Derivación de requisitos y construcción de trazabilidad entre artefactos del proceso de desarrollo

Cecilia Datko[1], Yanela Carllinni[2]
Analista de Sistemas en el Depto. Sistemas de la Dirección de Informática del Ministerio de Salud de la Provincia de Buenos Aires. [1]
Analista Funcional en Laboratorio de Investigación y Formación en Informática Avanzada (LIFIA). [2]
ceciliadatko@gmail.com, yanela.carllinni@gmail.com

Resumen. La trazabilidad es una actividad importante en el desarrollo de software. Permite analizar el impacto de cambios en requisitos originados por el cliente así como también estimar el esfuerzo al realizar el mantenimiento. También constituye un elemento importante en el proceso de mejora continua de la organización puesto que es la que permite tener una visión integradora de todos los artefactos que forman parte de cada etapa del proyecto [Huang 2012].

El presente trabajo propone un modelo de trazabilidad utilizando una estrategia de derivación de requerimientos a Caso de Uso partiendo de un conjunto de conceptos del dominio e implementa una herramienta para dar soporte automatizado a tal modelo.

La propuesta abarca el ciclo de desarrollo de software a partir del análisis y especificación de requerimientos, cubriendo codificación y testing.

Palabras Clave: Trazabilidad, Léxico Extendido del Lenguaje (LEL), Casos de Uso (UC).

1 Introducción

La trazabilidad es una asociación discernible entre dos o más entidades lógicas, tales como requisitos, elementos del sistema, verificaciones o tareas [CMMI 2010]. En otras palabras, se define como el seguimiento de un requerimiento, desde que nace hasta su mantenimiento, registrando cada estado o elemento que se produce a partir del mismo.

Mantener trazabilidad en un proyecto de desarrollo ofrece utilidades tales como:

- Permite verificar que la funcionalidad esperada ha sido incluida y que no existe funcionalidad innecesaria. Garantiza que todos los requerimientos sean diseñados, y que todos los diseños se codifiquen y se prueben.
- Permite realizar análisis de impacto y mejorar la gestión de cambios. Cuando ocurren cambios en el software, la trazabilidad hace que sea más fácil evaluar el impacto que dichos cambios podrían tener en otras partes del proceso de desarrollo.
- Mejora la comunicación y cooperación. Cada integrante del equipo almacena la información relacionada a su área dentro del proyecto de desarrollo y esta información puede ser utilizada por miembros y artefactos de otras áreas del proyecto, asegurando también la contribución de cada individuo.
- Incrementa la recolección de información acerca del proyecto, lo que permite obtener mediciones para mejorar la calidad del producto y evitar desviaciones en costes y plazos, o al menos detectarlas cuanto antes.

2 Objetivo

Independientemente del modelo de ciclo de vida seleccionado para desarrollar software, todos tienen en común las actividades de Análisis, Codificación y Verificación.

Mantener actualizada la traza representa un reto importante desde el punto de vista de la calidad ya que debe existir alguna manera de hacer un seguimiento de cómo se ha elaborado lo requerido durante todo el proceso de desarrollo.

La trazabilidad entre artefactos de software ayuda a obtener mayor información durante las etapas del ciclo de vida, facilitando la obtención de mediciones para analizar impactos ante posibles cambios y analizar, mejorar y evitar riesgos durante iteraciones actuales y futuras del proyecto. Cuanta más información se almacene respecto al ciclo de vida del proyecto, mayores y mejores métricas pueden obtenerse [Fenton 1991].

El objetivo de este trabajo es brindar un modelo de trazabilidad así como también una herramienta para soportarlo. Con este fin, se realizó la investigación y posterior implementación de:

- Un glosario que permite capturar el lenguaje de la aplicación con el que se escribirán los requerimientos [Antonelli 2003].

- Casos de Uso, que son los artefactos de análisis utilizados en los ciclos de vida iterativos e incrementales.
- Una técnica de derivación del glosario a Casos de Uso.
- La creación y administración de artefactos de verificación (incidencias).
- Trazabilidad entre la etapa de análisis y verificación.
- Trazabilidad entre la etapa de codificación y las etapas de análisis y verificación.

3 Modelo de traza

El presente capítulo describe el proceso para lograr un modelo de trazabilidad entre distintos artefactos de un proyecto de software. Dicha descripción se realiza detallando la relación entre los artefactos generados en cada etapa del desarrollo de un producto de software.

Se presenta un modelo integral que muestra los artefactos y las relaciones necesarias entre estos, siguiendo por el detalle de la relación entre cada uno de ellos: Léxico Extendido del Lenguaje (LEL, en inglés Language Extended Lexicon), Casos de Uso (UC, en inglés Use Case), Incidencias y Código fuente (Ver Fig. 1).

