Modelo para la Automatización de Relaciones de Trazabilidad en Procesos Scrum

Roberto Nazareno[1], Verónica Bollati[2], Horacio Leone[1], Silvio Gonnet[1]

[1]INGAR, Instituto de Desarrollo y Diseño
Universidad Tecnológica Nacional, CONICET
Avellaneda 3657, Santa Fe, Argentina
{rnazareno, hleone, sgonnet}@santafe-conicet.gov.ar

[2]Facultad Regional Resistencia
Universidad Tecnológica Nacional - CONICET
Resistencia, Argentina
vbollati@gmail.com

Abstract

La trazabilidad es considerada en el ámbito de las prácticas ágiles como un aspecto fundamental a estudiar para desarrollar sistemas de calidad. Sin embargo, los procesos ágiles ocurren en entornos donde no es frecuente encontrar un documento de especificación de requerimiento, no siendo posible aplicar técnicas clásicas de trazabilidad. En consecuencia, en este trabajo se propone un modelo de trazabilidad basado en las prácticas de Scrum, que permita representar las trazas existentes entre los artefactos generados durante procesos Scrum. La propuesta es validada empleando información disponible de la ejecución de un proceso ágil a escala industrial.

1. Introducción

Las técnicas o prácticas ágiles representan un conjunto de procesos de desarrollo, en el que tanto los requerimientos como el producto entregado, evolucionan de forma incremental a través de una serie de iteraciones cortas. Este tipo de proyectos se caracteriza por el énfasis en las interacciones humanas y colaboraciones, los procesos de desarrollo livianos, entregas frecuentes y con la documentación necesaria que aporte valor al proceso de desarrollo [1]. Siguiendo los valores propuestos en el manifiesto ágil [2], las prácticas ágiles, como Scrum [3], hacen hincapié en la necesidad de trabajar en estrecha colaboración con los clientes para dar valor al producto y para resolver los problemas a medida que surgen en lugar de confiar en los contratos rígidamente definidos. Por último, destacan la importancia de la evolución de las necesidades en lugar de seguir severamente un plan de desarrollo pre-establecido.

En el contexto de la agilidad, se considera a la trazabilidad como un aspecto fundamental para desarrollar sistemas de calidad [4] [5]. Para el desarrollo de este trabajo, definimos a la trazabilidad como el grado en el cual se puede establecer una relación entre dos o más productos del proceso de desarrollo (traza), especialmente en productos que posean una relación de predecesor-sucesor o superior-subordinado[6]. Considerar la trazabiliad en proyectos ágiles de gran tamaño es fundamental [7]. Proyectos con gran número de requerimientos requieren de técnicas y herramientas que faciliten su gestión. Las herramientas tradicionales de trazabilidad han demostrado su valor a lo largo de varias décadas de uso y pueden ayudar en la gestión de proyectos ágiles. Es muy posible que la trazabilidad no resuelva completamente los problemas de comunicación pero sí es un elemento clave para acercarse a la solución apropiada. Asumiendo que la trazabilidad contribuye a encontrar inconsistencias e impedimentos de manera rápida, contribuiría a mejorar la transparencia del proceso y dependencias entre artefactos para guiar el análisis de aquellos sistemas más complejos. Además la información de trazabilidad ayuda a la predicción del esfuerzo requerido para integrar el cambio [8] y brindar soporte a la evolución del proceso. La trazabilidad puede ser realizada de forma manual, semiautomática o automática [9]. Los modelos de trazabilidad automática tienden a seguir tres

direcciones para la creación de enlaces y el mantenimiento [10]. La primera consiste en aplicar minería de textos y técnicas de recuperación de información para recobrar los distintos enlaces entre los artefactos de software. La segunda línea deriva enlaces de trazabilidad desde los artefactos ya existentes. La tercera dirección es a partir de la automatización de transformaciones de modelos. Una vez que cada modelo es transformado, dicha transformación puede ser estudiada y se establecen enlaces de trazabilidad entre elementos del modelo o de dos versiones del modelo.

A partir de lo mencionado, el enfoque del presente trabajo se centra en automatizar la derivación de los enlaces de trazabilidad de artefactos generados en el proceso Scrum. La propuesta consiste en la definición de un modelo conceptual de procesos Scrum. El modelo captura la ejecución de un proceso Scrum y está definido en término de sus constructores esenciales: roles, eventos, y artefactos. A partir de este modelo se definen las distintas relaciones de trazabilidad que son posibles recuperar desde una ejecución del proceso.

A continuación se presentan los trabajos relacionados, luego, en la sección 3 se presenta un modelo de referencia de trazabilidad del proceso Scrum en el que se basa la presente propuesta y las distintas relaciones de trazabilidad que son factibles de recuperar. En la Sección 4 se presenta un caso de estudio basado en el análisis de la información disponible sobre la ejecución de un proyecto Scrum a escala industrial: el proyecto Moodle [11]. Por último, en la Sección 5 se presentan las conclusiones y trabajos futuros.

