The A.A.R.T. of Writing User Stories

A proposal for writing better Agile user stories.

A.A.R.T. = Actors, Action, Results, Tests

Proposed by Fred Henry Williams

www.williamstechnical.com

First Presented at

Agile Prague, 2013

(Link to video)

Summary:

Current user stories can be improved by clearly identifying, linking, and tracking Actors, Action, Results, and Tests.

Writing user stories in the form of:

Fred (Actor) drinks coffee (Action) to wake up. (Result)

...is more clear than the traditional user story, and easier to understand and write for non-native users of English.

Requiring at least one **Test**, written as user instruction for the Actor, increases the chances of detecting flawed requirements early.

Background:

Few developers and managers are comfortable writing user stories. So models have been provided for creating user stories. Typically, the user story model is something like this:

“As a (role) I want to (do something) so I can (get something).”

Users stories are the heart of Agile development practices. User stories must be well written, clear, concise. User stories show the true requirements for all stakeholders, and allow the prioritization, planning, tracking, development, testing and documentation of deliverables.

Problem Statement:

The current model unnecessarily distances developers from users, and relies on English language forms that might be unclear. Users are not named, actions can be buried under verbiage, and desired outcomes might be misunderstood or even ignored. Tests are rarely included with the user story until later in the process, losing valuable time in identifying and improving unclear requirements.

Roles are often imprecise, such as “user” or “administrator”. Even where user personas or specific users are well known, they are not clearly linked to the user story.

A key virtue of Agile is empathy. Developers must understand the users of the system, and their goals. Yet the form “As a (role)...)” invites role playing, rather than empathy. A common result is that inside the developer's head, the story too often is interpreted to mean “If I were playing that user or administrator role, here's how I would do it...”

User stories should really be considered “People's stories”. The more we can build empathy and understanding of the people using our products, and the situations and environments they experience, the more likely it is we will succeed in delighting our customers with excellent and innovative products that allow real people to solve real problems and achieve real goals.

Action (do something) may also be misunderstood following this template.

“I want to”, “I would like to”, “I must be able to”, and similar constructions are unwieldy, and can be difficult for non-native English speakers to parse correctly. Weak modal verbss are particularly prone to be misunderstood as suggestions, rather than requirements.

Results (get something) are also sometimes unclear in the Agile user stories following the standard template.

The result of a user's action might be part of a larger process, or may have multiple beneficiaries other than the user. The goal of the user can be lost in the process of understanding and implementing the action (do something). Lost in the wording, and distracted by focusing on enabling the tasks within the action process, developers may not be able to find a truly testable outcome for the user story.

When the desired result is unclear, or not stated in terms of the user's perspective, it's easy to lose sight of the reason the work is being done in the first place.

Tests should be written at the same time as requirements. But few teams manage to do this, because it's difficult and can introduce conflict into the process. If the actors, actions, and results are unclear, creating meaningful tests is not likely.

Because of the importance of testing to the Agile process, this lack of clear early tests is potentially deadly to Agile projects.

Proposal

Write user stories in the form of A.A.R.T.

Example:

Fred (**Actor**) drinks coffee (**Action**) to wake up (**Result**).

(Test) Instructions:

You can drink coffee when you want to wake up. Coffee contains caffeine, a stimulant. You can buy a cup of coffee, or make it yourself.

Warning: Do not drink more than 5 cups of coffee within 2 hours.

1. Buy or make coffee.
    * Add milk or sugar to your taste
2. Drink the coffee
    * Note: Hot coffee can burn you

Within 20 minutes, you will be more alert

Actors

Actors are real people performing some interaction with your system. They aren't playing a role, and it's certainly rare that they really want to use whatever you've made for them. People have other interests. In fact, those interests are probably the only reason they have to reluctantly interact with your system at all.

People are individuals, not roles.

As individuals, they have an environment where they live, and important goals in life. Understanding the actor's environment (where they encounter non-negotiable requirements) and goals is the reason we have user stories in Agile development.

Additional attention to your actors should improve your user stories, increasing empathy and understanding of their real motives and expectations.

The key innovation I'm proposing is for product managers, developers, designers, architects, and everyone else who should be thinking about the “user” and “role” to **give your users names**, and use those names instead of “As a (role)...”

