Mejorando la gestión de historias de usuario en eXtreme Programming*

Emilio A. Sánchez Patricio Letelier José H. Canós

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 Valencia - España
{emsanchez|letelier|jhcanos}@dsic.upv.es

Resumen Extreme Programming (XP) es una de las llamadas metodologías ágiles que más interés ha generado en el ámbito industrial y académico. Sin embargo, llevada a la práctica surgen inconvenientes no abordados explícitamente por la documentación disponible. En este trabajo se estudia la problemática de los requisitos, la cual en el contexto minimalista de XP se reduce a la especificación y seguimiento de las historias de usuario (User Stories). El planteamiento de XP al respecto es muy sencillo, pero debido a la gran cantidad de historias de usuario que puede tener el proyecto y a la volatilidad de los requisitos, su gestión puede llegar a ser complicada. En este trabajo damos algunas pautas para abordar las alternativas de evolución que puede sufrir la historia de usuario y presentamos el prototipo de una herramienta para gestionar historias de usuario que apoya dichas pautas.

1. Introducción

En la actualidad las metodologías ágiles están acaparando gran atención en al ámbito industrial y académico de ingeniería del software. Mucho se ha debatido respecto a este nuevo estilo de desarrollar software, caracterizado por su minimalismo en cuanto a artefactos generados, roles y actividades. Esto ha generado un cruce de críticas entre las comunidades asociadas, metodologías ágiles [1] frente a metodologías tradicionales (o peyorativamente llamadas metodologías "peso pesado", siendo las ágiles sinónimo de "peso ligero" [16]). Probablemente los exponentes más populares en ambas categorías son RUP (Rational Unified Process) [13] y XP (eXtreme Programming) [2], como metodologías tradicionales y ágiles respectivamente. Existen comparaciones entre ambas metodologías [15][11][12] e incluso propuestas de adaptación de RUP con prácticas XP[14].

XP es la metodología ágil de la cual se dispone mayor información, tanto en libros como en Internet, sin embargo, aún no se cuenta con datos rigurosos referentes a los resultados en su aplicación. Esfuerzos como el realizado por la red europea NAME [8] van dirigidos precisamente en esta dirección.

* Este trabajo ha sido financiado por el proyecto DOLMEN-SIGLO de la Comisión Interministerial de Ciencia y Tecnología, TIC2000-1673-C06-01.

Basándonos en nuestra experiencia en la aplicación académica de XP [5] en los dos últimos años y en los resultados obtenidos [6][7], hemos detectado algunos inconvenientes en la especificación y seguimiento de requisitos. En XP la gestión de requisitos es "extremadamente" simple, el cliente escribe y prioriza las historias de usuario que expresan requisitos funcionales y no funcionales del sistema. Los programadores estiman el esfuerzo asociado (no más de la duración de una iteración) y las dependencias entre ellas. Para planificar el trabajo desde el punto de vista técnico las historias de usuario son divididas en tareas para las cuales también se realiza una estimación (el esfuerzo asociado a estas tareas no debe superar los 3 días de programación). Teniendo en cuenta el esfuerzo asociado a las historias de usuario y las prioridades del cliente se define una *release* que sea de valor para el cliente y que tenga una duración de unos (pocos) meses. La *release* es dividida en iteraciones de no más de 3 semanas, asignando a cada iteración un conjunto de historias de usuario que serán implementadas. Esta sencilla práctica es denominada en XP "Juego de la Planificación". Durante el desarrollo de la iteración y en particular al final de cada iteración se realiza un seguimiento del plan de la iteración y de la release. Es de esperar situaciones relativas a historias de usuario tales como: aparición de nuevas, postergación en cuanto a la iteración en la cual se implementa, no finalización en la iteración planificada y modificación durante la iteración o una vez completada en una iteración previa.

