



Universidad Tecnológica Nacional

Facultad Regional Córdoba

Dirección de Posgrado

**Especialización en Ingeniería en Sistemas de
Información**

**Caracterización de Especificación de
Requerimientos en entornos Ágiles: Historias
de Usuario**

María Paula Izaurralde

D.N.I.:30471005

Email: paulaizaurralde@gmail.com

Córdoba, Febrero de 2013

Índice

Resumen.....	4
Abstract	4
1. Introducción	5
2. Introducción a los Requerimientos de Software	6
a. Requerimientos de Software	6
b. Ingeniería de Requerimientos de Software	7
c. Especificación de requerimientos	10
d. Guía para escribir buenos requerimientos.....	10
e. Atributos y características deseables de la especificación de requerimientos según IEEE	11
3. Introducción a las metodologías ágiles, Scrum e Historias de Usuario	15
a. Metodologías Ágiles	15
b. Introducción a Scrum.....	16
c. Las Historias de Usuario.....	20
d. El modelo INVEST.....	22
e. Las historias de usuario en el proceso de desarrollo de requerimientos	23
f. El desarrollo de los requerimientos en Scrum	24
4. Los requerimientos en las metodologías ágiles: comparación con las formas tradicionales de especificación.....	28
a. Ventajas de las Historias de Usuario por sobre los documentos de requerimientos tradicionales	28
b. Desventajas de usar Historias de Usuario.....	29
c. Las historias de usuario no son casos de uso	30
d. Las historias de usuario no son IEEE 830	31
e. Análisis del uso de las historias de usuario en la actualidad.....	34

f. Los requerimientos tradicionales y las historias de usuario pueden coexistir	35
5. Conclusiones	37
6. Trabajos Futuros	39

Índice de Figuras

Figura 1: Procesos de la Ingeniería de Requerimientos	7
Figura 2: Esquema básico de Scrum (Mountain Goat Software - Imágenes, 2005).....	17
Figura 3: Formato típico de una Historia de Usuario	21
Figura 4: Definición y Gestión de Requerimientos Ágiles (Moccia, Agile Requirements Definition and Management (RDM), 2012).....	25

Índice de Tablas

Tabla 1: Comparación entre la especificación de los requerimientos de software según el estándar IEEE 830 y las Historias de Usuario.....	34
---	----

Resumen

El presente trabajo revisa las prácticas de ingeniería para el desarrollo de requerimientos, con foco en la especificación de los requerimientos de software y las metodologías ágiles de desarrollo de software, en particular Scrum. Se analizan las características que poseen los requerimientos especificados según el estándar IEEE Std. 830-1998 para luego desarrollar una comparación con las historias de usuario. Se efectúa el análisis de la estructura de las historias de usuario, se analizan las ventajas, desventajas y también se describen las características que han sido definidas para ayudar a pensar y escribir historias de usuario. Por último se revisa el caso en donde los documentos tradicionales de especificación de requerimientos y las historias de usuario coexisten, aún cuando se ha adoptado *Scrum* como metodología para el desarrollo de software.

Palabras claves Metodologías Ágiles, Scrum, Historias de Usuario, INVEST, Ingeniería de Requerimientos, Especificación de Requerimientos, IEEE Std. 830-1998.

Abstract

A review of the best practices for requirements development with special focus on software requirements specification in agile methodologies, particularly Scrum. This study is an examination of the IEEE Recommended Practice for Software Requirements Specifications (830-1998) to summarize the content and qualities of good software requirements specifications. It is also a comparison of traditional software requirements specifications with user stories. User stories structure, advantages and disadvantages as well as the characteristics of a good quality user stories are reviewed here. Finally, the study includes an analysis of how user stories and requirements can work together combining the flexibility of user stories with the precision and manageability of traditional requirements.

Keywords: Agile, Scrum, User Stories, INVEST, Requirements Engineering, Requirements Specification, IEEE Std. 830-1998.

1. Introducción

El presente trabajo constituye la primera etapa del plan de tesis de maestría en Ingeniería en Sistemas de Información, “**Modelo Adaptable de Trazabilidad de Requerimientos de Software en Entornos Ágiles de gran escala**”. El mismo tiene por objetivo evaluar el impacto de no establecer mecanismos formales de trazabilidad de requerimientos a lo largo del ciclo de desarrollo y proponer un modelo de trazabilidad de requerimientos en metodologías ágiles. Dado que existen opiniones encontradas respecto de la utilidad de las matrices de trazabilidad de requerimientos de software (Ambler, Agile Requirements Modeling, 2010) (Ramesh & Jarke, 2001), el primer objetivo del plan de tesis de maestría es medir el impacto de no utilizar dichas matrices en proyectos ágiles de gran escala (Ambler, The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments, 2009). Pero para poder dar comienzo a dicho análisis es preciso determinar cómo se especifican los requerimientos en metodologías ágiles como *Scrum* (Schwabe, 2004) y qué características deben poseer dichas especificaciones para cumplir con las características de un buen requerimiento según la IEEE (Software Engineering Standards Committee of the IEEE Computer Society, 1998) (Software Engineering Standards Committee of the IEEE Computer Society, 1998).

En base a lo mencionado anteriormente se plantea como objetivo del presente trabajo determinar los principales atributos y características que debe cumplir un requerimiento ágil, revisando la bibliografía existente sobre técnicas de especificación de requerimientos de software en *Scrum* (Schwabe, 2004) e identificando las características de un buen requerimiento según IEEE (Software Engineering Standards Committee of the IEEE Computer Society, 1998) (Software Engineering Standards Committee of the IEEE Computer Society, 1998). El trabajo será elaborado en diferentes secciones, una primera sección que definirá los requerimientos de software, la ingeniería de requerimientos y el proceso de especificación de requerimientos de software. Una segunda sección en donde se hará una introducción a las metodologías ágiles, a *Scrum* (Schwabe, 2004) puntualmente y luego a las historias de usuario. Una tercera sección donde se hará una revisión de la bibliografía existente acerca de la práctica de especificación de requerimientos en *Scrum* (Schwabe, 2004) y se analizarán las diferencias respecto de las metodologías tradicionales. Por último, se desarrollarán las conclusiones del trabajo y detallarán los pasos a seguir en las próximas etapas.

2. Introducción a los Requerimientos de Software

a. Requerimientos de Software

El diccionario estándar de terminología de Ingeniería de Software ha definido a los requerimientos de software como (Standards Coordinating Committee of the Computer Society of the IEEE, 1990):

- (1) Una condición o capacidad requerida por un usuario para resolver un problema o alcanzar un objetivo.
- (2) Una condición o capacidad que debe ser poseída por un sistema o componente de un sistema para satisfacer un contrato, un estándar, una especificación u otro tipo de documento formalmente impuesto.
- (3) Una representación documentada de una condición o capacidad según 1 y 2.

Según la metodología que se utilice, existen varias clasificaciones de los tipos de requerimientos (Sandoval Carvajal, y otros, 2008). Una de ellas es la que distingue a los requerimientos entre funcionales y no funcionales (Sandoval Carvajal, y otros, 2008).

Requerimientos funcionales: Declaraciones de los servicios que proveerá el sistema, de la manera en que éste reaccionará a entradas particulares y de cómo se comportará en situaciones particulares. Pueden declarar lo que el sistema no debe hacer (Sandoval Carvajal & García Vargas, 2008).

Requerimientos no funcionales: Son aquellos requerimientos que no se refieren directamente a las funciones específicas que entregará el sistema, sino a las propiedades emergentes de éste tales como la fiabilidad, el tiempo de respuesta, la capacidad de almacenamiento, etc. En general se refieren al sistema como un todo y no a rasgos particulares, por lo tanto, una falla de un requerimiento no funcional, podría inutilizar un sistema (Sandoval Carvajal & García Vargas, 2008)

Cabe aclarar que el desarrollo de presente trabajo estará focalizado en los requerimientos funcionales únicamente.

b. Ingeniería de Requerimientos de Software

La ingeniería de requerimientos involucra descubrir cuáles son las metas, necesidades y expectativas de los *stakeholders*¹, ajustar las expectativas de los mismos y comunicarlas a los desarrolladores (Nuseibeh & Easterbrook).

La ingeniería de requerimientos está formada por una serie de procesos bien diferenciados (Ver figura: Figura 1: Procesos de la Ingeniería de Requerimientos) (Wiegiers, 1999).

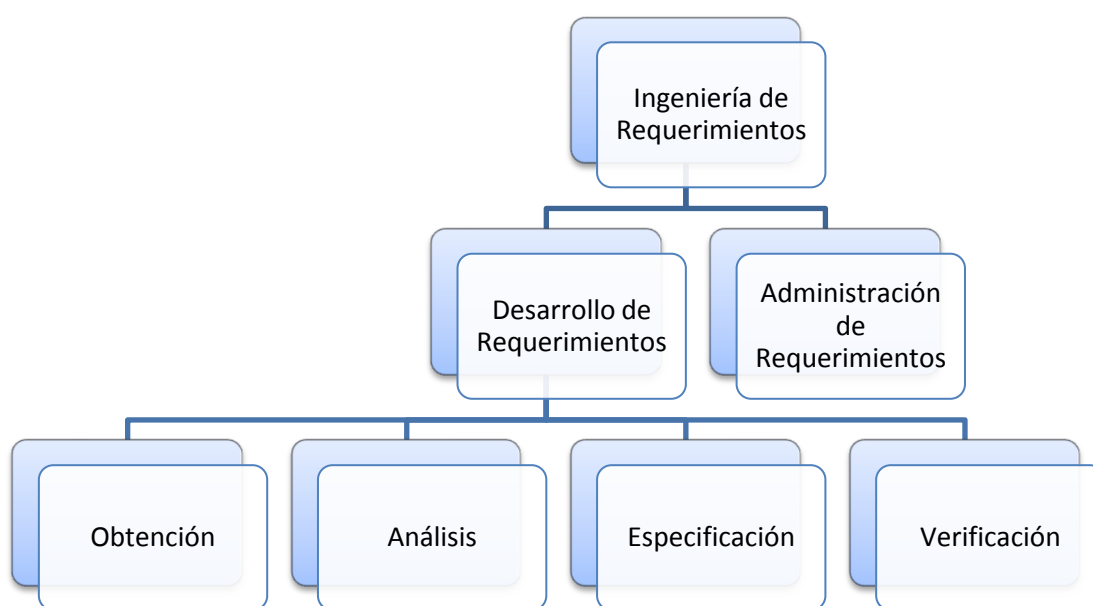


Figura 1: Procesos de la Ingeniería de Requerimientos

Los procesos de desarrollo de los requerimientos (Figura 1: Procesos de la Ingeniería de Requerimientos) abarcan todas las actividades relacionadas con la recopilación, evaluación y documentación de los requerimientos de software, incluyendo:

- Identificar las clases de usuario del producto esperado.