This works best when **Personas** are fully described and well understood. Where the Personas (or actual users, if you are so lucky) are presented in some details on a **wiki or webpage** we can also link to every instance of that Persona's interaction with the system, creating a richer picture of their interactions, goals, and tasks.

We can truly describe the problems they wish to solve, goals they want to achieve, and how they'll know they've got what they wanted within the persona description.

The persona's **Environment** must be described in some detail. The benefit of this is that you will discover environmental requirements that can be plainly stated in simple **Waterfall** requirements and **constraints**, and consistently tested with rigid parameters and standards. It is stipulated that all user stories involving the persona shall also conform to these environmental constraints. There is no need to try to re-work what is a simple unchanging and easily tracked Waterfall type requirement into a complex user story.

When the environment changes, you only need to change that constraint within the Persona's description – simultaneously updating the regression testing instructions.

This means that the user stories, which require more work and thought from project stakeholders than non-negotiable environmental requirements, can focus on the actions the actors take.

Alice and Bob Example:

Discussion of Alice and Bob Example:

Alice and Bob and Chuck are classic encryption characters. So we know something about their motives and environment from this story. The Test Instructions show how they could solve the problem with PGP and emails.

> * What if we changed the Actors and their environments? If Alice becomes Andrew, a notorious cyber-criminal, and Chuck becomes Cynthia, a police officer investigating Andrew's crimes, consider how their environmental constraints would change.
>
> * What if Bob has only a mobile device?
>
> * What if Alice is a 90 year old who wants to encrypt her diary until after her death?

Actors are people, so just a few changes to their descriptions would result in very different recommendations and solutions. What if Chuck develops meta-data collection?

A.A.R.T. User Story for 2 Actors:

> ***Alice*** *and* ***Bob*** *send email to each other that* ***Chuck*** *cannot read.*
>
> ***Test Instructions:***
>
> > ***Sending Encrypted Emails***
> >
> > *Encrypted email allows you to keep your messages safe from 3rd parties. You can use PGP public keys to encrypt private messages with your computer's mail client. (Installation instructions link) Open your mail client to write your messsage, then encrypt and send your message to another person who has a PGP public key.*
> >
> > 1. *Write your message.*
> >   ***Warning****: Your subject line CANNOT be encrypted. It may be read by anyone. PGP only encrypts your message body.*
> > 2. *Click the* ***Key*** *button to add your* ***public key***
> > 3. *Select your message recipient*
> >   *Note: If you have not already identified the recipient's* ***public key****, click* ***Download missing keys***
> > 4. *Type your encryption password*
> > 5. *Click* ***Send****.*
> >
> > *Your recipient receives your encrypted message, and uses your public key to decrypt it. Only you and your recipient can read this message.*

Extra discussion: Chuck's User Story:

> "**Chuck** intercepts meta-data from **Alice** and **Bob's** PGP encrypted emails."

Defining and Challenging Action

Actors can be plural. But in AARTful user stories, the Action must be singular.

In software, the Action is frequently going to be a single task within the system, such as saving a file on a server, or changing a record in a database. Many people perform these actions, and there can be many results (log files update, a popular file is subsequently distributed through a worldwide CDN, the saver of the file can find it again, and so on).

There may be multiple Actors performing this same action, and multiple results of this single action. (Certainly there would be many tests for such an action too.) But we want to describe one action, and describe how to test if this one action has the intended results.

The language we use to describe action can be simple present tense using the Subject Verb Object (SVO) word order. This is the first form of English taught to non-natives, and is the easiest to write and understand.

> **Subject – Verb - Object**
>
> **"Fred DRINKS coffee..."**
> **"Alice and Bob SEND encrypted emails..."**
> **"Richard SHARES a file..."**
> **"John and Susan NEVER EAT candy..."**
> **"Andy CANNOT DRINK alcohol..."**
> **(Note: Negative requirements should be included in your user stories.)**

Every Action can and should be questioned, and replaced with something better if that is reasonable for the actor's environment. Action is NOT a requirement, but a description of an assumption of the way the actor achieves a result.

When "Fred drinks coffee to wake up", he might be open to a different method. If there is no coffee available in Fred's environment, tea might be available. Maybe he'll try an energy drink instead. Fred might use vigorous exercise to wake up, or a quick painful pinch.