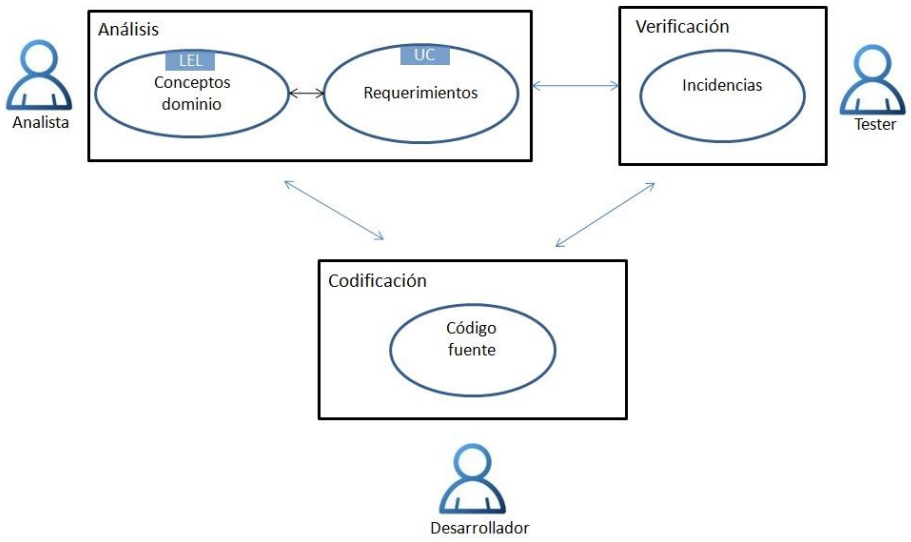

Figura 1. Modelo conceptual de Trazabilidad

El modelo planteado abarca:

- La creación y administración de artefactos de análisis (LEL y UC) a cargo del rol analista del producto.
- La implementación de una estrategia de derivación para transformar automáticamente LEL a UCs.

- La creación y administración de artefactos de verificación (incidencias), facilitada por la herramienta de tracking, a cargo de quien tenga el rol tester del producto.
- La creación de artefactos de desarrollo (código fuente) que es responsabilidad del rol desarrollador dentro del producto.
- La relación bidireccional entre la etapa de análisis y verificación que ocurre cuando se reporta una incidencia sobre un requerimiento determinado y a su vez cuando se indica, en un requerimiento particular, que el mismo presenta una incidencia previamente reportada en la misma herramienta.
- La relación bidireccional entre la etapa de codificación con las etapas de requerimientos y verificación que queda a cargo del desarrollador y de la funcionalidad que pueda llegar a ofrecer la herramienta de tracking seleccionada.

3.1 Esquema de integración

En el presente trabajo se plantea un esquema de integración y trazabilidad con dos objetivos fundamentales:

(i) Interrelacionar las herramientas de uso cotidiano, utilizadas en cada etapa de cualquier proyecto de software, a modo de hacer un seguimiento de los requerimientos desde su nacimiento hasta su mantenimiento en el producto de software.

(ii) La integración de los artefactos mencionados anteriormente: la captura de requerimientos utilizando LEL y Casos de uso, el código fuente y las incidencias en una misma herramienta: la herramienta de tracking.

La integración de los artefactos mencionados en la parte (ii) se logra con las interrelaciones entre las herramientas mencionadas en (i) alcanzando de esta forma, la trazabilidad buscada.

A continuación se detalla cada etapa perteneciente al modelo planteado y las relaciones entre ellas.

- La etapa de análisis comprende los conceptos del dominio capturados mediante LEL y los requerimientos definidos a través de Casos de Uso. Además, esta etapa incluye la integración automática entre ambos artefactos brindada por el enriquecimiento de una estrategia de derivación previamente seleccionada para transformar los conceptos de dominio (LEL) en requerimientos (Casos de Uso), lo que permite conocer también, qué concepto de dominio le da origen a cada Caso de Uso derivado.

- La etapa de codificación queda a cargo de los desarrolladores. Sin embargo, en este modelo se plantea la utilización de un servidor de versionado donde pueda alojarse el código fuente para su posterior integración con el resto de los artefactos.

- La etapa de verificación comprende a las incidencias que puedan surgir durante las pruebas de la codificación. Las mismas se registran en la herramienta de tracking para su correspondiente seguimiento.

3.2 Etapa de Análisis: LEL, UC y Relación LEL - UC

El Léxico Extendido del Lenguaje es un glosario cuyo objetivo es registrar la definición de los términos que se manejan en el dominio del problema.

Los términos, llamados símbolos en esta estrategia, se definen a través de dos atributos: noción e impactos. La noción describe la denotación del símbolo mientras que los impactos, la connotación.