2. Trabajos Relacionados

En los proyectos ágiles, las necesidades varían de uno a otro y por lo tanto sus restricciones de trazabilidad difieren de forma significativa. Espinoza y Garbajosa remarcan en [5] este hecho proponiendo un enfoque basado en modelos para la planificación de la trazabilidad en los proyectos ágiles. Estos autores abogan por realizar trazabilidad ajustada a los objetivos, permitiendo al usuario establecer enlaces de trazabilidad. Es bien conocido que los desarrolladores emplean una gran cantidad de esfuerzo en documentar y mantener la información de trazabilidad. Los estudios existentes realizados sobre trazabilidad de software tradicional están enfocados en reducir costos asociados con este esfuerzo manual, desarrollando ayudas automáticas en el establecimiento y mantenimiento de enlaces de trazabilidad entre artefactos de software [12] [13]. La trazabilidad automática aplica técnicas de recuperación de información para generar enlaces candidatos. De esta forma se busca reducir considerablemente el efecto de los enfoques manuales, para construir y mantener una matriz de requerimientos de traza, y de igual forma proporcionar trazabilidad en los documentos legados [14] [15]. Estas propuestas también reflejan que implementando trazabilidad automática, el trabajo de los desarrolladores no debería verse incrementado de manera significativa.

El escenario más común para el seguimiento de un proyecto ágil se representa en el modelo de información de trazabilidad (TIM), resumido en [1]. Este modelo establece relaciones de trazabilidad entre los test de aceptación y las historias de usuario. Las trazas son incorporadas como referencias cruzadas a las historias de usuarios en los test de aceptación, o viceversa. Además, existen trazas implícitas entre el código fuente que aprueba los tests de aceptación y las historias de usuario trazadas por esos tests. Sin embargo, el código es manipulado como un artefacto atómico y no permite la distinción de qué partes de él están trazadas con los distintos tests de aceptación. Por este motivo Cleland-Huang en [1] limita este enfoque al seguimiento de proyectos pequeños y pone en duda los beneficios reales del empleo de TIM brindando soporte en las distintas actividades que un modelo de trazabilidad debería facilitar, como ser análisis de impacto y conformidad del producto. Un modelo de trazabilidad debería definir sus potenciales usuarios, cuál es el posible uso de las trazas definidas, cuál es la información que se puede recobrar del análisis de las trazas capturadas y cómo se puede utilizar esa información para que sea útil en la práctica diaria.

Como respuesta a las cuestiones planteadas en los párrafos previos se propone un modelo conceptual de trazabilidad en procesos Scrum. Como punto de partida para la propuesta del modelo de trazabilidad se consideran los conceptos introducidos en el metamodelo de trazabilidad propuesto por Ramesh y Jarke [16]. Este metamodelo es ampliamente utilizado para la representación de la trazabilidad en procesos de desarrollo de software y permite definir trazas entre dos artefactos como tripletas (Object, TraceTo, Object). A partir del empleo de estas tripletas, nuestra propuesta representa la trazabilidad entre los artefactos (Object) generados en los distintos eventos involucrados en el proceso Scrum.

El modelo propuesto es validado a partir del estudio de procesos desarrollados a escala industrial con Scrum, en particular se analiza el proceso de desarrollo de la plataforma de aprendizaje Moodle [11]. Moodle está diseñado con el objetivo de brindar un sistema integrado que permita crear ambientes de aprendizaje personalizados. Esta plataforma se encuentra desarrollada por un equipo perteneciente a la organización desarrolladora de Moodle, por la comunidad internacional y además por una red de socios certificados. Está dirigido por colaboración abierta donde la comunidad sirve de apoyo al proyecto. Para llevar a cabo este proyecto se utiliza Scrum como proceso de desarrollo. A su vez, Moodle utiliza el sistema de seguimiento de proyectos e Issues Moodle Tracker [17]. Esta herramienta de gestión

extiende a Atlassian Jira, permitiendo almacenar y gestionar todos los Issues, errores, mejoras y solicitudes de características correspondientes al desarrollo del proyecto y sus sistemas relacionados. Tracker se conecta a diferentes bases de datos de subproyectos, permitiendo realizar consultas sobre el desarrollo de la aplicación para validar el modelo propuesto.

3. Modelo Conceptual de Scrum

En esta sección se propone un modelo conceptual de trazabilidad del proceso Scrum. El modelo representa los constructores esenciales de Scrum: *roles*, *eventos*, *artefactos* y las reglas que permiten asociar estos conceptos. Estos conceptos y sus relaciones nos permiten describir, de manera abstracta, las relaciones de trazabilidad implícitas en el proceso de desarrollo.

Los roles describen responsabilidades y niveles de autoridad de individuos, o gru-pos de individuos, que participan durante el proceso. En la Figura 1 se representan los roles definidos en Scrum: ScrumTeam, *ProductOwner*, *ScrumMaster*, y *Develop-mentTeam*. Para mantener la trazabilidad de un proceso ágil es preciso modelar los roles debido a que son estos los que tendrán la responsabilidad de llevar a cabo dife-rentes actividades durante la ejecución del proceso, siendo fuentes de los distintos artefactos generados.