All user stories are necessarily reductive. By linking to rich descriptions of user Personas, the user stories offer useful context for interpretation of real-world goals and requirements, perhaps allowing replacement of an assumed action with innovation.

Results

Results are the only reason anyone uses a system. I don't use software because I like to use software. I use it because it allows me to do other things.

Those results, the things that matter most to the Actor, must be stated explicitly. If the result is not testable...it is not a useful or realistic user story for requirements specification.

> In the example:
>
> *"**Fred** drinks coffee to wake up"*

The result is clear; Fred wakes up. If Fred drinks coffee and does not wake up, the test fails and another solution must be found.

Even better, by explicitly stating the result, developers can identifying short cuts and jumps...if Fred truly cares about waking up more than drinking coffe, perhaps an exotically named energy drink would be better and faster? Innovation can be encouraged by identifying the goal so clearly that the process can be circumvented when the circumstances are right.

By describing the results clearly, from the perspective of the Actor, simple black-box tests can be derived to confirm if the Actor is able to achieve the results promised.

Results must be stated in a testable way. Ambiguity produces vague results. Replacing descriptive adjectives and adverbs with numbers or comparisons can help. For perceptual requirements, declare the user Persona who must receive that perception. For system results, link to a summary description of the system environment and architecture.

Vague:	**Clear:**
Fred *wakes up.*	***Fred's*** *eyes blink less than 10x per minute.*
The system *is fast.*	***The system*** *returns search results faster than Google.*
The system *is user friendly.*	***Midred Thistlebottom*** *says the system is useful.*

Tests

Defining at least one test, from the user's perspective, helps you find incomplete, ambiguous, or overly complex requirements early. Additional tests can and should be added according to environmental and other non-Agile requirements, but the first draft of a user story should ideally contain just one obvious user oriented test written in the form of instructions for a user story's actor.

By making the test an explicit part of the user story, and placing it directly after the expected result, the clear link between user goals (expected Results) and testing is clear.

I additionally propose that writing the test in the form of user instructions offers many advantages.

Write the instructions simply, with a description telling the Actor why they should perform an action, what the action is, where if can be found, and when to use it. Steps describe how to perform the action. The result statement tells the Actor what they'll get for following the instruction.

When the user story has been developed, user documentation requires only validation and a header starting with the gerund of the Action. (“Drinking Coffee” or “Saving Files”).

Sharing Files Example:

"Sharon and Martha share files so they can see each other's children pictures."

Test Instructions

Sharing Pictures and Videos

When you share your picture and video files on your Profile page, any of your friends can see the files.

1. Login to your **Facebook** account
2. On your **Profile** page, click **Add Pictures and Videos**
3. Select a file you want to share
4. Click **OK**.

Your pictures and videos display on your Profile page for any Facebook friend to see.

Sharing Files Example Discussion:

Notice the tendency to find an existing solution to the problem. As a major concern about my proposal, and potential weakness, this is most serious. So this discussion is not posturing, I really do want feedback.

* A valuable advantage for writing user instruction is it forces you to identify: Why the persona should use the feature; What the feature does; Where the Persona can find the functionality; When to use it; and Who uses it (if that is unclear). The How-to steps will be changed to match the user interface before testing, and can be approximate at the beginning. The result statement must be clear, and match the result described in the user story.

Linking

JIRA/Confluence allows you to connect wiki pages with user story task tracking.

Create a wiki page for each Persona, including that persona's environmental constraints . The constraints can be stated Waterfall style since they are not negotiable.

When developers work on a AARTful user story, they can immediately refer to the wider goals and experiences of the linked personas to understand the motives for performing an action.

(screenshots of examples from JIRA/Confluence)

Conclusion:

We're not done yet, my friends!

A.A.R.T. is an attempt to improve a weakness in current user stories. Some companies have already adopted this standard into their Agile methodology and report good results.

I am offering this to the development community for feedback and testing so I can improve this idea.

All those offering substantive comments and suggestions, especially those who tell me how I am wrong, will be gratefully acknowledged.

Please contact: fred (at) williamstechnical (dot) com

Thank you,

Fred Williams

October 2013

Acknowledgements (so far):

David Stula, Kentico
Colin Vipurs, Shazam
Czech Society for Scientific and Technical Communications (CSVTK)
Frank McAleer, Williams Technical
Pavel Zelezny, Williams Technical
Dean Aaron Johnson, UNYP