¹ Grupo de personas afectadas o responsables por la salida de una actividad. *Stakeholders* de un proyecto son sus miembros, proveedores, clientes, usuarios finales, etc. (Software Engineering Institute, 2010). *Stakeholders* relevante es aquel que se encuentra involucrado en actividades especificadas en el plan (Software Engineering Institute, 2010). *Stakeholder* es también aquel cuyos intereses pueden verse afectados positiva o negativamente por la *performance* del proyecto. Aquel que puede ejercer influencia sobre el proyecto, miembros del equipo o sus entregables (Bourse, 2009).

- Extraer las necesidades de los individuos que representan cada clase de usuario.
- Comprender las tareas y metas del usuario y los objetivos de negocio con los que esas tareas se alinean.
- Analizar la información recibida de los usuarios para distinguir sus objetivos de tarea de requerimientos funcionales, requerimientos no-funcionales, reglas de negocio, etc.
- Destinar partes de los requerimientos de alto nivel a definir componentes de software en la arquitectura sistema.
- Comprender la importancia de los atributos de calidad.
- Negociar las prioridades de implementación.
- Traducir las necesidades de usuario escritas dentro de las especificaciones y modelos de requerimientos.
- Examinar los requerimientos documentados para asegurar el conocimiento común de los requerimientos presentados por los usuarios y corregir cualquier problema antes de que el grupo de desarrolladores los acepte.

La iteración es una clave para el éxito del desarrollo de los requerimientos.

La Administración de Requerimientos (Figura 1: Procesos de la Ingeniería de Requerimientos) implica establecer y mantener actualizado un acuerdo con el cliente de los requerimientos para el proyecto de software (Wieggers, 1999) e incluye las siguientes actividades:

- Definir el punto de partida de los requerimientos.
- Revisar y evaluar el impacto de cada requerimiento cambiado antes de aprobarlo.
- Seguir cada requerimiento en su diseño, código fuente y pruebas.
- Agrupar los requerimientos según rendimiento y actividad de cambio durante todo el proyecto.

Debido a que el presente trabajo se enfoca únicamente en el proceso de especificación de requerimientos, es deseable entender la definición de dicho proceso según modelos internacionales tal como es el CMMI (Software Engineering Institute, 2010). El Desarrollo de los Requerimientos es

un área de proceso del CMMI² (Software Engineering Institute, 2010) que pertenece al nivel 3 en la representación por etapas y está ubicado dentro de la categoría de proceso de Ingeniería para la representación continua. Tiene como propósito producir y analizar los requerimientos del cliente, del producto y de los componentes del producto.

Según ha sido definido por el modelo, los objetivos y prácticas específicas para esta área de proceso incluyen (Software Engineering Institute, 2010):

Objetivo Específico N°1: Definir los Requerimientos del Cliente: Las necesidades, expectativas, restricciones e interfaces de los interesados son recogidas y traducidas en requerimientos del cliente.

- Práctica Específica 1.1: Obtener las necesidades del cliente
- Práctica Específica 1.2: Transformar las necesidades identificadas en requerimientos del cliente.

Objetivo Específico N°2: Definir los Requerimientos del Producto: Los requerimientos del cliente son refinados y elaborados para desarrollar los requisitos del producto y de componentes del producto.

- Práctica Específica 2.1: Establecer Requerimientos de Producto y Componentes de Producto.
- Práctica Específica 2.2: Asignar los Requerimientos para cada componente del producto.
- Práctica Específica 2.3: Identificar Requerimientos de Interfaz.

Objetivo Específico 3: Analizar y evaluar los requerimientos definidos

- Práctica Específica 3.1: Establecer y mantener conceptos y escenarios operativos.
- Práctica Específica 3.2: Establecer y mantener definiciones de funcionalidades requeridas y atributos de calidad.
- Práctica Específica 3.3: Analizar los Requerimientos para asegurar que son necesarios y suficientes.
- Práctica Específica 3.4: Analizar los Requerimientos para equilibrar las necesidades y restricciones de los *stakeholders*.
- Práctica Específica 3.5: Validar los Requerimientos

² CMMI: Capability Maturity Model Integration

c. Especificación de requerimientos

Es el proceso de grabado o el registro de los requerimientos en una o más formas, incluyendo el lenguaje natural y formal, representaciones simbólicas o gráficas (Tuffley, 2005). La especificación de los requerimientos es el paso en donde los resultados de la identificación de los requerimientos se “retratan” (Brackett, 1990).

Tradicionalmente los requerimientos se han representado en una forma puramente textual. Sin embargo, se están utilizando otras técnicas tales como la construcción de modelos y prototipos, que demandan una descripción más detallada de los requerimientos (Borland, 2005).

Dado que muchas veces los usuarios no son capaces de pensar en todas las situaciones posibles en las cuales el software puede ser utilizado, es tarea de los desarrolladores documentar los requerimientos desde un punto de vista que admita la posibilidad de verificarlos (*testing*). En este proceso se darán a conocer posibles situaciones que no se hubiesen tenido en cuenta la primera vez (Borland, 2005).

d. Guía para escribir buenos requerimientos

No exista una fórmula para escribir buenos requerimientos, es la experiencia la que enseña a elaborar mejores requerimientos. Los documentos de requerimientos deben evitar la jerga técnica y emplear terminología que pueda ser comprendida por los usuarios. A continuación se describen algunas recomendaciones para escribir buenos requerimientos (Wiegers, 1999):

- Escribir oraciones completas
- Usar la voz activa
- Usar los términos consistentemente y crear un glosario
- Descomponer los requerimientos de alto nivel, agregar detalle y claridad para evitar la ambigüedad
- Especificar los requerimientos de manera consistente
- Definir la condición o la acción que causa que el sistema ejecute un comportamiento específico
- Identificar los actores

- Usar tablas, listas, figuras e imágenes para presentar la información
- Enfatizar la información que es importante
- Evitar el uso de términos vagos y subjetivos

e. Atributos y características deseables de la especificación de requerimientos según IEEE

El proceso de especificación de requerimientos de software tiene por objetivo obtener documentación no ambigua y completa de los requerimientos de software. Los beneficios de la especificación de los requerimientos de software se describen a continuación:

- Establecen una base de acuerdo entre los clientes y los proveedores acerca de lo que el software debe hacer.
- Reducen el esfuerzo de desarrollo: Dado que se especifican y validan previo al desarrollo.
- Constituyen una base para estimar costos y calendario.
- Constituyen una línea base para efectuar actividades de verificación y validación.
- Facilitan la transferencia de los requerimientos de software a otros interesados.
- Sirven de base para elaborar mejoras posteriores al producto.

Una característica importante sobre la especificación de los requerimientos de software es que debe ser escrita por uno o más representantes del proveedor³, uno o más representantes del cliente⁴ o por ambos.

Aspectos básicos que debe especificarse de los requerimientos según IEEE (Software Engineering Standards Committee of the IEEE Computer Society, 1998) (Software Engineering Standards Committee of the IEEE Computer Society, 1998):

- a) Funcionalidad: Lo que el software debe hacer.
- b) Interfaces externas: Cómo debe interactuar el software con las personas, hardware, software, etc.

³ Proveedor: Persona o personas que desarrollan un producto. En el contexto de esta práctica, el cliente y el proveedor pueden ser personas de la misma organización (Software Engineering Standards Committee of the IEEE Computer Society, 1998) (Software Engineering Standards Committee of the IEEE Computer Society, 1998).

⁴ Cliente: Persona o personas que pagan por el producto y usualmente (no siempre) deciden los requerimientos (Software Engineering Standards Committee of the IEEE Computer Society, 1998) (Software Engineering Standards Committee of the IEEE Computer Society, 1998).

- c) Performance: Velocidad, disponibilidad, tiempo de respuesta, tiempo de recuperación ante fallos, etc.
- d) Atributos: consideraciones de portabilidad, mantenibilidad, seguridad, etc.
- e) Restricciones de diseño: estándares, lenguajes de programación, políticas de integración, sistemas operativos, etc.

Características de una buena especificación de requerimientos de software:

- Correcta: Los requerimientos son correctos en tanto y cuanto representen lo que realmente se espera del software.
- No ambigua: Los requerimientos son ambiguos cuando pueden interpretarse de diferentes formas. Cuando en la especificación de los requerimientos de software se utilizan términos que puede tener distintas acepciones dependiendo del contexto, se sugiere agregar un glosario y anexarlo a la especificación.
 - Dificultades del lenguaje natural: El lenguaje natural es ambiguo por naturaleza. Se recomienda solicitar revisión de las especificaciones para identificar y corregir las ambigüedades.
 - Lenguajes de Especificación de Requerimientos: Una forma de evitar la ambigüedad en la especificación de los requerimientos de software consiste en utilizar lenguajes de especificación. Existen procesadores de lenguaje que pueden identificar errores semánticos y sintácticos. Sin embargo, lleva tiempo aprender a utilizarlos.
 - Herramientas de Representación: En general, tanto los métodos y los lenguajes de especificación como así también las herramientas de soporte a dichos métodos y lenguajes, pueden categorizarse de la siguiente manera: Objetos, Procesos o Comportamiento. Los enfoques orientados a objetos organizan a los requerimientos en objetos del mundo real, e identifican atributos y los servicios que los objetos exponen. Los enfoques basados en procesos organizan a los requerimientos en una jerarquía de funciones que comunica flujo de datos. Los enfoques en comportamientos en cambio, describen el comportamiento del sistema en nociones abstractas, tales como funciones matemáticas o máquinas de estado.

