo avanzar mucho debido a que la capacidad estuvo limitada por licencias y dedicaciones a otros proyectos	Documentación Obsoleta	Deuda de *Testing*	Mala integración y manejo de liberaciones

12 Anexo 3 - Resultados de la Teoría Justificada

A continuación se presentan los resultados obtenidos durante el proceso de la aplicación de la adaptación de la Teoría Justificada. Se presenta el mapeo de las frases que han sido encontradas en las retrospectivas para un determinado factor de Deuda Técnica.

1. **Deuda de *Testing*:** en la Tabla 12-1 se presentan las frases asociadas al factor deuda de *testing*. Por cada *sprint* (1-15) se exponen las frases que se identificaron en las instancias de reflexión.

Tabla 12-1. Frases codificadas por *Sprint* para el Factor Deuda de *Testing*.

Sprint	Frases
1	No se tenía la descomposición de tareas estimadas. Calcular el total de horas estimadas en tareas. Capacidad estimada del equipo para ese *sprint*. Impacto de atraso del equipo en áreas de soporte.
2	Aplazamos la misma historia 2 veces. Separar capacidad en los próximos *sprints*. Estimación pesimista. Planificación dentro del 20% de capacidad estimada. Descomponer tareas de complejidad >=21.
3	Historias no completadas en su totalidad dentro del *sprint*. Modificaran los criterios de aceptación. La historia vuelve al *back log*. Planificar y resolver los *bugs* en los próximos *Sprints*. Sobrecarga de trabajo en el líder técnico. LT no debería estar asignado al 100% de tareas de su capacidad. Registrar el tiempo dedicado a apoyo a otro miembro del equipo en la tarea en la cual realizo ese trabajo. Revisión de código y preparen algo para presentar en la retrospectiva, como puede ser unificar la forma en que está implementando algo.
4	Definir dinámica con la cual se planificarán, asignarán y corregirán los bugs. Planificación solo dedicar un % de horas para solución de *bugs*. Como atacarlos: cada integrante lo maneja a su convenir. Aprueban y priorizan: el PM apruebe y priorice cada *bug*.

Sprint	Frases
	Cada integrante se auto asigna los *bugs*. Los bugs que nadie tome serán asignados por el PM. Las mejoras/*refactoreos* se registraran como tareas en la historia del equipo a medida que se vayan viendo. Avisar cuando quede estabilizado el ambiente del <Cliente>. No dejarlo para cuando este la UI porque se producirá una sobrecarga del área. Como se gestiona al faltar el SM. Sería bueno contar con las siguientes métricas en cada reunión de retrospectiva.
5	Se definió el flujo de cambio de estados según lo que se había conversado en la retrospectiva pasada. Nos estamos quedando cortos en las estimaciones. Considerar el riesgo de las operaciones complejas al momento de estimar las tareas considerando el tiempo de investigación del código anterior. Cuidar la asignación de tareas y el riesgo que implican. A medida de que se vaya incorporando la UI, se ajustarán y redondearán las historias. Hay que considerar que las historias irán en aumento de complejidad, por lo que hay que mejorar la estimación. Interacción con UI, se incluirán de forma de ayudar al equipo en la estimación de las historias. UI se incluirá en las reuniones de seguimiento. Es importante que QC pueda realizar los test cases a tiempo. QC tratará de ir un *Sprint* adelante con la realización de los test cases, sobre historias del *back log*. Test Cases aportan valor o es suficiente con el *checklist* y los criterios de aceptación, optimizar el tiempo de QC. Exponer las vulnerabilidades en el sitio que afecten la información de los clientes, definir temas pendientes. Es importante que cada uno haga el seguimiento y actualice información respecto a los impedimentos que tienen asignados. Es importante que el equipo pueda alimentar la información de las historias de cada *Sprint*.
6	Reservar tiempo en el próximo *sprint* para que se ataque en parte los puntos anteriores. Identificar qué corresponde cambiar y definir quien hace cada cosa. Lo mejor sería que cada uno ataque un punto completo. Al comenzar a integrar el diseño a una funcionalidad, no se contaba con el total de los insumos. No todas las funcionalidades quedan correctamente finalizadas y existe un re trabajo cuando se quiere utilizar un nuevo insumo ya integrado.

