Identifying Technical Debt Cost Factors in Reflection activities of an AgileProjects

Cecilia Nacimento, Santiago Matalonga
Universidad ORT Uruguay, ORT
Montevideo, Uruguay
{cnacimento; smatalonga}@uni.ort.edu.uy

Jean Carlo Rossa Hauck
Universidade do Sul de Santa Catarina (UNISUL)
Palhoça, Santa Catarina, Brasil.
jean.hauck@unisul.br

Absract – The technical debt metaphor has received wide attention by both practitioners and researchers alike. The metaphor has been proven useful to explain in economy's terms aspects which occur during the software development process. This work studies the applicability of the metaphor in reflection activities of agile projects. To achieve this, a qualitative research methodology has been carried out to study the records of a live agile development projects. Our results confirm the applicably of the proposal by the identification of terms related to the metaphor in this records. Moreover, our study has further validated the applicability of cost factors used in technical debt measurement reported in the literature.

Keywords—Technical Debt; Agile Development; Project Reflection; Open Coding

I. INTRODUCCION

La metáfora de Deuda Técnica se presenta como un término de interés para la comunidad de investigación y profesional de la ingeniería de Software. La metáfora es aplicada para explicar fenómenos que ocurren durante la producción de software. Por ejemplo, cuando en vistas a cumplir con una fecha de entrega se dejan de lado controles de calidad, se dice que el proyecto está incurriendo en Deuda Técnica. El proyecto deberá decidir en un futuro si pagarla (implementar los controles o mejorar el código), o asumirla (usualmente asumiendo costos de retrabado en mantenimiento). Como se nota en el ejemplo, la metáfora utiliza términos económicos de uso común para la explicación y comunicación de estos fenómenos.

La literatura reporta aplicaciones de la metáfora para explicar fenómenos de evaluación de arquitecturas, tomas de decisiones y costos de mantenimiento entre otros.

En este trabajo pretendemos evaluar su aplicación en instancias de desarrollo ágil de software. A nuestro leal saber y entender, no existen aún en la literatura trabajos que se centren en la evaluación de la metáfora de deuda técnica en instancias de reflexión de metodologías agiles de desarrollo.

En este trabajo aplicamos la técnica de Open Coding para analizar los registros de retrospectivas de un proyecto real de desarrollo de software que estaba siguiendo un proceso Scrum. Open Coding es una técnica de análisis cualitativo que se usa como herramienta en la metodología de Teoría justificada [1]. Durante esta técnica el objetivo de investigaciones es definido ampliamente y mediante iteraciones de codificación y abstracción se genera conocimiento que luego puede ser contrastado con la literatura existente.

En caso presentado en este trabajo, luego de 4 iteraciones con los registros de retrospectiva se identificaron constructos que podían relacionarse a factores de costo de Deuda Técnica. Estos factores de costo habían sido previamente identificados por otros investigadores de nuestro grupo de investigación.

Como resultados del presente trabajo se destacan la identificación de la aplicabilidad de la metáfora en instancias de reflexión, la confirmación de los factores de deuda técnica.

Este artículo está organizado de la siguiente manera. En la sección II se presentan los trabajos relacionados. Incluyendo un introducción a la metáfora, y los resultados de los factores de Deuda Técnica previamente alcanzados por el grupo y que son relevantes para comprender el resultado de la investigación reportada en este artículo. La sección III, describe el método de investigación seleccionado (Open codig/Teoría justificada) y presenta las adaptaciones realizadas en esta instancia. La sección IV presenta los resultados. Y finalmente la sección IV resume aportaciones del trabajo y presenta nuestras futuras líneas de investigación.

II. TRABAJOS RELACIONADOS

A. Breve Introduccion al Concepto de 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 paralicé el desarrollo futuro del proyecto. Estos elementos conforman la Deuda Técnica" [2].

Se presentan nuevas definiciones de dos reconocidos autores involucrados en la investigación de la metáfora. 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" [4].

En los últimos años se produjo un incremento en el conocimiento empírico acerca del tema, es a partir de aquí que se comienza a ampliar el espectro de fenómenos de la Ingeniería de software en que la metáfora de Deuda Técnica puede explicar. Se puede hacer referencia a Deuda Técnica a nivel de código [5], de diseño y arquitectura [6], de pruebas [7]. El proceso de Ingeniería de Software sobre el que existe mayor información y estudios realizados a hoy, es sobre Deuda Técnica generada a nivel de código, la mayoría de los estudios refieren a la forma en que pobremente se escribe el código [8], [9]. 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 por 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 [10].