XP propone fichas en papel para registrar las historias de usuario y tareas asociadas, sin dar mayores guías respecto a la información que debe ser recolectada ni cómo debe gestionarse dicha información. Por otra parte, además de las restricción máxima de esfuerzo asociado (3 semanas de programación) no hay más pautas respecto a la granularidad de una historia de usuario, con lo cual, tratándose de requisitos funcionales y no funcionales, el número puede ser considerable incluso para sistemas pequeños. Aunque existen algunos trabajos que han mostrado su interés por registrar en soporte informático las historias de usuario [3][10], en ellos no se ha dado solución al tratamiento de las posibles situaciones antes mencionadas.

El objetivo del presente trabajo es presentar un método para especificar y gestionar historias de usuario utilizando una herramienta *ad hoc* desarrollada por los autores y denominada Volt.

El resto del artículo está organizado como se indica a continuación. En la sección 2 se describe el artefacto historia de usuario de XP y se propone una plantilla esencial representada en XML. La sección 3 muestra la funcionalidad de un sistema de control de versiones y en particular se analiza la herramienta Subversion [17] y cómo ésta puede servir para el versionado de historias de usuario. La sección 4 presenta nuestra propuesta de método para gestión de historias de usuario y la herramienta que estamos desarrollando, la cual está basada en Subversion. Finalmente en la sección 5 se elaboran las conclusiones y el trabajo futuro.

2. Historias de usuario

El primer problema que encontramos al intentar trasladar las historias de usuario desde las fichas en papel a un sistema informático radica en la necesidad de definir qué información deben contener las mismas y en qué forma se va a representar dicha información.

Respecto al primer punto apenas existen indicaciones que orienten a la hora de crear las historias de usuario. Beck [2] afirma que la única información que se debe recoger es el nombre de la historia y una descripción de la misma. Aun así en el mismo libro se incluyen plantillas para las historias de usuarios cuyo uso se ha desaconsejado [9]. Otros, por su parte, mantienen que se debería registrar la cantidad mínima de información que sea útil para el proyecto de desarrollo [18]. Creemos que la sola descripción de la historia de usuario no es suficiente, pero manteniendo la simplicidad de XP cualquier otra información asociada debe estar justificada con respecto al beneficio que aporte. Es más la información de una historia de usuario podría variar y ajustarse a las características específicas del proyecto. Por otro lado, pudimos comprobar en un experimento anterior [7], que el usuario no se siente cómodo ante el trabajo adicional que supone decidir cuáles son los datos relevantes para conformar sus historias de usuario, prefiriendo que se le ofrezca una plantilla a rellenar. Con el fin de no hacer más complicado el proceso de desarrollo al usuario, la herramienta que presentaremos en la sección 4 provee una plantilla configurable para las historias de usuario que incluye al menos la información que consideramos esencial en cualquier proyecto de desarrollo. Esta plantilla es el resultado de estudiar las historias de usuario dadas como ejemplo en libros [2][18], en información en Internet y nuestra propia experiencia de aplicación de XP [6][7].

Una vez conocida la información que compondrá las historias de usuario debemos elegir el medio más adecuado para representar estos datos. Para tomar esta decisión hemos de tener en cuenta el tratamiento que la herramienta va a hacer de esta información. Dado que se va a realizar control de versiones sobre las historias de usuario es recomendable utilizar un formato en el que la información de las historias se almacene de forma textual. Esta representación permitirá al sistema de control de versiones mostrar las diferencias entre una versión y otra de forma inteligible. Partiendo de este requisito decidimos representar nuestras historias de usuario utilizando XML, de forma similar a la representación de escenarios utilizada por Breitman [3], ya que cumple la propiedad deseada además de ofrecer cualidades interesantes. La utilización de XML como lenguaje de representación nos permite, además de facilitar el control de versiones, disponer de un formato sencillo de tratar por herramientas informáticas.