Sprint	Frases
	Se crearan tareas asociadas a las mejoras y ajustes que hay que realizar en las funcionalidades a las cuales ya se les integro el diseño. Se tratara de reportar bugs genéricos de insumos o ajustes que falten para ser resueltos por <Nombre>, para su reúso luego. Se incluirá a <Nombre> en las reuniones del próximo *Sprint*. Las propuestas de mejoras serán centralizadas en el *Product Manager*, será él quien las apruebe o no. Todas las estimaciones en horas estuvieron mucho más altas respecto a la dedicación real que llevaron. La estimación en puntos de complejidad estuvo demasiado elevada. Esto se debió al miedo en quedar cortos en la estimación y que luego no fuera suficiente el tiempo para culminar con las historias del *sprint*. Falta de registro de la totalidad de horas. Preparar mejor la presentación para las reuniones. El equipo presenta y el único que tiene voz es el PM. El equipo toma notas y luego conversa o trasmite las mismas al PM al final de la reunión o en otra instancia. Velar por no perder el foco en el objetivo de la reunión de cierre y evitar las interrupciones externas.
7	Reservar un tiempo para hacer un análisis y revisar el código anterior para identificar y cubrir todos los casos. Si se detecta algo muy grande que exceda la estimación se separara lo nuevo y se escribirá una nueva historia para esa parte. Unificar criterios generales de forma que no se generen bugs similares en distintas funcionalidades. Dedicar tiempo para ver todo el proyecto de forma de encontrar criterios y realizar los ajustes generales. La cantidad de *bugs* reportados se explica porque se prueban varios *Sprint* a la vez. Se piensa que en futuros *Sprint* bajara y que los existentes se van a resolver más rápido. Reservar tiempo en la próxima planificación para resolver los bugs priorizando los que son parte de la primera liberación al cliente. Sacarle más jugo a las reuniones diarias con UI o arte. Comunicación más fluida entre el equipo y arte para minimizar las intervenciones del PM. Solicitar a arte apoyo en la diagramación de ciertas pantallas. Evaluar si vale la pena realizar ciertas reuniones o no. Definir objetivos para las reuniones y velar para que se cumpla al final.
8	Han quedado varias historias afuera de las comprometidas en la planificación.

Sprint	Frases
	Contemplar el riesgo en las historias en la planificación tanto sea en la estimación como en las comprometidas a entregar. Tomar la experiencia de los *Sprint* anteriores para ajustar la estimación. Para las historias complejas, reservar una tarea de análisis para que se pueda identificar tempranamente si la estimación estuvo muy errada para ajustar la planificación. Redistribuir tareas de forma tal de que se pueda llegar a completar al menos una de las historias que no se llegaba al cierre del *Sprint*. Velar por todo el grupo de forma de poder identificar y destrabar impedimentos para llevar adelante su tarea.
9	Lecciones aprendidas (*bugs, refactoreo* y mejora de código), se definieron como acciones a llevar adelante durante todos los *Sprint*. No se completan todas las historias comprometidas para el *Sprint*. Nos estamos quedando cortos en las estimaciones. Obtener comparación de estimaciones históricas y lo que realmente llevo. Se trabajara sobre ello para mejorar las estimaciones. Tener en cuenta los imprevistos al momento de la planificación. Agregar una tarea a cada historia que indique realizar las pruebas cruzadas.
10	*Sprint* demasiado cortó. Incomodidad en el equipo por sprint a causa de fiestas y licencias. En caso parecido realizar un sprint de mayor duración o dedicarse a correcciones de bugs o *refactoreo* de código. Verificar la tendencia en próximo *sprint*, si continua escalar a la gerencia. Se mantiene una cantidad importante de *bugs* abiertos. Próximo *sprint* incorporar como tarea dentro de la historia el *testing* cruzado para bajar *bugs* reportados por QC.
11	Estimación y complejidad de las historias se está mejorando. Cantidad de bugs abiertos daremos prioridad en el *sprint* 13. Identificar aquellos que se puedan asignar a <Nombre> de UI. Respecto a Identificación y definición de estándares de diseño y generar el documento que sea input del equipo se definió que la prioridad es finalizar todas las funcionalidades. Ver todo con arte y MV, de allí identificar mejoras que serán aplicadas a todo el sitio. Estamos justos en el tiempo por lo cual hay que ajustar bien las planificaciones. Evaluar si es mejor proponer todos los temas y elegir solamente uno para atacar en el próximo *Sprint*.
12	Este *Sprint* fue muy justo y presionado por la entrega de la tercera entrega al <Cliente> El equipo dedico tiempo extra y cumplió con lo comprometido.