B. Factores de costo de Deuda Técnica

Algunos de los trabajos de investigación actuales en la metáfora de Deuda Técnica se centran en identificar métricas para cuantificar la Deuda Técnica en los proyectos de software. Por ejemplo, [11] utilizan los resultados de un analizador estático de código, Findbugs[1], como medida indirecta de la deuda técnica de código. Por otro lado, [12] han usado la herramienta Sonar[2] como medida de la deuda técnica. Sonar, además de utilizar resultados de analizadores estáticos de código, complementa esta información con información de contexto del proyecto provista por el usuario. Por último, Kruchten [13] también utiliza indicadores basados en los diagramas estáticos de arquitectura para establecer métricas de deuda técnica de arquitectura que permitan la selección de alternativas arquitectónicas.

Dentro de este marco, nuestro grupo de investigación inicio un mapeo sistemático de la literatura con el objetivo de identificar todos los factores de costo que pudiesen ser utilizados en ecuaciones de costo/beneficio para la toma de decisiones. Este trabajo extiende una revisión sistemática anterior del grupo [8].

Esta sección no pretende ser un reporte de un mapeo sistemático, sino un resumen del proceso de investigación y la presentación de los resultados relevantes para este trabajo. Un reporte preliminar de la iteración 2014 se encuentra en [14].

Un mapeo sistemático tiene como objetivo establecer un mapa conceptual del conocimiento en un área [15]. El objetivo de este mapeo era identificar las tendencias de investigación en Deuda Técnica de manera de mantener al equipo de trabajo actualizado, e intentar identificar factores de costo que pidieran utilizarse en métricas de seguimiento de Deuda Técnica.

El mapeo fue realizado utilizando los siguientes strings de búsqueda “technical debt”; “design debt”; “analysis debt”, “requirements debt”, “testing debt”, “configuration management debt”, “process debt”, “architecture debt”, ”architectural debt”, “people debt”, “documentation debt”, “code quality debt”. Y fue ejecutado en las siguientes bases de datos: ACM; Springer; IEEE; SciVerse; Citeseerx.

En total se identificaron 59 factores que se presentan en la tabla siguiente:

Tabla 1 Factores de Costo de Deuda Técnica

Id Factor	Factores de Costo Descripción
1	Compromiso a corto y largo plazo (nivel código y diseño)
2	Falta de conocimiento/experiencia
3	Bajo refactoring y rediseño
4	Time to market
5	Refactoring
6	Deterioro de la arquitectura
7	Deterioro del código /diseño
8	Requerimientos mal definidos y/o cambiantes
9	Falta de tiempo / budget
10	Pobre codificación en etapa de desarrollo
11	Documentación Obsoleta
12	Mal control de versionado
13	Excesivo testing manual
14	Tiempo
15	Cambios rápidos en tecnologías
16	Poco esfuerzo y poca motivación
17	Gerenciamiento de proyectos caótico
18	Obsolescencia tecnológica
19	Calidad
20	Mala Planificación
21	Añejamiento de Código
22	Deuda de Testing
23	Smells
24	Falta de transparencia hacia otros stakeholders

[1] http://findbugs.sourceforge.net/

[2] http://www.sonarqube.org/evaluate-your-technical-debt-with-sonar/

Id Factor	Factores de Costo Descripción
25	Nuevos requerimientos en entornos legados
26	Cambios en el equipo
27	Falta de fondos para realizar los cambios necesarios
28	Mala integración y manejo de liberaciones
29	Compromiso en una dimensión en detrimento de otra
30	Migración de sistemas legacy
31	Cambio de estándares de calidad en el tiempo
32	Nuevas oportunidades de negocio
33	Costos de implementación / retrabajo
34	Pobre adherencia a estándares de desarrollo
35	Costo de Implementación
36	Retrabajo
37	Atajos de Diseño
38	Compromiso a corto y largo plazo (Proceso)
39	Compromiso a corto y largo plazo (nivel arquitectura)
40	Código Obsoleto
41	Estructura del sistema
42	Decaimiento de código
43	Complejidad de código
44	Cobertura de pruebas unitarias
45	Código duplicado
46	Deuda de diseño
47	Defectos
48	Falta de mejora de procesos
49	Falta de conocimiento
50	Energía
51	Corrección de defectos
52	Adherencia a estándares de código (des adherencia)
53	Pasaje de defectos en un sprint (Defects carry over)
54	Testear al final
55	Deuda Funcional (funcionalidades prometidas no entregadas)
56	Código legado pobre
57	Inexperiencia
58	Adquisición de software legado
59	Imprudencia