- **Completa:** La especificación de los requerimientos de software es completa si incluye los siguientes elementos:
 - Todos los requerimientos, ya sea funcionales, no funcionales, restricciones de diseño, atributos o interfaces externas deben ser identificados y especificados.
 - Deben identificarse las respuestas esperadas del software a toda clase de datos de entrada y en toda clase de situaciones posibles. Es preciso identificar el resultado esperado ante una entrada válida y una entrada inválida.
 - Debe etiquetarse y referenciarse todas las figuras, tablas y diagramas y deben definirse todos los términos y unidades de medida.
 - Las especificaciones de requerimientos de software que dejan aspectos a ser determinados posteriormente no son completas.
- **Consistente:** Las especificaciones de los requerimientos de software no deben contradecir ni generar conflictos con otros requerimientos previamente especificados (contratos, documento de requerimientos de marketing, etc.).
- **Priorizada:** Los requerimientos no son igualmente importantes. Algunos requerimientos serán esenciales mientras que otros podrán ser deseables. Es recomendable dejar explícita dicha importancia en la especificación, como así también algún indicador de cuán estable es la especificación respecto de posibles cambios en el futuro. Un modo de priorizar los requerimientos es agruparlos en requerimientos esenciales, condicionales y opcionales. Esencial implica que el software no será útil a menos que estos requerimientos hayan sido implementados. Condicionales son aquellos requerimientos que mejorarían el producto, pero que no lo haría inútil si no estuvieran. Finalmente, opcionales son aquellos requerimientos que de ser implementados, excederían las expectativas de los usuarios finales.
- **Verificable:** Un requerimiento es verificable si existe un proceso efectivo que una máquina o persona puedan llevar a cabo para chequear que el software cumple con los requerimientos de software. Algunos ejemplos de requerimientos no verificables se describen a continuación: requerimientos cuya especificación incluya frases tales como: “funciona bien”, “buena interfaz de usuario”, “debe ocurrir frecuentemente”, etc. No son verificables dado que no pueden definirse “bien”, “buena” y “frecuentemente”. Para ser verificables debe contener definiciones de términos y cantidades mensurables.

- **Modificable:** Las especificaciones de los requerimientos de software son modificables siempre y cuando sea posible efectuar los cambios de manera completa, simple y consistentemente, manteniendo la estructura y estilo. Por este motivo, es conveniente no repetir las especificaciones de los requerimientos en diferentes lugares, dado que ello puede llevar a que no sean actualizados consistentemente.
- **Que se pueda hacer un seguimiento del requerimiento (trazabilidad):** Debe poder hacerse un seguimiento desde el requerimiento inicial a los diferentes elementos de trabajo que se crean a lo largo del desarrollo del mismo. Existen diferentes sentidos desde el cual puede efectuarse el seguimiento de un requerimiento:
 - De adelante hacia atrás: etapas posteriores de desarrollo. Es importante poder identificar unívocamente la especificación de un requerimiento.
 - De atrás hacia adelante: es importante establecer y mantener las referencias del requerimiento a los diferentes artefactos de trabajo.

3. Introducción a las metodologías ágiles, Scrum e Historias de Usuario

a. Metodologías Ágiles

En febrero de 2001 nace el término “ágil” aplicado al desarrollo de software y se creó “The Agile Alliance”⁵, una organización sin fines de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones a adoptar dichos conceptos (Canós, Letelier, & Penadés). El punto de partida fue el Manifiesto Ágil (Beck & al, 2001), según el cual se valora:

Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas (Beck & al, 2001).

La gente es el principal factor de éxito de un proyecto software. La motivación y los equipos auto gestionados son muy importantes; como así también la interacción entre las personas (Canós, Letelier, & Penadés).

Desarrollar software que funciona más que conseguir una buena documentación (Beck & al, 2001).

La regla a seguir es no producir documentos a menos que sean necesarios para tomar una decisión importante. Estos documentos deben ser cortos y centrarse en lo fundamental (Canós, Letelier, & Penadés).

La colaboración con el cliente más que la negociación de un contrato (Beck & al, 2001). Los requerimientos del software no siempre van a poder colectarse al inicio del ciclo de desarrollo. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito (Canós, Letelier, & Penadés).

Responder a los cambios más que seguir estrictamente un plan (Beck & al, 2001). La habilidad de responder a los cambios que puedan surgir a los largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser exacta sino flexible (Canós, Letelier, & Penadés).

⁵ www.agilealliance.com

En las metodologías ágiles, aunque se valoran los elementos de la derecha, se valoran más aún a los elementos de la izquierda (Beck & al, 2001). Estas declaraciones son la base de la filosofía “ágil” de varias metodologías de desarrollo de software entre las que se encuentran *Scrum* (Schwabe, 2004), *XP* (Programación Extrema) (Well, 2009), *Lean Software Development* (Poppendiek & Poppendiek, 2003) (Desarrollo de Software “*Lean*”), *Kanban* (Peterson, 2009), entre otras.

b. Introducción a Scrum

Scrum es un proceso de desarrollo de software iterativo⁶ e incremental⁷, donde las iteraciones son cortas, entre 2 y 4 semanas, y los equipos de trabajo acuerdan con sus clientes la lista priorizada de requerimientos que deben completar en cada iteración. Es iterativo dado que el trabajo es planeado en iteraciones y el resultado de una iteración es mejorado en iteraciones posteriores. Es incremental en cuanto a que las funcionalidades completadas son entregadas durante el desarrollo y no al finalizar el proyecto (Cohn, Using Stories with Scrum, 2009). Cabe destacar el concepto de funcionalidad completa o lista para entregar, ya que la misma implica que el requerimiento debe estar desarrollado y probado de manera que pueda ser entregado inmediatamente a un cliente o interesado. Otro concepto importante es que estas metodologías requieren que exista una fluida comunicación entre el equipo de trabajo y los interesados, logrando de esta forma que los cambios puedan ser implementados de forma rápida y económica (Schwabe, 2004). El esquema básico de *Scrum* se muestra en la siguiente figura (Mountain Goat Software - Imágenes, 2005).

⁶ Iterativo: un proceso iterativo es aquel que genera progreso a través de refinamientos sucesivos. Un equipo de desarrollo toma una parte del software sabiendo que no es completo y progresivamente lo perfecciona hasta obtener un producto satisfactorio. En cada iteración el software es mejorado (Cohn, User Stories Applied: For Agile Software Development, 2009).

⁷ Incremental: un proceso incremental es aquel en el cual el software es construido y entregado en partes. Cada parte, o incremento, representa un subconjunto completo de funcionalidades (Cohn, User Stories Applied: For Agile Software Development, 2009).



Figura 2: Esquema básico de Scrum (Mountain Goat Software - Imágenes, 2005)

Scrum muestra progreso a través de iteraciones, las cuales han sido denominadas *sprints*. Los *sprints* tienen duración fija y pueden extenderse hasta un mes. Al iniciar un *sprint*, el equipo determina la cantidad de trabajo que puede completar durante la próxima iteración. El trabajo a completarse en un *sprint* es extraído de los ítems de producto, en inglés *product backlog*. El trabajo que el equipo considera puede completar durante la iteración se organiza en los que se conoce como Ítems de Iteración, en inglés *sprint backlog*. Se efectúan reuniones diarias para evaluar el progreso obtenido y efectuar los ajustes que hagan falta con el fin de completar el trabajo comprometido al finalizar la iteración.

A continuación se explicarán los elementos básicos de *Scrum*:

- Los Equipos: Los equipos *Scrum* deben estar conformados por 4 a 7 personas. Si bien se valora que existan personas con determinados perfiles (programadores, *testers*, especialistas en bases de datos, etc.), lo más importante es que exista una identidad de equipo. El equipo es responsable de cumplir con el trabajo asumido. Las responsabilidades de los miembros del equipo abarca pruebas, análisis, arquitectura, diseño, programación, planificación y estimación (Ambler & Holitza, Understanding Agile Roles, 2012).

- *Scrum Master*: El *scrum master* lidera el equipo y elimina los obstáculos que impiden a los miembros del equipo avanzar con sus actividades. Ayuda al equipo a focalizarse en el trabajo que se debe desarrollar y cumplir los objetivos de la iteración.
- Dueño del producto o *product owner*: representa la voz del cliente ya que conoce sus necesidades y expectativas. Puede aclarar los detalles de la solución propuesta y es responsables de que exista una lista priorizada de ítems del producto. Algunas responsabilidades adicionales del dueño del producto: dar visibilidad acerca del estado del proyecto y representar al equipo ante los demás *stakeholders*. Elaborar estrategias de desarrollo. Establecer objetivos de corto y largo plazo. Dar a conocer las necesidades de los clientes y demás *stakeholders* a los miembros del equipo. Obtener, priorizar y gestionar los requerimientos del producto. Establecer fechas de entrega del producto. Contestar las preguntas de los equipos y participar en la toma de decisiones. Aceptar o rechazar el trabajo desarrollado por los equipos (Ambler & Holitza, Understanding Agile Roles, 2012).
- Las ceremonias: Como se mencionó anteriormente, una de las características fundamentales de las metodologías ágiles y particularmente de *Scrum*, es el fomento de la comunicación cara a cara entre los miembros del equipo y los interesados claves. Esta comunicación está establecida en el marco de trabajo a través de reuniones que tienen objetivos claramente identificados, y que se mencionan a continuación:
 - Reunión de planificación de la iteración: Esta reunión se realiza una vez al comienzo de cada iteración y su objetivo es identificar los ítems de producto que se desean entregar al finalizar la misma. Luego de seleccionar los ítems de producto a completarse durante la iteración, deben identificarse las tareas necesarias para producir cada uno de esos ítems de manera que los mismos estén listos para ser entregados, esto implica no sólo el desarrollo de los ítems sino también todas las actividades de verificación y validación requeridas para alcanzar el criterio de calidad preestablecido.
 - Reuniones de avance y compromiso diarios: Estas reuniones se realizan todos los días y son acotadas en su duración. El objetivo de estas reuniones es que cada miembro del equipo informe sobre el avance realizado en el día anterior,

el trabajo a realizar en el día de la reunión y los problemas que pudiera tener que impidan que realice el trabajo.

- Reunión de revisión de la iteración: Esta reunión se realiza una vez al finalizar cada iteración. El objetivo de esta reunión es mostrar a los interesados del proyecto todos los ítems de producto que se encuentran listos para ser entregados.
- Reunión de identificación de lecciones aprendidas y definición de acciones para la siguiente iteración también conocidas como retrospectivas: Esta reunión se realiza una vez al finalizar cada iteración. El objetivo de esta reunión es que los miembros del equipo expongan sus comentarios acerca de los problemas que hubieron durante la iteración, las cosas que resultaron bien y deberían repetirse, como así también las nuevas ideas que podrían llevarse a cabo en la siguiente iteración (Schwaber, 2001).
- Ítems del Producto o *product backlog*: Lista de las funcionalidades que se esperan encontrar en el producto software. Cuando arranca un proyecto, los dueños del producto y los equipos escriben las primeras historias, lo cual es suficiente para arrancar con la primera iteración. A medida que el proyecto avanza y se va ganando más conocimiento del dominio y de las necesidades de los usuarios finales, se va completando la lista.
- Puntos de Historia: son una unidad de medida creada para expresar el tamaño global de una actividad. Un error común es tratar a los puntos de historia sólo como una medida de complejidad, cuando en realidad los puntos de historia son una combinación de complejidad, esfuerzo, riesgos, etc. Otro aspecto importante es que los puntos de historia, a diferencia de las estimaciones de tiempo, son relativas. Una forma común de trabajar es elegir un tema que parece ser pequeño y dar un valor de referencia, a partir de allí las otras funcionalidades se estiman en relación a la referencia (Gomez, 2009).