Sprint	Frases
	Hay preocupación de poder cumplir con todo a tiempo y las próximas planificaciones estarán condicionadas por la información. Sigue creciendo la lista de bugs y en los próximos *Sprint* no habrá tiempo para resolverlos. Se atacaran los *bugs* una vez finalizado el desarrollo de las funcionalidades de la última liberación. En caso que haya un *bug* critico a resolver de inmediato notificar y asignar al equipo para ser resuelto en el *Sprint* en curso.
13	Quedan varios temas para destrabar con el <Cliente> lo cual afecta definición de las historias pendientes y datos de pruebas.
15	En este *Sprint* no se pudo avanzar mucho debido a que la capacidad estuvo limitada por licencias y dedicaciones a otros proyectos.

2. **Calidad:** las frases asociadas a la deuda de calidad se presentan en la Tabla 12-2, las mismas se encuentran ordenadas por *sprints* de acuerdo a como se identificaron durante el proceso de codificación abierta.

Tabla 12-2. Frases codificadas por *Sprint* para el Factor Calidad

Sprint	Frases
1	No se tenía la descomposición de tareas estimadas. Impacto de atraso del equipo en áreas de soporte.
2	Separar capacidad en los próximos *sprints*. Estimación pesimista. Planificación dentro del 20% de capacidad estimada. Descomponer tareas de complejidad >=21.
3	Historias no completadas en su totalidad dentro del sprint. Modificaran los criterios de aceptación. La historia vuelve al *back log*. Sobrecarga de trabajo en el líder técnico. LT no debería estar asignado al 100% de tareas de su capacidad. Registrar el tiempo dedicado a apoyo a otro miembro del equipo en la tarea en la cual realizo ese trabajo.
4	Definir dinámica con la cual se planificarán, asignarán y corregirán los *bugs*.

Sprint	Frases
	Planificación solo dedicar un % de horas para solución de *bugs*. Como atacarlos: cada integrante lo maneja a su convenir. Aprueban y priorizan: el PM apruebe y priorice cada *bug*. Cada integrante se auto asigna los *bugs*. Los bugs que nadie tome serán asignados por el PM. Estabilización del ambiente de la empresa. Las mejoras/*refactoreos* se registraran como tareas en la historia del equipo a medida que se vayan viendo. Avisar cuando quede estabilizado el ambiente del <Cliente>. No dejarlo para cuando este la UI porque se producirá una sobrecarga del área. Como se gestiona al faltar el SM. Sería bueno contar con las siguientes métricas en cada reunión de retrospectiva.
5	Se definió el flujo de cambio de estados según lo que se había conversado en la retrospectiva pasada. Nos estamos quedando cortos en las estimaciones. Cuidar la asignación de tareas y el riesgo que implican. Las historias de este Sprint no fueron completadas óptimamente. UI se incluirá en las reuniones de seguimiento. *Test Cases* aportan valor o es suficiente con el *checklist* y los criterios de aceptación, optimizar el tiempo de QC. Exponer las vulnerabilidades en el sitio que afecten la información de los clientes, definir temas pendientes.
6	Identificar qué corresponde cambiar y definir quien hace cada cosa. Lo mejor sería que cada uno ataque un punto completo. Al comenzar a integrar el diseño a una funcionalidad, no se contaba con el total de los insumos. Se incluirá a <Nombre> en las reuniones del próximo *Sprint*. Las propuestas de mejoras serán centralizadas en el *Product Manager*, será el quien las apruebe o no. Todas las estimaciones en horas estuvieron mucho más altas respecto a la dedicación real que llevaron. La estimación en puntos de complejidad estuvo demasiado elevada. Esto se debió al miedo en quedar cortos en la estimación y que luego no fuera suficiente el tiempo para culminar con las historias del *sprint*. Falta de registro de la totalidad de horas. Preparar mejor la presentación para las reuniones. El equipo presenta y el único que tiene voz es el PM.

