LA GUÍA DE HISTORIAS DE USUARIO

JAVIERGARZAS.COM

233 grados de TI


HISTORIAS DE USUARIO
© JAVIER GARZAS @JGARZAS
FUNCIONALIDAD RESUMIDA EN UNA TARJETA
FUNCIONALIDAD LO MÁS PEQUEÑA!
SI BIEN LA PARTE MÁS IMPORTANTE ES LA NO ESCRITA... LA CONVERSACIÓN A LA QUE LLEVAN
PERO, RECUERDA, LA CLAVE ESTÁ EN EL VALOR QUE APORTA
OH YEAH
233 grados de TI

Esta guía ha sido posible gracias al apoyo de los amigos de…

www.r2docuo.com/es/

wwww.velneo.es

v 1.0 - Feb 2021

He creado esta guía para ayudar a la comunidad

Todos deberíamos dedicar tiempo a causas sociales, a ayudar con aquello que podemos aportar y que otros necesitan. Si puedes, hazlo. En nuestro caso, en mi caso, algo muy destacado que puedo y debo aportar a la comunidad es conocimiento y experiencia.

Por ello, como llevo haciendo desde hace muchos años, he querido dejaros, totalmente gratis, una guía más, en este caso 38 páginas en abierto hablando sobre Historias de Usuario.

Y que este documento se una a tantos otros pdf, cursos online gratis (como el de Mirada o el de 233 Academy) o videos (como el más de un centenar del canal de YouTube) que donarlos libremente me ha parecido siempre que era una responsabilidad que tenía y tengo con la comunidad Ágil.

Puedes complementar esta guía con... el curso GRATIS: "Una introducción a la Agilidad para todos los públicos y edades"

Inscríbete aquí

Cómo me puedes ayudar

Si me quieres ayudar a seguir creando contenido:

- Comparte esta guía, así me ayudas a luchar contra el Lado Oscuro y disminuir las malas prácticas que hay por el mundo.
- Suscribete al canal de YouTube (aquí), te va a llevar dos segundos, es gratis y me ayudas a difundir más el conocimiento.
- También te puedes suscribir a la lista de correo de mi blog, para estar al tanto de nuevas cosas, noticias, etc.

Sobre el autor, Javier Garzás

Cursé estudios postdoctorales y fui investigador invitado en la Universidad Carnegie Mellon (Pittsburgh, EE. UU), investigando en Agilidad. Soy doctor, Ph.D., en informática, calificado cum laude por unanimidad e Ingeniero en Informática (premio extraordinario).

He sido programador, jefe de proyectos, consultor, director de informática (CIO), diseñador de muchos UMLs, etc., he pasado por casi todas las dedicaciones del desarrollo software.

Pionero en España en agilidad, mi primer proyecto ágil en empresa fue en 2001 (con eXtreme Programming).

Desde hace años, me dedico a ayudar a organizaciones, a que trabajen mejor (en tiempo, con productividad, agilidad, calidad y felicidad).

Hasta la fecha, a lo largo de mi carrera profesional he trabajado para, o en, más de 100 empresas (en España, Colombia, Chile, Venezuela y EEUU) entre las que están algunas como:

- Ingeniería – Aeronáutica – Automoción – Software Crítico: GMV – División de Control de Satélites, FICOSA, CESA (Sistemas Aeronáuticos).
- Retail: Inditex, Pull and Bear, Leroy Merlin
- Constructoras: Ferrovial.
- Aseguradoras y Sector Financiero: Ocaso, Banco Santander, BBVA, Aegon, Produban, Cajasur, Mapfre, ING.
- Operadoras: Telefónica, Vodafone
- Información y Comunicaciones: El Mundo, Wolters Kluwer.
- Tecnologías de la Información: Velneo, Vintegris, Indra, Freepik.
- ERPs: DANTIA, VisualTrans

- Consultoras: Accenture, Altran Innovación S.L, Viewnext, Ibermática, Bilbomática, Dantia.
- Sector Energético: PVH Storage.
- Administración Pública: Dirección General de Tráfico, Ministerio de Administraciones Públicas (MAP), Informática de la Comunidad de Madrid (ICM), Sistemas Técnicos de Loterías (STL), Agencia Tributaria (AEAT), LANTIK, Informática de la Seguridad Social, Catastro, Informática del Gobierno Vasco (EJIE).
- Otros Organismos Públicos: ONCE
- Alimentación: DIA, S.A.
- Deporte: F. C. Barcelona, Federación Española de Fútbol.
- Transporte: Pullmantur.
- Certificadoras: AENOR, EQA.

Además, soy profesor de la Universidad Rey Juan Carlos.

En lo que refiere a certificaciones, muchas, algunas de ellas: Certified Management 3.0 Practitioner (y co-owner de la propia Management 3.0), Scrum Master certificado por Jeff Sutherland (co-creador de Scrum), Advanced Agile certificado por Alistair Cockburn (uno de los padres de la agilidad), Certificado DAD (Disciplined Agile delivery) por Scott Ambler (creador del modelo), etc.

Te dejo mis redes para estar en contacto:

- javier.garzas@233gradosdeti.com
- twitter: @jgarzas
- web: www.javiergarzas.com, blog en el que escribo desde hace más de 10 años
- Instagram: https://www.instagram.com/javiergarzas/
- YouTube: http://youtube.com/c/JavierGarzas
- LinkedIn: es.linkedin.com/in/jgarzas

Libros relacionados

Si no estás familiarizado con Scrum, o con la Agilidad en general, te recomiendo leer antes estos libros, podrás encontrarlos en:

http://233gradosdeti.com/publicaciones-233/

23 Historias de Equipos Ágiles.

Peopleware y Equipos Ágiles.

Gestión de *proyectos* Ágil…. y las experiencias de más de 12 años de proyectos ágiles.

Cómo sobrevivir… A la planificación de un *proyecto* ágil.

Índice

1. MÓDULO 1: ENTENDIENDO QUÉ ES UNA HISTORIA DE USUARIO

Una historia de usuario describe funcionalidad que será útil para el usuario[1] (y esta primera definición la iremos matizando). Y aunque normalmente las historias de usuario suelen escribirse en post-it o tarjetas, son mucho más que eso.