Durante el proceso de codificación y abstracción llevado adelante para esta investigación, surgió como categoría núcleo que las decisiones con las que se estaba encontrando el equipo de desarrollo podían representarse en términos de estos factores. Por tanto, para este estudio se hace necesario la explicación previa de la fuente de estos 59 factores que serán utilizados como objetivo de abstracción del proceso de investigación descrito en la sección III.

C. Instancias de reflexion en proyectos de desarrollo ágil

Los proyectos de desarrollo que utilizan metodologías ágiles se presentan ya como una alternativa real al desarrollo de software. Como lo indica una reciente encuesta [16], cerca del 40% de los proyectos de desarrollo se identifican con una metodología ágil.

El termino agilidad vienen de la firma del manifiesto ágil[3], 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 asa mejorarlo continuamente".

La práctica de reflexiones en proyectos es anterior a las metodologías agiles, y el esfuerzo por incorporar prácticas de reflexión en metodologías ágiles ha venido por trabajos que han primero intentado implementar procesos de reflexión utilizados en metodologías tradicionales [17].

En los últimos años ha habido más atención hacia la aplicación de técnicas que mejoren los resultados de las instancias de reflexión. En [18], 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 [17] – cuando este último sigue una metodología distinta de investigación.

Otros métodos para la realización de actividades de reflexión son propuestos por [19] y [20]. Sin embargo, a nuestro leal saber entender únicamente en el trabajo de [21], se vincula el concepto de Deuda Técnica con el de actividades de reflexión. Sin embargo en este último trabajo, 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 presentamos 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.

D. Introduccion a la técnica Open Coding

La Figura 1 presenta el proceso general de Teoría Justificada (de aquí en adelante GT por sus siglas en inglés) propuesta por [22] para su aplicación en Ingeniería de software. Esta sección describe la metodología y la técnica de codificación utilizada.

[3] Agilemanifest.org

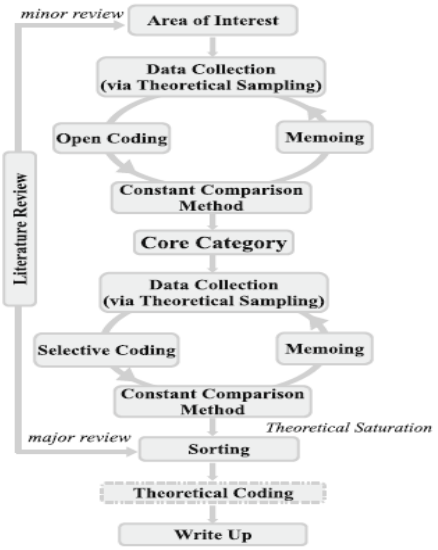


Figura 1 Proceso Grounded theory propuesto por Hoda [23]

Área de Interés: el método GT tiene la intención de generar teoría es por esto que recomienda abstenerse de generar una pregunta de investigación con antelación. Lo que se le solicita al investigador es identificar un área de interés general.

Revisión Mínima de la Literatura: el método GT 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.

Recolección de Datos: muestreo teórico se denomina en GT 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.

Reclutamiento de participantes: 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.

Entrevistas y Observaciones: el dictamen en GT 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.

Análisis de Datos: se puede comenzar el análisis de datos llamado codificación en el GT 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.

Método de Comparación Constante: 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 GT, la comparación que se utilizó de nuevo para agrupar estos códigos y producir un nivel de abstracción más alto, llamado conceptos en GT.

Core Categoría: 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.

Codificación selectiva: 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 GT. 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.

Theorical 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.