Sprint	Frases
	El equipo toma notas y luego conversa o trasmite las mismas al PM al final de la reunión o en otra instancia. Velar por no perder el foco en el objetivo de la reunión de cierre y evitar las interrupciones externas.
7	Si se detecta algo muy grande que exceda la estimación se separara lo nuevo y se escribirá una nueva historia para esa parte. Dedicar tiempo para ver todo el proyecto de forma de encontrar criterios y realizar los ajustes generales. La cantidad de bugs reportados se explica porque se prueban varios *Sprint* a la vez. Se piensa que en futuros *Sprint* bajara y que los existentes se van a resolver más rápido. Reservar tiempo en la próxima planificación para resolver los bugs priorizando los que son parte de la primera liberación al cliente. Sacarle más jugo a las reuniones diarias con UI o arte. Comunicación más fluida entre el equipo y arte para minimizar las intervenciones del PM. Solicitar a arte apoyo en la diagramación de ciertas pantallas. Evaluar si vale la pena realizar ciertas reuniones o no. Definir objetivos para las reuniones y velar para que se cumpla al final.
8	Han quedado varias historias afuera de las comprometidas en la planificación. Contemplar el riesgo en las historias en la planificación tanto sea en la estimación como en las comprometidas a entregar. Tomar la experiencia de los *Sprint* anteriores para ajustar la estimación. Para las historias complejas, reservar una tarea de análisis para que se pueda identificar tempranamente si la estimación estuvo muy errada para ajustar la planificación. Redistribuir tareas de forma tal de que se pueda llegar a completar al menos una de las historias que no se llegaba al cierre del *Sprint*. Velar por todo el grupo de forma de poder identificar y destrabar impedimentos para llevar adelante su tarea.
9	No se completan todas las historias comprometidas para el Sprint. Nos estamos quedando cortos en las estimaciones. Obtener comparación de estimaciones históricas y lo que realmente llevo. Se trabajara sobre ello para mejorar las estimaciones. Tener en cuenta los imprevistos al momento de la planificación. Agregar una tarea a cada historia que indique realizar las pruebas cruzadas.
10	Incomodidad en el equipo por *sprint* a causa de fiestas y licencias. En caso parecido realizar un *sprint* de mayor duración o dedicarse a correcciones de bugs o *refactoreo* de código.

Sprint	Frases
	Verificar la tendencia en próximo sprint, si continua escalar a la gerencia. Se mantiene una cantidad importante de *bugs* abiertos.
11	Estimación y complejidad de las historias se está mejorando. Identificar aquellos que se puedan asignar a <Nombre> de UI. Estamos justos en el tiempo por lo cual hay que ajustar bien las planificaciones. Evaluar si es mejor proponer todos los temas y elegir solamente uno para atacar en el próximo *Sprint*.
12	Sigue creciendo la lista de bugs y en los próximos *Sprint* no habrá tiempo para resolver los mismos. En caso que haya un bug critico a resolver de inmediato notificar y asignar al equipo para ser resuelto en el *Sprint* en curso.
13	Quedan varios temas para destrabar con el <Cliente> lo cual afecta definición de las historias pendientes y datos de pruebas.

3. **Documentación Obsoleta:** las frases asociadas a la documentación obsoleta se presentan en la Tabla 12-3, las mismas se encuentra divididas por sprint según fueron observadas durante el proceso de codificación abierta.

Tabla 12-3. Frases codificadas por Sprint para el Factor Documentación Obsoleta.