Cada símbolo pertenece a una de las siguientes cuatro categorías: sujeto, objeto, verbo o estado. (Ver Tabla 1).

Nombres	Identifica todos los nombres con los cuales el símbolo es identificado dentro del dominio	**Tipo**	Verbo/Estado/ Objeto/Sujeto
Sinónimos	Identifica con que otros nombres puede identificarse al símbolo		
Noción	Describe qué es el símbolo		
Impacto	Describe qué hace el símbolo / que le hacen		

Tabla 1. Símbolo del LEL

La construcción del LEL comienza por obtener información del dominio. A partir de esta información se elabora una lista de símbolos. Estos símbolos se deben clasificar. Luego de clasificarlos, se los define y como producto de la definición se pueden descubrir sinónimos, por lo cual se deben reorganizar los símbolos. La información debe ser validada por los expertos del dominio y controlada por el ingeniero de requerimientos [Leite1990].

Para obtener automáticamente requerimientos funcionales (UCs) y trazarlos con los conceptos del dominio registrados en el glosario, se seleccionó e implementó la estrategia de derivación que propone [Antonelli 2012], estos requerimientos pueden ser enriquecidos y validados en fases posteriores del proyecto así como también pueden incorporarse nuevos requerimientos, Casos de Uso, que no formen parte de la estrategia de derivación, es decir, que no tienen origen a partir del diccionario LEL.

La esencia de la traza se basa en adquirir conceptos del dominio mediante LEL y obtener Casos de Uso que deriven de dicho glosario.

Los Casos de Uso representan interacciones dentro de la aplicación, mientas que los símbolos de tipo verbo representan acciones dentro del alcance de la misma. Esto muestra que cada símbolo de tipo verbo se puede derivar en un Caso de Uso.

La identificación del UC se establece mediante el nombre del verbo. El objetivo en cada símbolo de de tipo verbo se representa con la noción, la cual se utilizará para

describir el objetivo en el contexto del Caso de Uso. El comportamiento (Impactos) en un símbolo de tipo verbo, describe las acciones necesarias para lograr el objetivo, de esta forma se puede derivar este comportamiento para definir el escenario principal del UC.

Es necesario definir un rol que lleve a cabo las acciones del UC. Los símbolos de tipo sujeto están relacionados naturalmente con los verbos, dado que los impactos en el sujeto incluyen las acciones que los verbos realizan. De aquí puede observarse que los actores primarios son los sujetos que realizan acciones indicadas por los verbos.

Los símbolos de tipo estado son los candidatos para ser las pre y postcondiciones del UC. Para ello, es necesario identificar los símbolos de tipo estados del LEL que están relacionados con el verbo que representa al UC y así derivarlos a las pre y postcondiciones del Caso de Uso. Puede ocurrir que LEL no tenga símbolos de tipo estados relacionados con cada verbo, en estas situaciones las pre y postcondiciones quedaran sin definir en el UC.

3.3 Etapa de Codificación: Relación UC – Código Fuente

La relación que se establece entre UCs y código fuente quedará a cargo del desarrollador, quien mediante la plataforma de desarrollo que utilice, implementará cada uno de los Casos de Uso definidos en la etapa de análisis.

La herramienta de tracking, seleccionada para dar soporte al modelo de trazabilidad, ofrece un módulo de conexión con Subversion para el control de versionado. Esta funcionalidad permite registrar, en la herramienta, los comentarios realizados en la operación de commit durante el desarrollo, y que mencionan a una incidencia en particular. Del mismo modo, se realizó una extensión para almacenar los comentarios de los commits que mencionan UCs. Al momento de guardar en el servidor de versionado el código que implementa un Caso de Uso, el desarrollador deberá mencionar su identificador para lograr así la traza entre UC-Código Fuente.

3.4 Etapa de Verificación: Relación Incidencia – UC – Código Fuente

Las incidencias que surjan durante la etapa de verificación (testing) deben ser reportadas en una herramienta de tracking y queda a cargo del tester realizar esta tarea.

Al momento de guardar en el servidor de versionado el código fuente que resuelve una incidencia, el desarrollador deberá indicar el identificador de la incidencia para obtener la traza entre la incidencia y el código fuente.

La posibilidad de conocer las incidencias que se detectaron sobre un requerimiento en particular, así como también qué artefactos de desarrollo fueron involucrados para resolver incidencias reportadas, dependen de la funcionalidad que pueda brindar la herramienta de tracking utilizada para el seguimiento de cada proyecto. En el presente trabajo se realizó la extensión de la herramienta de tracking seleccionada con la posibilidad de mantener la traza entre Incidencias, Caso de Usos y Código fuente.