- Velocidad de un equipo de desarrollo⁸: al finalizar las iteraciones, los equipos contabilizan los puntos de las historias de usuario finalizadas en la iteración. El total de puntos de historia completados es lo que se conoce como velocidad del equipo.
- Planificación del *release*: Un *release* está compuesto por una o más iteraciones. La planificación de un *release* se refiere a determinar el balance entre el tiempo calendario y el conjunto de funcionalidades que se necesitan en ese tiempo calendario. Durante esta etapa se priorizan las historias de usuario en función de cuán requeridas son para los usuarios y cuán cohesivas son las historias en entre sí. Por ejemplo, una historia para hacer “*zoom in*” puede carecer de sentido si no se implementa la historia para hacer “*zoom out*” (Cohn, An Overview, 2009).
- Criterios de Aceptación: definen los límites de las historias de usuario y los parámetros que ayudarán a determinar que las historias de usuario han sido terminadas y que funcionan de acuerdo a las expectativas del dueño del producto (Leffingwell & Behrens, 2009).

Podemos ver que este marco de trabajo permite al equipo de desarrollo interactuar fluidamente y tomar decisiones con respecto a su modalidad de trabajo, de manera que logren el objetivo de entregar un producto al final de cada iteración (Schwaber, 2001).

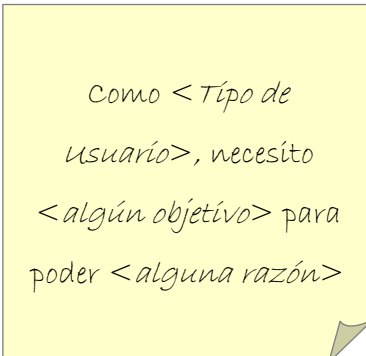
Con esta información confirmamos lo dicho anteriormente en relación a los principios fundamentales en los que se basa *Scrum*: entregas cortas (rapidez) y entrega de funcionalidades completas potencialmente entregables en cada iteración las cuales deben cumplir con cierto nivel de calidad preestablecido.

c. Las Historias de Usuario

Siguiendo con la definición de requerimientos de software de la IEEE (Standards Coordinating Committee of the Computer Society of the IEEE, 1990) citada en la sección Introducción a los Requerimientos de Software, podemos concluir que las historias de usuario son requerimientos ya que expresan el problema que el sistema o producto software debe

⁸ La *velocidad* de una iteración es una métrica de *Scrum* (Cohn, What is Scrum?, 2009). Existen otras métricas que no serán abordadas por no estar relacionadas con el objetivo de este trabajo.

resolver. Las Historias de Usuario son un enfoque de requerimientos ágil que se focaliza en establecer conversaciones acerca de las necesidades de los clientes. Son descripciones cortas y simples de las funcionalidades del sistema, narradas desde la perspectiva de la persona que desea dicha funcionalidad, usualmente un usuario (Cohn, Mountain Goat Software). Poseen las siguientes características: una descripción escrita que será utilizada para planificar y posteriormente disgregar los detalles con el dueño del producto, las conversaciones propiamente dichas con el dueño del producto y las pruebas que han de determinar si las historias están finalizadas o no (Cohn, An Overview, 2009). Generalmente se las escribe en post-its (notas), y se las dispone en una pared o una mesa para facilitar de este modo la planificación y discusiones que se llevan a cabo durante la misma. La parte más importante de las historias de usuario es la conversación que se genera entorno a las mismas. Estas notas representan los requerimientos del cliente y típicamente se las escribe de la siguiente manera:



*Como <Tipo de
usuario>, necesito
<algún objetivo> para
poder <alguna razón>*

Figura 3: Formato típico de una Historia de Usuario

Uno de los beneficios de las Historias de Usuario es que pueden ser escritas en diferentes niveles de detalle. Es posible escribir historias que cubren múltiples funcionalidades. Estas historias más grandes son llamadas Épicas y dado que típicamente no pueden ser finalizadas en una iteración, se las divide en múltiples Historias de Usuario (Cohn, **User Stories Applied: For Agile Software Development, 2009**).

Los dueños del producto son responsables de que exista una pila de producto compuesta de Historias de Usuario, sin embargo, no necesariamente son ellos quienes deben escribirlas. Lo

importante es que participen en las discusiones que aportan mayor detalle sobre las necesidades de los usuarios (**Cohn, What is Scrum?, 2009**).

Las Historias de Usuario son escritas a lo largo de todo el proyecto de desarrollo. Usualmente al comenzar un proyecto se lleva a cabo un *workshop* donde participan todos los miembros del equipo, con el fin de crear una pila de producto que describa las funcionalidades que van a desarrollarse en el curso de los siguientes tres a seis meses (**Cohn, What is Scrum?, 2009**).

El modelo INVEST presenta una serie de atributos a tener en cuenta para lograr escribir buenos requerimientos en metodologías ágiles.

d. El modelo INVEST

El modelo INVEST (por sus siglas en inglés *Independent, Negotiable, Verifiable, Estimable, Small, Testable*) es la clave para pensar y escribir buenas historias de usuario. Las historias deben ser Independientes, Negociables, Valiosas, Estimables, Pequeñas y Testables (Leffingwell & Behrens, 2009).

- **Independiente:** Esto significa que las historias de usuario deben poder implementarse, probarse y entregarse por sí mismas sin depender de otras funcionalidades (Leffingwell & Behrens, 2009). La dependencia entre las historias de usuario hace que sea más difícil planificar, priorizar y estimar. A menudo, se pueden reducir las dependencias haciendo una combinación de historias, o partiéndolas de forma diferente.
- **Negociable:** A diferencia de los requerimientos tradicionales, las historias de usuario no son obligaciones contractuales (Leffingwell & Behrens, 2009), sino más bien el compromiso de establecer conversaciones con el dueño del producto para convenir los detalles de los requerimientos y así poder implementarlos, probarlos y validarlos. Este es el proceso de negociación mediante el cual el equipo de desarrollo reconoce las necesidades del negocio, pero también aporta sus ideas en base a la colaboración y retroalimentación.
- **Valiosa:** El objetivo de los equipos *scrum* es proveer valor a los clientes y usuarios con los recursos disponibles, en el tiempo disponible. Ese es el motivo que hace que ésta sea la característica más importante del modelo INVEST. Los ítems del producto son priorizados en

función del valor que cada historia proveerá a los clientes, usuarios y demás *stakeholders* del producto. Una forma muy eficaz de generar historias valiosas es hacer que el cliente la escriba.

- **Estimable:** Los desarrolladores necesitan poder estimar una historia de usuario para que se pueda priorizar y planificar. Los problemas que pueden impedirle a los desarrolladores estimar una historia son: falta de conocimiento del dominio (en cuyo caso se necesita más Negociación / Conversación); o si la historia es muy grande (en cuyo caso se necesita descomponer la historia en historias más pequeñas).
- **Pequeña:** Una buena historia debe ser pequeña en esfuerzo, generalmente representando no más de 2-3 personas/semana de trabajo. Una historia que es más grande va a tener más errores asociados a las estimaciones y al alcance. Las historias de usuario deberían ser lo suficientemente pequeñas como para poder ser implementadas en una iteración.
- **Verificable:** No se desarrolla lo que no puede ser probado. Si no puede probarse, nunca va a saberse si se ha terminado. Si una historia no puede ser verificada probablemente sea muy compleja, o tenga muchas dependencias con otras historias. Para lograr que las historias de usuario hayan sido probadas antes de finalizar la iteración, algunos equipos de desarrollo emplean un enfoque que comienza creando los casos de prueba y luego continúa con la codificación, este enfoque es conocido como Desarrollo Guiado por Pruebas, o en inglés *Test Driven Development*.

e. Las historias de usuario en el proceso de desarrollo de requerimientos

Aunque CMMI (Software Engineering Institute, 2010) es independiente del enfoque de desarrollo que se utiliza, en la versión 1.3 se agregaron anotaciones específicas para ayudar a quienes emplean metodologías ágiles a interpretar las prácticas descriptas en el modelo. El área de proceso Desarrollo de Requerimientos hace una mención a las metodologías ágiles en donde se especifica que, al igual que en otros enfoques, en ágiles las necesidades de los clientes también son obtenidas, elaboradas, analizadas y validadas. Los requerimientos se documentan de diferentes maneras, historias de usuario, escenarios, casos de uso, ítems de productos y en los resultados obtenidos al finalizar cada iteración (código funcionando). Los requerimientos a desarrollarse en cada iteración son derivados en base a las prioridades y evaluaciones de riesgos de lo que queda pendiente en la pila

del producto. Los detalles a documentar de los requerimientos estarán guiados por las necesidades de coordinación (entre los miembros de un equipo, entre equipos y en iteraciones posteriores) y el riesgo de perder información de lo que se ha aprendido hasta el momento (Software Engineering Institute, 2010). Como ya ha sido mencionado, los detalles de las historias de usuario son ajustados por medio de conversaciones que son llevadas a cabo entre el dueño del producto y los equipos de desarrollo y, si fuera necesario registrar ciertos detalles de estas conversaciones podrían adjuntarse documentos a las historias de usuario (Leffingwell & Behrens, 2009).

f. El desarrollo de los requerimientos en Scrum

Los mismos principios que se utilizan en el desarrollo de software bajo metodologías ágiles pueden aplicarse también a la definición y gestión de requerimientos (Moccia, Agile Requirements Definition and Management (RDM), 2012). El problema para la mayoría de las organizaciones es esforzarse en mantener una pila de producto actualizada para que los ítems del producto puedan ser tomados por los equipos de desarrollo (Moccia, Agile Requirements Definition and Management (RDM), 2012). La definición y gestión de requerimientos ágiles ha sido diseñada específicamente para resolver este problema. El objetivo es alimentar la pila del producto a un ritmo mayor al que los equipos de desarrollo pueden generar código. Este marco de trabajo puede utilizarse tanto para la generación de requerimientos justo a tiempo o *just in time* (JIT) como para el armado de un repositorio de requerimientos que han de desarrollarse en el futuro. Se utiliza un ciclo similar al ciclo *Scrum* que emplean los equipos de desarrollo. La diferencia es que este ciclo se encuentra dos o tres etapas adelantado a los equipos de desarrollo (Ver Figura 4: Definición y Gestión de Requerimientos Ágiles (Moccia, Agile Requirements Definition and Management (RDM), 2012)). El objetivo es crear un proceso en el cual se definan, revisen, organicen y comuniquen los requerimientos. El proceso comienza identificando y construyendo una pila de requerimientos. Esta pila es una lista de elementos que deben definirse en orden para poder alimentar la pila del producto.