Ron Jeffries comentaba[2] que una historia no es solo una descripción de una funcionalidad, normalmente en un post-it, una historia de usuario además está formada por otras dos partes más:

- La conversación que conllevan, ya que son una herramienta para interactuar.
- El cómo se confirma su implementación, las pruebas y verificación de la misma.

Las historias de usuario, frente a mostrar un «cómo», solo dicen el «qué». Es decir, muestran funcionalidad que será incorporada al producto, pero no cómo se incorporará. Por ello, las cuestiones técnicas no son historias de usuario (estas son lo que llamamos tareas).

Una historia de usuario describe funcionalidad, describe el valor que suponemos (lo de suponemos te lo explicaré cuando hablemos del Hypothesis-Based Development) que algo aportará al usuario una vez que esté y sea operativo en nuestro producto y, sobre todo, debe fomentar la interacción.

El formato típico (no necesariamente obligatorio) de una historia de usuario es:

> **Como** *<rol, actor, persona o sistema externo que interactúa con la funcionalidad que queremos crear y añadir al producto>* **quiero** *<funcionalidad>* **para** *<beneficio qué me va a aportar que esa funcionalidad esté en el producto>*

Dicho de otra manera, para complementar lo anterior, tenemos tres partes, la primera es QUIÉN, la segunda es QUÉ y la tercera el PARA QUÉ.

[1] https://cutt.ly/gkEBoH2
[2] http://xprogramming.com/articles/expcardconversationconfirmation/

Ejemplos:

- Como usuario quiero poder añadir mi fecha de nacimiento al formulario de alta para poder recibir promociones el día de mi cumpleaños.
- Como posible comprador quiero leer revisiones de los productos para decidir si los compro.
- Como turista quiero ver ofertas de coches de alquiler para reservar la oferta más ventajosa.
- Como pasajero frecuente quiero poder introducir mi número de cliente para ver las millas que tengo y poder hacer uso de descuentos.
- Como editor quiero poder comentar un post para poder aportar mi punto de vista sobre el mismo

1.1. No importa tanto el formato, importa la conversación a la que llevan

La mayoría de malos usos que hoy te puedes encontrar con historias de usuario suelen venir de olvidar, como te contaba antes, que su objetivo no es dejar necesidades por escrito, sino fomentar la interacción, colaboración, hablar, etc.

Otro foco típico de críticas viene del riesgo de olvidar centrarse en el verdadero beneficio, valor, que aportará la historia de usuario. Esto lo puedes ver muy claro cuando la gente deja fuera de la historia (esto es muy típico) el «para / *so that*», como si fuera

opcional, lo que deja fuera la razón, la ventaja que el usuario obtendrá cuando se añada esa nueva funcionalidad.

1.2. Historias vs Tareas

Supongo que con poco que estés metido en agilidad, en Scrum, etc., tendrás claras las principales diferencias entre historias de usuario y tareas.

Típicamente, vamos con el resumen, las historias son un «qué» queremos, las tareas un «cómo», las historias son funcionalidad que aporta valor al usuario, las tareas suelen ser aspectos técnicos, las historias están en el Product Backlog, son responsabilidad del Product Owner, las tareas están en el Sprint Backlog, y son responsabilidad del equipo, etc.

Bueno, y resumido lo anterior, voy con una diferencia entre historias y tareas, que no he nombrado y que más que una diferencia es una buena práctica poco conocida: una historia es algo en lo que por lo general… trabaja más de una persona. Y una tarea es algo en lo que por lo general… trabaja una sola persona.

Una historia de usuario implicará a varios perfiles, a varias personas: desarrolladores, testers, diseñadores, etc. Y para organizar ese «cómo» hacer la historia aparecen las tareas. Y en las tareas, lo normal, es que haya una relación 1 a 1 entre tarea y persona que la hace.

1.3. Una historia de usuario no es un requisito software

Con mucha frecuencia se equiparan las historias de usuario con los viejos requisitos. Pero por definición, las historias de usuario no suelen, ni deben, tener el nivel de detalle que tiene la especificación de un requisito.

Si una historia de usuario tiene demasiada información, o incluso ocupa varios folios… no es una historia de usuario. Por ello se recomienda que las historias de usuario se escriban en un espacio reducido, y que el soporte físico, el post-it o la tarjeta, tenga apenas una frase.

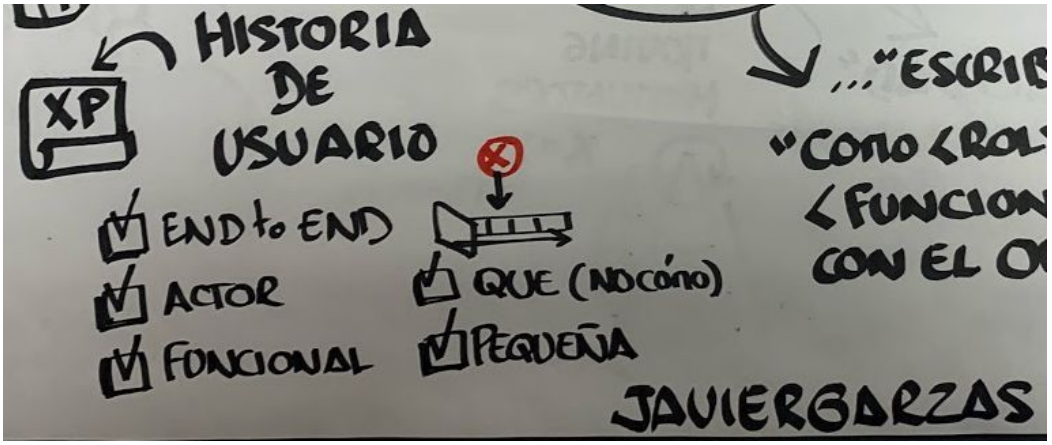
Una historia de usuario debe ser pequeña, memorizable, y que pudiera ser desarrollada en pocos días. Pero claro, al tener una única frase, es imposible que una historia de usuario contenga toda la información necesaria para desarrollar. En tan reducido espacio no pueden contener el diseño, las pruebas, normativa, etc. Ni tampoco detalles para codificar.