4 Herramienta

Este capítulo presenta los motivos por los cuales se seleccionó la herramienta de tracking MantisBT para dar soporte al modelo de trazabilidad buscado. Luego se muestra, en detalle, el modelo creado en el presente trabajo, indicando las funcionalidades incorporadas por el plugin Honey en MantisBT, teniendo en cuenta pautas de usabilidad [Nielsen 1994] y accesibilidad [W3 2013] para lograr dicho modelo.

4.1 Descripción general

La herramienta de tracking seleccionada para lograr el modelo de trazabilidad entre las distintas áreas de un proyecto de software fue MantisBT [MantisBT 2012]. Las características que llevaron a elegir esta herramienta por sobre otras son las siguientes:

- Está desarrollada en PHP, pudiéndose utilizar en cualquier plataforma que lo soporte.
- Es una herramienta de alta difusión en la actualidad.
- Se destaca por su facilidad y flexibilidad de instalar y configurar, permitiendo configurar workflows adaptables a cada proyecto.
- Para proyectos que requieren un manejo de casos de prueba más estructurado o automatizado, MantisBT posee la posibilidad de integrarse con una herramienta open source, TestLink, que permite registrar y ejecutar casos de prueba y relacionar el resultado de su ejecución con issues de MantisBT.
- Posee integración con SVN mediante un módulo específico para tal fin, quedando registrado, en la herramienta, el historial de cambios en las incidencias con cada operación de commit [SVN-MantisBT 2012]. Esta característica permitió que se pueda utilizar un repositorio SVN para involucrar a los artefactos de desarrollo y a los desarrolladores en el modelo de trazabilidad buscado.
- Es Open Source, permitiendo la facilidad de extensión. Esta característica permitió la creación del plugin Honey, dentro de MantisBT, con el cual se implementó el modelo propuesto.

4.2 Modelo de trazabilidad detallado

Interrelacionando herramientas de uso cotidiano para el desarrollo de un producto de software, que intervienen en las distintas etapas de dicho producto, se logra hacer un seguimiento de los requerimientos desde su concepción hasta que son implementados en la etapa de codificación y luego verificados en la etapa de prueba.

La elección de las herramientas y artefactos que éstas producen permitió lograr la trazabilidad bidireccional, objetivo del presente trabajo (Ver Apéndice B).

A continuación se presenta el modelo completo de trazabilidad logrado con las funcionalidades incorporadas a MantisBT (Ver Fig. 2)

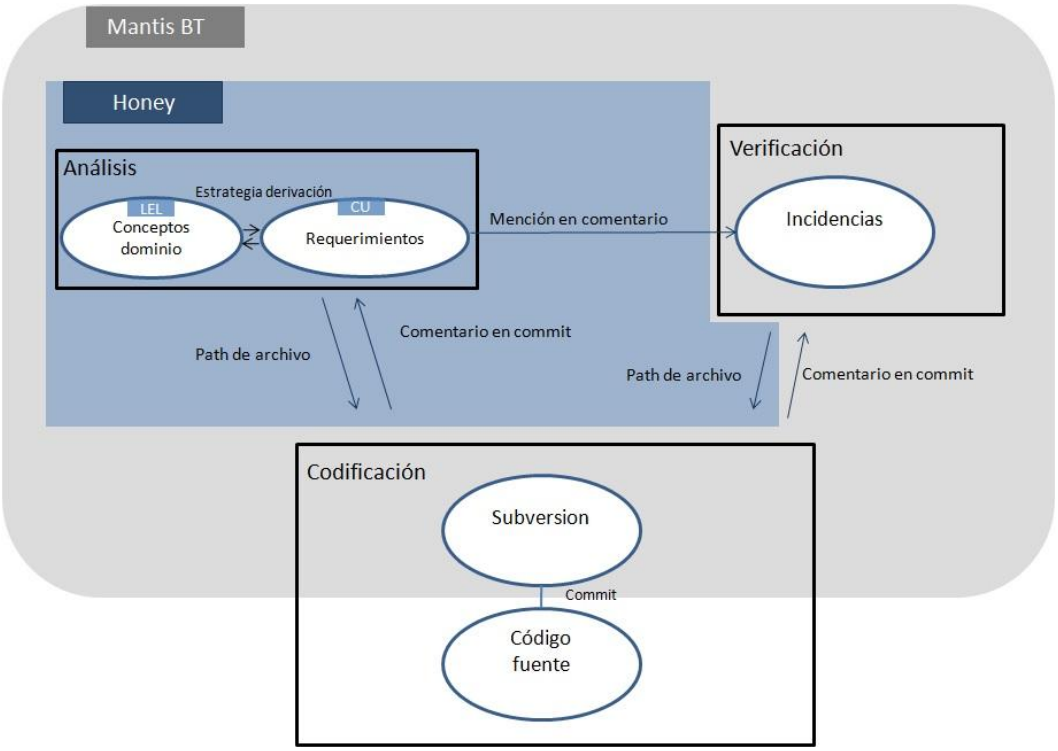

Figura 2. Aporte de Honey al modelo de trazabilidad deseado

El modelo de trazabilidad logrado comprende:

- Creación y gestión de artefactos de análisis (LEL y UCs).
- Relación entre artefactos de análisis. A partir de un concepto del dominio (símbolo LEL) se puede crear un UC, quedando registrada la relación entre el UC y el símbolo que lo creó y viceversa.
- Relación entre artefactos de análisis y artefactos de pruebas. En MantisBT se registran las incidencias encontradas en la etapa de verificación del producto, donde se prueba la funcionalidad implementada. En el presente trabajo se realizó una extensión con la que se pueden crear Casos de Uso manualmente y a través de una estrategia de derivación. Una vez que el rol tester registra una incidencia, la misma puede ser asociada a un Caso de Uso por medio de una nota. De esta manera, el rol de tester puede indicar qué funcionalidad, o mejor dicho, qué Caso de Uso es el que está fallando y se logra la relación UC-INCIDENCIA que permite conocer todas las incidencias generadas para un Caso de Uso particular.
- Relación entre artefactos de pruebas y artefactos de desarrollo. MantisBT posee un módulo de integración con Subversion que registra notas en las incidencias con el comentario realizado en la operación de commit que las mencione. Si bien esta

funcionalidad es útil, no permite hacer un seguimiento de manera tal que pueda conocerse qué artefactos de desarrollo se involucraron en la resolución de cierta incidencia en el producto. Para lograrlo se incorporó a dicha nota, mediante el plugin Honey, la posibilidad de que se registre la ruta de los artefactos de desarrollo utilizados para resolver una incidencia en la herramienta de tracking.

- Relación entre artefactos de análisis y artefactos de desarrollo. Con objetivo de trazar los artefactos de desarrollo con los de análisis, se utilizó como base la extensión desarrollada en el punto anterior para asociar los Casos de Uso con los artefactos de desarrollo que lo implementan. Para establecer la relación es necesario que el comentario del commit mencione al Caso de uso; como resultado quedarán registradas las rutas de los archivos commiteados como nota del UC mencionado.

Etapa de Análisis: Relación LEL - UC.

Basándonos en la estrategia de derivación propuesta por [Antonelli 2012], la trazabilidad entre los artefactos de análisis se realizó de la siguiente forma:

Se incorporó una nueva funcionalidad en MantisBT, a través del plugin Honey, que permite ingresar y administrar los distintos conceptos del dominio, símbolos, capturados durante el análisis de requerimientos, creándose así el glosario LEL sobre la herramienta.

Luego, se implementó y enriqueció un proceso de derivación de LEL a UCs, brindando la posibilidad de incorporar a la herramienta de tracking, de manera automática, todos los UCs y actores que puedan ser derivados del LEL, quedando registrado en cada entidad creada (UC y actor) el símbolo que le dio origen.

Por otro lado, se detectó que los datos proporcionados por la estrategia de derivación de LEL a UC, no son suficientes para especificar completamente un UC. Honey permite enriquecer los UCs derivados con reglas del negocio, interfaces, escenarios alternativos y relaciones con otros UCs. Además, se incorpora la funcionalidad para crear Casos de Uso sin derivar y administrarlos (agregando notas, modificando datos, etc.), brindando también la posibilidad de relacionarlos con aquellos que fueron derivados (relaciones extend e include). Para lograr dicha funcionalidad se agregó al plugin Honey la administración de reglas y actores.

Una vez lograda la derivación de LEL a UCs, puede ocurrir que surjan cambios que impacten directamente en el LEL. Esta situación implicaría modificar el glosario previamente creado y realizar una nueva derivación.

Parte de la labor en el presente trabajo fue evaluar las distintas alternativas e implementarlas evitando dejar información del dominio inconsistente.

Adicionalmente, Honey posee la funcionalidad de almacenar todos los datos generados en cada derivación (UCs, actores y relaciones entre éstos). Esta característica podría permitir, por ejemplo, ver el historial de derivaciones o restaurar a una derivación previa (posible trabajo fututo).