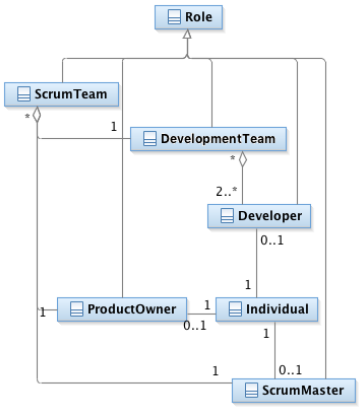


Figura 1. Roles definidos en Scrum.

Los eventos son ocurrencias en el tiempo de un conjunto de reuniones o acciones, utilizados con el objetivo de sincronizar las diferentes etapas por las que atraviesa un proceso Scrum. Son importantes en el modelado debido a que es en ellos donde la trazabilidad tiene lugar en el espacio temporal. En la Figura 2 se representa a los eventos y las relaciones entre sí mediante una asociación predecessor – successor. La realiza-ción de los distintos eventos requieren de artefactos y geenran nuevos artefactos, por tal motivo conforman tripletas de traza: (artefacto fuente; evento enlace de traza; artefacto destino).

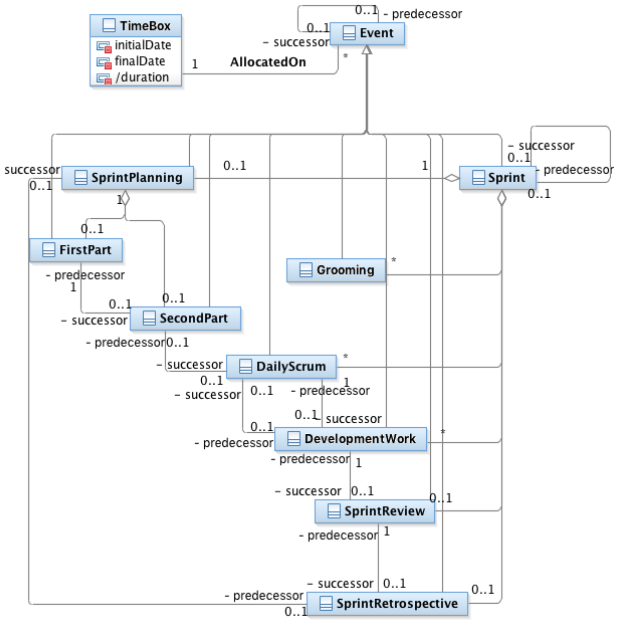


Figura 2. Eventos definidos en Scrum.

Un artefacto de software es la definición de un producto de trabajo. Los artefactos pueden estar compuestos por otros artefactos y son productos de trabajo concretos consumidos, producidos o modificados por los distintos eventos del proceso Scrum. Como se mencionó en el párrafo previo, un Artefact (Figura 3) es el origen y/o destino de los distintos tipos de trazas. Como resultado de la ejecución de distintos eventos se identifican relaciones entre los artefactos, por ejemplo, un ProductBacklogItem es refinado mediante un evento de Grooming en un conjunto de ProducBacklogItem. Este refinamiento es representado en la Figura 3 por la asociación RefinedOn. Algunos de estos enlaces pueden estar reificados y ser modelados como artefactos. Por ejemplo, en la Figura 3 se define una tripleta de trazabilidad entre ProductIncrement (object fuente), SprintBacklog (TraceTo), y ProductBacklogItem (object destino). SprintBacklog resulta de la ejecución de un Sprint reificando el enlace entre fuente y destino. Este concepto permite trazar qué ítem fue considerado en un incremento realizando un Sprint particular. Es posible definir una regla de derivación en OCL para inferir esta relación. En la ecuación (1) se especifica dicha regla.

```
context ProductBacklogItem:source:
Set(ProductIncrement)
derive:self.sprintBacklog.productIncrement
                                        (1)
context ProductIncrement:target:
Set(ProductBacklogItem)
derive:
self.sprintBacklog.productBacklogItem
```