Sprint	Frases
1	Calcular el total de horas estimadas en tareas. Capacidad estimada del equipo para ese sprint.
2	Aplazamos la misma historia 2 veces. Estimación pesimista.
3	Historias no completadas en su totalidad dentro del sprint. La historia vuelve al *back log*. Sobrecarga de trabajo en el líder técnico. LT no debería estar asignado al 100% de tareas de su capacidad. Registrar el tiempo dedicado a apoyo a otro miembro del equipo en la tarea en la cual realizo ese trabajo.
4	Planificación solo dedicar un % de horas para solución de *bugs*. Avisar cuando quede estabilizado el ambiente del <Cliente>. No dejarlo para cuando este la UI porque se producirá una sobrecarga del área.

Sprint	Frases
5	Nos estamos quedando cortos en las estimaciones. Considerar el riesgo de las operaciones complejas al momento de estimar las tareas considerando el tiempo de investigación del código anterior. Las historias de este *Sprint* no fueron completadas óptimamente. A medida de que se vaya incorporando la UI, se ajustarán y redondearán las historias. Hay que considerar que las historias irán en aumento de complejidad, por lo que hay que mejorar la estimación. Interacción con UI, se incluirán de forma de ayudar al equipo en la estimación de las historias, Es importante que QC pueda realizar los test cases a tiempo. QC tratará de ir un Sprint adelante con la realización de los test cases, sobre historias del *back log*.
6	Reservar tiempo en el próximo sprint para que se ataque en parte los puntos anteriores. Se incluirá a <Nombre> en las reuniones del próximo Sprint. Esto se debió al miedo en quedar cortos en la estimación y que luego no fuera suficiente el tiempo para culminar con las historias del sprint.
7	La cantidad de bugs reportados se explica porque se prueban varios *Sprint* a la vez. Se piensa que en futuros *Sprint* bajara y que los existentes se van a resolver más rápido. Reservar tiempo en la próxima planificación para resolver los bugs priorizando los que son parte de la primera liberación al cliente.
8	Han quedado varias historias afuera de las comprometidas en la planificación. Contemplar el riesgo en las historias en la planificación tanto sea en la estimación como en las comprometidas a entregar. Para las historias complejas, reservar una tarea de análisis para que se pueda identificar tempranamente si la estimación estuvo muy errada para ajustar la planificación. Redistribuir tareas de forma tal de que se pueda llegar a completar al menos una de las historias que no se llegaba al cierre del *Sprint*. QC propuso *testing* cruzado de forma de testear funcionalidades que QC no puede ver. Primer filtro para poder reducir la cantidad de bugs reportados por QC.
9	No se completan todas las historias comprometidas para el *Sprint*. Nos estamos quedando cortos en las estimaciones. Agregar una tarea a cada historia que indique realizar las pruebas cruzadas.
10	Sprint demasiado cortó. Incomodidad en el equipo por sprint a causa de fiestas y licencias.
11	Estamos justos en el tiempo por lo cual hay que ajustar bien las planificaciones.
12	Este Sprint fue muy justo y presionado por la entrega de la tercera entrega al <Cliente>. El equipo dedico tiempo extra y cumplió con lo comprometido.

Sprint	Frases
	Hay preocupación de poder cumplir con todo a tiempo y las próximas planificaciones estarán condicionadas por la información. Sigue creciendo la lista de bugs y en los próximos *Sprint* no habrá tiempo para resolver los mismos.
15	En este Sprint no se pudo avanzar mucho debido a que la capacidad estuvo limitada por licencias y dedicaciones a otros proyectos.

4. ***Time to market*:** las frases asociadas a la deuda de time to *market* se presentan en la Tabla 12-4, las mismas se encuentran ordenadas por *sprints* de acuerdo a como se identificaron durante el proceso de codificación abierta.

Tabla 12-4. Frases codificadas por Sprint para el Factor Time to *Market*.

