Historias de usuario

# 

# cartulina

Cómo <rol\_usuario> quiero <función\_del\_sistema> para <valor\_del\_negocio>

Anotar en la parte trasera: cómo testear el producto de la historia de usario

## las historias de usuario pueden describir:

* Características
* Requerimientos no funcionales
* Bug fixes

# Cómo escribir buenas historias de usuario (vídeo)

### Minutos Ágiles (2): Historias de Usuario

<https://www.youtube.com/watch?v=-mbAXwB1q2M>

<http://librosweb.es/eventos/codemotion-roma-2013/como-escribir-buenas-historias-de-usuario/>

5:38

# Reglas

* Independiente de otras
* Granuladas pequeña
* Negociable
* Estimable
* Verificable (Testeable) TOKEN = 15’
* Valiosa

## Independientes

Deben ser atómicas en su definición. Es decir, se debe intentar que no dependa de otras historias para poder completarla.

## Negociables

Son entidades vivas. Deben ser ambiguas en su enunciado para poder debatirlas, dejando su concreción a los criterios de aceptación.

## Valiosas

Deben ser valoradas por el cliente, para poder saber cuánto aporta al Valor de la aplicación y junto con la estimación convertirse en un criterio de prioridad.

## Estimables

Aunque sea siempre un poco como leer de una bola de cristal, deben poder ser estimadas. Tener su alcance lo suficientemente definido como para suponer una medida de trabajo en la que pueda ser completarla.

## Pequeñas

Para poder realizar una estimación con cierta validez y no perder la visión de la Historia de Usuario, se recomienda que sean mayores de dos días y menores de dos semanas.

## Verificables

Este es el gran avance de las Historias de Usuario. Junto al cliente se acuerdan unos Criterios de Aceptación que verifican si se ha cumplido con las funcionalidades descritas y esperadas.

-------------------------------------------------------------------------------------------------------------------------------

User Stories Applied: For Agile Software Development

By Mike Cohn

**Historias de usuario.**

… de usuario, como su nombre indica: valiosas para el usuario.

# Chapter 1. An Overview

Software requirements is a communication problem.

Those who want the new software (either to use or to sell) must communicate with those who will build the new software.

To succeed, a project relies on information from the heads of very different people: on one side are customers and users and sometimes analysts, domain experts and others who view the software from a business or organizational perspective; on the other side is the technical team.

* If either side dominates these communications, the project loses.
* When the business side dominates, it mandates functionality and dates with little concern that the developers can meet both objectives, or whether the developers understand exactly what is needed.
* When the developers dominate the communications, technical jargon replaces the language of the business and the developers lose the opportunity to learn what is needed by listening.

**What Is a User Story?**

A user story describes functionality that will be valuable to either a user or purchaser of a system or software.

User stories are composed of three aspects:

1. a written description of the story used for planning and as a reminder: **card**
2. **conversations** about the story that serve to flesh out the details of the story
3. tests that convey and document details and that can be used to determine when a story is **complete**

Las tres C’s: Card, Conversation, and Confirmation

cards "represent customer requirements rather than document them." This is the perfect way to think about user stories:

While the card may contain the text of the story, the details are worked out in the Conversation and recorded in the Confirmation.

Ejemplo OK:

A user can search for jobs.

A company can post new job openings.

A user can limit who can see her resume.

Ejemplo KO: no son valiosas para el usuario

The software will be written in C++.

The program will connect to the database through a connection pool.

**Where Are the Details?**

It's one thing to say "A user can search for jobs." It's another thing to be able to start coding and testing with only that as guidance. Where are the details?

Many of these details can be expressed as additional stories. In fact, it is better to have more stories than to have stories that are too large.

When a story is too large it is sometimes referred to as an epic. Epics can be split into two or more stories of smaller size.

However, we do not continue splitting stories until we have a story that covers every last detail.

Similarly, the user story does not need to be augmented in typical requirements documentation.

The conversation is the key, not the note on the story card. Neither the developers nor the customer can point to the card three months later and say, "But, see I said so right there."

Stories are not contractual obligations. As we'll see, agreements are documented by tests that demonstrate that a story has been developed correctly.

# Chapter 15: Using stories with Scrum

## Adding Stories to Scrum

### Stories and the Product Backlog

I have had great success expressing Scrum backlog items in the form of user stories (Cohn 2003).

Rather than allowing product backlog items to describe new features, issues to investigate, defects to be fixed, and so on, the product backlog is constrained to only user stories.

Each story in the product backlog must describe some item of value to a user or to the product owner.

By constraining the product backlog to only user stories it becomes much easier for the product owner to prioritize the backlog.