Ejemplo:

1. El analista registra en MantisBT los siguientes símbolos

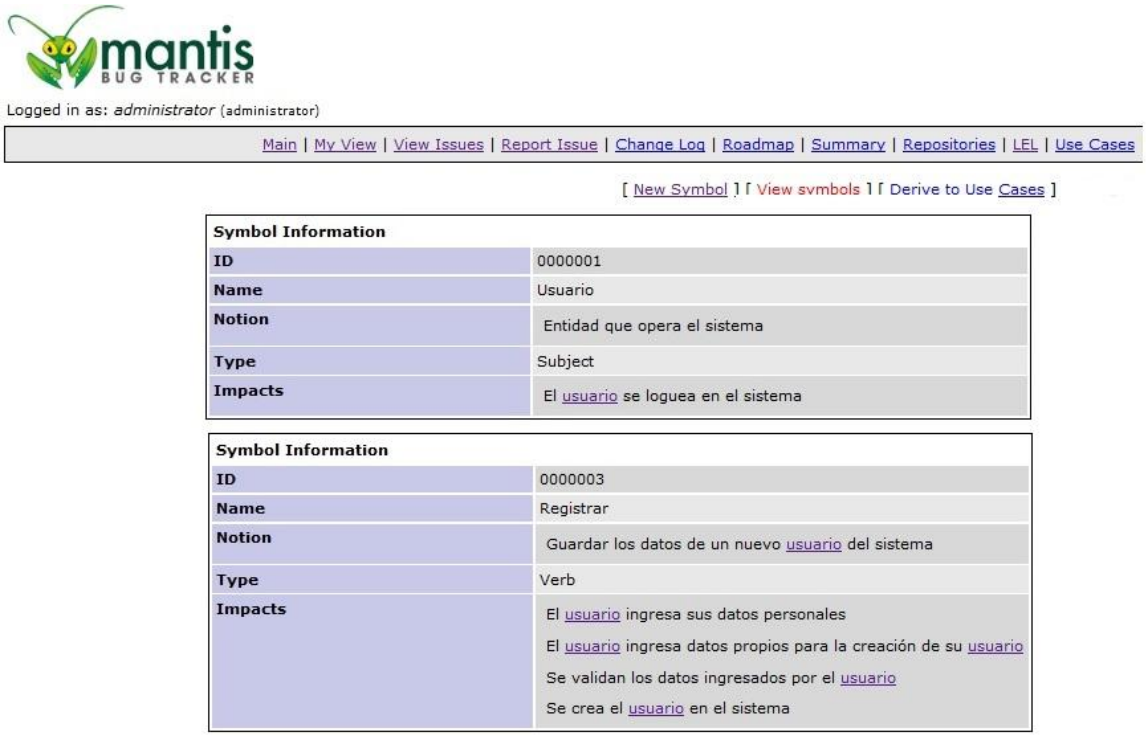

Figura 3. Símbolos creados en glosario LEL

2. El analista realiza la derivación de LEL a Casos de uso mediante la opción de menú “Derive To Use Case”

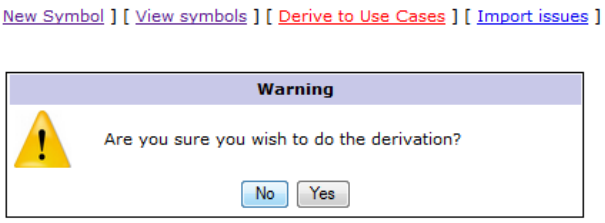

Figura 4. Confirmación para derivar LEL a UCs

3. MantisBT crea el Actor y Caso de Uso como puede observarse en las siguientes imágenes:

Figura 5. Caso de Uso y Actor derivados desde LEL

4. El Caso de Uso puede enriquecerse con actores, reglas, relaciones e interfaces.

Etapa de Implementación: Relación UC - Código.

La relación que se establece entre el código fuente de un producto y los Casos de Uso que especifican la funcionalidad de dicho producto debe realizarse explícitamente por el desarrollador. Sin embargo, en el presente trabajo se analizaron herramientas e incorporaron extensiones que facilitan la traza bidireccional entre la especificación de los requerimientos (UCs) y el código fuente.

Parte de la extensión desarrollada con el plugin Honey ayuda al desarrollador en su tarea de establecer las relaciones entre los artefactos de las áreas de Análisis y Codificación.

Al momento de seleccionar una herramienta de tracking a extender, uno de los puntos requeridos era que la herramienta tenga la posibilidad de conectarse a repositorios de Subversion, de esta manera se conocería la ubicación de los artefactos de desarrollo utilizados durante la codificación del producto. MantisBT cuenta con esta integración. La cual permite registrar el paso a paso de lo que ocurre con incidencias reportadas en la herramienta, sin embargo no brinda la posibilidad de relacionar a artefactos de análisis (Casos de Uso) con los elementos de desarrollo (código fuente), es decir:

- ¿Qué artefactos de desarrollo se modificaron o agregaron para desarrollar un nuevo UC?
- Si se desea modificar la funcionalidad de un UC ya desarrollado ¿puedo saber qué artefactos de desarrollo involucran ese UC?

