The new user story backlog is a map

Why the flat user story backlog doesn't work, and how to build a better backlog that will help you more effectively explain your system, prioritize, and plan your releases.

10/08/2008

This is Gary.

Gary and I worked together for a day to build a user story map - a better version of a product backlog. Building a user story map helps us focus on the big picture - the product as a whole instead of getting myopically focused on an individual story.

When it comes time to prioritize, Gary did so with the entire context of the system in view. In Gary's case he originally set out to build Mimi - short for Music Industry Marketing Interface. However, when it came time to prioritize we focused on the band member promoting a gig in a club. It was too big for a single story - too big for an "epic" really... it took most of the map to think about all the details of designing a promotional mailer, building up an audience list, then sending out the promotion, and tracking responses. By the end of the mapping exercise, all the other music related things Mimi could have done seemed like a tall order. What's more, building a piece of commercial software that made the job of sending out mailers easily and quickly seemed like a good idea. And it turns out it was.

Mimi shipped after a lot of sweat and effort from Gary, Dave Hoover, and the fine folks at Obtiva. As of this writing, Mimi currently sends close to four million messages per month. The user experience is a work of art. It's a sexy piece of software. And, the user story backlog we built that first day of discussion still describes the same high level view of Mimi we saw then - minus the stories we moved out of the release of course.

Flat backlogs don't work for me

One of the more troubling things, for me, about common Agile development practice is the idea of the flat user story backlog. It just seems like a dumb idea to me. Well not completely dumb, in that if I'm going to build software incrementally in small pieces, I'll need to figure out what piece to start with. The backlog puts those pieces in build order - from highest value to lowest value - which is great if you're a project manager - which I'm not.

I'm writing this from the plane as I fly back from a client site. We had what I feel like was a pretty successful couple days of story writing and release planning. I had fun - and I was pleased to see the group really get into it and feel good about what they came up with. I've been using a practice I [now] call story mapping. I first wrote about this approach in the article "How You Slice It." The article was published in January 2005 - but by the time it was written, I'd been using the practice for a couple years.

Years later, I still love the approach... so much so that I'm in danger of seeing it as a "silver bullet" - which is an early warning sign that I'm becoming a dogmatic fool. However, I'm seeing more and more people arrive at similar
approaches - so at least I know I'm not too crazy.

Let me describe what's wrong with story writing, then generally what a story map is - and why it solves problems for me.

The flat backlog is poor explanation of what a system does

Arranging user stories in the order you'll build them doesn't help me explain to others what the system does. Try handing your user story backlog to your stakeholders or users when they ask you the question "what does the system you're building do?"

For my money, trying to understand the system - the whole system - is the difficult part of software development. One of the most common complaints I hear from Agile teams is that they lose the big picture - that is if they ever had it in the first place.

I tried to explain to a co-worker once what I thought was wrong with a story backlog. The story goes something like this:

"We spend lots of time working with our customers. We work hard to understand their goals, their users, and the major parts of the system we could build. Then we finally get down to the details - the pieces of functionality we'd like to build. In my head a see a tree where the trunk is built from the goals or desired benefits that drive the system; big branches are users; the small branches and twigs are the capabilities they need; then finally the leaves are the user stories small enough to place into development iterations."

"After all that work, after establishing all that shared understanding I feel like we pull all the leaves off the tree and load them into a leaf bag - then cut down the tree."

"That's what a flat backlog is to me. A bag of context-free mulch."

I need that context in order for me to really tell a story about the system.

For a new system, the flat backlog is dismal at helping me determine if I've identified all the stories

Given a stack of index card, or a spreadsheet full of user stories, I can stare at them for hours, discuss them for hours... but I always walk away with the nagging feeling that there's something I'm missing. Of course I know I can't think of everything - but I just wish I felt a little more comfortable.

How often are you blindsided by overlooking important pieces of functionality when scoping a new piece of software?
Release planning with a flat backlog is difficult

For me, for the typical project I get involved with, the number of stories in a backlog is dozens at a minimum - often over a hundred. I work hard, especially when we're just starting out, to keep the stories high level. Even then it's common to end up with a 120 or so stories in a first cut of a backlog. Stepping through each one of these and making an in-out decision is tedious. Some of the most miserable hours of my life have been spent sitting in meetings with groups prioritizing their backlog. Uuugh.

Build a story map

Arranging user stories into a helpful shape - a map has worked well for me.