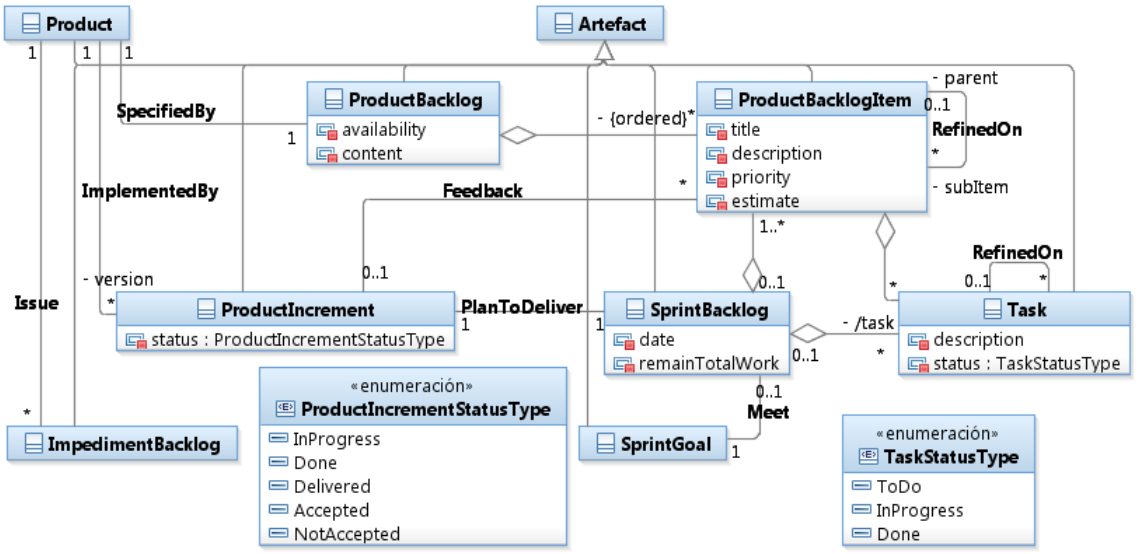


Figura 3. Artefactos definidos en Scrum.

3.1. Relaciones entre Roles, Eventos y Artefactos

Las relaciones entre los distintos constructores de Scrum permiten a los Stakeholders responder algunas preguntas sobre cómo un proceso Scrum fue llevado a cabo. Por ejemplo, ¿Cuál Stakeholder fue responsable de un artefacto? En Scrum, un rol es responsable de un artefacto; esto simplifica trazar a los distintos responsables de los artefactos. Sin embargo, otros roles pueden usar y actualizar artefactos con el permiso adecuado. En una contribución previa, [18], se incluyen las vistas que representan las relaciones entre Roles, Eventos, y Artefactos.

3.2. Usos y relaciones de trazabilidad en Scrum

Pohl [19] compara los diversos aspectos de la trazabilidad, usos, relaciones, documentación, presentación, y trazabilidad específica al proyecto. En [20] Spanoudakis y Zisman organizan los tipos de relaciones de trazabilidad en 8 grupos principales, por cuestiones de espacio se muestran los siguientes: *dependencia*, *generalización/refinamiento*, *evolución*, *satisfacción* y *solapamiento*. A continuación detallaremos los tipos de relaciones de las que el modelo hará uso.

Dependencia: es una relación en la que manifiesta que si un artefacto A depende de un artefacto B, cuando el elemento B es modificado implica que el elemento A se ve afectado por esta relación. En un Sprint de Scrum se identifican relaciones de este tipo entre ProductBacklogItems y entre Tasks dependientes entre sí (asociación DependsOn en Figura 4). Este tipo de relación es transitiva, por lo que puede ser inferida según lo enunciado en la ecuación (2).

```
context Item:dependsOn: Set(Item)
derive: self->clousure(target)
```

(2)

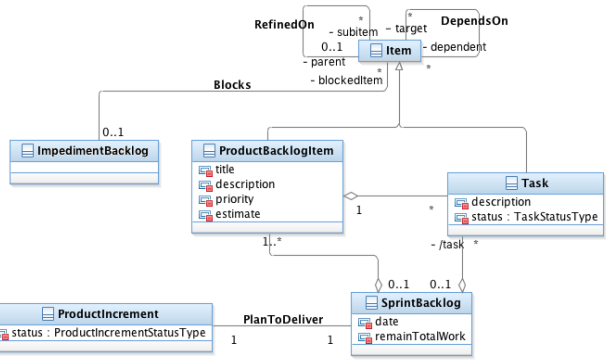


Figura 4. Relación de Dependencia.

El ImpedimentBacklog es una lista de dificultades, impedimentos u obstáculos que limitan el rendimiento del DevelopmentTeam. Si una Task fue planeada para el Sprint en curso y no fue finalizada durante el DevelopmentWork, es registrada como un impedimento, dicha traza (asociación Blocks en Figura 4) tiene a ImpedimentBacklog como artefacto fuente y a Task como artefacto destino (blockedItem en Figura 4).

También, durante un Sprint, se da la relación de dependencia entre ProductBacklogItems, o Tasks, y ProductIncrements. Si un ítem es considerado en un Sprint para producir un incremento de producto, entonces, ese incremento depende de tal ítem. Este hecho fue representado en la ecuación (1) mediante las propiedades

derivadas source y target. Además, si ese ítem es parte de un ítem más general (asociación RefinedOn en Figura 4), el incremento también dependerá del ítem más general. En la ecuación (3) es definida una regla de derivación en OCL que permite especificar esta relación:

```
context ProductIncrement: targetInf:
Set(Item)
derive: self.target->clousure(parent)
```
(3)
```
context Item: sourceInf:
Set(ProductIncrement)
derive: self->clousure(subitem).source
```

Generalización/refinamiento: este tipo de relación es usado para identificar como elementos complejos de un sistema pueden ser combinados para formar otros elementos, y como un elemento puede ser refinado por otro elemento. En Scrum este tipo de traza puede ser generada por el evento Grooming, en este evento los ProductBacklogItem son refinados en nuevos ProductBacklogItems o Tasks, en este evento también las Tasks pueden ser refinadas en nuevas Tasks (Figura 5). En la ecuación (4) se especifan los ProductBacklogItems refinados por un evento grooming.

