Horizontal and Vertical User Stories - Slici the Cake

Written by Ned Kremic

User stories are an agile approach to requirements.
User story is description of functionality as seen from user perspective of the system (*"I want to withdraw $100 from my checking account"*), not from pers developer of the system (*"the system shall tokenize the bank card number..."*). Which is exactly what its name implies "the user story" not just "the story"! The primary purpose of the user story is to let the user express in clear, unambiguous, non-technical language what she really wants, so that all stakehold a story implementation could understand it.

What are Vertical and Horizontal Stories?

In the last few years when agile adoption has started to spread rapidly, we have seen many references to vertical and horizontal stories. What are those, a need for distinction between them.
The term "vertical" came from now famous analogy of the complex system under consideration, to a cake (Bill Wake 2003).

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, b 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 pres 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 vertica 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 c there's no presentation layer.

~ Bill Wake

The end user really doesn't care how the system is structured and built, she only values the functionality she is interested in. Therefore "withdrawing $100 checking account" might be very complex operation that needs to go through all technological layers, but at the end it must satisfy user need "to withdraw

Vertical Stories

Rather than building the system, layer by layer, in which case the end user will have a chance to experience the system only when it is fully done, the idea i user story by user story, where each one will cross all horizontal layers.
Analogy is that each user story represents one tiny slice of cake, hence its name: **vertical story = slice of cake**.
Therefore "*As a user of ATM, I want to withdraw cash from my bank account*" and "*As a user of ATM, I want to pay my utility bill*" are the vertical user stories.

Vertical user story is not a scenario.

One user story may contain many positive and negative scenarios, and they should be part of acceptance criteria for that user story.
From the user story mentioned above, we can derive at least ten positive scenarios, and all of them at the end will result in satisfied user walking away fro $100 cash.

User enters the wrong password, then corrects it
User enters the amount larger than the daily withdrawal limit, then corrects it.
User enters the amount larger than the her account balance, then corrects it.

Similarly we can come up with five times more negative scenarios that will result in user being rejected withdrawal from her account.

Wrong bank card
expired bank card
customer forgot her password
account in negative balance
ATM has no money

Vertical stories are written in a unambiguous language that is easily understood by all stakeholders of the system, and encourages the conversation betw
All those scenarios mentioned above can be easily communicated between the product owner and the team (developer and tester), and specified as an a
criteria.

Horizontal Stories

Back to our analogy between the complex software system and the cake. Cake consists of several layers, similar to software system that consists of sev
architectural layers.. Horizontal stories are stories that belong to one architectural layer of the system, such as DB, network, security, middleware applicat
Those are the stories or requirements that are easily understood only by that particular layer specialists.

Even though we may call them stories, and write them as stories, they are in fact requirements, as rarely horizontal stories can deliver the value to the enc
without interaction with other layers of the software system.

Therefore: **Horizontal user stories = Traditional IEEE 830 requirements**

For instance DBA has focus and responsibility only for the DB layer and operations relevant to DB.
UI designer is interested in visual representation and user experience, and is not concerned with DB normalization, query optimization and store procedur
Although particular layer functions are absolutely necessary and have perfect sense to developers, alone they don't bring tangible value to the end user.

"The system shall tokenize the credit card number"
"The system shall check the credit limit against the NFDB"
"The system shall transfer the approved funds to the receiving institution"

The major caveats of the "horizontal" approach are:

The feedback loop is long, which significantly increases the risk. The entire cake must be baked before the customer realizes that the cake she got is not
wanted (in the best case she will get what she has asked for, but at the end it is not what she had in mind).

Maybe she preferred walnuts rather than almonds in one of the cake layers. Or she wanted pink icing for her daughter birthday rather than the blue one, th
assumed because icing colour had not been specified.
If she had a chance to sample one slice, she would have realized those gaps early on.

It is difficult to prioritize horizontal stories/requirements and potentially release the most important ones or the most required features first (Fast time-to-
For example, during the initial pilot roll-out of ATM machines, most bank customers have asked for ability to withdraw their money 24/7 and check their b
based on the marketing research, just withdrawing the cash would accounts for over 90% of intended usage of ATM
It could have been easily achieved with two vertical stories (epics) and delivered to the market early:
"*As a user of ATM, I want to withdraw cash from my bank account*"
"*As a user of ATM, I want to get the current balance of my bank account*"
Knowing this fact, the prudent approach would be to release the first version of ATM, only with ability to dispense the cash (and we can call it "the cash m
bank would be able to release it to the market much sooner (3-6 months) rather than waiting to implement the full functionality (12-15 months)

However, application layer developers working based on the horizontal requirements specified for release 1, scheduled to be released in 15 months, have on "processTransaction" functionality that will cover fund transfers from one account to the other one, bill payments and deposits, rather than the cash w

If competition is just about to release the two most demanding ATM functions in four months, and our full-fledged ATM will be released in 15, the bank is losing rather large number of its customers.