Para resolver el anterior problema hay que entender que las historias de usuario son lo que son porque su objetivo es, entre otros, lograr la interacción con el equipo y con el usuario, por encima de documentar (Manifiesto Ágil).

1.4. Algunas pistas para saber es, o no, una historia de usuario

Hay un tema cuanto menos inquietante. El tema de… cómo saber si eso que estais llamando historia de usuario… realmente lo es.

Me voy a saltar el *rollo* teórico y voy con unas cuantas cosas que si no se cumplen es raro que ese ítem sea una historia de usuario, por mucho que lo llamemos así y así lo etiquetemos en Jira.



Si ese ítem es historia de usuario…

- Su desarrollo, su incorporación al incremento, es «end to end», es decir, no supone la creación de un trozo de *frontend*, o de *backend*, o de lo que sea, es un

desarrollo de todo lo necesario para que la historia, una vez desarrollada, sea usable y esté testeada.

- Tiene un actor, una persona o sistema externo, que interactuará con ella, una vez incorporada al incremento.
- Es una funcionalidad, no es un «aumentar el tiempo de respuesta», o un «mejorar la seguridad».
- Cuenta un qué se espera, negocio, no un cómo hacerlo.
- Es pequeña, en lo que refiere a que se espera terminarla en pocos días. Aunque también, ya que estamos, son pequeñas en su descripción, apenas unas pocas líneas en *pro* de conversar sobre ella.

1.5. Historias de Usuario sospechosas: Como Product Owner quiero…

Las supuestas historias de usuario, las historias de usuario *sospechosas*, son un coladero para mantener el «Cascada Lifestyle» (estilo de vida cascada) vistiéndolo de Ágil.

Como *disclaimer* antes de que sigamos, puedes, por supuesto, no hacer caso a nada de lo que te cuente ahora y tener un modelo de trabajo súper potente, feliz, y que no es Ágil (aunque es difícil en estos tiempos). Pero, en cualquier caso, lo que yo te voy a contar son cosas que suenan muy raras, muy raras, muy dudosamente ágiles, cosas como…

La historia de usuario me mira. Yo la miro a ella. Jugamos con las miradas. Ahora sé que me está engañando…

El juego que habrá dado, y seguirá dando. El famoso formato «Como *<rol>* quiero *<funcionalidad>* para *<cuál es el beneficio que obtengo con ello>*», cuyo objetivo inicial era forzarnos a que las historias tuvieran un *<rol>* a quien ofrecer su funcionalidad (evitando caer en colar Tareas por Historias) y que pensáramos al hacerlas en el beneficio que obtenía ese rol con esa historia, pensar en el valor, en ese para *<cuál es el beneficio que obtengo con ello>*.

Hay muchas retorcidas y canallas maneras de retorcer el «Como *<rol>* quiero…», pero entre todas ellas destaca el… *«Como Product Owner quiero…»*

¿Cómo que como Product Owner quiero? Pero vamos a ver, si una historia es un incremento de valor a un producto (ojo, en un desarrollo software o en otra índole) será un futuro usuario (o algún rol más específico que encaje dentro del rol de usuario) de la historia el que quiera esa funcionalidad… no tú. Es que hay que pensar en el incremento de valor funcional (que una historia es funcional) al usuario, no en el «yo quiero» ¿Cómo que yo quiero?

Yo reflexionaría sobre los formatos, porque nos gustan mucho los formatos, así no tenemos que pensar y ya tenemos una regla que seguir y que piensa por nosotros. La idea de las historias era un muy pequeño aporte de valor funcional (no una tarea cómo hacer Testing u otra), y como es aporte de valor es… a un usuario del producto (sea cual sea el producto).

Ya estoy viendo a más de uno, después de leer esta guía, reemplazando automáticamente en Jira (Buscar – Reemplazar) los «Como Product Owner quiero» por «Como Usuario quiero», ay, que duro es esto…

1.6. Historias de usuario y Pruebas de Aceptación

Los Criterios de Aceptación, o Condiciones de Satisfacción, de una historia de usuario (o un ítem de manera general) normalmente refieren a escenarios de prueba para comprobar la historia de usuario una vez implementada. Hablan de su comportamiento esperado una vez implementada. De hecho, los criterios de aceptación complementan la información, funcional, de la historia de usuario.

A los criterios de aceptación hay quien los llama pruebas de aceptación, y muchas veces se utilizan como sinónimos, aunque, teorizando mucho hay quien te puede decir que no son lo mismo, pero en el día a día suelen ser sinónimos.

Los criterios de aceptación, o las pruebas de aceptación, por ejemplo, refieren más a escenarios de Test y, típicamente, los puedes ver escritos en Gherkin[3], bajo la llamada «especificación con ejemplos»[4].

Si nos metemos en terminología, y para no enfadar a los académicos de la «Real Academia del Agile», también hay quien diferencia a las pruebas de aceptación de los criterios de aceptación llamando a estos últimos, Definition of Done…

1.7. Historias de usuario y Definition of Done (DoD)

Aunque suene *raruno*, la traducción de DoD sería algo así como «Definición de Terminado». Y el DoD vendría a ser una lista de condiciones que debe cumplir una historia de usuario, una vez implementada, para que el equipo la considere terminada.

¿Qué cosas suelen ser condiciones en un DoD? Pues cada uno debiera buscar las que mejor le vengan, pero aquí, si miro, por ejemplo, en estas historias de usuario que tengo delante veo cosas como, cobertura de pruebas unitarias, si a habido «pair review», umbrales de métricas (en este caso en Sonar), etc.

1.8. ¿Diferencias entre el Definition of Done y los Criterios de Aceptación?

Dicho lo anterior, deberían estar claros, no obstante convendría resaltar que, típicamente, los criterios de aceptación son específicos para cada historia de usuario, mientras que el DoD puede ser común para varias historias, ya que refiere al incremento, a cómo queda el producto una vez que esas historias le son incorporadas. Aunque esto no es una regla inamovible, ya que depende del producto, o productos, con los que trabaje el equipo.

Por continuar con las aclaraciones, los antiguos del lugar pueden explicarte que el DoD viene a ser los viejos, ojo, que suelto la palabra, requisitos, *ejem*, no funcionales, mientras que los criterios de aceptación vienen a ser las pruebas funcionales de aceptación.