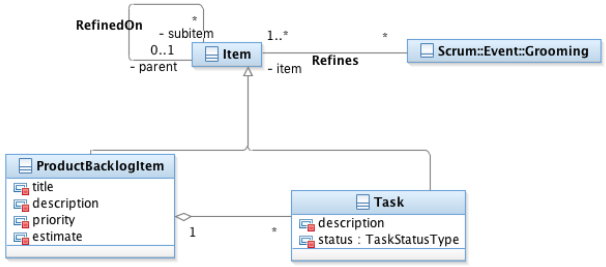


Figura 5. Relación de Generalización/refinamiento.

```
context Item: refinedOnGrooming:Set(Item)
derive:
self.refinedOn->select(i| self.grooming->
  intersection(i.grooming)->notEmpty())
```
(4)
```
context Item: refinedOn: Set(Item)
derive: self->clousure(subitem)
```

Evolución: Un elemento e1 evoluciona a un elemento e2 si e1 fue reemplazado por e2 durante el desarrollo, mantenimiento, o evolución del sistema. En Scrum esta relación de traza surge durante un Sprint en los DailyScrum y DevelopmentWork cuando luego de haber trabajado con un SprintBaclog y haber generado un ProductIncrement, se transforman los ProductBacklogItems y Tasks (Items) para alcanzar los objetivos del proceso de desarrollo. Los Items evolucionan a una nueva versión, asociación Replace en Figura 6. En la ecuación (5) se especifican todos los ítems sucesores y predecesores de un ítem dado.

```
context Item: successor: Set(Item)
derive: self->closure(newVersion)
```
(5)
```
context Item: predecessor: Set(Item)
derive: self->closure(oldVersion)
```

Satisfacción: en este tipo de relaciones un elemento e1 satisface un elemento e2 si e1 reúne las expectativas, necesidades, o deseos de e2; o si e1 cumple con una condición representada por e2. En Scrum este tipo de trazas relacionaría al ProductIncrement y al SprintBacklog en el evento llamado SprintReview donde el ProductOwner comprueba si el ProductIncrement con Status Done satisface el SprintGoal propuesto para ese Sprint (Figura 7).

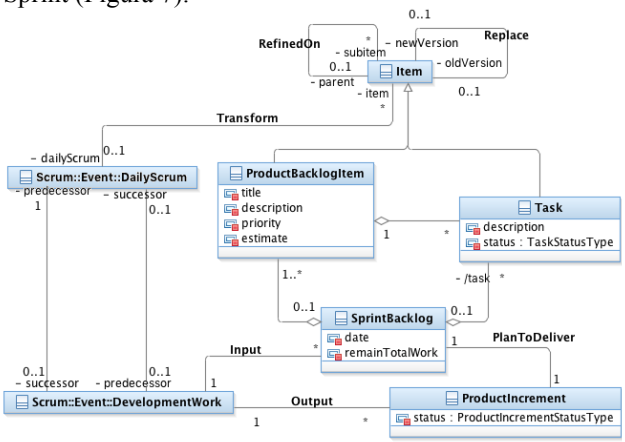


Figura 6. Relación de evolución.

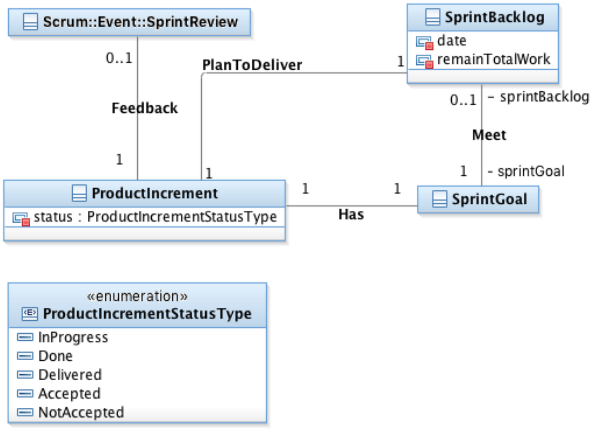


Figura 7. Relación de satisfacción.

Solapamiento: un elemento e1 es solapado con el elemento e2 si e1 y e2 refieren a características comunes del sistema o de su dominio. Las relaciones de solapamiento son utilizadas entre "declaraciones" de requerimientos, casos de uso, y modelos de análisis de

objetos, también estas relaciones pueden trazar al código con requerimientos. En Scrum dos ProductBacklogItems pueden estar solapados, o también solaparse con alguna Task. Cuando un Sprint finaliza, el ProductOwner puede agregar nuevos ProductBacklogItems al ProductBacklog, donde sería necesario mantener las trazas de cada requerimiento. Dichos Items pueden tener características similares con otros existentes en el listado de requerimientos, y cuando se identifica la similitud, es definida entre tales elementos.