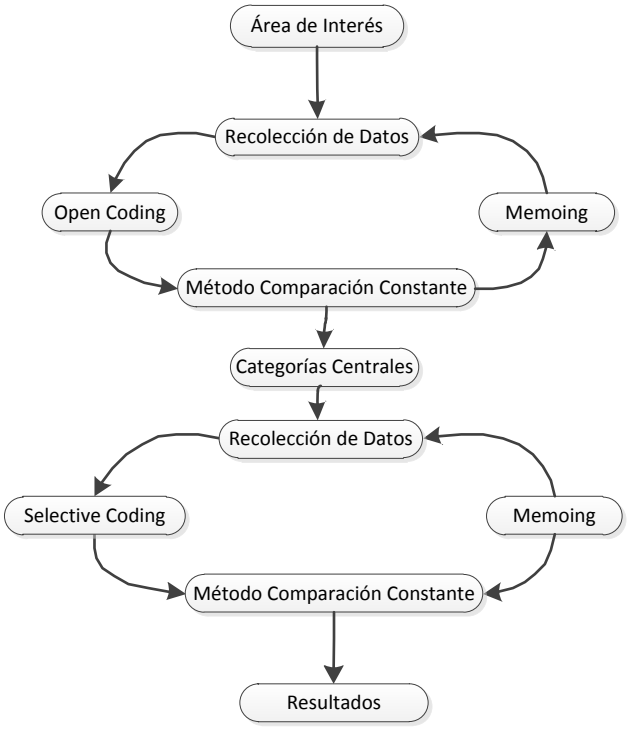
Write–up: el último paso en GT es escribir la teoría, que sigue el teórico generado como resultado de la clasificación y codificación teórica [23].

III. PROCESO DE IDENTIFICACION DE FACTORES

El objetivo del presente trabajo es identificar si en las instancias de retrospectivas de un proyecto de desarrollo ágil, se identifican constructos que pueden asociarse a la metáfora o a fenómenos que la literatura indica pueden ser explicados con la metáfora de Deuda Técnica.

Como los investigadores involucrados habían participados en instancias de la revisión sistemática ya mencionada [8] [9] [11], no podemos afirmar que el conocimiento del fenómeno de Deuda técnica fuese nuevo para nosotros. Sin embargo su aplicación en instancias de retrospectiva si lo era.

Teniendo en cuenta esto último y la evidencia con que el quipo cuenta se decidió adaptar la técnica poniendo foco en el proceso de codificación y abstracción. El proceso de investigación que se utilizamos para este trabajo es una adaptación del propuesto por [23] para la aplicación de teoría justificada en Ingeniería de Software.



A. *Área de Interes*

El área de interés se define como el estudio de la aplicación la Metáfora de Deuda Técnica en Proyectos ágiles de desarrollo. Para iniciar el ciclo se tomo un nivel de abstracción anterior al de nuestra pregunta de investigación para ver si a través de los informes de retrospectiva era posible identificar otras instancias de aplicación de la metáfora.

B. *Recoleccion de Datos*

Los datos para la realización de la codificación abierta (Open Coding) surgen de informes de retrospectivas de un proyecto de desarrollo ágil de la Industria uruguaya de Software. El proyecto aplicaba la metodología de desarrollo Scrum.

Se dispone de 15 documentos que representan las 15 iteraciones que se realizaron en ese proyecto. En estos informes el equipo de desarrollo describió

C. *Open Coding, Constant Comparison y Memoing*

Se realizaron 3 iteraciones sobre los informes de retrospectivas. La primera pasada se identificó que tareas se podrían tomar como generadoras de Deuda Técnica en el proyecto. En la segunda se revisaron las tareas contra los informes y las propias tareas identificadas en la pasada anterior. De esta forma se depuraron y se consolidaron las tareas de manera de obtener el conjunto relevante al estudio.

D. *Categoría núcleo*

Una vez que se identificaron las tareas relevantes de cada sprint se mapean con los factores de Deuda Técnica identificados en un estudio previo [xx], en esta etapa se identifican los factores que pueden ser provocados en cada una de las tareas. A partir de las relaciones surgen las categorías centrales, las mismas son:

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

E. *Selective Coding, Constant Comparison Method, Memoing*

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.

F. *Resultados Obtenidos*

Esta sección presenta los resultados obtenidos del proceso de codificación y abstracción. La Tabla 1 presenta el resultado final del mapeo de los conceptos asociados a la metáfora con los factores de costo de deuda técnica.