2.1. Representación en XML

A continuación se presenta el DTD a partir del cual se pueden instanciar documentos XML para cada una de las historias y que constituye nuestra plantilla esencial.

```
<?xml version="1.0"?>
<!ELEMENT user_story (story_id,name,creation_date,customer,
                      priority,dependence*,estimation, risk,
                      release,iteration,upgrade*,base?,
      description)>
<!ELEMENT story_id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT creation_date (#PCDATA)>
<!ELEMENT customer (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
<!ELEMENT dependence (#PCDATA)>
<!ELEMENT estimation (#PCDATA)>
<!ELEMENT risk (#PCDATA)>
<!ELEMENT release (#PCDATA)>
<!ELEMENT iteration (#PCDATA)>
<!ELEMENT upgrade (#PCDATA)>
<!ELEMENT base (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

Este DTD pone de relieve cuáles has sido los datos que hemos considerado relevantes en nuestras historias de usuario.

- **story_id**, identificador unívoco para la historia.
- **name**, nombre de la historia de usuario.
- **date**, fecha en la que la historia de usuario fue creada.
- **customer**, cliente que ha confeccionado la historia de usuario.
- **priority**, prioridad asignada por el cliente a la historia de usuario.
- **dependence**, dependencia con otras historias de usuario establecida por el personal de desarrollo.
- **estimation**, estimación realizada por el grupo de desarrollo del esfuerzo necesario para implementar la historia de usuario.
- **risk**, riesgo asociado a la implementación de la historia de usuario de acuerdo a la experiencia del equipo de desarrollo.
- **release**, release que incorporará la funcionalidad especificada por la historia de usuario.
- **iteration**, iteración en la que el cliente desea que se implemente la historia de usuario.
- **upgrade**, identificador de la historia de usuario que actúa como mejora o corrección de la actual.
- **base**, identificador de la historia de usuario modificada o ampliada por la actual.
- **description**, texto explicativo de la historia de usuario.

Aunque algunos campos pueden parecer complejos a la vista del cliente, sobre todo los campos *upgrade* y *base*, cabe reseñar que el cliente sólo debe rellenar los campos *name*, *priority*, *release*, *iteration* y *description*. El equipo de desarrollo se encargara de los campos *dependence*, *risk* y *estimation*. El resto de campos

se rellenaran y tratarán de forma automática por la herramienta. De esta forma la utilización de la herramienta pretende ser fiel a la filosofía XP procurando no complicar inútilmente el proceso de desarrollo, centrándose en ofrecer un entorno sencillo, claro y conciso para cada tipo de usuario.

3. Control de versiones

Los sistemas de control de versiones se han utilizado a lo largo del tiempo en proyectos de desarrollo de software. Cierto es que, normalmente, tan solo se realiza control de versiones sobre código fuente, siendo este el ámbito en el que los sistemas de control de versiones han demostrado su eficacia y enorme utilidad. Estos sistemas permiten almacenar la historia de los ficheros con el fin de poder recuperar antiguas versiones de los mismos. Además, cada vez que se registra un cambio se almacena un comentario explicativo sobre la razón de ser del mismo junto con el usuario que lo produce.

Para realizar su cometido los sistemas de control de versiones almacenan los ficheros en un repositorio. Este repositorio se puede considerar como un servidor de ficheros que "recuerda" cada cambio hecho a los archivos.