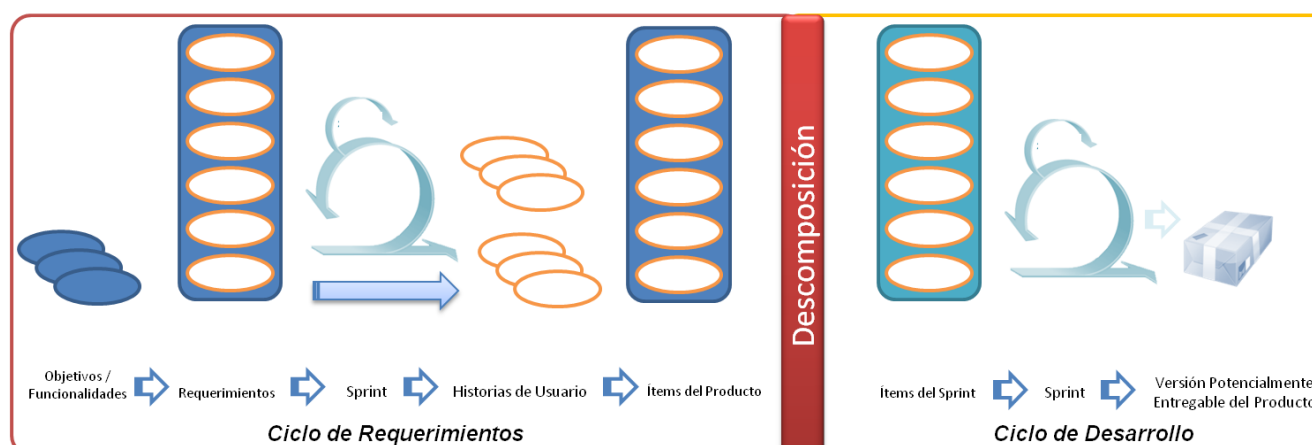




Figura 4: Definición y Gestión de Requerimientos Ágiles (Moccia, Agile Requirements Definition and Management (RDM), 2012)

El objetivo final puede ser una lista de historias de usuario, requerimientos funcionales, etc. El equipo de requerimientos decide, basado en las estrategias y objetivos del negocio, qué cosas deben definirse. Al igual que los equipos de desarrollo, el equipo de requerimientos puede planear su iteración, llevar a cabo el trabajo y finalmente tener una reunión de revisión. Si los resultados de la iteración cumplen con las expectativas planteadas, entonces pueden moverse a la pila del producto. En muchos casos, las organizaciones desarrollarán documentos que no necesariamente terminarán en la pila del producto, sino que serán consultados por los equipos durante el desarrollo. Aquí es donde la trazabilidad de los ítems del producto a cualquier otro documento externo se torna importante. Otra de las partes importantes de este marco de trabajo es la Descomposición. La Descomposición es el proceso por el cual los ítems del producto son comunicados y refinados juntamente con los equipos de desarrollo. En *Scrum* esto es conocido como preparación de la pila del producto o *backlog grooming*.

g. Buenas prácticas para la definición de requerimientos en metodologías ágiles

A continuación se presenta una guía para la escritura de buenas historias de usuario (Cohn, Guidelines for Good Stories, 2009), (Ambler, Agile Modeling Best Practices).

- Participación activa de los *stakeholders*: Los *stakeholders* deben proveer información en tiempo y forma para poder tomar decisiones y deben participar activamente a lo largo del proceso de desarrollo.
- Visión arquitectónica: Al iniciar un proyecto ágil se deberá llevar a cabo modelado arquitectónico de alto nivel para poder identificar la viabilidad técnica de la solución.
- Especificaciones Ejecutables: Preferentemente priorizar las especificaciones ejecutables a la documentación estática. Especificar los requerimientos en la forma de casos de pruebas ejecutables y el diseño como pruebas unitarias.
- Modelado iterativo: al comenzar cada iteración habrá cierta carga de modelado como parte de la planificación de la iteración.
- Bueno suficiente: Los modelos y documentación generada deben ser buenos y suficientes.
- Requerimientos priorizados: los equipos de desarrollo implementan las historias de usuario según su prioridad. Estas prioridades son especificadas por los *stakeholders*.
- Visión de Requerimientos: Al iniciar un proyecto nuevo se deberá invertir tiempo en identificar el alcance del proyecto y en crear una lista inicial de requerimientos priorizados.
- En proyectos grandes y complejos resulta difícil saber por dónde comenzar a identificar las historias de usuario. Una alternativa es comenzar por identificar los roles y los objetivos que persiguen estos roles con el *software*.
- Las historias de usuario pueden descomponerse en partes más pequeñas.
- Escribir historias de usuario cerradas: una historia cerrada es aquella que finaliza con el cumplimiento de un objetivo y que permite a los usuarios sentir que han alcanzado algo.

 <p>ESPECIALIZACIÓN EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN</p>	<p>Carrera de Postgrado de Especialización en Ingeniería en Sistemas de Información</p> <hr/> <p><i>María Paula Izaurralde</i></p>	 <p>Universidad Tecnológica Nacional Facultad Regional Córdoba</p>
---	---	---

- Evitar incluir características de la interfaz de usuario en la descripción de la historia de usuario.
- Hacer foco en las áreas más importante: prestar mayor atención a las cosas que estarán sucediendo en el futuro cercano.
- Utilizar otras formas de especificación de requerimientos si fuera necesario.
- Escribir en voz activa.
- Dejar que los clientes sean quienes escriban las especificaciones en la medida de lo posible.
- No enumerar las historias de usuario.

4. Los requerimientos en las metodologías ágiles: comparación con las formas tradicionales de especificación

a. Ventajas de las Historias de Usuario por sobre los documentos de requerimientos tradicionales

A continuación se listarán las ventajas de las historias de usuario por sobre la especificación de los requerimientos que siguen el estándar de especificación de la IEEE830-1998 (Software Engineering Standards Committee of the IEEE Computer Society, 1998) (Cohn, *Why User Stories?*, 2009).

- Las historias de usuario enfatizan la comunicación verbal por sobre la escrita: la desventaja de documentar los requerimientos de software es que los clientes pueden obtener por resultado lo que el equipo de desarrollo ha interpretado, y no precisamente lo que el cliente necesita. En *Scrum*, el objetivo es documentar lo menos posible en las historias de usuario, por el contrario, se escribe lo estrictamente necesario para recordar que deben establecerse las conversaciones con los clientes para definir los detalles de la implementación.
- Las historias de usuario son entendidas por ambos, los clientes y/o usuarios y los equipos de desarrollo: Una de las ventajas de las historias de usuario por sobre las especificaciones de los requerimientos de software según el estándar IEEE 830 (Software Engineering Standards Committee of the IEEE Computer Society, 1998) (Software Engineering Standards Committee of the IEEE Computer Society, 1998), es que las mismas pueden ser comprendidas tanto por los clientes y/o usuarios finales como por los miembros del equipo de desarrollo (Cohn, *What stories are not*, 2009). Los documentos de especificación de requerimientos de software ERS (en inglés *SRS, Software Requirements Specifications*) (Software Engineering Standards Committee of the IEEE Computer Society, 1998) (Software Engineering Standards Committee of the IEEE Computer Society, 1998), contienen detalles técnicos que pueden resultar difíciles de comprender para los clientes y usuarios finales. Del mismo modo, contienen información específica del negocio que puede resultar incomprensible a los miembros del equipo de desarrollo. Las historias de usuario en cambio, se escriben de modo tal que se expone el valor

agregado a los usuarios, lo cual es comprendido tanto por las personas del negocio como por desarrolladores con perfil técnico (Cohn, Why User Stories?, 2009):

- Las historias de usuario poseen el tamaño necesario para efectuar las estimaciones: (Cohn, User Stories Applied: For Agile Software Development, 2009) las historias de usuario tienen el tamaño adecuado para poder estimar y priorizar los requerimientos en diferentes *releases*. Los documentos de especificación de requerimientos de software pueden resultar más complejos de priorizar y estimar dada la cantidad de requerimientos listados y la fuerte interrelación entre ellos.
- Las historias de usuario sirven para trabajar en iteraciones: una de las ventajas más importantes de las historias de usuario es que son compatibles con el desarrollo iterativo. Esto quiere decir que no es necesario escribir todas las historias antes de comenzar un proyecto, sino que pueden escribirse un conjunto de historias, desarrollarse (codificarlas y probarlas) y luego continuar definiendo otro conjunto.
- Las historias de usuario fomentan el diferimiento de la toma de decisiones (detalles) hasta poseer mejor entendimiento de las necesidades: dado que las historias de usuario se trabajan en iteraciones, es posible diferir la definición de los detalles tanto de negocio como técnicos, hasta tanto se decida comenzar a trabajar en dichas historias de usuario.
- Las historias de usuario fomentan el desarrollo participativo: muchos proyectos han fallado debido a la falta de participación de los usuarios; las historias de usuario involucran y comprometen a los usuarios en el proceso de desarrollo del software. Dado que las historias de usuario evitan la jerga técnica que utilizan los desarrolladores, son totalmente comprensibles para los clientes y usuarios.
- Construyen conocimiento sobre el dominio: fomenta y prevalece la comunicación cara a cara, las historias de usuario promueven la adquisición de conocimiento en todos los miembros del proyecto.

b. Desventajas de usar Historias de Usuario

En secciones previas se analizaron los motivos por los cuales sí conviene utilizar historias de usuario para definir los requerimientos de software, a continuación se describen algunas desventajas según (Cohn, Why User Stories?, 2009):

En proyectos grandes con muchas historias de usuario, se hace más difícil establecer y entender las relaciones entre las historias. Sin embargo, Mike Cohn (Cohn, Why User Stories?, 2009) sugiere organizar las historias según los roles que necesitan y ejecutan determinadas funcionalidades. Definir las historias y los roles en alto nivel de detalle y luego comenzar a refinar las historias cuando los equipos de desarrollo puedan tomarlas para comenzar a trabajar.

En algunos proyectos puede requerirse agregar información a las historias de usuario, como ser la definición y mantenimiento de matrices de trazabilidad. El autor sugiere armar matrices de trazabilidad entre las historias implementadas en cada iteración y los casos de prueba creados para verificarlas.