[3] https://www.javiergarzas.com/2014/12/bdd-behavior-driven-development-1.html
[4] https://www.javiergarzas.com/2015/08/atdd-bdd-como-aclarando-el-lio-de-siglas-en-testing.html

En cualquier caso, ambos, tanto el DoD como los criterios de aceptación deben estar listos para considerar que una historia de usuario está terminada.

Te recomendamos que visites este vídeo en YouTube sobre DoD vs Criterios de Aceptación.

Haz click aquí

Tres términos relacionados, que hay quien usa como sinónimo: criterios de aceptación, pruebas de aceptación y Definition of Done.

Yo no me complico la vida, uso pruebas de aceptación (como escenarios funcionales idealmente en Gherkin) y DoD, como condiciones de carácter más técnico y no funcionales.

Pero, eso sí, es importante resaltar el uso de esas dos especificaciones, de manera separada, para dejar bien clara una historia de usuario y, ojo, haciéndolo con cuidado para no caer en *Lado Oscuro* y acabar escribiendo *sábanas* de texto que acompañan a una historia de usuario (interacción por encima de documentación).

1.9. NO asocies Historia igual a funcionalidad operativa liberable al usuario

No lo asocies, no siempre, porque puede ser que SÍ, pero puede ser que NO.

Si asocias siempre que historia de usuario es igual a funcionalidad operativa liberable al usuario vas a caer en el problema de hacer historias muy grandes, que ya te contaré luego que traen muchos problemas.

Para que se entienda bien, partimos de la base de que no todo es historia de usuario, para llamarse incluso como tal, tiene que ser pequeña, «end to end», suponer un incremento funcional en el producto, terminar en «working software (o solutions)», etc.

Toda historia de usuario, una vez terminada, debería acabar en el producto, como una nueva funcionalidad operativa, PERO, no obligatoriamente liberable, es decir, contándolo

con otra terminología, toda Historia de Usuario debe terminar en PRODUCCIÓN o… en PRE-PRODUCCIÓN.

Ojalá todas terminaran en producción, es decir, se liberaran al usuario, que podría disfrutar pronto de ellas, nos daría información para ajustar próximos incrementos, etc. Pero en el mundo real, complicado mundo real, esto no siempre es posible.

Ejemplo tonto, queremos hacer un formulario de inscripción a algo, una historia de usuario trata sobre un campo para registrar tus datos, otra sobre un botón de confirmación de los anteriores. Siendo (por las circunstancias del ejemplo) dos Historias separadas, hasta que el producto, el incremento, no tenga las dos no tiene sentido liberar la nueva funcionalidad que aportan de manera conjunta.

Si las dos entrasen en el mismo Sprint, quizás sí, pero si por alguna razón una no entra, la otra, la que se termina antes no tiene sentido liberarla en producción, si en pre-producción.

¿Obvio, no? Pues no debe serlo.

¿Problema, frecuente, de no entender lo anterior? Hacer una historia grande que contenga la funcionalidad de registro y el botón de confirmación.

En este ejemplo (pensado para que se entienda) podría pensarse que total es funcionalidad simple y podría hacerse en pocos días, pero si no es así, y la historia (que no sé si llamarla historia) grande nos lleva muchos días… hemos hecho mal, mala práctica, por no entender esto.

Por esta razón (y hay otras tantas), me encuentro muchos equipos que no entienden esto y hacen historias de usuario gigantes.

1.10. Los peligros de «escribir» Historias de Usuario

Las palabras son peligrosas, muy peligrosas, muchas veces revelan lo que pide nuestro subconsciente luchando contra nuestra consciencia (e incluso luchando contra la moda y la opinión pública).

Al igual que «escalar agilidad» es peligroso, porque induce a «añadir», cuando la mayoría de las veces «hay que quitar», más correcto sería «des-escalar agilidad», y ello lleva a muchos a meter y meter procesos y cosas que poco ágiles son (no voy a entrar en algunos frameworks de escalado que *ejem*…), «escribir historias de usuario» incita a eso… a escribir.

Y si me pongo a «escribir» mucho texto en una historia de usuario, que te quede claro… ¡ya no es una historia de usuario!, es un requisito camuflado de los de toda la vida, y te has cargado una de las principales esencias de la historia: reducir documentación escrita por interacción y hablar.

Claro, todo aquel al que he escuchado (y te aseguro que son much@s) lo de «hay que escribir mejores historias de usuario» porque «tenemos problemas al implementarlas / de entendernos con los Stakeholders / X» comienza, si no lo remedia (o lo remedio a tiempo yo) un camino hacia el *Lado Oscuro*.

1.11. Complementando la descripción de las historias con Gherkin

Las historias de usuario,o *features* (en el mundo BDD puedes encontrar las historias de usuario bajo el nombre de features, hay quien dice que no son exactamente lo mismo), lógicamente, las escribe negocio, es responsabilidad del Product Owner.

En BDD (Behavior-Driven Development), lo que haremos será, con el objetivo de automatizar el testing, aparte de tener historias en post-it, como ha sido de toda la vida, cada historia de usuario se escribirá usando un lenguaje llamado Gherkin (la traducción es pepinillo).

Gherkin es un lenguaje entendible por «negocio», lo que se llama un DSL (Domain-Specific Language), más o menos cercano al lenguaje natural. Está pensado para que lo entienda «negocio», los usuarios, etc. Su idea es contar cómo se «comporta» el software sin entrar en detalles de implementación.

Es un lenguaje muy simple. Sólo tienen 5 sentencias:

- Feature
- Scenario

- Given
- When
- Then

Así que Gherkin[5] nos sirve para documentar y para automatizar lo que luego se convertirá en test, haciéndolo con un lenguaje que puede entender negocio, o cualquier Product Owner, y que además puedes usar en otros idiomas más allá del inglés.

Muchas veces, esos escenarios serán el paso intermedio entre la historia de usuario y la implementación de una prueba automatizada.

[5] Gherkin da para mucho más, yo no quería extender este punto y solo te lo he mencionado, pero te recomiendo que profundices en este punto. Puedes descargar la Guia de Gherkin aqui.

2. MÓDULO 2: DIVIDIENDO HISTORIAS DE USUARIO

2.1. ¿Por qué las Historias de Usuario deben ser pequeñas?