Tabla 2. Factores, descripción y frecuencia

Id Factor	***Factor de Deuda Técnica***	***Frecuencia***
22	Deuda de Testing	104
19	Calidad	76
11	Documentación Obsoleta	42
4	Time to market	40
30	Migración de sistemas legacy	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	Retrabajo	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
53	Pasaje de defectos en un sprint (Defects carry over)	8
18	Obsolescencia tecnológica	7
33	Costos de implementación / retrabajo	7
54	Testear al final	6
57	Inexperiencia	6
39	Compromiso a corto y largo plazo (nivel arquitectura)	5
3	Bajo refactoring y rediseño	4
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 / budget	3
44	Cobertura de pruebas unitarias	3
47	Falta de mejora de procesos	3
55	Deuda Funcional (funcionalidades prometidas no entregadas)	3
42	Complejidad de código	2
45	Deuda de diseño	2
56	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
50	Energía	1
51	Corrección de defectos	1
59	Imprudencia	1

A continuación se explica la asociación de los factores que recibieron más menciones.

Deuda de Testing

Las frases asociadas a la deuda de testing fueron se presentan en la siguiente tabla.

Tabla 3 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 sprint 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 esta 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 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 Seria 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 Riesgos se repaso, riesgo 4 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 seria 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>, para su reusó luego 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 mas 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 mas rápido Reservar tiempo en la próxima planificación para resolver los bugs priorizando los que son parte de la primera liberación al cliente Sacarle mas jugo a las reuniones diarias con UI o arte Comunicación mas fluida entre el equipo y arte para minimizar las intervenciones del PM Solicitar a arte apoyo en la diagramación de ciertas pantalla 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	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 corto 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 esta 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 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 Se atacaran los bugs una vez finalizado el desarrollo de las funcionalidades de la ultima 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

Calidad

Las frases asociadas a la deuda de Calidad se presentan en la siguiente tabla:

Tabla 4 Frases codificadas por Sprint para el Factor Calidad

Sprint	***Frases***
1	No se tenia la descomposición de tareas estimadas Impacto de atraso del equipo en áreas de soporte
2	Separar capacidad en los próximos sprint 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 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 Seria 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 Riesgos se repaso, riesgo 4 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 seria 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 mas 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	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 mas rápido Reservar tiempo en la próxima planificación para resolver los bugs priorizando los que son parte de la primera liberación al cliente Sacarle mas jugo a las reuniones diarias con UI o arte Comunicación mas fluida entre el equipo y arte para minimizar las intervenciones del PM Solicitar a arte apoyo en la diagramación de ciertas pantalla 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 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 esta 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

Documentación Obsoleta

Las frases asociadas a la documentación obsoleta se presentan en la siguiente tabla:

Tabla 5 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
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 mas 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 corto 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 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

Time to market

Las frases asociadas a la deuda de Time to market se presentan en la siguiente tabla:

Tabla 6 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 sprint 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 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 corto
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

Migración de sistemas legacy

Las frases asociadas a Migración de sistemas legacy se presentan en la siguiente tabla:

Tabla 7 Frases codificadas por Sprint para el Factor Lke-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 Riesgos se repaso, riesgo 4 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 mas fluida entre el equipo y arte para minimizar las intervenciones del PM Solicitar a arte apoyo en la diagramación de ciertas pantalla
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 esta mejorando Estimación y complejidad de las historias se esta 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 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

Costo de Implementación

Las frases asociadas al costo de implementación se presentan en la siguiente tabla:

Tabla 8 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 seria 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 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 deberían 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

Añejamiento de Código

Las frases asociadas al factor añejamiento de código ser presentan en la siguiente tabla:

Tabla 9 Frases codificadas por Sprint para el Factor Añejamiento de código

Sprint	*Frases*
4	Seria 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
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 esta 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 ultima 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

Falta de transparencia hacia otros stakeholders

Las frases asociadas a la deuda de Calidad fueron:

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 implico 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
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 esta 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 ultima 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

Como se observa en las tablas anteriores (de 3 a 9), de los registros de las retrospectivas de un proyecto real de desarrollo es posible identificar conceptos que pueden ser asociados a la metáfora de Deuda Técnica. Eso es una conformación más de que la metáfora es útil para explicar fenómeno que ocurren durante el desarrollo de software. Además, el proceso de mapeo confirma, en un proyecto real de desarrollo, la validez de los factores de costo de deuda técnica. Estos factores fueron identificados en la literatura académica mediante un proceso de revisión sistemática, y 42 de ellos fueron observados en las discusiones de retrospectivas del proyecto.

IV RESUMEN Y FUTURAS LINEAS DE INVESTIGACIÓN

En este trabajo se presentó la utilización de una metodología cualitativa para realizar el análisis de los registros de retrospectivas de un proyecto real de producción de software.