La utilización del repositorio permite que varias personas trabajen simultáneamente sobre los mismos ficheros sin que los cambios de un desarrollador sobreescriban los realizados por otro. Esto se consigue aislando a los usuarios unos de otros. Cada usuario del sistema de control de versiones trabaja sobre una copia local del repositorio a la que tan solo él tiene acceso.

Un escenario clásico en la utilización de estos sistemas es el siguiente. Los usuarios A y B obtienen una copia local actualizada del repositorio. Cada uno trabaja sobre su copia local independientemente del otro. Cuando el usuario A finaliza su trabajo registra los cambios en el repositorio. Dado que nadie más había registrado cambios desde que A obtuvo su copia local el sistema integra los cambios de A en el repositorio. Cuando B finaliza su trabajo también pide actualizar el repositorio. Si los cambios de B se han producido en ficheros diferentes a los realizados por A el sistema integrará los cambios automáticamente. Por otra parte, si se han producido cambios en ficheros modificados por A el sistema informa de un posible conflicto mostrando las diferencias entre los archivos. En este momento es responsabilidad del usuario el descartar alguno de los cambios o mezclar las dos versiones del fichero si los cambios se produjeron en líneas distintas y no producen conflicto entre ellos.

3.1. Subversion

Subversion [17] es el sistema de control de versiones libre que hemos elegido como base para nuestra herramienta. Subversion nació como sustituto funcional de CVS [4]. Su principal intención es la de corregir todos los defectos que CVS arrastra desde sus comienzos. Al mismo tiempo pretende ser lo más parecido posible a CVS para permitir que los usuarios de este sistema puedan comenzar a utilizar Subversion sin necesidad de un esfuerzo adicional.

Muchas son las ventajas que ofrece Subversion sobre CVS. Entre ellas destacan: versionado de directorios, versionado de metadatos, ramificado eficiente o actualizaciones atómicas. Dos han sido las características que nos llevado a adoptar este sistema de control de versiones en nuestra herramienta, ellas se describen a continuación:

Hooks Los "hooks" son programas que pueden ser ejecutados antes o después de realizar ciertas acciones con el repositorio. En nuestro caso esta funcionalidad nos permite validar cada historia de usuario con su DTD antes de registrar cualquier cambio en el repositorio.

Interoperabilidad Entendiendo como interoperabilidad la facilidad con la que Subversion es accesible desde otras herramientas. En este caso Subversion se ofrece como un conjunto de librerías en C con una API bien definida. Esto facilita su utilización desde otras aplicaciones y lenguajes.

3.2. Versionando historias de usuario

Al poner a prueba el sistema de versionado utilizando las historias de usuario extraídas de nuestros experimentos con alumnos, pudimos comprobar que todavía era necesaria flexibilidad en cuanto a su composición. El control de versiones mostró el comportamiento esperado almacenando el historial de los artefactos y mostrando de forma inteligible los cambios que se producían de una versión a otra mediante formato XML. Pero el principal problema apareció al intentar utilizar el sistema para registrar cambios en los requisitos.

Cuando el usuario se enfrenta a un nuevo requisito debe registrarlo como una nueva historia de usuario, pero cuando se trata de modificar un requisito existente apenas se encuentra información en la bibliografía sobre cómo abordar el problema. Con un sistema de control de versiones lo más sencillo parece ser modificar directamente la historia de usuario y registrar el cambio en el repositorio. Pero actuar de esta forma tiene una repercusión negativa en la estimación de esa historia de usuario que debe ser realizada de nuevo. Siguiendo la filosofía XP un cambio en los requisitos se podría registrar como una nueva historia de usuario, si se adopta esta opción y queremos mantener la trazabilidad entre los requisitos nos vemos obligados a almacenar referencias entre la antigua historia de usuario y su correspondiente modificación.