Por ponernos en contexto, un poco por refrescar cosas, el ciclo de vida ágil es un tipo especial del veterano ciclo de vida iterativo e incremental[6]. Entre otras matizaciones, un ciclo de vida iterativo e incremental podríamos decir que es ágil si la parte incremental se lleva al extremo, en lo que refiere a cortos periodos de tiempo, entrega de valor (o valor listo para ser entregado) y funcionalidad a usuarios, cada pocos días o incluso horas.

El ideal es la entrega continua de funcionalidad, valor, al usuario. Y si la naturaleza de tu desarrollo no posibilita esas puestas en producción frecuentes, al menos, valor o funcionalidad en pre-producción o en condiciones de ser usado[7].

Lo de «entrega continua» es una de las muchas diferencias de un ciclo de vida ágil frente al cascada, en el cascada se entregaba después de mucho tiempo, en el ágil se entrega poquito muy frecuentemente (y lo que se entrega es útil, típicamente funcional, para el usuario).

Lo más normal es que para ir realizando esos incrementos de valor, o funcionalidad a los usuarios, se haga uso de las historias de usuario, que representan funcionalidad pequeña. Pero sí que quiero destacar dos cosas: manteniendo las historias de usuario pequeñas y funcionales (lo que en inglés se conoce como “end to end”).

¿Y por qué tanta obsesión en que las historias de usuario sean pequeñas? Y añado, pequeñas a la vez que funcionales, de hecho, si las historias no son funcionales… no son historias.

[6] https://www.javiergarzas.com/2012/10/iterativo-e-incremental.html
[7] https://www.javiergarzas.com/2016/11/sprint-termina-paso-produccion.html

Te voy a dejar 4 razones, entre muchas otras, que me parecen fundamentales para que te obsesiones con hacer historias de usuario pequeñas (y funcionales).

2.1.1 «Feedback» frecuente con usuario

Cuanto antes vea el usuario lo que estamos haciendo antes nos aseguramos de que es eso lo que quiere, y si no lo es, cambiaremos cosas pequeñas no cosas grandes difíciles de cambiar, y de las que incluso ya ni nos acordamos.

Así nos será más fácil acertar con qué quiere de verdad el usuario, qué le aporta valor.

Podía haberlo puesto en un punto más, pero como está muy relacionado con este lo dejo aquí… historias de usuario pequeñas nos dan más capacidad de cambio, de adaptarnos a las necesidades. Y nos dan más opciones para ir dando lo más esencial al usuario cuanto antes.

2.1.2 Saber de VERDAD dónde estamos

Aquí hablo del tradicional problema del seguimiento del «proyecto». Antiguamente, se hacían cosas muy raras como porcentajes de avance en función del tiempo transcurrido, etc., que tenían fiabilidad escasa. Realmente sabemos que algo está terminado cuando… lo está.

Historias de usuario pequeñas me muestran el avance REAL en cortos periodos de tiempo, sacándome del *lado oscuro* de ver avances en función de horas.

2.1.3 Integraciones menos complicadas

Todos hemos sufrido lo que es una integración, hay «proyectos» en los que la integración ha durado más que el desarrollo. Tú, o un equipo, hace una cosa y yo, u otro equipo, hace otra y una cosa depende de la otra y en algún momento hay que juntarlas. Y todos sabemos que cuánto más tiempo pasa más difícil es unir.

Historias de usuario pequeñas llevan a pocos días para su realización y llevan a integraciones más frecuentes y por ello menos desastrosas.

2.1.4 Testeo frecuente

Muy en línea con lo anterior, dejar el Testing para el final, para dentro de mucho no es buena cosa. Si tengo historias de usuario pequeñas, testearé muy frecuentemente, y lo que el Testing encuentre aún será posible de resolver, pero si Testeo en real (o pre-producción) cada mucho tiempo, encontraré cosas que será muy difícil de resolver.

2.2. Estrategias para descomponer y hacer pequeñas historias de usuario

Un clásico, cuando alguien empieza a usar historias de usuario, sobre todo si viene de usar grandes requisitos, o si ni siquiera usaba requisitos, enfrentarse a que la historia de usuario debe ser lo más pequeña posible y aportar valor al usuario...llegado ese momento a mucha gente le cuesta dividir, piensa que es imposible, que «su caso es especial, que son diferentes»[8].

Te recomendamos que visites el siguiente video, sobre cómo descomponer Historias de Usuario:

Haz click aquí

Si estás en una de esas, por si te vale y ayuda, te voy a dejar algunos patrones, estrategias, para dividir al máximo historias de usuario.

[8] https://www.javiergarzas.com/2017/02/caso-especial-somos-diferentes.html

2.2.1 Estrategia 1: Descomposición por flujo de trabajo

Si las historias de usuario conllevan un flujo de trabajo, una estrategia es detectar los pasos individuales y que cada paso (que aporte valor al usuario) sea una historia. Por ejemplo, lo siguiente, que perfectamente puede ser una Epic:

> Como X (el rol que sea) quiero comprar M (lo que sea), con el objetivo de Y (lo que sea)
>
> Dado lo anterior, podríamos identificar los siguientes pasos:
>
> Como X quiero logearme, con el objetivo de que el sistema recuerde mis datos y compras previas
>
> Como X quiero ver la lista de productos en oferta, con el objetivo de…
>
> Como X quiero seleccionar X, con el objetivo de…

Súper importante, en todo, pero especialmente en esta primera estrategia usar Mapas de Historias de Usuario.

2.2.2 Estrategia 2: Descomponer por reglas de negocio

Una historia de usuario muy grande, mejor dicho, una Epic, normalmente, puede conllevar una serie de reglas de negocio. Así, la anterior podría descomponerse en:

> Como responsable del servicio quiero rechazar pedidos por debajo de 10 euros, porque no son rentables
>
> Como responsable del servicio quiero rechazar pedidos desde fuera de España, porque los gastos de envío hacen que no sean rentables

2.2.3 Estrategia 3: Descomponer por medio del flujo feliz / infeliz

Esto que suena así un poco raro, habla de hacer historias de usuario con funcionalidad cuando todo va bien e historias de usuario cuando las cosas van mal. Si pensamos en flujos felices e infelices podría salir:

> Como X quiero iniciar sesión, para… (feliz)
>
> Como X, quiero restablecer mi contraseña, para… (infeliz)

Hasta cierto punto esta estrategia (bueno, y todas) es complementaria, conviene recordar, complementar, etc., a la especificación con ejemplos.