4. Caso de Estudio

Con el fin de validar la propuesta, se presenta un caso de estudio basado en el proyecto Moodle [11], el cual adopta prácticas de Scrum para el desarrollo de su plataforma de software. Moodle utiliza el sistema de seguimiento de proyectos Moodle Tracker [14], basado en Atlassian Jira. Esta herramienta de gestión posee una base de datos donde se almacenan todas las historias de usuario, test, errores, tareas y otros artefactos correspondientes al proyecto. Muchos de estos artefactos se generan como Issues en Moodle Tracker, donde cada Issue representa el trabajo a realizar. La Tabla 1 incluye distintos tipos de artefactos generados en un proyecto Moodle (columna derecha) y el correspondiente concepto de Scrum que especializan (columna izquierda).

En la sección previa se definieron distintas relaciones de trazabilidad a partir de los artefactos de Scrum. Estas relaciones pueden recuperarse desde la base de seguimiento de los proyectos, aplicando el modelo definido en la Sección 3. Para probar la aplicabilidad del modelo se recuperó la base del proyecto Moode. La base de datos contiene 51017 issues (Tabla 2), de los cuales 12074 estarían formando parte del product backlog ítem y 6607 de las tareas que lo componen.

Tabla 1. Artefactos de Moodle en Scrum.

Artefacto en Scrum	Especialización en Moodle
ProductBacklog	ListOfIssue
ProductBacklogItem	Issue (type = Epic, New Feature, Improvement)
Task	Issue (type = Task, Sub-task)
SprintBacklog	IssuesOnSprint
SprintGoal	Roadmap
ProductIncrement	ReleaseCandidate
ImpedimentBacklog	Issue (type = Bug)
Product	Release

Tabla 2. Cantidad de Issues según el Tipo.

Issue Type	# cantidad
Epic	126
New Feature	3268
Improvement	8680
Task y Sub-task	6607
Bug	32336
Issues	51017

En la Sección 3 se definió la relación de dependencia entre ProductIncrement e Item (ProductBacklogItem o Task en Figura 4). A partir de la especialización de los artefactos en el caso de estudio, este tipo de relación se daría entre ReleaseCandidate e Issue (Tabla 1). Por ejemplo, en la Figura 8 se incluye la descripción de un product backlog ítem, un Issue de tipo Epic, el issue MDL-25544. Según la información disponible en la herramienta, el issue MDL-25544 afecta el Release Candidate 2.0 (un ProductIncrement). Este issue contiene 13 issues (Figura 9), de los cuales dependen las versiones: 2.1.8, 2.2.5, 2.3.2, 2.7.5, 2.8.3, 2.92.9, 2.8, 2.4.1, y 2.7. En consecuencia, aplicando la ecuación (3) a MDL-25544, resultarían que sourceInf incluye a todas estas versiones.

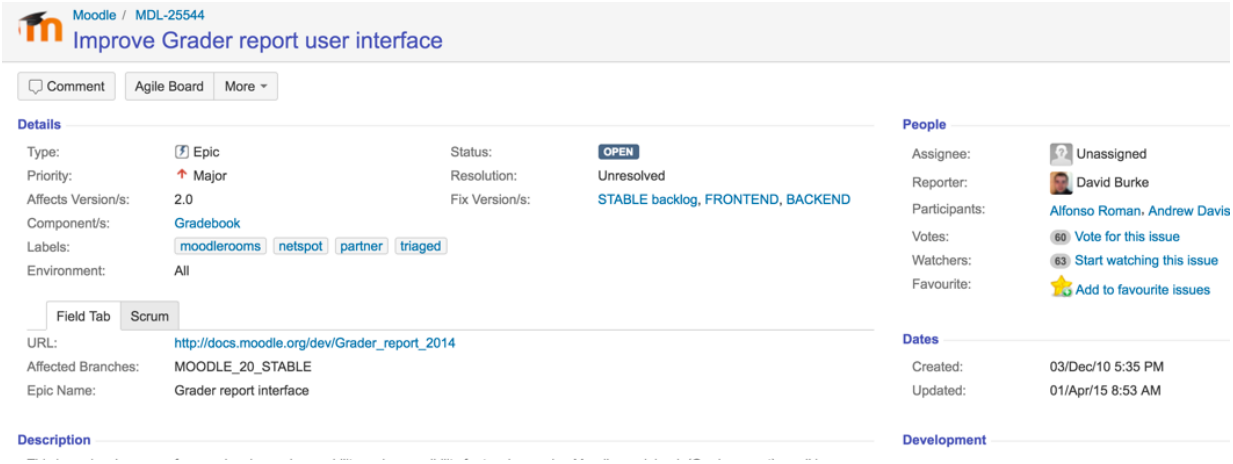
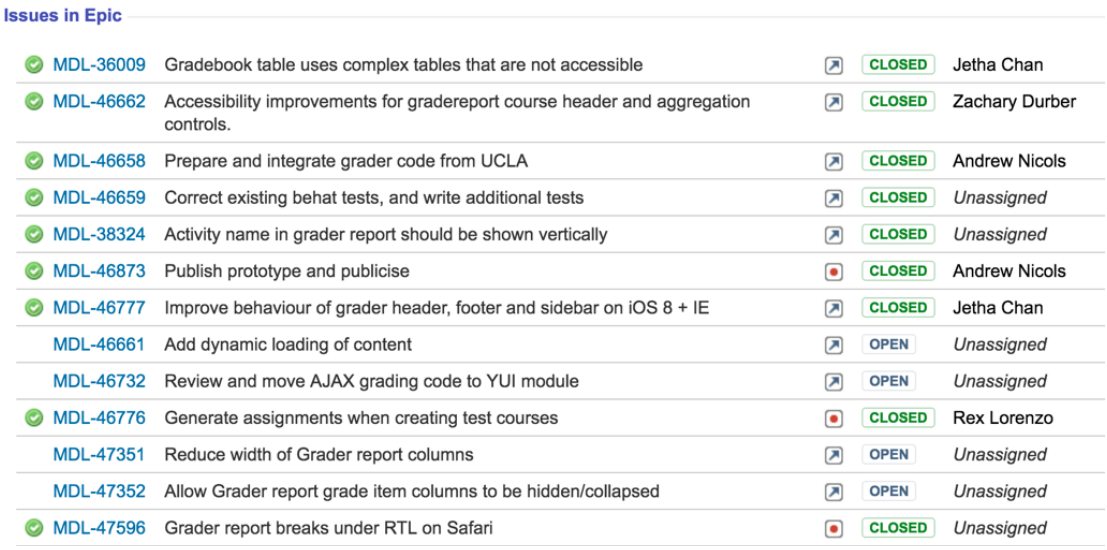


