Backlog vs. Specification: Why Backlog Beats Specification?

By Balazs Tornai

Specifications are well-loved by managers, because they detail every tiny step up-front. A specification builds a nice hierarchy, and it makes you think that once you hand it over to the developers, they follow it to the letter. Magic happens, and you get a perfect product ready to go.

When it comes to manufacturers, it may even work out like that, but in a knowledge-based economy, it never will. No matter how good the specifications are, there is a lot of obscurity in software development. You will keep learning new information in the development phase, which could shake your specifications to the core.

Technology and languages change at a mind-bending pace in software development. The same volatility goes for business expectations. A great idea might lose its value in just a few months.

This is why writing a specification could be wasted time. In a specification-driven collaboration, no one wants to push one unavoidable change after another through massive documentation. This doesn't help to build a cooperative relationship.

So, when should you write a specification? Small projects can use small specifications to satisfy small needs.

What could you do instead?

Use Backlog.

In this post we're covering:

Backlog definition

The Backlog consists of well-defined units prioritized among each other.

High-priority units are continuously refined, while lower-priority units aren't developed all the way until a business decision is made about introducing them to the market.

A Backlog is better than a specification

The Backlog is a special beast with some necessary and unique attributes. In exchange, it allows for a lot of flexibility as opposed to the classic specification, where both parties are interested in not being flexible. This is a typical conversation when following a specification:

"I planned everything precisely!"

“I’m not paying for anything else!”

“This was never part of the project!”

“It’s a basic feature; it was obviously meant to be there!”

The flexibility of a backlog alleviates both parties from trying to push their interests against each other.

How is backlog so flexible?

The smallest unit in a backlog is a user story. A user story follows some baseline rules, but as long as you keep to these rules, your backlog will hold your business and the development process together more effectively than any other documentation has before.

A user story describes processes from the end-user point of view. This is the base for estimations, and it is used to define developer tasks.

A user story is the smallest unit that breaks down the software in a clear way. Developer subtasks aren’t part of the equation.

Subtasks built around user stories cover everything you could expect from a great specification.

It’s even better because user stories include quality control elements, which is hard to effectively put into a specification.

As soon as you have a few user stories, the developers can get to work, especially if they have clear priorities.

Importance and priority always differ between any two items. I know, everything is important, but one thing is always more important for a working product.

The key of the Backlog is that it consists of prioritized user stories.

You don’t want to use user stories in every possible scenario. There are technical tasks that can’t be defined as user stories. You just define them as separate Backlog items.

When you set up priorities, it doesn’t matter if it’s a user story or a different item. You just need to make sure everyone involved is clear on what item to work on right now and on which one to do next.

You can only do this with a Backlog, as long as you stick to the rules.

What do you get out of a backlog?

One of the gifts of backlog is that developers won't surprise management. Managers will always know what they're working on and why, and the devs won't stray off the path either.

And again, planning a backlog can't be ad hoc. You absolutely need to stick to all five base rules of agile planning; otherwise, everything will fall apart.

Once your backlog items are ready and you have set up a priority, you just have to let your developers set up their subtasks in sync with that and get them done.

You always need to estimate your subtasks in the unit previously agreed upon.

The prognosticator

Besides continuous prioritization, you need to make sure your estimates are fairly reliable. It seems to be a contradiction since you can't plan every subtask of every backlog item up-front. Most of them aren't even split to subtasks.

And yet, it's a valid expectation to give some kind of estimate.

The trick of a backlog is that you can use relative estimations.

A backlog is handy at extrapolating expected time and resource requirements for items that haven't been planned yet.

With some experience, you can write backlog items and set up priorities. Based on a randomly chosen backlog item, you can do fair estimates on whether they take similar effort, twice as much, half as much, and so on.

It's not a precise estimate, but that's not the point. The idea is that you can figure out early and with low effort whether an item will be easy to deliver or if you have trouble coming.

When it's not in the extremes, you'll get a clearer picture once you put some effort into planning.

With relative estimates, management gets updates on the current state of the backlog and the estimated arrival of the high-priority items, without putting effort into planning that is likely to be tossed out anyway. It could be due to a change in the business environment, technology rules or restrictions, user requirements, etc.

The person handling the Backlog will be continuously working on updating the remaining items with the new information from one iteration to another. This will keep you updated on everything, and the estimates on time and resources will become more precise every time.

According to the last 10 years' worth of data in a well-run agile project, the time planning takes is about 25-45%.

Conclusion

Backlog…

-Allows you to plan in sync with the five levels of agile principles.

-Demands continuous prioritization.

-Allows for flexibility in line with market or technological needs.

-Recognizes “go” and “no go” situations.