2.2.4 Estrategia 4: Desglose por plataformas

En este caso hablamos de descomposición por plataformas, como tablets, teléfonos móviles, web, etc. Por ejemplo…

Como miembro del equipo quiero ver la Junta de Scrum, así que sé cómo estamos haciendo en equipo

Podemos identificar las siguientes plataformas:

> Como X quiero ver Y vía Web, para…
>
> Como X quiero ver Y en un App, para…

2.2.5. Estrategia 5: Desglose por parámetros de entrada

En este caso la división se basa en tipos de datos o en parámetros que se supone que deben manejar. Tomemos, por ejemplo, una función de búsqueda para una tienda en línea:

> Como X quiero buscar productos, para…

Pensando en parámetros de entrada…

> Como X quiero buscar un producto por su referencia, para…
>
> Como X quiero buscar productos por precios, para…

También aquí puedes usar como referencia «métodos para optimizar el uso de la funcionalidad», «buscar por voz», «buscar por teclado», etc.

2.2.6. Estrategia 6: Desglose por operaciones

Las historias de usuario suelen implicar las típicas Alta – Baja – Modificación y Consulta (el CRUD). Y esto también sirve como estrategia de división…

> Como X quiero dar de alta productos por su referencia, para…
>
> Como X quiero eliminar productos, para…

2.2.7. Estrategia 7: Desglose por escenarios de prueba

Esta estrategia es útil cuando es difícil desglosar grandes historias de usuarios basadas únicamente en funcionalidad. En ese caso, ayuda a preguntar cómo se va a probar una

pieza de funcionalidad. ¿Qué escenarios hay que comprobar para saber si funciona? Lleva un sistema de planificación de tareas:

> Como X quiero comprar productos, para que…

Si consideramos esta funcionalidad en función de posibles escenarios, podemos dividir el elemento en:

> Como X quiero que si un producto ya está comprado no se pueda volver a vender, para…
>
> Como X quiero que si un producto no está en almacén no se pueda vender, para…

De nuevo, resalto en esta estrategia leer y usar la especificación con ejemplos.

2.2.8. Estrategia 8: Desglose por roles

Aquí la idea es recordar que muchas veces «usuario» es demasiado genérico, y que ayudará mucho especificar, si existen, tipos de usuarios. Así que en esa «Como X quiero…» detalla qué diferentes X puede haber.

2.3. El método de la Hamburguesa para dividir Historias de Usuario

Te voy a contar un viejo método que llevo usando con equipos desde hace años, para dividir (bien) historias de usuario. El método se conoce como Hamburguesa y es originario de Gojko Adzic.

2.3.1 Paso 1 – Identifica las tareas para implementar la Historia

Lo primero es que identifiques los «ingredientes», es decir, los pasos técnicos que habría entre los dos panes. Estamos buscando el «end to end» de la historia. Esos pasos técnicos o tareas van a formar diferentes capas dentro de la hamburguesa.

Esto, además, nos va ayudar a recordar que un ingrediente, un trozo de lechuga, no es una historia, es una tarea. Y que no podemos caer en esa trampa a la hora de dividir Historias, porque nos estaríamos haciendo trampas.

Si estáis haciendo un taller de dividir historias de usuario con este método, utilizar post-it poniendo cada tarea en un post-it.

2.3.2 Paso 2 – Identifica diferentes opciones

Una vez que habéis identificado una primera lista de tareas para rellenar la Hamburguesa, cuando ya tenéis una primera serie de capas entre los dos panes, lo que tenéis que hacer es buscar diferentes alternativas por cada nivel, por cada capa.

Para buscar alternativas, pensar en pasos técnicos, tareas, de diferente calidad y similar resultado.

2.3.3 Paso 3 – Ordena y une los ingredientes

Ordena de izquierda a derecha las diferentes opciones por cada capa, las diferentes alternativas de tareas o pasos técnicos, en función de su calidad.

Quitar pasos repetidos, pasos sin sentido, etc.

2.3.4 Paso 4 – Recorta la Hamburguesa

Analizar ahora las diferentes capas.

En función de la calidad de cada opción y en función del coste de cada opción, desechar tareas que tengan poco sentido.

No olvidéis, no obstante, que vamos a intentar orientarnos por MVPs[9], es decir, opciones viables pero mínimas.

[9]https://www.javiergarzas.com/2016/02/aclarando-con-una-checklist-el-concepto-de-producto-minimo-viable-o-mvp.html

He intentado simplificarlo con un ejemplo real para que lo entendais, pero, como los «ingredientes» son tan específicos de cada sitio, al final he preferido dejaros este híper simplificado, aunque quede de poco realista. No obstante, también os dejo el siguiente, que es de Gojko Adzic[10], el creador del método, que es más realista… pero se entiende peor.

2.3.5 Paso 5 – Dale un bocado

Limpia la hamburguesa, ahora darle un bocado, es decir, elegir de cada capa una opción, haciendo una selección de «ingredientes» que nos permita ir desde un pan al otro.

Obviamente, este no es el único método para ayudarte a dividir historias de usuario. Si este no te encaja… prueba con algún otro.

[10] https://gojko.net/2012/01/23/splitting-user-stories-the-hamburger-method/

3. MÓDULO 3: CREANDO HISTORIAS DE USUARIO

3.1. El Dual-Track

El “Dual Track” era el concepto que nos advertía de que «TODAS la necesidades» (no lo llames requisitos) no se conocen antes de empezar a crear el producto, sino que se van descubriendo poco a poco.

Dos líneas de trabajo: en una nos centramos en descubrir esas «necesidades» (Discovery Track) y en la otra en construir (Construction Track).

El trabajo del “Discovery Track” alimenta el Product Backlog. El “Discovery Track” tiene como misión tener listas las historias de usuario antes de que arranque el Sprint de construcción en el que se implementarán, lo que incluye la información necesaria complementaria, UX (User Experience), UI (User Interfaces) y similares.