Figura 8. Un Issue de tipo Epic (Product backlog ítem).



Issues in Epic

MDL-36009	Gradebook table uses complex tables that are not accessible	CLOSED	Jetha Chan
MDL-46662	Accessibility improvements for gradereport course header and aggregation controls.	CLOSED	Zachary Durber
MDL-46658	Prepare and integrate grader code from UCLA	CLOSED	Andrew Nicols
MDL-46659	Correct existing behat tests, and write additional tests	CLOSED	*Unassigned*
MDL-38324	Activity name in grader report should be shown vertically	CLOSED	*Unassigned*
MDL-46873	Publish prototype and publicise	CLOSED	Andrew Nicols
MDL-46777	Improve behaviour of grader header, footer and sidebar on iOS 8 + IE	CLOSED	Jetha Chan
MDL-46661	Add dynamic loading of content	OPEN	*Unassigned*
MDL-46732	Review and move AJAX grading code to YUI module	OPEN	*Unassigned*
MDL-46776	Generate assignments when creating test courses	CLOSED	Rex Lorenzo
MDL-47351	Reduce width of Grader report columns	OPEN	*Unassigned*
MDL-47352	Allow Grader report grade item columns to be hidden/collapsed	OPEN	*Unassigned*
MDL-47596	Grader report breaks under RTL on Safari	CLOSED	*Unassigned*

Figura 9. Issues que componen al epic MDL- 25544 (Refinamiento de product backlog ítem).

Otro tipo de relación de trazabilidad definido es Evolución (ver Figura 6). Se obtuvieron del product backlog los issues de tipo New Feature, debido a que estos son catalogados como requerimientos del sistema. Luego, se filtraron aquellos issues creados antes del 01/01/2005 y cuyo estado es "Reopend". El resultado de dicha consulta fue de dos (2) Isssues, el MDL-1071 y el MDL-1967. El ítem MDL-1071 (Figura 10) a lo largo de su vida tuvo más de 70 cambios, tales cambios y evoluciones lo afectaron de diferentes maneras. En la primer ocurrencia de MDL-1071 de la Figura 10 se incluye el issue original. En la última fila se describe su última versión. Se puede apreciar la evolución de distintas propiedades, como Summary, Assignee, y Affects Versions.

5. Conclusiones

En este trabajo se presentó un modelo para la automatización de las relaciones trazabilidad de Scrum. El modelo propuesto es definido a partir de los conceptos clásicos de trazabilidad y es integrado a los constructores esenciales de Scrum. A partir de los constructores de Scrum, principalmente artefactos y eventos, se analizan distintos tipos de relaciones de trazabilidad, tales como: Dependencia, Generalización, Evolución, Satisfacción, y Solapamiento. Luego mediante el análisis de la información disponible de proyectos de desarrollo ágiles se validó la propuesta. En particular se analizó el proyecto de desarrollo de Moodle. Se puede decir que en en este proceso en particular, y en los distintos procesos capturados con herramientas como Jira, las trazas son creadas entre los diferentes artefactos. Sin embargo, la existencia de tales enlaces no implica que exista trazabilidad, debido a que el proceso de trazabilidad (uso, planeación de la estrategia y mantenimiento) no es tenido en cuenta en tales herramientas. Es imprescindible para proyectos ágiles de escala industrial definir la recuperación de los enlaces y la planificación de la trazabilidad para que su uso sea realmente efectivo, y no un sobretrabajo.