Sprint	Frases
1	Capacidad estimada del equipo para ese *sprint*.
2	Aplazamos la misma historia 2 veces. Separar capacidad en los próximos *sprints*. Estimación pesimista. Planificación dentro del 20% de capacidad estimada. Descomponer tareas de complejidad >=21.
3	Historias no completadas en su totalidad dentro del sprint. La historia vuelve al *back log*. Sobrecarga de trabajo en el líder técnico. LT no debería estar asignado al 100% de tareas de su capacidad.
4	Como atacarlos: cada integrante lo maneja a su convenir. Los *bugs* que nadie tome serán asignados por el PM.
5	Nos estamos quedando cortos en las estimaciones. Considerar el riesgo de las operaciones complejas al momento de estimar las tareas considerando el tiempo de investigación del código anterior. Cuidar la asignación de tareas y el riesgo que implican. Hay que considerar que las historias irán en aumento de complejidad, por lo que hay que mejorar la estimación. Interacción con UI, se incluirán de forma de ayudar al equipo en la estimación de las historias UI se incluirá en las reuniones de seguimiento.

Sprint	Frases
	Es importante que cada uno haga el seguimiento y actualice información respecto a los impedimentos que tienen asignados. Es importante que el equipo pueda alimentar la información de las historias de cada Sprint.
6	Se incluirá a <Nombre> en las reuniones del próximo *Sprint*. Todas las estimaciones en horas estuvieron mucho más altas respecto a la dedicación real que llevaron. La estimación en puntos de complejidad estuvo demasiado elevada. Esto se debió al miedo en quedar cortos en la estimación y que luego no fuera suficiente el tiempo para culminar con las historias del *sprint*. Falta de conocimiento respecto a lo que se enfrentarían al integrar el diseño.
7	Reservar un tiempo para hacer un análisis y revisar el código anterior para identificar y cubrir todos los casos. Si se detecta algo muy grande que exceda la estimación se separara lo nuevo y se escribirá una nueva historia para esa parte.
8	Han quedado varias historias afuera de las comprometidas en la planificación. Contemplar el riesgo en las historias en la planificación tanto sea en la estimación como en las comprometidas a entregar. Tomar la experiencia de los *Sprint* anteriores para ajustar la estimación. Para las historias complejas, reservar una tarea de análisis para que se pueda identificar tempranamente si la estimación estuvo muy errada para ajustar la planificación. Redistribuir tareas de forma tal de que se pueda llegar a completar al menos una de las historias que no se llegaba al cierre del *Sprint*. Velar por todo el grupo de forma de poder identificar y destrabar impedimentos para llevar adelante su tarea.
9	Lecciones aprendidas (*bugs, refactoreo* y mejora de código), se definieron como acciones a llevar adelante durante todos los *Sprint*. No se completan todas las historias comprometidas para el *Sprint*. Nos estamos quedando cortos en las estimaciones.
10	Sprint demasiado cortó.
11	Estimación y complejidad de las historias se está mejorando. Evaluar si es mejor proponer todos los temas y elegir solamente uno para atacar en el próximo Sprint.
13	Quedan varios temas para destrabar con el <Cliente> lo cual afecta definición de las historias pendientes y datos de pruebas.

5. **Migración de sistemas *legacy*:** en la Tabla 12-5 se muestran las frases asociadas a Migración de sistemas *legacy*, las mismas surgen de la aplicación de codificación abierta y se presentan por *sprints*.

Tabla 12-5. Frases codificadas por *Sprint* para el Factor Like-to-like migration