Por último, si bien las historias de usuario fomenta la comunicación cara a cara y promueven el aprendizaje por medio de la participación activa de los clientes y/o usuarios finales, en proyectos grandes con múltiples equipos de desarrollo distribuidos geográficamente, si no se documenta cierto tipo de información el conocimiento puede perderse.

c. Las historias de usuario no son casos de uso

Inicialmente introducidos por Ivar Jacobson en el año 1992, los casos de uso son asociados al proceso unificado de desarrollo (Jacobson, Booch, & Rumbaugh, 1999). Un caso de uso es una descripción general de las interacciones entre el sistema y uno o más actores, ya sean usuarios o sistemas externos. Las descripciones de los casos de uso se dividen típicamente en cursos normales y cursos alternativos. El curso normal de un caso de uso representa la secuencia más importante de transacciones que satisfacen las necesidades del usuario. Los cursos alternativos son variantes al curso normal (Firesmith) (Firesmith).

Una de las diferencias más notables entre los casos de uso y las historias de usuario es el alcance de las mismas (Cohn, What stories are not, 2009). Si bien ambas describen el valor que agregan al negocio, las historias de usuario son más acotadas dado que deben ser desarrolladas en iteraciones cortas.

Otra de las diferencias entre las historias de usuario y los casos de uso es su longevidad. Los casos de uso son artefactos permanentes que continúan existiendo a lo largo del desarrollo y mantenimiento del software. Las historias de usuario en cambio no sobreviven más allá de las iteraciones en las que fueron desarrolladas (Cohn, What stories are not, 2009).

Por último, las historias de usuario y los casos de uso tienen diferentes objetivos. El objetivo de los casos de uso es llegar a un acuerdo sobre lo que en él se ha especificado. Las historias de usuario en cambio, se escriben con el propósito de facilitar la planificación de los *releases* y servir como recordatorio de que deben llevarse a cabo una serie de conversaciones sobre las necesidades de negocio (Cohn, What stories are not, 2009).

d. Las historias de usuario no son IEEE 830

La IEEE⁹ publicó un conjunto de guías y recomendaciones para escribir especificaciones de requerimientos de software. Este documento, conocido como estándar IEEE 830 (Software Engineering Standards Committee of the IEEE Computer Society, 1998) (Software Engineering Standards Committee of the IEEE Computer Society, 1998), fue revisado por última vez en el año 1998. Las recomendaciones de la IEEE incluyen tópicos como organización del documento de especificación de requerimientos, el rol de los prototipos¹⁰, y las características que debe poseer un buen requerimiento. Los mismos fueron revisados en la primera sección de este trabajo. A continuación se describen una serie de razones por las cuáles algunos autores como Mike Cohn (Cohn, What stories are not, 2009), sostienen que las historias de usuario no deben seguir las recomendaciones del estándar IEEE 830.

Una de las características más distinguibles del estándar IEEE 830 (Software Engineering Standards Committee of the IEEE Computer Society, 1998) (Software Engineering Standards Committee of the IEEE Computer Society, 1998) es que los requerimientos deben ser escritos de la siguiente manera: “El sistema debe...”. A continuación se muestra un fragmento de una especificación de requerimientos de software según el estándar IEEE 830 (Cohn, What stories are not, 2009):

⁹ Institute of Electrical and Electronics Engineers (IEEE).

¹⁰ Prototipo: Formulario, instancia de un producto, servicio, componente de un servicio preliminar que sirve de modelo para etapas posteriores o para la versión final de un producto o servicio (Software Engineering Institute, 2010).

X.1 El sistema debe permitir a las compañías pagar con tarjetas de crédito por el servicio de ofertar trabajos.

X.1.1 El sistema debe aceptar las siguientes tarjetas de crédito: Visa, MasterCard y American Express.

X.1.1 El sistema debe cargar el pago en la tarjeta de crédito antes de publicar la oferta de trabajo en el sitio.

X.1.2 El sistema debe entregar un número de confirmación único.

La documentación de los requerimientos del software a este nivel es tediosa, propensa a errores y consume mucho esfuerzo de armado (Cohn, What stories are not, 2009). Por otro lado, los documentos de especificación de requerimientos escritos de esta manera son muy difíciles de leer.

Desafortunadamente resulta imposible escribir todos los requerimientos del software de esta forma. Existe un ciclo de retroalimentación muy efectivo e importante cuando los usuarios pueden ver el software que se está construyendo. Cuando esto ocurre, surgen nuevas ideas, los usuarios cambian de opinión respecto de lo que esperaban inicialmente. Estos cambios son necesarios para acabar construyendo la solución que satisface las necesidades de los clientes.

Otra de las diferencias entre los requerimientos según IEEE 830 y las historias de usuario es que en el modo tradicional, los costos del producto no pueden calcularse sino hasta haber finalizado el documento. El escenario más típico sería el de un analista que invierte dos o tres meses escribiendo un documento de requerimientos. Cuando éste es entregado a los desarrolladores para que se efectúen las estimaciones, puede ocurrir que se necesiten 24 meses en vez de los 6 meses que esperaba el cliente. En este caso, se invirtió mucho esfuerzo y tiempo de los analistas para especificar un producto que no podrá ser desarrollado en 6 meses. Las estimaciones de las historias de usuario se llevan a cabo conjuntamente con la definición de las mismas. En *Scrum*, los clientes conocen la velocidad de los equipos de desarrollo, las estimaciones de las historias de usuario y el costo aproximado de un punto de historia, entonces es posible determinar cuánto esfuerzo aproximado y tiempo calendario serán requeridos para finalizar un conjunto de funcionalidades.

Con el objetivo de resumir las ventajas, desventajas y diferencias que se analizaron hasta ahora, se presenta a continuación una tabla comparativa entre la especificación de los requerimientos de software según el estándar IEEE830 y las Historias de Usuario.

**Especificación de
Requerimientos de Software
según el Std. IEEE 830 -1998**

**Especificación de
Requerimientos de Software
con Historias de Usuario
(Modelo INVEST)**

Priorización y Estimación	Difícil dada la fuerte interrelación entre los requerimientos y la gran cantidad de requerimientos identificados al comenzar un proyecto (Cohn, Why User Stories?, 2009).	Fáciles de priorizar y planificar (Cohn, Why User Stories?, 2009).
Participación de los clientes y/o usuarios finales	Al inicio, durante la definición de los requerimientos. Al finalizar el proyecto para llevar a cabo las pruebas de aceptación (Cohn, Why User Stories?, 2009).	Activamente durante todo el proceso de desarrollo (Cohn, Why User Stories?, 2009).
Acerca de la Especificación	Los requerimientos deben escribirse siguiendo ciertas pautas con el fin de cumplir con las cualidades de un buen requerimiento. Deben ser: correctos, no ambiguos, completos, verificables, consistentes, clasificables (prioridad), modificables y explorables (trazabilidad) (Software Engineering Standards Committee of the IEEE Computer Society, 1998).	Los detalles se analizan al momento de implementar las historias. Mientras tanto, las historias se escriben en alto nivel siguiendo las pautas del modelo INVEST según el cual las historias deben ser: independientes, negociables, valiosas, estimables, pequeñas y verificables (Leffingwell & Behrens, 2009).
Flexibilidad de cambios (alcance, prioridad)	Poco flexible. Estricto control de cambios (Figueroa, 2012).	Muy flexibles (Cohn, Why User Stories?, 2009)
Enfoque	Foco en las capacidades y	Describen lo que el usuario desear

Comunicación	<p>limitaciones del sistema. Se resta importancia a la interacción del usuario o el contexto de los negocios relacionados con el usuario o la empresa (Figuerola, 2012).</p>	<p>ser capaz de hacer. Se centran en el valor que viene de usar el sistema en lugar de una especificación detallada de lo que el sistema debe hacer (Figuerola, 2012).</p>
	<p>Estrecha comunicación escrita entre el analista de negocios y el equipo de desarrollo (Figuerola, 2012).</p>	<p>Los detalles de los requerimientos de la historia se trabajan en colaboración durante las conversaciones con los usuarios. Las historias de usuarios fueron concebidas como un medio para fomentar la colaboración (Figuerola, 2012).</p>

Tabla 1: Comparación entre la especificación de los requerimientos de software según el estándar IEEE 830 y las Historias de Usuario

e. Análisis del uso de las historias de usuario en la actualidad

La comunicación de los requerimientos de software es un problema (Cohn, An Overview, 2009). Las personas que lo solicitan, ya sea porque lo pagan o porque lo van a utilizar, deben comunicarse con las personas que lo construyen ya que para que el proyecto sea exitoso, se necesitan diferentes puntos de vista (los usuarios finales, clientes y expertos del dominio y los equipos de desarrollo). Si no existe un balance, entonces el proyecto puede fracasar ya sea porque el negocio domina la capacidad de lo que equipo puede construir o porque domina el aspecto técnico y se pierden de vista las necesidades del negocio. Para que los proyectos sean exitosos se necesitan mecanismos que permitan a ambas partes trabajar en forma conjunta (Cohn, An Overview, 2009). Por mucho tiempo se ha entendido a la captura de los requerimientos del proyecto como una fase temprana en el proceso de desarrollo, una vez completada esta etapa se sientan las bases a partir de las cuales se llevará a cabo el proyecto. La realidad demuestra que los clientes rara vez conocen sus propias necesidades con suficiente profundidad como para definir las a priori, y a esto se une que,

durante la vida del proyecto, las necesidades y prioridades de los clientes cambian. A menudo necesitan ver lo que las empresas de software son capaces de hacer y cómo resuelven éstas sus problemas a nivel técnico para poder descubrir nuevas necesidades u oportunidades que desconocían que eran capaces de implementar. Las organizaciones de software pueden protegerse de estos cambios estableciendo mecanismos de control, pero este enfoque puede concluir en clientes poco satisfechos, que verán el desarrollo del proyecto como algo rígido que no se adapta a sus necesidades y que si lo hace es repercutiendo en costos añadidos al presupuesto del proyecto (Corral, 2007).

Cuando un cliente requiere un nuevo producto software, en vez de seguirse los procedimientos que tiempo atrás seguían los gerentes de producto o analistas de negocio escribiendo documentos extensos de especificación de requerimientos de software, las metodologías ágiles comienzan con un enfoque más liviano en donde se escriben descripciones breves de lo que el sistema debe hacer. Estas descripciones son las Historias de Usuario. (Ambler & Holitza, Getting Started with Agile, 2012).