Lo anterior es, aparentemente, fácil de entender, pero cuando veo a algún equipo intentando aplicar lo anterior… esto es lo que muchos equipos NO suelen entender:

3.1.1 Dos tipos de trabajo, no dos equipos

Mientras se trabaja en el desarrollo del producto (típicamente, no exclusivamente, software), la creación de producto busca el control de la deuda técnica, calidad, la velocidad, etc. Y en el descubrimiento se busca aprender lo más rápido posible, lo que implica tirar ideas.

El descubrimiento es «parte» de la creación del producto, no algo separado. Descubrimiento y creación forman parte del mismo proceso. Y el descubrimiento es ágil, sigue un proceso ágil y le aplican los mismos principios e ideas que al desarrollo de producto, por ejemplo, software.

El trabajo del equipo de descubrimiento es cercano al de creación del producto. Un diseñador, UX, debería estar en contacto frecuentemente con aquellos que están creando el producto.

3.1.2 Error de no entender esto… pensar que un Sprint Review es para Product Owners, cuando es para Stakeholders

Te advierto, que a ojo, 8 de cada 10 que dicen «hacer Scrum» no conocen esto del «Dual Track», menos aún han interiorizado su significado real. Pero en cualquier caso, más avanzado o no, una evidencia de trabajar en equipos separados, es establecer el «Sprint Review» como el momento de la validación del trabajo de creación.

No, no, no hay dos equipos, Product Owner vs Equipo Técnico (quizá te quede más claro si en este ejemplo cambias Product Owner por Tester), y hay interacción frecuente… al Sprint Review se llega con todas las validaciones hechas por parte del Product Owner.

El Sprint Review es para enseñar a los Stakeholders no al Product Owner. Como diría ese mítico documento llamado Guía de Scrum… «During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint».

3.1.3 Descubrir implica tirar ideas

Y esas ideas se tiran antes de convertirse en parte del producto… o después, lo que vuelve a resaltar la unión de la creación de producto y del descubrimiento. De hecho, hay quien comenta[11] que realmente todo es descubrimiento, hasta el desarrollo del producto.

Esto nos llevaría a ideas como el *Hypothesis-Based Development, desarrollo basado en Hipótesis* y a que si el trabajo de creación se guía por la velocidad el de descubrimiento se guía por métricas, como el número de aprendizajes.

3.2. Hypothesis-Based Development (desarrollo basado en Hipótesis)

Lo del desarrollo basado en Hipótesis viene ser lo de llegar al desarrollo de nuevas ideas, productos, servicios, o lo que sea, por medio de experimentos, iterando hasta llegar a un resultado o hasta llegar a comprobar que la idea no era viable. Si conoces con profundidad los conceptos bajo el paraguas “ágil” esto debe sonarte.

[11]http://jpattonassociates.com/dual-track-development/

Podría citar decenas de ideas, de estos últimos años, que bajo otros nombres, cada una con su enfoque, plantean una idea como esta, desde el TDD, el BDD, con Continuous Delivery, el Lean Startup e incluso hasta el #noprojects.

En vez de planes rigurosos, inamovibles, a largo plazo, lanzar experimentos, rápidamente, de manera frecuente, para aprender rápido. Como en este mundo es muy difícil adivinar el futuro a largo plazo, pon ideas pequeñas a funcionar rápido y aprende de su éxito o fracaso.

Detrás de todo esto incluso subyace la idea de hacer uso del milenario "método científico", ya sabes, sintetizadamente: observar, formular una hipótesis, experimentar, tener indicadores de éxito o fracaso y si es necesario… volver a empezar.

Un muy buen caso de estudio sobre todo esto es el que hace ya tiempo publicó Airbnb[12], en el que muestran su manera de aplicar el "Hypothesis-Based Development", el desarrollo basado en Hipótesis, a la mejora del flujo que va desde que alguien busca una casa y la reserva, si te interesa esto te recomiendo que lo leas con calma.

3.3 Ten pocas Historias

Hay que controlar el tamaño de los Product Backlogs, que, dicho de otra manera, viene a ser tener en el Product Backlog solo las historias de usuario (podría generalizarse a Ítems, pero bueno) del próximo Sprint o de los próximos (pocos) Sprints, siempre teniendo en mente la idea de tener pocas historias.

Este es un tema con el que me toca «luchar» con bastante frecuencia. Mucha gente se agarra al «es que tengo compromisos con mi cliente» para justificar proyectos cerrados (que ya no son Ágiles, por mucho Sprint que se tenga), de los de toda la vida: comprometer un conjunto de historias en una fecha (más allá del Sprint).

Y, aunque algunas veces lo anterior sea cierto, en mi experiencia el 90% de las veces no es tan así. Los clientes suelen pedir compromisos en fechas pero solo fijan el alcance a niveles de abstracción bastante altos, Epics o superiores, y somos nosotros los que nos ponemos a sacar, desde el inicio, centenares de Historias para cumplir con esas Epics, cuando esas historias podrían irse creando poco a poco, en Dual Track y flexibilizando ese alcance «cerrado».

[12]https://www.javiergarzas.com/2016/10/hypothesis-based-development-desarrollo-basado-hipotesis.html

3.3.1. Tener pocas Historias… huele a Dual-Track

Tener pocas historias huele a Dual Track, es decir, a que lo que se va a ir incorporando al producto se va ideando casi a la vez que se van sacando incrementos, versiones del producto en pre-producción o producción, siendo esas versiones, y el choque con el mundo real, una entrada importante para crear historias de usuario, frente a la desconexión que supone sacar centenares de historias, las de muchos Sprints, y olvidarse de la «inspección – adaptación» continua.

3.3.2 Tener muchas Historias lleva por el camino del desperdicio

De entre muchas razones para tener Backlogs ligeros esta me parece tremendamente importante (e interesante).

El trabajo superfluo[13] es un desperdicio de los complicados… porque aparenta añadir valor pero, realmente, es desperdicio.

El trabajo superfluo son necesidades, tareas, trabajos, etc., que generan los tiempos innecesarios entre que llega una necesidad (una historia) y se resuelve. Y que podrían haberse evitado.