Para enfrentar con éxito esta falta de información referente al método de desarrollo creamos una pequeña guía que sirva de orientación al usuario cuando se enfrente con este problema. Tras estudiar el conjunto de historias de usuario decidimos no limitarnos a una sola forma de modificar las historias sino seguir las siguientes pautas:

- Si la modificación que deseamos realizar no implica cambios en el contenido de la historia entonces crearemos una nueva tarea de programación para cubrir esa funcionalidad. De esta forma registramos las mejoras y modificaciones detectadas por el equipo de desarrollo pero que no afectan a los requisitos iniciales registrados en la historia de usuario.

- Si la modificación implica cambios en la información de la historia tomaremos la decisión de acuerdo a la granularidad del cambio.
  - Si la modificación es de una granularidad similar a la propia historia de usuario implica que nos encontramos ante una nueva historia de usuario. En este caso se necesitan mantener referencias entre las dos historias para disponer de trazabilidad.
  - Por otra parte si el cambio no alcanza el nivel de una nueva historia se modificará la historia de usuario afectada y será el control de versiones el encargado de registrar ese cambio.

4. Herramienta Volt

Volt es una herramienta que permite gestionar historias de usuario a través de un navegador web incluyendo control de versiones. Para ello la herramienta codifica cada historia de usuario en XML siguiendo el formato descrito anteriormente y las registra en un repositorio Subversion. Al mismo tiempo la herramienta permite gestión de usuarios ofreciendo una interfaz personalizada a cada tipo o rol de usuario. A continuación se detallan las principales características de Volt.

Figura 1. Aviso del control de versiones

4.1. Control de historias de usuario

Cuando los usuarios graban sus cambios estos se registran en el control de versiones. Si un usuario consulta una historia de usuario que ha sido modificada

en el control de versiones el sistema le informa de los cambios. En la figura 1 podemos ver como el usuario "Juan L" perteneciente al rol "Desarrollo" consulta la historia número cuatro y el sistema la informa de un cambio realizado desde su última visita.

A su vez cada usuario puede además consultar el historial de cambios sufrido por los artefactos desde su creación hasta la versión actual. Como muestra la figura 2 el sistema almacena y muestra a petición del usuario todos los estados por los que ha pasado una historia de usuario desde su creación.

Consideramos que es un aspecto muy importante que este tipo de herramientas orienten al usuario respecto del proceso de desarrollo a seguir. Por esta razón la herramienta consta de una sección en la cual se explica a través de una serie de hipertextos los pasos a seguir durante la creación, modificación y mantenimiento de sus historias de usuario.

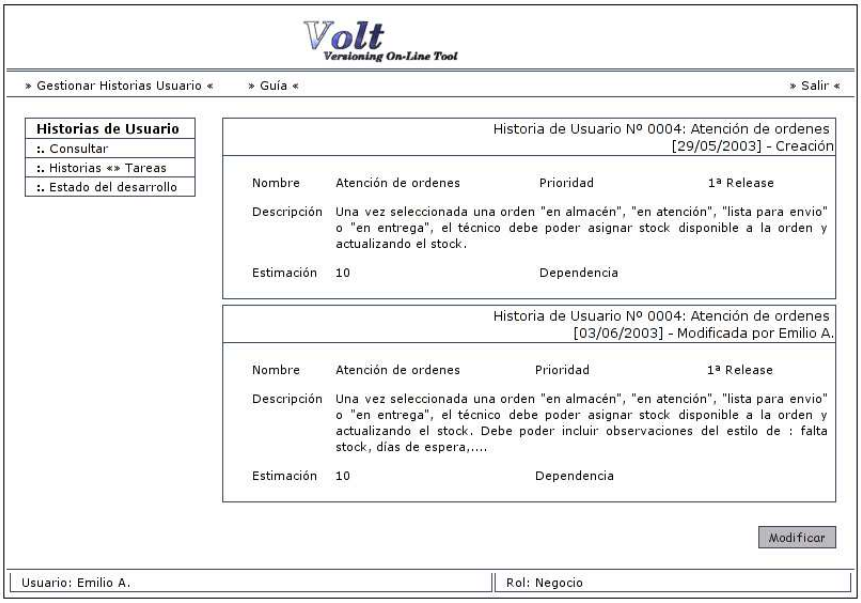

Figura 2. Historial de una historia de usuario

5. Conclusiones y Trabajo Futuro

Desde que comenzó el interés por las metodologías ágiles se ha intentado incorporar en ellas prácticas y métodos establecidos en las metodologías tradicionales con la idea que éstos también podrían ser necesarios y efectivos. Este trabajo está en esta línea pues pretende integrar una gestión de requisitos y un control de versiones más elaborado que el planteado por XP. Sin embargo, para