Sprint	Frases
4	Estabilización del ambiente de la empresa. Avisar cuando quede estabilizado el ambiente del <Cliente>. No dejarlo para cuando este la UI porque se producirá una sobrecarga del área.
5	A medida de que se vaya incorporando la UI, se ajustarán y redondearán las historias. Se agregará una tarea de validación de la funcionalidad contra el ambiente del <Cliente>. Interacción con UI, se incluirán de forma de ayudar al equipo en la estimación de las historias Es importante que QC pueda realizar los test cases a tiempo. QC tratará de ir un *Sprint* adelante con la realización de los test cases, sobre historias del back log. Exponer las vulnerabilidades en el sitio que afecten la información de los clientes, definir temas pendientes.
6	Al comenzar a integrar el diseño a una funcionalidad, no se contaba con el total de los insumos. No todas las funcionalidades quedan correctamente finalizadas y existe un re trabajo cuando se quiere utilizar un nuevo insumo ya integrado. Se crearan tareas asociadas a las mejoras y ajustes que hay que realizar en las funcionalidades a las cuales ya se les integro el diseño. Falta de conocimiento respecto a lo que se enfrentarían al integrar el diseño.
7	Comunicación más fluida entre el equipo y arte para minimizar las intervenciones del PM. Solicitar a arte apoyo en la diagramación de ciertas pantallas.
8	Han quedado varias historias afuera de las comprometidas en la planificación. Contemplar el riesgo en las historias en la planificación tanto sea en la estimación como en las comprometidas a entregar.
11	Testing cruzado la experiencia fue positiva. Se detectaron ajustes menores a realizar pero nada desde el lado de la codificación. Estimación y complejidad de las historias se están mejorando. Ver todo con arte y MV, de allí identificar mejoras que serán aplicadas a todo el sitio. Estamos justos en el tiempo por lo cual hay que ajustar bien las planificaciones.

Sprint	Frases
	Evaluar si es mejor proponer todos los temas y elegir solamente uno para atacar en el próximo *Sprint*.
12	Este *Sprint* fue muy justo y presionado por la entrega de la tercera entrega al <Cliente>. El equipo dedico tiempo extra y cumplió con lo comprometido. En caso que haya un bug critico a resolver de inmediato notificar y asignar al equipo para ser resuelto en el *Sprint* en curso.
14	Corrección de *bugs* y temas de seguridad.

6. **Costo de Implementación:** las frases asociadas al costo de implementación se presentan en la Tabla 12-6, se exponen por *sprints* según surge el resultado de la aplicación de la codificación abierta.

Tabla 12-6. Frases codificadas por *Sprint* para el Factor Costo de Implementación.

Sprint	Frases
3	Planificar y resolver los bugs en los próximos *Sprints*.
5	Definir dinámica con la cual se planificarán, asignarán y corregirán los bugs. Planificación solo dedicar un % de horas para solución de *bugs*. Cada integrante se auto asigna los *bugs*. Los bugs que nadie tome serán asignados por el PM. Las mejoras/*refactoreos* se registraran como tareas en la historia del equipo a medida que se vayan viendo. Avisar cuando quede estabilizado el ambiente del <Cliente>. No dejarlo para cuando este la UI porque se producirá una sobrecarga del área.
6	Reservar tiempo en el próximo sprint para que se ataque en parte los puntos anteriores. Identificar qué corresponde cambiar y definir quien hace cada cosa. Lo mejor sería que cada uno ataque un punto completo. Al comenzar a integrar el diseño a una funcionalidad, no se contaba con el total de los insumos. No todas las funcionalidades quedan correctamente finalizadas y existe un re trabajo cuando se quiere utilizar un nuevo insumo ya integrado. Se crearan tareas asociadas a las mejoras y ajustes que hay que realizar en las funcionalidades a las cuales ya se les integro el diseño. Se tratara de reportar bugs genéricos de insumos o ajustes que falten para ser resueltos por <NOMBRE> (que rol tenia), para su reusó luego.

Sprint	Frases
	Las propuestas de mejoras serán centralizadas en el *Product Manager*, será el quien las apruebe o no.
9	Obtener comparación de estimaciones históricas y lo que realmente llevo. Se trabajara sobre ello para mejorar las estimaciones.
10	Se mantiene una cantidad importante de bugs abiertos.
11	Cantidad de *bugs* abiertos daremos prioridad en el sprint 13.
13	Siguiente Sprint se debería resolver *bugs*. La forma de planificarlos es agruparlos y asignarlos a quien haya desarrollado para poder hacer una estimación del tiempo de resolución.
14	Corrección de *bugs* y temas de seguridad.

7. **Añejamiento de Código:** como resultado de la aplicación de codificación abierta surgen las frases asociadas al factor añejamiento de código, las mismas se presentan en la Tabla 12-7 ordenadas por *sprint*.

