In XP, we think of requirements of coming in the form of user stories. It would be easy to mistake the story card for the “whole story,” but Ron Jeffries points out that stories in XP have three components: Cards (their physical medium), Conversation (the discussion surrounding them), and Confirmation (tests that verify them).

A *pidgin language* is a simplified language, usually used for trade, that allows people who can’t communicate in their native language to nonetheless work together. User stories act like this. We don’t expect customers or users to view the system the same way that programmers do; stories act as a pidgin language where both sides can agree enough to work together effectively.

But what are characteristics of a good story? The acronym “INVEST” can remind you that good stories are:

- **I** – Independent
- **N** – Negotiable
- **V** – Valuable
- **E** – Estimable
- **S** – Small
- **T** – Testable

Independent

Stories are easiest to work with if they are *independent*. That is, we’d like them to not overlap in concept, and we’d like to be able to schedule and implement them in any order.

We can’t always achieve this; once in a while we may say things like “3 points for the first report, then 1 point for each of the others.”

Negotiable… and Negotiated

A good story is *negotiable*. It is not an explicit contract for features; rather, details will be co-created by the customer and programmer during development. A good story captures the essence, not the details. Over time, the card may acquire notes, test ideas, and so on, but we don’t need these to prioritize or schedule stories.

Valuable

A story needs to be *valuable*. We don’t care about value to just anybody; it needs to be valuable to the customer. Developers may have (legitimate) concerns, but these framed in a way that makes the customer perceive them as important.

This is especially an issue when splitting stories. Think of a whole story as a multi-layer cake, e.g., a network layer, a persistence layer, a logic layer, and a presentation layer. When we split a story, we’re serving up only part of that cake. We want to give the customer the essence of the whole cake, and the best way is to slice vertically through the layers. Developers often have an inclination to work on only one layer at a time (and get it “right”); but a full database layer (for example) has little value to the customer if there’s no presentation layer.

Making each slice valuable to the customer supports XP’s pay-as-you-go attitude toward infrastructure.

Estimable

A good story can be *estimated*. We don’t need an exact estimate, but just enough to help the customer rank and schedule the story’s implementation. Being estimable is partly a function of being negotiated, as it’s hard to estimate a story we don’t understand. It is also a function of size: bigger stories are harder to estimate. Finally, it’s a function of the team: what’s easy to estimate will vary depending on the team’s experience. (Sometimes a team may have to split a story into a (time-boxed) “spike” that will give the team enough information to make a decent estimate, and the rest of the story that will actually implement the desired feature.)

Small

Good stories tend to be *small*. Stories typically represent at most a few person-weeks worth of work. (Some teams restrict them to a few person-days of work.) Above this size, and it seems to be too hard to know what’s in the story’s scope. Saying, “it would take me more than a month” often implicitly adds, “as I don’t understand what-all it would entail.” Smaller stories tend to get more accurate estimates.

Story descriptions can be small too (and putting them on an index card helps make that happen). Alistair Cockburn described the cards as tokens promising a future conversation. Remember, the details can be elaborated through conversations with the customer.

Testable

A good story is *testable*. Writing a story card carries an implicit promise: “I understand what I want well enough that I *could* write a test for it.” Several teams have reported that by requiring customer tests before implementing a story, the team is more productive. “Testability” has always been a characteristic of good requirements; actually writing the tests early helps us know whether this goal is met.

If a customer doesn’t know how to test something, this may indicate that the story isn’t clear enough, or that it doesn’t reflect something valuable to them, or that the customer just needs help in testing.

A team can treat non-functional requirements (such as performance and usability) as things that need to be tested. Figure out how to operationalize these tests will help the team learn the true needs.

For all these attributes, the feedback cycle of proposing, estimating, and implementing stories will help teach the team what it needs to know.

SMART Tasks

There is an acronym for creating effective goals: “SMART” –

- **S** – Specific
- **M** – Measurable
- **A** – Achievable
- **R** – Relevant
- **T** – Time-boxed

(There are a lot of variations in what the letters stand for.) These are good characteristics for tasks as well.

Specific

A task needs to be *specific* enough that everyone can understand what’s involved in it. This helps keep other tasks from overlapping, and helps people understand whether the tasks add up to the full story.

Measurable

The key *measure* is, “can we mark it as done?” The team needs to agree on what that means, but it should include “does what it is intended to,” “tests are included,” and “the code has been refactored.”

Achievable

The task owner should expect to be able to *achieve* a task. XP teams have a rule that anybody can ask for help whenever they need it; this certainly includes ensuring that task owners are up to the job.

Relevant

Every task should be *relevant,* contributing to the story at hand. Stories are broken into tasks for the benefit of developers, but a customer should still be able to expect that every task can be explained and justified.

Time-Boxed

A task should be *time-boxed*: limited to a specific duration. This doesn’t need to be a formal estimate in hours or days, but there should be an expectation so people know when they should seek help. If a task is harder than expected, the team needs to know it must split the task, change players, or do something to help the task (and story) get done.

Conclusion

As you discuss stories, write cards, and split stories, the INVEST acronym can help remind you of characteristics of good stories. When creating a task plan, applying the SMART acronym can improve your tasks.

[I developed the INVEST acronym, and wrote this article in April and August, 2003. Thanks to Mike Cohn for his encouragement and feedback.]

Related Articles

- **I**ndependent Stories in the INVEST Model
- **N**egotiable Stories in the INVEST Model
- **V**aluable Stories in the INVEST Model
- **E**stimable Stories in the INVEST Model
- **S**mall – Scalable – Stories in the INVEST Model
- **T**estable Stories in the INVEST Model

Postscript – Added 2-16-2011

I've been asked a few times where the INVEST acronym came from. I consciously developed it: I sat down and wrote every attribute I could think of applying to good stories: independent, small, right-sized, communicative, important, isolated, etc. This gave me a page full of words. Unfortunately, I haven't kept that list.

Then I grouped the words into clusters:

- Isolated, independent, separate, distinct, non-overlapping, …
- Important, valuable, useful, …
- Discrete, triggered, explicit, …
- …

The categories were a little fuzzy; I had about ten.

I identified "centers"; words that captured the essence of their category. Some clusters had two or three plausible centers. That was ok, as I just wanted their first letter for the acronym.

Then it was scramble time: take the three or four clusters that had to be there, plus some of the less important ones, and scramble the initials to find a word that fit. I wanted a word that had 4-6 letters, with no repeats.

I tried a lot of combinations. I remember at one point that if I had a G I could make VESTIGE or VESTING. Having VESTIN_, I realized I could turn it around to get INVEST, which sounded much better than VESTIGE:) So INVEST won, and it's been popular enough that I'm sure I made the right choice.