que este tipo de extensiones sean bienvenidas y valoradas en el contexto de las metodologías ágiles deben mantener su espíritu en cuanto a sencillez y pragmatismo. En nuestro caso esto lo hemos conseguido mediante una definición de un método simple y de una herramienta de apoyo amigable e intuitiva que soporta dicho método.

Durante las primeras fases del trabajo en las que todavía no se había dado forma a la herramienta y realizábamos el control de versiones directamente desde línea de comandos ya pudimos observar las bondades de nuestro método. El control de versiones nos permite sin necesidad de esfuerzo extra seguir la evolución de las historias de usuario a lo largo de todo su ciclo de vida. Con el desarrollo de Volt la interacción con el sistema de control de versiones se facilita. Además la incorporación de la guía en la propia herramienta permite que los usuarios puedan aprender el método de desarrollo al mismo tiempo que trabajan.

A pesar de que Volt ofrece valor añadido cuando se utiliza en un proceso de desarrollo siguiendo XP, pensamos que sería interesante extenderla hasta conformar un entorno integrado para el apoyo a XP. En la actualidad tan sólo se cubren los aspectos relacionados con las historias de usuario pero en próximas versiones se ampliará el marco de acción, cubriendo más actividades, tales como: el juego de la planificación, seguimiento del proceso, refactorización, pruebas, etc.

Respecto a la validación del método y la herramienta propuesto, tenemos planificado incorporarlos el próximo año académico en las prácticas de nuestras asignaturas donde hemos estado trabajando con XP y así conseguir una primera evaluación.

Referencias

1. Agile Alliance, página Web: `www.agilealliance.org/`
2. Beck K.. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999.
3. Breitman, K. y Leite, J.. Managing User Stories. International Workshop on Time-Constrained Requirements Engineering 2002
4. CVS, página Web: `www.cvshome.org`
5. Laboratorio de Sistemas de Información, página Web: `www.dsic.upv.es/asignaturas/facultad/lsi/`
6. Letelier P., Canós, J.H. and Sánchez E.A. An Experiment Comparing RUP and XP. Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering, XP2003, Génova, pp. 41-46, LNCS 2675, Springer-Verlag, 2003.
7. Letelier P., Canós, J.H. and Sánchez E.A. Working with Extreme Programming in a Software Development Laboratory. Aceptado en la 15th Conference on Software Engineering and Knowledge Engineering, SEKE 2003, San Francisco, Julio 2003.
8. Red Europea del VI programa marco NAME (Network of Agile Methods Experience), página Web: `name.case.unibz.it`
9. Robinson, W. Story andTask Cards. 09/01/1999 Página Web: `http://www.xprogramming.com/xpmag/story_and_task_cards.htm`
10. Pinna, S., Mauri S., Lorrai. P, Marchesi. M y Serra N.. XPSwiki: An Agile Tool Supporting the Planning Game. 4th International Conference, XP 2003.

11. Pollice G. RUP and XP Part I: Finding Common Ground, página Web: `therationaledge.com/content/mar_01/f_xp_gp.html`
12. Pollice G., RUP and XP Part II: Valuing Differences, página Web: `therationaledge.com/content/apr_01/f_xp2_gp.html`
13. Rational Unified Process, página Web: `www.rational.com/products/rup/index.jsp`
14. Borrador de un capítulo de libro ilustrando la adaptación de RUP usando los principios ágiles, página Web: `objectmentor.com/resources/articles/RUPvsXP.pdf`
15. Smith J. A Comparison of RUP and XP. Rational Software White Paper, página Web: `www.rational.com/media/whitepapers/TP167.pdf`
16. Smith R. Why UML Methodologists Are Throwing Their Weights Around. Artículo on-line en DevX.com, página Web: `archive.devx.com/uml/articles/Smith03/Smith03-1.asp`
17. Subversion, página Web: `subversion.tigris.org`
18. Wake W.. Extreme Programming Explored. Addison-Wesley, 2001.