Las prácticas ágiles parten de la base de que ni los clientes ni los desarrolladores tienen conocimiento del producto al iniciar el proyecto. Los desarrolladores de software que adoptan metodologías ágiles entienden y aceptan que los requerimientos van a evolucionar a lo largo del proyecto. Por consiguiente, no escriben requerimientos correctos, no ambiguos, completos, consistentes, priorizados, verificables, modificables y mapeables a diferentes artefactos de trabajo. En cambio escriben requerimientos ágiles, expresados en alto nivel de detalle (Williams, 2004).

f. Los requerimientos tradicionales y las historias de usuario pueden coexistir

A pesar de que las metodologías ágiles fueron diseñadas para utilizar historias de usuario, el desarrollo de las aplicaciones está dirigido en su mayoría por requerimientos tradicionales (Varhol, 2012). Muchas de las organizaciones que terciarizan sus productos utilizan requerimientos como medio para definir y acordar lo que se debe construir. En estos casos, los equipos de desarrollo han tenido que encontrar el modo de trabajar con ambos a la vez. Las historias de usuario proveen flexibilidad en el diseño y en la implementación. Los requerimientos proveen precisión y facilidad de gestión (Varhol, 2012). Surge de este modo la siguiente pregunta ¿pueden coexistir ambos? La

respuesta es un enfático sí. Los equipos deben estructurar el proyecto con el objetivo de aprovechar al máximo las ventajas de cada uno. Una forma de hacerlo es utilizando los requerimientos como el acuerdo entre los clientes y/o usuarios y el equipo de desarrollo, ya que los requerimientos son estructurados, objetivos y pueden definir exactamente lo que se debe implementar; y las historias de usuario, ya que pueden construir el contexto que ayuda a los equipos de desarrollo a entender cómo se va a utilizar la aplicación. Las historias de usuario pueden administrarse en forma conjunta con los requerimientos sin necesidad de invertir demasiado esfuerzo adicional si ambos representan las mismas funcionalidades desde diferentes puntos de vista; el del uso: las historias de usuario y el sistémico: los requerimientos (Suscheck, 2012). Esto significa que, una historia de usuario podría estar relacionada con múltiples requerimientos y que un requerimiento podría a su vez, ser explicado por múltiples historias de usuario (Varhol, 2012). Los programadores utilizan las historias de usuario para diseñar e implementar las funcionalidades, y los requerimientos para agregar los detalles que sean necesarios. Los *testers*¹¹ escriben los casos de prueba que verifican los requerimientos, porque éstos últimos representan el acuerdo con el cliente, sin embargo, vuelven a las historias de usuario para comprender mejor cómo estructurar los casos de prueba y de este modo asegurarse que las funcionalidades más importantes han sido probadas en profundidad. Para que esto sea posible se deben relacionar de algún modo las historias de usuario y los requerimientos (trazabilidad) y asegurarse de que todos los requerimientos han sido mapeados a una o más historias de usuario. Se parte de un documento de requerimientos del producto que luego es descompuesto en funcionalidades, las cuales conocemos como historias de usuario (Langenfild, 2011).

¹¹ Un *tester* es aquel que se dedica a realizar pruebas a nuevas aplicaciones o a modificaciones de aplicaciones existentes.

5. Conclusiones

Una primera conclusión a la que se ha arribado durante el desarrollo de este trabajo es la importancia de saber diferenciar lo que es requerimiento de software de su especificación. El requerimiento expresa el problema que el sistema o producto software debe resolver. La especificación del requerimiento en cambio es el registro del requerimiento en una o más formas, a saber: Especificación de Requerimientos de Software (ERS), Casos de Uso, Escenarios, Historias de Usuario, etc.



Con el objetivo de definir los atributos y características deseables para la especificación de los requerimientos en metodologías ágiles, se revisaron los objetivos y las buenas prácticas de desarrollo de requerimientos según el modelo CMMI (Software Engineering Institute, 2010) y el estándar IEEE830 (Software Engineering Standards Committee of the IEEE Computer Society, 1998). Inicialmente se había planteado la hipótesis de que las historias de usuario podrían ser compatibles con los atributos del estándar de la IEEE, sin embargo se encontró bibliografía que dice exactamente lo opuesto. Por tal motivo se continuó la investigación y se encontró que existe un modelo llamado INVEST, que define las características para escribir las historias de usuario.

En metodologías ágiles, al igual que en otras metodologías tradicionales, existen mecanismos para obtener, analizar, especificar y verificar los requerimientos de un sistema. Los dueños del producto representan la voz del cliente dado que conocen sus necesidades y expectativas y son responsables de la gestión y priorización de los requerimientos. Los requerimientos pueden ser especificados en historias de usuario y validados al finalizar cada iteración. A diferencia de las metodologías tradicionales, en *Scrum* (Schwabe, 2004) no es preciso esperar a tener el producto definido completamente para poder comenzar con el desarrollo. Pueden escribirse las historias de usuario que definen las funcionalidades que van a desarrollarse en el transcurso de los próximos tres a seis meses.

Durante el desarrollo del trabajo también se ha visto que las historias de usuario constituyen un enfoque de requerimientos que hace énfasis en la comunicación y participación de los usuarios en el proceso de desarrollo. Los clientes rara vez conocen sus propias necesidades y durante el transcurso del proyecto esas necesidades y prioridades pueden cambiar. Las historias de usuario favorecen el trabajo en iteraciones lo cual facilita la adaptación a los cambios. También se ha visto

que expresar los requerimientos de software como historias de usuario puede ocasionar algunos problemas. El arte de escribir buenas historias de usuario resulta muy difícil para quienes recién comienzan a emplear metodologías ágiles, los errores cometidos en esta etapa pueden llevar a desarrollar casos de prueba equivocados, a mal interpretar los requerimientos o peor aún a desarrollar el producto incorrecto (Kaczor, 2011) (Kaczor, 2011). Es por ello que se han creado modelos como el INVEST (Leffingwell & Behrens, 2009). que ayudan a escribir buenas historias de usuario.

Aún cuando las historias de usuario pueden pensarse como un reemplazo de los documentos de requerimientos tradicionales, es preciso recordar que la parte escrita de las historias de usuario permanece incompleta hasta tanto se efectúen las discusiones con los dueños del producto. Durante el desarrollo del este trabajo se ha visto que hay autores como Mike Cohn que opinan que las historias de usuario no son obligaciones contractuales, y que por lo tanto no es necesario escribir los detalles conversados con los dueños del producto durante las reuniones de planificación. Las especificaciones del producto residen en lo que se ha implementado al finalizar cada iteración, en el código funcionando y en los casos de prueba. Pero también se ha visto que en algunos casos las historias de usuario se utilizan como punteros a requerimientos, típicamente en diagramas de flujos, hojas de cálculo donde se especifique cómo calcular un valor o cualquier otro artefacto que consideren el dueño del producto o los miembros del equipo (Cohn, Mountain Goat Software) (Cohn, Mountain Goat Software). Incluso en algunos casos coexisten los documentos de especificación de requerimientos y las historias de usuario.

 <p>ESPECIALIZACIÓN EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN</p>	<p>Carrera de Postgrado de Especialización en Ingeniería en Sistemas de Información</p> <hr/> <p><i>María Paula Izaurralde</i></p>	 <p>Universidad Tecnológica Nacional Facultad Regional Córdoba</p>
---	---	---

6. Trabajos Futuros

Este trabajo fue realizado en el marco del plan de tesis de maestría en Ingeniería en Sistemas de Información, “**Modelo Adaptable de Trazabilidad de Requerimientos de Software en Entornos Ágiles de gran escala**”. La investigación llevada a cabo en este trabajo permite concluir la primera etapa de investigación del plan de tesis, cuyo objetivo era analizar las distintas formas en que pueden encontrarse definidos los requerimientos en proyectos que operan utilizando metodologías ágiles y recopilar información acerca de las prácticas de gestión de requerimientos en modelos de calidad conocidos. Para poder avanzar con el plan de tesis se llevará a cabo una investigación acerca de la aplicación actual de la de trazabilidad de los requerimientos de software en proyectos ágiles de gran escala para luego proponer un modelo de trazabilidad de requerimientos, el cual será aplicado y validado en un caso real.

Bibliografía

- Ambler, S. W. (s.f.). *Agile Modeling Best Practices*. Recuperado el 12 de Febrero de 2013, de Agile Modeling: <http://www.agilemodeling.com/essays/bestPractices.htm>
- Ambler, S. W. (22 de Diciembre de 2009). *Agile Requirements at Scale*. Recuperado el 19 de Junio de 2012, de https://www.ibm.com/developerworks/mydeveloperworks/blogs/ambler/entry/agile_requirements_at_scale?lang=en
- Ambler, S. W. (2010). *Agile Requirements Modeling*. (Ambysoft) Recuperado el 19 de Junio de 2012, de <http://www.agilemodeling.com/essays/agileRequirements.htm>
- Ambler, S. W. (23 de Marzo de 2009). *Agile Scaling Factors*. Recuperado el 18 de Mayo de 2012, de developerWorks: https://www.ibm.com/developerworks/mydeveloperworks/blogs/ambler/entry/agile_scaling_factors?lang=en
- Ambler, S. W. (2010). *IBM agility@scale: Become as Agile as You Can Be*. New York: IBM Corporation.
- Ambler, S. W. (s.f.). *Introduction to Test Driven Development (TDD)*. (Agile Data) Recuperado el 9 de Junio de 2012, de <http://www.agiledata.org/essays/tdd.html>
- Ambler, S. W. (s.f.). *Introduction to User Stories*. Recuperado el 30 de Noviembre de 2012, de Agile Modeling: <http://www.agilemodeling.com/artifacts/userStory.htm>
- Ambler, S. W. (2009). *The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments*. United States of America: IBM Corporation.
- Ambler, S. W., & Holitza, M. (2012). *Agile for Dummies*. New Jersey: John Wiley & Sons, Inc.
- Ambler, S. W., & Holitza, M. (2012). Getting Started with Agile. En S. W. Ambler, & H. Matthew, *Agile for Dummies* (págs. 15-24). New Jersey: John Wiley & Sons, Inc.
- Ambler, S. W., & Holitza, M. (2012). Understanding Agile Roles. En S. W. Ambler, & M. Holitza, *Agile for Dummies* (págs. 11-14). New Jersey: John Wiley & Sons, Inc.
- Appleton, B., Cowham, R., & Berczuk, S. (18 de September de 2007). *Lean Traceability: a smattering of strategies and solutions*. Recuperado el 21 de Agosto de 2012, de <http://www.cmcrossroads.com/agile-scm/9089-lean-traceability-a-smattering-of-strategies-and-solutions>
- Beck, K., & al, e. (2001). *Manifiesto por el Desarrollo Ágil de Software*. (Agile Alliance) Recuperado el 8 de Agosto de 2012, de <http://agilemanifesto.org/iso/es/>
- Bolton, M. (14 de Marzo de 2008). *Requirement Traceability Matrix and Agile Testing*. Recuperado el 21 de Agosto de 2012, de <http://tech.groups.yahoo.com/group/agile-testing/message/13320?threaded=1&p=7>

Borland. (2005). Mitigating Risk with Effective Requirements Engineering: How to improve decision-making and opportunity through effective requirements engineering. *Part II in a series about understanding and managing risk*.