With all backlog items in terms she can understand, it becomes easy for the product owner to make decisions to trade one feature for another.

### Stories in the Sprint Review Meeting

The use of stories benefits Scrum during the sprint review meeting because stories make it simpler to assess what parts of a sprint have been completed.

On a Scrum project that uses a random collection of technical tasks, requirements, issues and bug fixes as its product backlog, it can be difficult for the team to demonstrate that each item made its way into the product built during that sprint.

When the entire product backlog is composed of stories describing items of customer or user value, it is generally easier to demonstrate those items.

### Stories and the Daily Scrum Meeting

I have found stories beneficial to daily scrum meetings because they help ensure focus remains on customer and end-user desires.

Because there is no upfront requirements or analysis phase that precedes a sprint, sprints are started with only a partial understanding of exactly what will be built. The team may know they are planning to add a search screen but they may not know what fields will be searchable, how search criteria may be combined, and so on.

Stories are useful because they remind the team about the intent behind what is being developed. In the midst of a sprint the team can use the story (as well as ongoing discussions with the product owner about the story) to determine whether they have gone far enough, or perhaps too far, in programming a particular story.

# Chapter 2. Writing Stories

To create good stories we focus on six attributes. A good story is:

* Independent
* Negotiable
* Valuable to users or customers
* Estimatable
* Small
* Testable

User Story Kata:

<http://historiasdeusuario.com/userstorykata/quiero-llamar-a-mi-madre>

## Independent

Care should be taken to avoid introducing dependencies between stories.

Dependencies between stories lead to prioritization and planning problems.

Dependencies between stories can also make estimation much harder than it needs to be.

When presented with this type of dependency, there are two ways around it:

* Combine the dependent stories into one larger but independent story =>
* Find a different way of splitting the stories

## Negotiable

Stories are negotiable.

They are not written contracts or requirements that the software must implement.

Story cards are short descriptions of functionality, the details of which are to be negotiated in a conversation between the customer and the development team.

Because story cards are reminders to have a conversation rather than fully detailed equirements themselves, they do not need to include all relevant details.

However, if at the time the story is written some important details are known, they should be included as annotations to the story card,

Ver ejemplo libro.

If we think about the story card as a reminder for the developer and customer to have a conversation, then it is useful to think of the story card as containing:

* a phrase or two that act as reminders to hold the conversation
* notes about issues to be resolved during the conversation

Ver ejemplo libro.

## Valuable to Purchasers or Users

It is tempting to say something along the lines of "Each story must be valued by the users." But that would be wrong.

Many projects include stories that are not valued by users. Keeping in mind the distinction between user (someone who uses the software) and purchaser (someone who purchases the software)

stories like the following might be valued by purchasers contemplating buying the product but would not be valued by actual users:

“The development team will produce the software in accordance with CMM Level 3.”

What you want to avoid are stories that are only valued by developers. For example, avoid stories like these:

“All connections to the database are through a connection pool.”

## Estimable

It is important for developers to be able to estimate (or at least take a guess at) the size of a story or the amount of time it will take to turn a story into working code.

There are three common reasons why a story may not be estimable:

1. Developers lack domain knowledge.
2. Developers lack technical knowledge.
3. The story is too big.

- If the developers do not understand a story as it is written, they should discuss it with

the customer who wrote the story.

- the developers do not understand the technology involved.

The solution in this case is to send one or more developers on what Extreme Programming calls a spike, which is a brief experiment to learn about an area of the application.

During the spike the developers learn just enough that they can estimate the task.

The spike itself is always given a defined maximum amount of time (called a timebox), which allows us to estimate the spike.

In this way an unestimatable story turns into two stories: a quick spike to gather information and then a story to do the real work.

## Small

Story size does matter because if stories are too large or too small you cannot use them in planning.

Epics are difficult to work with because they frequently contain multiple stories.

## Splitting Stories

Epics typically fall into one of two categories:

* The compound story
* The complex story

- A compound story is an epic that comprises multiple shorter stories

- the complex story is a user story that is inherently large and cannot easily be disaggregated into a set of constituent stories.

If a story is complex because of uncertainty associated with it, you may want to split the story into two stories: one investigative and one developing the new feature.

Complex stories are also common when developing new or extending known algorithms [One team in a biotech company had a story to add novel extensions to a standard statistical approach called expectation maximization.]

## Testable

Stories must be written so as to be testable.

Successfully passing its tests proves that a story has been successfully developed. If the

story cannot be tested, how can the developers know when they have finished coding?

Whenever possible, tests should be automated. This means strive for 99% automation, not

10%

Untestable stories commonly show up for nonfunctional requirements,