Las respuestas a estas preguntas se obtienen manteniendo la trazabilidad entre artefactos de desarrollo y análisis. Para ello se incorporaron en el plugin Honey las siguientes funcionalidades, logrando trazar las etapas de Análisis y Codificación:

- Posibilidad de identificar Casos de Uso en el commit de un artefacto de desarrollo: Consiste en incorporar en el comentario del commit la identificación de uno o más Casos de Uso registrados en MantisBT (creados manualmente o bien derivados del LEL, utilizando el plugin Honey).
- Registro de Path de archivos en las notas de los artefactos registrados en MantisBT: Honey guarda todos los paths de archivos commiteados que referencian a Casos de Uso creados en la herramienta de tracking. Las rutas de los archivos relacionados con un UC también se mantiene visible al usuario de la herramienta.

Ejemplo:

1. El desarrollador implementa el UC 0000001 en una plataforma de desarrollo.

2. El desarrollador selecciona un artefacto o conjunto de artefactos en su plataforma de desarrollo: alta_user.java y validar.js.

3. El desarrollador procede a guardar los cambios realizados, generando una nueva versión del/los artefacto/s mediante la operación de commit.

4. La operación de commit solicita el ingreso de un comentario: “clases que implementan el uc #1”.

5. El sistema automáticamente registra una nota en el UC 0000001 como la siguiente:

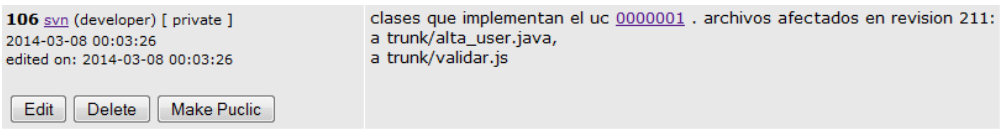

Figura 6. Nota en Caso de Uso que muestra texto y path de archivos commiteados

Etapa de Verificación: Relación Incidencia –UC – Código.

Como se indicó anteriormente, MantisBT permite establecer la relación entre el código fuente y las incidencias reportadas en la herramienta, sin embargo, esto no permite conocer qué artefactos de desarrollo fueron necesarios para resolver determinada incidencia o BUG.

Para poder relacionar los artefactos de desarrollo con las incidencias registradas en MantisBT, se desarrolló en Honey la funcionalidad que permite que los paths de archivos, involucrados en la operación de commit, queden registrados en la herramienta de tracking. Así logra establecerse la traza entre el código fuente del producto y las incidencias reportadas en la herramienta.

Queda como trabajo futuro la implementación de una funcionalidad que permita referenciar un Caso de Uso dentro de una nota de una incidencia reportada en la herramienta de tracking, Esto implicaría la alteración del código fuente de MantisBT.

Ejemplo:

1. El desarrollador selecciona un artefacto o conjunto de artefactos en su plataforma de desarrollo: alta_user.java y validar.js

2. El desarrollador procede a guardar los cambios realizados, generando una nueva versión del/los artefacto/s mediante la operación de commit.

3. La operación de commit solicita el ingreso de un comentario: “clases modificadas para corregir la issue #11”.

4. El sistema automáticamente registra una nota en la issue 0000011 como la siguiente:

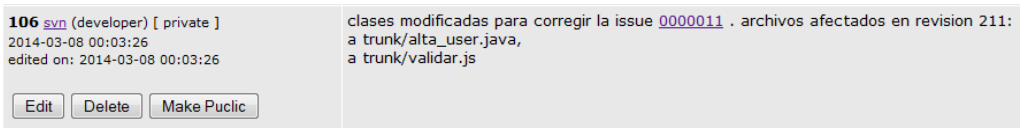

Figura 7. Nota en Incidencia que muestra texto y path de archivos commiteados

Por otro lado, MantisBT brinda la posibilidad de mencionar incidencias en las notas de otra incidencia, generando una relación directa entre estas. Esta característica facilitó el trabajo de realizar la extensión necesaria en Honey, de manera tal que, dentro de las notas de un Caso de Uso creado en MantisBT se pueda hacer referencia a una o más incidencias, generándose la relación INCIDENCIA – UC.

Ejemplo:

1. El tester detecta un bug en la operación correspondiente a generar un usuario en el sistema (UC 0000001) y lo registra en MantisBT (issue 0000011).

2. El tester procede a guardar una nota en el UC 0000001 con el siguiente texto: "se detectó el bug #11 durante el testing".