Bourse, L. (22 de Septiembre de 2009). *Who is a stakeholder?* Recuperado el 6 de Marzo de 2013, de Voices of Project Management: Independent ideas and insights by and for project practitioners: http://blogs.pmi.org/blog/voices_on_project_management/2009/09/who-is-a-stakeholder.html

Brackett, J. W. (1990). Software Requirements. *SEI Curriculum Module SEI-CM-19-1.2*. Boston University.

Canós, J., Letelier, P., & Penadés, M. C. (s.f.). *Métodologías Ágiles en el Desarrollo de Software*. Recuperado el 8 de Agosto de 2012, de <http://www.willydev.net/descargas/prev/ToDoAgil.Pdf>

Cohn, M. (2009). An Overview. En M. Cohn, *User Stories Applied for Agile Software Development* (págs. 3-15). Indiana: Addison - Wesley.

Cohn, M. (2009). Guidelines for Good Stories. En M. Cohn, *User Stories Applied for Agile Software Development* (págs. 75-83). Indiana: Addison Wesley.

Cohn, M. (s.f.). *Mountain Goat Software*. (Mountain Goat Software) Recuperado el 1 de Diciembre de 2012, de User Stories: <http://www.mountaingoatsoftware.com/topics/user-stories>

Cohn, M. (2009). *User Stories Applied: For Agile Software Development*. Indiana: Addison Wesley.

Cohn, M. (2009). Using Stories with Scrum. In M. Cohn, *User Stories Applied for Agile Software Development* (pp. 165-176). Indiana: Addison-Wesley.

Cohn, M. (2009). *What is Scrum?* Recuperado el 3 de Febrero de 2013, de Mountain Goat Software: <http://www.mountaingoatsoftware.com/topics/scrum>

Cohn, M. (2009). What stories are not. In M. Cohn, *User Stories Applied for Agile Software Development* (pp. 133-144). Indiana: Addison-Wesley.

Cohn, M. (2009). Why User Stories? In M. Cohn, *User Stories Applied for Agile Software Development* (pp. 145-155). Indiana: Addison-Wesley.

Corral, R. (12 de Noviembre de 2007). *Exprimiendo Scrum: Scrum y la gestión de requisitos*. (Los pensamientos, peleas y descubrimientos de Rodrigo Corral con Scrum, C++, C#, ASP.Net, Team System, Sql Server, Sharepoint, la arquitectura, la gestión de proyectos y el desarrollo de software en general...) Recuperado el 6 de Junio de 2012, de <http://geeks.ms/blogs/rcorral/archive/2007/11/12/exprimiendo-scrum-scrum-y-la-gesti-243-n-de-requisitos.aspx>

Craig. (15 de Enero de 2008). *Requirements Traceability: Project Leadership, Requirements Management and Product Design*. Recuperado el 9 de Junio de 2012, de <http://www.betterprojects.net/2008/01/requirements-traceability.html>

Figuerola, N. (Febrero de 2012). *Definiendo Requerimientos: Tradicional vs. Caso de Uso vs. Historias de Usuario*. Recuperado el 16 de Febrero de 2013, de <http://articulosit.files.wordpress.com/2012/04/requerimientos.pdf>

Firesmith, D. G. (s.f.). *Use Cases: the Pros and Cons*. Recuperado el 14 de Diciembre de 2012, de Knowledge System Corporation: <http://www.ksc.com/articles/usecases.htm>

Fritzsche, M., & Keil, P. (2007). Agile Methods and CMMI: Compatibility or Conflict? *e-Infomatica Software Engineering Journal*, 1 (1).

Gomez, D. (16 de Junio de 2009). *Qué significa puntos de historia*. Recuperado el 24 de Enero de 2013, de DosIdeas: <http://www.dosideas.com/noticias/metodologias/624-que-significa-puntos-de-historia.html>

Gotel, O. C., & C.W, F. A. (1994). *An Analysis of the Requirements Traceability Problem*. London: Imperial College of Science, Technology & Medicine.

Hazrati, V. (6 de Junio de 2008). *Traceability Matrix in an Agile Project*. Recuperado el 21 de Agosto de 2012, de <http://www.infoq.com/news/2008/06/agile-traceability-matrix>

Ibañez, J. (s.f.). *Gestión de requerimientos IV: trazabilidad*. Recuperado el 21 de Agosto de 2012, de <http://blogs.salleurl.edu/project-management/gestion-de-requerimientos-iv-trazabilidad/>

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley.

Kaczor, K. (3 de Agosto de 2011). *5 Common Mistakes We Make Writing User Stories*. Recuperado el 1 de Diciembre de 2012, de Scrum Alliance: <http://www.scrumalliance.org/articles/366--common-mistakes-we-make-writing-user-stories>

Langenfeld, C. (8 de Julio de 2011). *Using Rally to Map High Traceability User Stories: PRD to SRS*. Recuperado el 20 de Enero de 2013, de Rally Blogs: <http://www.rallydev.com/community/agile-blog/using-rally-map-high-traceability-user-stories-prd-srs>

Leffingwell, D., & Behrens, P. (2009). *A User Story Primer*. Recuperado el 24 de Enero de 2013, de <http://trailridgeconsulting.com/files/user-story-primer.pdf>

Moccia, J. (27 de Enero de 2012). *Agile Requirements Definition and Management (RDM)*. Recuperado el 11 de Febrero de 2013, de <http://onespring.net/blog/agile-requirements-definition-and-management-rdm/>



Moccia, J. (10 de Febrero de 2012). *Agile Requirements Definition and Management*. Recuperado el 11 de Febrero de 2013, de Scrum Alliance: Transforming the world of work: <http://scrumalliance.org/articles/398-agile-requirements-definition-and-management>

Mountain Goat Software - Imágenes. (2005). *Mountain Goat Software - Imágenes*. (Mountain Goat Software) Recuperado el 22 de Agosto de 2012, de <http://www.mountaingoatsoftware.com/system/asset/file/18/ScrumLargeNoLabels.png>

North, D. (2 de Enero de 2009). *Behaviour-Driven Development*. Recuperado el 9 de Junio de 2012, de <http://behaviour-driven.org/>

Nuseibeh, B., & Easterbrook, S. *Requirements Engineering: A Roadmap*. London: Department of Computing.

- Peterson, D. (2009). *What is Kanban?* Recuperado el 11 de Febrero de 2013, de Kanban Blog: <http://www.kanbanblog.com/explained/index.html>
- Poppendiek, M., & Poppendiek, T. (2003). *Lean Software Development: An Agile Toolkit*. New Jersey: Addison Wesley.
- Product Arts. (2009). *Agile Requirements - So What's Different?* Recuperado el 9 de Junio de 2012, de <http://www.product-arts.com/articlelink/204-agile-requirements-so-whats-different>
- Products Arts. (2009). *A Tour of Requirements Documents*. Recuperado el 9 de Junio de 2012, de <http://www.product-arts.com/articlelink/138-a-tour-of-requirements-documents>
- Pugh, K. (2010). *Lean-Agile Acceptance Test-Driven Development*. Boston: Addison-Wesley.
- Ramesh, B., & Jarke, M. (2001). Toward Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering*, 27 (1), 58-93.
- Sandoval Carvajal, M. M., & García Vargas, M. A. (2008). *La Trazabilidad en el proceso de requerimientos de software*. Recuperado el 21 de Agosto de 2012, de <http://www.iiis.org/CDs2008/CD2008CSC/CISCI2008/PapersPdf/C601UZ.pdf>
- Schwabe, K. (2004). *Agile Project Management with Scrum*. Microsoft Press.
- Schwaber, K. (2001). *Scrum Practices*. NJ: Prentice Hall.
- Software Engineering Institute. (2010). Appendix D: Glossary. En *CMMI for Development v1.3* (págs. 433-468). CarnegieMellon.
- Software Engineering Institute. (2010). *CMMI for Development, Version 1.3*. Massachusetts: CMMI Product Team.
- Software Engineering Institute. (2010). Requirements Development. In S. E. Institute, *CMMI for Development, Version 1.3* (pp. 325-340). Pennsylvania: CMMI Product Team.
- Software Engineering Institute. (2010). Requirements Development. In *CMMI for Development, Version 1.3*. Pennsylvania: CMMI Product Team.
- Software Engineering Institute. (2010). Requirements Management. En *CMMI for Development, Version 1.3* (págs. 341-347). Pennsylvania: CMMI Product Team.
- Software Engineering Standards Committee of the IEEE Computer Society. (25 de Agosto de 1998). *IEEE Recommended Practice for Software Requirements Specifications*. Recuperado el 6 de Diciembre de 2012, de IEEE Std 830-1998: <http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf>
- Standards Coordinating Committee of the Computer Society of the IEEE. (1998). *IEEE Recommended Practices for Software Requirements*. New York.
- Standards Coordinating Committee of the Computer Society of the IEEE. (1990). *IEEE Standard Glossary of Software Engineering Terminology*. New York: IEEE.
- Suscheck, C. (17 de Enero de 2012). *Defining Requirement Types: Traditional vs. Use Cases vs. User Stories*. Recuperado el 20 de Enero de 2013, de Agile Techwell:

 ESPECIALIZACIÓN EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN	Carrera de Postgrado de Especialización en Ingeniería en Sistemas de Información <i>María Paula Izaurralde</i>	 Universidad Tecnológica Nacional Facultad Regional Córdoba
--	--	--

<http://agile.techwell.com/articles/weekly/defining-requirement-types-traditional-vs-use-cases-vs-user-stories>

Tabares, M. S., Barrera, A. F., & Arroyave, J. D. (Diciembre de 2007). Un método para la trazabilidad de requisitos en el proceso unificado de desarrollo. *Revista EIA: Escuela de Ingeniería de Antioquia*, págs. 69-82.

Tuffley, A. (2005). CIT3190 IT Project Course. *Requirements Elicitation and Management*.

Varhol, P. (2012). Conference Paper: Agility with Traceability: Blending Requirements and User Stories. *Quality Engineered Software & Testing Conference & Expo*. Chicago.

Well, D. (28 de Septiembre de 2009). *Extreme Programming: A gentle introduction*. Recuperado el 11 de Febrero de 2013, de Extreme Programming: <http://www.extremeprogramming.org/>

Wells, D. (1999). *User Stories*. Recuperado el 14 de Enero de 2013, de Extreme Programming: <http://www.extremeprogramming.org/rules/userstories.html>

Wieggers, E. K. (1999). Software Requirements Management. En *Software Requirements* (pág. 350). Washington: Microsoft Press.

Williams, L. (2004). *Agile Requirements Elicitation*. Recuperado el 20 de Enero de 2013, de <http://agile.csc.ncsu.edu/SEMaterials/AgileRE.pdf>