La metodología de investigación utilizada tiene su origen la teoría justificada, que es un proceso cualitativo de investigación cuyo objetivo es la generación de teoría a partir de registros y la observación de un fenómeno. En el trabajo aquí descrito se realizó el análisis de 15 sprints, que representan todo el proyecto. El proceso de codificación y abstracción de conocimiento permitió confirmar la aplicabilidad de la metáfora de Deuda Técnica para explicar fenómenos que ocurren durante la producción de software.

Además, en los registros de las retrospectivas, fue posible abstraer la presencia de los factores de costo de Deuda Técnica. Confirmando la validez de un trabajo de investigación previo del grupo.

El próximo paso consiste en la utilización de los factores durante las instancias de reflexión de metodologías agiles de desarrollo como mecanismo para hacer visible los posibles problemas a los que el proyecto se encuentra. Es decir, aplicar la metáfora a las instancias de retrospectiva para evaluar el impacto de los problemas y las soluciones planteadas, y observar si la misma es útil para mejorar el proceso de producción de software.

REFERENCIAS

[1] Glaser B (2010) Grounded theory institute: methodology of Barney G Glaser, 2010. URL http://groundedtheory.org/, accessed on April 2

[2] Cunningham W., "The wycash portfolio management system," OOPSLA '92, Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum), pp. 29, 1992.

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

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

[5] 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.

[6] B. Curtis, J. Sappidi, and J. Subramanyam, "Mensuring the Structural Quality of Business Applications," in 2011 AGILE Conference, 2011, pp. 147-150.

[7] S. Muhammad, A. L. I. Shah, M. Torchiano, A. Vetro, and M. Morisio, "Exploratory testing as a source of testing technical debt," 2013.

[8] A. Villar and S. Matalonga, "Definiciones y tendencia de deuda técnica: Un mapeo sistemático de la literatura," in *Memorias de la XVI Conferencia Iberoamericana de Ingeniería de Software CIbSE 2013*, 2013, pp. 33–46.

[9] Matalonga S., Villar A., Nacimento C., "Deuda Técnica: ¿Cuáles son los límites de la metáfora?" [Online] Available: http://www.ort.edu.uy/fi/pdf/documento10fi.pdf

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

[11] N. Zazworka, A. Vetro', C. Izurieta, S. Wong, Y. Cai, C. Seaman, and F. Shull, "Comparing four approaches for technical debt identification," *Softw. Qual. J.*, Apr. 2013.

[12] .-L. Letouzey, "The SQALE method for evaluating Technical Debt," *2012 Third Int. Work. Manag. Tech. Debt*, pp. 31–36, Jun. 2012.

[13] R. L. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, "In Search of a Metric for Managing Architectural Technical Debt," *2012 Jt. Work. IEEE/IFIP Conf. Softw. Archit. Eur. Conf. Softw. Archit.*, pp. 91–100, Aug. 2012.

[14] Villar A, Matalonga S, Factors affecting Technical Debt Raw data from a systematic literature map. Reporte de Investigación nro 12. Universidad ORT Uruguay. Disponible en: http://www.ort.edu.uy/index.php?id=AAAHBH

[15] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, "Using Mapping Studies in Software Engineering," vol. 2, 2007.

[16] Beguel, Nagappan. Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. ESEM '07 Proceedings of the First International Symposium on Empirical

[17] C. J. Stettina and W. Heijstek, "Five Agile Factors : Helping Self-management to Self-reflect," in *European Software Process Improvement Conference EuroSPI 2011*, 2011, pp. 84–96.

[18] 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.

[19] M. Ringstad, T. Dingsøyr, and N. Brede Moe, "Agile Process Improvement: Diagnosis and Planning to Improve Teamwork," in in Systems, Software and Service Process Improvement SE - 15, vol. 172, R. O'Connor, J. Pries-Heje, and R. Messnarz, Eds. Springer Berlin Heidelberg, 2011, pp. 167–178.

[20] E. Bjarnason and B. Regnell, "Evidence-Based Timelines for Agile Project Retrospectives – A Method Proposal," in *13th International Conference, XP 2012*, 2012, pp. 177–184

[21] C. B. 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.

[22] Glaser, Barney G., and Anselm L. Strauss. The discovery of grounded theory : strategies for qualitative research. Chicago: Aldine Pub. Co, 1967.

[23] R. Hoda, J. Noble, and S. Marshall, "Developing a grounded theory to explain the practices of self-organizing Agile teams," Empirical Software Engineering, vol. 17, no. 6, pp. 609–639, Apr. 2012.