3. El sistema automáticamente registra una nota en el UC 0000001 como la siguiente:

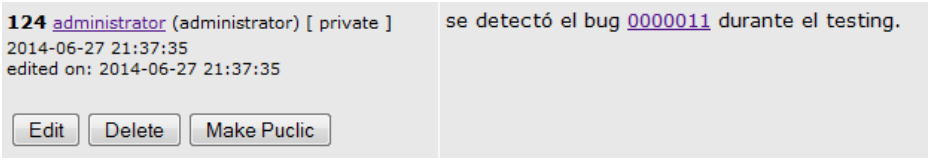

Figura 8. Nota en Caso de Uso que referencia a una Incidencia

4.3 Usabilidad y accesibilidad en Honey

El plugin Honey fue expuesto a un Test de Usabilidad realizado por un experto y se realizaron las mejoras pertinentes en base al resultado de dicho test. Además, se tuvieron en cuenta pautas de accesibilidad (Ver Apéndice C).

5 Conclusión

En la actualidad existen proyectos que logran obtener trazabilidad gracias al inconmensurable esfuerzo de sus integrantes. No se encuentran herramientas gratuitas y Open Source que brinden la integración lograda en el presente trabajo.

La contribución aportada consiste en la definición de un modelo conceptual de trazabilidad que permite integrar las etapas de Análisis, Codificación y Testing presentes en el proceso de desarrollo de cualquier proyecto de Software. Siguiendo por el desarrollo de una herramienta que aplica a dicho modelo y permite abstraerse de la plataforma con la que se codifique el producto.

La información registrada en el modelo planteado contribuye a la incorporación futura de la etapa de Diseño, la cual escapa al alcance del presente trabajo.

El plugin Honey permitió interconectar las herramientas utilizadas en las distintas etapas del proyecto, en un único software: la herramienta de tracking MantisBT.

Para poder implantar este modelo en un proceso de desarrollo debe utilizarse la guía de instalación e integración que ofrece el Apéndice A.

En etapas tempranas del proceso de desarrollo se suelen utilizar herramientas para capturar los conceptos del dominio del problema. La elección, investigación e implementación del glosario LEL fue motivada por la existencia de una estrategia de

derivación para obtener requerimientos a partir de los conceptos del dominio. El desarrollo e incorporación de dicha funcionalidad dentro del modelo incrementó aún más la trazabilidad buscada.

Por último, cabe destacar que la creación del plugin Honey no beneficia sólo a los miembros de un área específica sino que beneficia a todos los integrantes del proyecto, colaborando con:

- Las tareas de los coordinadores de proyecto en la obtención de métricas que permiten hacer un seguimiento del producto y controlar los riesgos.
- El esfuerzo de los analistas y testers para controlar la calidad del producto que llega al usuario final.
- El trabajo cotidiano de los desarrolladores, brindando conocimiento de la funcionalidad afectada por los artefactos de desarrollo, evitando posibles errores.
- Las necesidades empresariales para obtener información en tiempo real con el fin de fidelizar a los clientes.
- El desarrollo tecnológico en plataformas informáticas y la obtención de información en tiempo real.

6 Referencias Bibliográficas

[Antonelli 2003]: Antonelli, L., Rossi G., Oliveros, A.: Traceability en la elicitación y especificación de requerimientos, Tesis presentada a la Facultad de Informática de la Universidad Nacional de La Plata como parte de los requisitos para la obtención del título de Magister en Ingeniería de Software, Bueno Aires, Argentina, Febrero (2003).

[Antonelli 2012]: Antonelli, L., Rossi, G., Leite, J.C.S.P., Oliveros, O.: Deriving requirements specifications from the application domain language captured by Language Extended Lexicon. In proceedings of the Workshop in Requirements Engineering (WER), Buenos Aires, Argentina, Abril (2012).

[CMMI 2010]: CMMI para Desarrollo, Versión 1.3, Guía para la integración de procesos y la mejora de productos. Tercera edición, Junio (2010).

[Fenton 1991]: Fenton, E. Norman. Software Metrics A rigorous approach .Chapman & Hall, Primera Edición (1991).

[Huang 2012]: Huang, J.: Software and Systems Traceability (2012).

[Leite1990]: Leite, J., Franco, A.. "O Uso de Hipertexto na Elicitaçao de Linguagens da Aplicaçao" Anais de IV Simpósio Brasilero de Engenharia de Software, SBC, Mayo (1990).

[MantisBT 2012] Dynamic Plugin Requirements, http://www.mantisbt.org, accedido en octubre de 2012.

[Nielsen 1994]: Nielsen, J.: Heuristic evaluation. Usability Inspection Methods (1994).

[SVN-MantisBT 2012] Subversion (SVN) Repository Integration with Mantis Bug Tracker, http://www.unitz.com/u-notez/2009/10/subversion-svn-integration-mantisbt, accedido en diciembre de 2012.

[W3 2013]: http://www.w3.org/, accedido en septiembre de 2013.