Why Transitioning to the Vertical Stories is Difficult

Even though the concept behind the user stories is so simple, almost self-explanatory, their adoption and effective use in practice is quite complex for nev that are transitioning to agile.
There are two major obstacles that most agile teams are facing when transitioning to user stories:

Resistance to Change

The user stories, even as simple as they are, present major paradigm shift for most new team members who are coming from traditional technologically l systems.

The truth is that they see their world in a horizontal way, as they have been taught, trained, specialized and then worked for years on horizontal layers!
Most data architects and DBA in order to properly design the database and normalize the tables, would like to see all detailed requirements upfront. Simila with the solution and infrastructure architects, as well as specific domain (horizontal layer) subject matter experts. Notion of incremental development an based on vague user stories are in most cases so strange that majority of them have hard time even contemplating.

Complexity of the System - The Iceberg Phenomenon

Although the cake analogy to complex software system is fantastic way to easily describe the horizontal requireme user stories, it does present very simplistic view of the system.
In reality most software systems are not shaped as a regular cake, rather like the iceberg. Just 1/8 of the system is therefore tangible to the end user. The remaining 7/8 of the system is invisible hidden under the water.
The challenge is describing, implementing and integrating 7/8 of the system that is laying under the water, with the derived from 1/8 of the system.

Consider this typical iceberg user story:
"As a user of Expedia travel site, I would like to see the list of flights between the two selected cities, with total, tax inclu

There is enormous complexity of this user story that lays beneath the water. Integration between the various airline systems, tax calculations based on the province or state of residence, performance requirements etc.
This user story is typical epic that cannot be implemented in one iteration. It takes practice and experience to split s along vertical lines, implement them in a single iteration and still deliver the value to the user.
Natural tendency of most teams would be to split that epic flow into numerous horizontal user stories, and integrat together at the end. However this is not the optimal solution and probably the worst way to split the story.

Splitting the story along the horizontal layers, means losing all advantages of tiny vertical stories mentioned above.

It is beyond the scope of this article to analyze the optimal ways to split the vertical user stories, however as a guideline, the teams should always judge th created story against the INVEST criteria. If any of the letters is missed, the user story is flawed and should be rewritten.
http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Building the System by Stacking the Horizontal Layers

It is almost impossible or impractical to align all horizontal stories across different layers, and adjust them to always changing business priorities.

With horizontal stories, we can consider that the software system is complete when each horizontal layer fulfills a list of requirements (horizontal stories) integrated together.
The user will have a chance to see and experience the system once it is fully assembled.
The best outcome would be if the user gets "what she has asked for". However if what she got is not what she had wanted and doesn't satisfy users goals

“change of scope”, and the whole vicious cycle will go again...

Building the System by Joining the Vertical Slices

On the contrary, with vertical user stories, the software system is built incrementally, implementing user story by user story, where each story fulfills the u
History has taught us, that users may have different and probably better opinions once they see and experience each vertical slice. And we expect that us
change that slice either because she didn't get what she had wanted, or she has seen the potential for even better improvement. Again the original ask is i
paramount is always that the user gets what she wants, and that is the best value for each slice.

Going even further, we can say that the software system is not done when the scope is completed, rather when it fulfills the most important user goals (B
can withdraw the cash from ATM 24/7, which attributes to 90% of ATM usage),
That goal may be satisfied and software released well before the entire scope is completed (cash withdrawals, deposits, transfers, bill payments, stateme
That powerful approach, is the major driver behind the rapid time-to-market delivery of agile software.

~~Ned Kremic~~ is Project Management (PMP) and Agile/Lean (CSM/CSP) Consultant based in Toronto, Ontario.
His mission is to help progressive organizations seeking high-impact results, to radically improve productivity, time-to-market and quality for software dev
He was instrumental in building and leading the high-performance teams that were utilizing the best technology practices, combined with advanced agile
development processes and achieved outstanding results that outperformed not only the traditional projects in corporate portfolios, but rather the industr
performance results of the other agile projects.
Your best chance to succeed with transition to Agile/Lean development and management process and achieve exceptional performance results is with s
has successfully done it over and over again.
Explore this site, and if you are ready to go from good to great, and you think that Ned might be right for your organization to show you the way to get ther
~~us a call~~.

Featured Articles

- Horizontal and Vertical User Stories – Slicing the Cake
- High Productivity And Exceptional Quality Are Just Half The Story
- Agile Performance Metrics
- Agile Estimation

Most Read Articles

- Why is Agile Time to Market (TTM) Delivery 50% Faster?
- Why are Agile teams 25% more productive?
- The Quest for Exceptional Software Quality

More Info

Search here

About Ned