Tabla 12-7. Frases codificadas por *Sprint* para el Factor Añejamiento de Código.

Sprint	Frases
4	Sería bueno contar con las siguientes métricas en cada reunión de retrospectiva.
5	Es importante que QC pueda realizar los test cases a tiempo. Test Cases aportan valor o es suficiente con el *checklist* y los criterios de aceptación, optimizar el tiempo de QC.
6	No se debe perder el foco, respetar los bocetos que fueron validados por el cliente.
7	Los bugs trabajados fueron ajustes de diseño y re trabajo que implico la incorporación de los mismos. Identificación y definición de estándares.
8	QC propuso *testing* cruzado de forma de testear funcionalidades que QC no puede ver. Primer filtro para poder reducir la cantidad de bugs reportados por QC.
9	Lecciones aprendidas (bugs, *refactoreo* y mejora de código), se definieron como acciones a llevar adelante durante todos los Sprint. *Testing* cruzado en el sprint 10 para mejorar la calidad.
10	Se mantiene una cantidad importante de *bugs* abiertos.

Sprint	Frases
11	*Testing* cruzado la experiencia fue positiva. Se detectaron ajustes menores a realizar pero nada desde el lado de la codificación. Cantidad de bugs abiertos daremos prioridad en el sprint 13. Ver todo con arte y MV, de allí identificar mejoras que serán aplicadas a todo el sitio.
12	Hay preocupación de poder cumplir con todo a tiempo y las próximas planificaciones estarán condicionadas por la información. *Testing* cruzado está siendo útil como un primer filtro de detección de errores. Sigue creciendo la lista de bugs y en los próximos *Sprint* no habrá tiempo para resolver los mismos. Se atacaran los bugs una vez finalizado el desarrollo de las funcionalidades de la última liberación. En caso que haya un bug critico a resolver de inmediato notificar y asignar al equipo para ser resuelto en el *Sprint* en curso.
14	Corrección de *bugs* y temas de seguridad.

8. **Falta de transparencia hacia otros *stakeholders*:** Las frases asociadas a la deuda de Falta de transparencia hacia otros *stakeholders* fueron las que se presentan en la Tabla 12-8.

Tabla 12-8. Frases codificadas por Sprint para el Factor Falta de Transparencia hacia otros *Stakeholders*.

Sprint	Frases
3	Planificar y resolver los bugs en los próximos *Sprints*
4	Aprueban y priorizan: el PM apruebe y priorice cada *bug* Cada integrante se auto asigna los *bugs* Los *bugs* que nadie tome serán asignados por el PM
7	Los bugs trabajados fueron ajustes de diseño y re trabajo que implicó la incorporación de los mismos Unificar criterios generales de forma que no se generen *bugs* similares en distintas funcionalidades La cantidad de *bugs* reportados se explica porque se prueban varios *Sprint* a la vez Reservar tiempo en la próxima planificación para resolver los bugs priorizando los que son parte de la primera liberación al cliente

Sprint	Frases
8	QC propuso *testing* cruzado de forma de testear funcionalidades que QC no puede ver Primer filtro para poder reducir la cantidad de bugs reportados por QC
9	*Testing* cruzado en el sprint 10 para mejorar la calidad
10	Próximo sprint incorporar como tarea dentro de la historia el *testing* cruzado para bajar bugs reportados por QC
11	Testing cruzado la experiencia fue positiva. Se detectaron ajustes menores a realizar pero nada desde el lado de la codificación Cantidad de bugs abiertos daremos prioridad en el sprint 13
12	Testing cruzado está siendo útil como un primer filtro de detección de errores Sigue creciendo la lista de bugs y en los próximos *Sprint* no habrá tiempo para resolver los mismos Se atacaran los bugs una vez finalizado el desarrollo de las funcionalidades de la última liberación En caso que haya un bug critico a resolver de inmediato notificar y asignar al equipo para ser resuelto en el *Sprint* en curso
14	Corrección de *bugs* y temas de seguridad Status general del proyecto y la proximidad con la fecha de finalización