It's a simple idea really. A small story map might look something like this:

At the top of the map are "big stories." I call them user activities (borrowing the term from UX people like Larry Constantine and Don Norman). An activity is sort of a big thing that people do – something that has lots of steps, and doesn't always have a precise workflow. If I was building an email system (which I'd never be foolish enough to do) I might have an activity called: "managing email", and "configuring email servers", and "setting up out of office responses."

A story for an "activity" might read: As a consultant I want to manage my email so I can keep up with clients, colleagues, and friends.

But that's way too big of a story to put into an iteration or sprint.

That story breaks down into other stories like "send message," "read message," "delete message," "mark message as spam" - stuff like that. I call these user tasks. (Again a word used by UX people.) For lots of Agile people "tasks" refer to the things that developers do to finish user stories. Really a task is something that someone does to reach a goal. A user task is what users do to reach their goals, developer tasks are what developers do to create stories, Ant tasks are what ant does to... well... do whatever you're doing with Ant.

I simply arrange the small things under the big things in a bit of a grid form.

I'm always imagining time moving left to right (because it does in my western world. My counterparts from top down or right to left worlds can translate as needed). For instance when arranging stories in the map, if a person using the system typically does one thing after another, then I'll put the early thing on the left, and the later thing...
on the right. I do this with the big things and the little things - the activities and the tasks.

When teaching this, people often tell me “the users can perform these in any order. What order should I put them in?” I’ll ask them to “explain to me what the system does at a high level - just tell me the activities.” They then recite them to me. “That’s the order” I say. In fact, the order you’d explain the behavior of the system in is the correct order. We’re building a map that lets us tell a really big story about the system. Build the map in a way that helps you tell the story.

Keep your epics - but stop calling them that because it bothers me

The really big stories can be considered "epics" as Mike Cohn describes them. They're stories - just really big ones - too big to estimate and build. When an epic gets in your backlog, and it's time to discuss it in more detail, I often see people remove the epic from the backlog, decompose it, and replace the pieces they've identified. This is where I cringe. Recall my cutting down the tree and keeping the leaves in a mulch bag story above.

That big story was context. It was my simple way of thinking about the whole activity that people were doing. It's my quick way of explaining to others what the system is about.

And, I hate that word "epic." I haven't written beowulf here. There's no hero with a magic weapon slaying a monster. It's just my user "managing email" - a relatively basic thing from my user's perspective. At least the terms "activity" and "user task" give me some idea of what kinds of stories they are. That said, I'm not fond of the term "user story" either, but I've come to accept it. It beats the crap out of requirement. I still have trouble looking a C-level exec in the eye and explaining to him that "we've broken your system down into a series of epics and stories." Upon saying something like that I imagine he's hastily rethinking my daily consulting rate. All the sticky notes and index cards aren't helping either.

Walk your map to test it - to get the big picture

Given a fully assembled map, I have something I can hang on a wall, or lay out on a tabletop and actually have a discussion with. I can walk the map from beginning to end with a user, stakeholder, or developer and tell a story about the users of the system and what they're doing. I can skim along the top of the map, and just touch on the high points. I can dig down into the map to discuss the details.

Talking through the map with users and others helps me find things I've missed. I often hear "you've missed a couple steps here" from users when doing this.

I can annotate the map with pain points or opportunities. As I talk through the map with a user it's common to hear him say, "this here is really a problem with the system today."

When working with users they often correct me. Yesterday I was working with a potential user of a new system - she was a compliance officer for a major company. I write down a process step as she said it and placed it on the table. I'd been using green cards for big things, yellow for smaller things. She quickly corrected me - "no, that should be a yellow card." In less than an hour's discussion she had caught onto the modeling approach and could, in
very few words, supply me with a lot of information about the thing she was doing with the system. She knew yellow was smaller than green, and could tell me that the thing she was doing in the system wasn't as big as I thought it was.

Building and walking story maps leaves me more comfortable than ever before that I "get it" - that I haven't missed something big.

**Your software has a backbone and a skeleton - and your map shows it**

I find that the big things on the top of the story map look a little like vertebrae. And the cards hanging down look a little like ribs. Those big things on the top are often the essential capabilities the system needs to have. I refer to them as the "backbone" of the software. I stole this term from Dr. Dan Rawsthorne who might use the term slightly differently than I do.

When it comes time to prioritize stories, I don't prioritize the backbone. It just "is." I do prioritize the ribs - the stories hanging down from the backbone. Place them high to indicate they're absolutely necessary, lower to indicate they're less necessary. When you do this, you'll find that all the stories placed high on the story map describe the smallest possible system you could build that would give you end to end functionality. This is what Alistair Cockburn refers to as the "walking skeleton". I always try to build this first.