Esto es un principio Lean (que suena muy Ágil), si abres algo tienes que cerrarlo cuanto antes, sino, el tenerlo abierto mucho tiempo, genera trabajo superfluo: llamadas de clientes, correos, recordar de qué iba aquello que hay que hacer (y que lleva abierto mucho tiempo), quejas, el desgaste de priorizar mucha información, etc.

Tener muchas historias implica tener abiertas historias que se harán dentro de mucho y eso genera trabajo superfluo, es decir, desperdicio: tener que ordenar todas esas historias, volver a recordar de qué iba aquello cuando toque implementarlas, etc.

Podríamos enumerar más razones que justifican tener los Product Backlog a dieta. Hay quien, incluso, ha realizado estudios sobre como tener mucha información implica sobrestimaciones[14].

[13]https://www.javiergarzas.com/2018/02/lecciones-lean-de-guerrilla-cuidado-con-el-trabajo-superfluo.html
[14]https://www.javiergarzas.com/2016/12/la-informacion-irrelevante.html

¿Complicado tener pocas historias e ir creando historias poco a poco? No sé, pero para esto, y otras, está el rol del Product Owner ¿no?

4. MÓDULO 4: COMPLEMENTANDO LAS HISTORIAS DE USUARIO

Cuando hablamos del Product Owner, siempre a casi todo el mundo le vienen a la cabeza únicamente dos conceptos: las Historias de usuario y el Product backlog.

Pero conviene recordar que si bien las historias de usuario son importantes, pieza clave en un ciclo ágil, pensadas para recoger funcionalidad, etc., no son suficientes para definir plenamente un nuevo producto software, nos van a faltar cosas.

Cosas como, por ejemplo, la «no funcionalidad», las restricciones que le llaman algunos, aspectos como el tiempo de respuesta, requisitos de seguridad, etc. También nos van a venir bien bocetos sobre el diseño, borradores de las pantallas, etc.

4.1. Temas, epopeyas e historias de usuario

Suele ser habitual hacer uso de diferentes niveles de abstracción, con mayor o nivel de detalle, para ir definiendo objetivos que esperamos a menor o mayor plazo de tiempo.

Y en este sentido, en el mayor nivel de abstracción (menor nivel de detalle – mayor horizonte temporal) tenemos los «temas». Por ejemplo, decir que en la «release 3» afrontaremos «la afiliación de usuarios a nuestro portal», y poco más, es un tema.

Un Tema debiera ser más un Outcome, que vendría a ser algo que queremos cambiar con el producto que estamos desarrollando. Un objetivo que tengamos, un problema que tienen los usuarios y que queremos resolver.

Mientras que una Historia es más un Output, que viene a ser lo que producimos, la salida, una de las muchas maneras de afrontar la resolución de los problemas que cuentan los Temas que tienen los usuarios.

Por ello, los temas son una herramienta de la gente de negocio para recoger necesidades a nivel de Roadmap[15]

Además de las historias de usuario, podemos hacer uso de las «Epopeyas» o epics. Una epopeya es una historia de usuario más grande, sigue el mismo formato que una historia de usuario pero su alcance es mayor. Las epopeyas recogen objetivos de alto nivel, objetivos de carácter más estratégico.

Normalmente las epopeyas se dividen en historias de usuario, que son más manejables y más pequeñas, ya que realizar una epopeya nos podría llevar más de una iteración o Sprint.

Por último, las historias de usuario a su vez se dividen en tareas. Las tareas ya son cosas muy concretas a realizar durante una iteración, con el objetivo de completar una historia de usuario.

4.2. Mockups, complementando gráficamente historias de usuario

Los Mockups son una *"cooliana"* manera de llamar a aquello que el que más y el que menos hemos tenido que hacer, o interpretar, alguna vez: un boceto de las pantallas de la futura aplicación.

Los Mockups son un soporte gráfico a los requisitos, si bien, de manera general, el término se refiere a «maquetado».

Aunque la idea es de toda la vida (en el campo de los casos de uso, o de los requisitos tradicionales, esto está muy trillado), el término Mockups se está escuchando cada vez más en proyectos ágiles, ya que es una herramienta que complementa a las historias de usuario.

Muchas veces, junto con las historias de usuario y el Product Backlog, los Mockups se están convirtiendo en una herramienta complementaria para el Product Owner.

[15] https://www.javiergarzas.com/2012/06/roadmap-agil-1.html

Por lo general, complementar las Historias de Usuario y el Product Backlog con los Mockups suele dar buenos resultados, con dos consideraciones:

- Que los aspectos gráficos no hagan perder el enfoque funcional, o que las historias de usuario sigan siendo la pieza clave, y el Mockups un complemento, no al revés.
- Que no sean un motivo de generación de documentación que no aporta valor, conllevando, además, un coste en mantenimiento de «papel». Recuerda, documentación sí, pero documentación útil, simple e inteligente.

4.3. La técnica Personas para mejorar historias de usuario

Desde hace un tiempo, nosotros, internamente, cuando hacemos las reuniones internas de planificación, frente al tablero Scrum, a la hora de comentar el sentido de una historia de usuario o tarea, solemos asociarle un usuario real.

Yo suelo comentar: «cuando hablamos de esta tarea pensar en –Manolo–, porque es a él a quien va dirigido». El tal «Manolo» siempre es alguien real que todos conocemos, eso ayuda mucho a que todos tengamos claro qué queremos lograr con esa tarea o historia de usuario.

La anterior técnica, la empezamos a usar porque nos parecía de lo más razonable y nos ayuda mucho a no *irnos por las ramas*. Tiempo después nos dimos cuenta de que, además, la anterior técnica la usaba mucha más gente y que se denominaba «Personas», inventada, o más bien bautizada así, por un tal Alan Cooper.

La técnica Personas (en inglés también se le llama «personas») se basa en una idea bastante simple a la vez que eficiente: al trabajar con historias crear un actor arquetipo asociado, un personaje muy cercano a la realidad.

Al utilizar la técnica «personas» describiríamos diferentes clientes o usuarios, actores reales o muy cercanos a ser reales, para acercar la realidad al requisito, que muchas veces queda demasiado frío.

Los actores son roles, y las «personas» son «reales», serían Javier, Ana, etc.

El objetivo es llevar a los usuarios a la vida mediante el uso de personajes con nombres reales, personalidades, motivaciones y a veces incluso una foto.