Project	Key	Summary	Issue Type	Status	Assignee	Affects Version/s
Moodle	MDL-1071	Add a forum option to allow anonymous posting	New Feature	In Progress	Martin Dougiama	2.0
Moodle	MDL-1071	Implement user disguises	New Feature	Open	Andrew Nicols	1.7, 1.8, 1.9, 2.0, 2.5, 2.8.1, 2.9.4, 3.0.2
Moodle	MDL-1071	Implement user disguises	New Feature	Reopened	Andrew Nicols [dobedobedoh]	1.7, 1.8, 1.9, 2.0, 2.5, 2.8.1, 2.9.4, 3.0.2

Figura 10. Vista parcial de la evolución del Issue MDL-1071.

En herramientas de seguimiento de proyectos, como el Tracker de Moodle y Jira, existe información que sólo se puede recuperar mediante la formulación de consultas explícitas, lo que dificulta la recuperación automática de las trazas. Por ello, se define como trabajo futuro la implementación de la recuperación de relaciones de trazabilidad en Scrum en herramientas como Jira, Eclipse o Netbeans. Contar con su implementación en entornos de desarrollo es imprescindible para facilitar la tarea de los desarrolladores, gestores de proyectos, o clientes. A partir de tal implementación, se buscará luego la implementación de técnicas, como trazabilidad justo a tiempo (Just In Time), que faciliten la recuperación, representación y evaluación de los enlaces en forma automática y que cada Stakeholder haga uso de dichas técnicas.

Asimismo, a partir del modelo propuesto, se prevee el desarrollo de nuevas herramientas simples, tales como wikis y herramientas de rastreo de cambios, que permitan utilizar las trazas en el desarrollo de un proceso Scrum sin perder agilidad. El uso de estas herramientas no pretenderían sobreponerse a los principios ágiles porque buscarían facilitar la tarea del desarrollo y de la satisfacción del cliente.

6. Agradecimientos

Este trabajo ha sido financiado en forma conjunta por CONICET y la Universidad Tecnológica Nacional. Se agradece el apoyo brindado por estas instituciones.

7. Referencias

[1] J. Cleland-Huang, “Traceability in Agile Projects,” in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer London, 2012, pp. 265–275.

[2] “Manifesto for Agile Software Development,” 2001.

[3] Scrum Alliance, “Core Scrum.” 13-Dec-2012.

[4] L. Cao and B. Ramesh, “Agile Requirements Engineering Practices: An Empirical Study,” *IEEE Softw.*, vol. 25, no. 1, pp. 60–67, Jan. 2008.

[5] A. Espinoza and J. Garbajosa, “A study to support agile methods more effectively through traceability,” *Innov. Syst. Softw. Eng.*, vol. 7, no. 1, pp. 53–69, Mar. 2011.

[6] “Systems and software engineering – Vocabulary,” *ISOIECIEEE 247652010E*, pp. 1–418, 2010.

[7] L. Williams, “What Agile Teams Think of Agile Principles,” *Commun ACM*, vol. 55, no. 4, pp. 71–76, Apr. 2012.

[8] “Estimating the Implementation Risk of Requirements in Agile Software Development Projects with Traceability Metrics - Springer.” [Online]. Available: http://link.springer.com/chapter/10.1007%2F978-3-319-16101-3_6. [Accessed: 23-May-2016].

[9] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. Maletic, and P. Mäder, “Traceability Fundamentals,” in *Software and Systems Traceability*, Springer London, 2012, pp. 3–22.

[10] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni, “Model Traceability,” *IBM Syst J*, vol. 45, no. 3, pp. 515–526, Jul. 2006.

[11] “The Moodle Project,” 2014. [Online]. Available: www.moodle.org.

[12] P. Arkley, P. Mason, and S. Riddle, “Position paper: Enabling traceability,” in *Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh, Scotland*, 2002, pp. 61–65.

[13] Y. Zhang, R. Witte, J. Rilling, and V. Haarslev, “Ontological approach for the semantic recovery of traceability links between software artefacts,” *IET Softw.*, vol. 2, no. 3, pp. 185–203, Jun. 2008.

[14] J. Cleland-Huang, R. Settimi, E. Romanova, B. Berenbach, and S. Clark, “Best Practices for Automated Traceability,” *Computer*, vol. 40, no. 6, pp. 27–35, Jun. 2007.

[15] A. Mahmoud, “Toward an effective automated tracing process,” in *2012 IEEE 20th International Conference on Program Comprehension (ICPC)*, 2012, pp. 269–272.

[16] B. Ramesh and M. Jarke, “Toward reference models for requirements traceability,” *IEEE Trans. Softw. Eng.*, vol. 27, no. 1, pp. 58–93, Jan. 2001.

[17] “The Moodle Tracker,” 2014. [Online]. Available: https://tracker.moodle.org/secure/Dashboard.jspa.

[18] R. Nazareno, S. Gonnet, H. Leone, "Trazabilidad de procesos SCRUM", in Simposio Argentino de Ingeniería de Software (ASSE 2015), pp. 284-298, 2015.

[19] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[20] G. Spanoudakis and A. Zisman, “Software traceability: a roadmap,” *Handb. Softw. Eng. Knowl. Eng.*, vol. 3, pp. 395–428, 2005.