**Plan using your backbone**

When it's time to plan releases, it's usually not important to prioritize backbone items against each other. For instance if I were to build a high level backlog for a car it might look something like this:

- engine
- transmission
- brakes
- suspension
- ...

It would be stupid to ask stakeholders to prioritize that: "what's more important, the engine or the transmission?" – or "we don't have enough time in this release, could we release without brakes and add them later?" These items are essential - and we'll need all of them to deliver a **minimum viable product** - and MVP.

Where the prioritization comes in is below this level: 4-cylinder engine or 6-cylinder engine? brakes with anti-locking or brakes without? sport suspension or not? It's how we build up those backbone items - prioritize their
characteristics that matters.

When you prioritize a story map, you’ll move cards or stickies up and down to indicate high or low. Lately I take a long strip of masking tape and line off the story map - creating horizontal swim lanes for each release. Then I move the stories up and down into each lane, and even vary their height in the lane.

Keep your map displayed to communicate the big picture

Building a story map helps you initially understand the functionality. Ask yourself when in your project do you not need to understand your functionality any longer? I know it’s not a fair question, but I do find some folks that spend a great deal of time building this sort of thing to understand the problem, then tossing it all out in favor of putting stories into a flat-backlog. Cutting down the tree and loading the leaves into a mulch bag.

I find a story map hung as an information radiator becomes a constant point of discussion about the product we’re building. When the project is running, it becomes our sprint or iteration planning board. We identify or mark off stories to build in the next iteration directly on the map. During the iteration we’ll place just the stories we’re working on into a task wall to managing their development - but the story map lives on the planning wall reminding us what the big picture is, and how far we’ve come.

When we're building software incrementally, story by story, we'll choose them from the story map left to right, and top to bottom. We'll slowly move across the backbone, and down through the priorities of each rib. We're slowly building up the system not a feature at a time, but rather by building up all major features a little at a time. That way we never release a car without brakes.

A different backbone may be in order for adding features to an existing product

When adding features to an existing product, it already has a backbone - and sufficient functionality to have released. I find it useful to stilly identify the backbone to help me get context - to help me see where new features are being placed.
On some projects adding just a few features to a large existing product, it’s been difficult to talk people into building up a story map for just a few features. In these cases, I’ll simply prioritize the new features, then for each feature build a little story map to prioritize the user stories that make up that features. Each little story map may have 10 or so cards - but they’re still arranged left to right, and top to bottom so we can focus on building a little-tiny-walking-skeleton of the features as early as possible.

**It's a pattern - not an innovation**

Although I tried not to, I became indignant when I heard someone describe this concept back to me because a major consultancy was now teaching this as best practice for managing backlogs. "They stole my idea!" I thought. But then I had to remind myself that I’ve seen lots of others doing very similar things. I know when I started at ThoughtWorks many years ago, I hadn't been working there long before I ran into Luke Barrett who’d been building almost exactly the same model collaboratively with users the same way I had.

I always remind myself of the litmus test for a pattern. If you explain someone a concept and they say "what a cool idea!" it’s not a pattern. But if they say "we’re doing something like that too!" it’s a pattern. I’ve seen this often enough now that I believe it’s a pattern.

When teaching I like showing Indi Young’s Mental Model that arranges user research in a similar left to right flow, with feature ideas placed adjacent to the user behavior they’re relevant to. I like Todd Warfel's Task Analysis Grid. It’s like one vertebrae of a story map with lots of cool context, and all the user stories neatly prioritized below. Todd uses color to indicate priority, not height.

---

**Learning more about story maps**

If you'd like to learn more about this, consider showing up at one of the conferences where I’ll be teaching the practice. This year I'll be at Spool's UI13 in October, SD Best Practices 2008 in November, and Better Software's Agile Development Practices 2008 in Orlando. A half day class should give you all you need to know to get started.

If you're impatient, you can try to figure it out from my presentation slides. And, go back to my original article on building story maps.

---

**What now?**

Thanks for reading this blog/essay/rant! I’d love to hear from you, so please rate this entry, and consider leaving a comment. Or, send email to me directly.

If this blog entry was interesting, please take a look at all blog entries.
If you find my brand of thinking valuable, please consider visiting me at a conference, or public class. Or feel free to contact me directly for agile coaching and training.