Thermo Fisher Scientific: Lesson #6 – Story Writing Proves Difficult

One of the surprises during the Thermo Fisher Scientific engagement was that the team initially struggled with writing Epics and Stories.

The problem was not that Stories were inherently difficult to write. Rather, the Scrum approach to describing work conflicted with some of the habits that hardware engineers develop through years of working with requirements and specifications.

To understand the issue, it is important to distinguish between a requirement and a deliverable.

In software development, that distinction is often blurred.

Suppose I write the User Story:

As a City Planner, I want to sort property addresses by Zip Code.

The statement simultaneously describes a requirement and implies the deliverable. The requirement is the ability to sort addresses by Zip Code. The deliverable is the software enhancement that implements that capability.

Because the language used to describe the requirement and the resulting deliverable is essentially the same, software engineers often do not think about the distinction explicitly.

Hardware development is different.

Suppose I write the requirement:

“As a child, when I drop the object, it bounces back toward my hand.”

The requirement describes a desired behavior. The deliverable, however, is a rubber ball.

The language used to describe the requirement and the language used to describe the deliverable are completely different.

As a result, hardware engineers become accustomed to treating requirements and specifications as distinct concepts.

Interestingly, this was not the source of the Thermo Fisher team's difficulty.

The team had no trouble distinguishing between functional requirements and deliverables. In fact, they naturally excluded functional requirements from the Stories they wrote.

The real challenge involved non-functional requirements.

When I explained that Stories should describe deliverables, the team repeatedly produced Stories that described non-functional requirements instead. These requirements were important, but they did not identify any specific deliverable that the team would produce.

As a result, I found myself reviewing each Story and asking a simple question:

What is the deliverable?

If the Story did not clearly result in a component, subsystem, prototype, design, test fixture, document, or some other tangible outcome, it needed to be rewritten.

This pattern recurred often enough that it caught my attention.

In my experience, software teams usually adopt Story writing with relatively little difficulty. The Thermo Fisher team required considerably more coaching before they became comfortable focusing Stories on deliverables rather than on non-functional requirements.

The good news is that the issue was short-lived. By the end of the first day, the team had largely mastered the technique and Story writing was no longer a significant obstacle.

Nevertheless, the experience raised an interesting question.

Was this challenge unique to Thermo Fisher Scientific, or was it a more general characteristic of hardware engineering teams? At the time, I had no way to know.

The lesson was therefore worth remembering.

Lesson #6: Hardware teams may initially struggle with Story writing because they tend to express non-functional requirements where Scrum expects Stories to describe deliverables.

The issue is not a major roadblock, but it is something Agile practitioners should anticipate when introducing Scrum into a hardware-development environment.

In my next post, I'll discuss another surprise from the Thermo Fisher engagement: why the Product Owner wrote relatively few Stories.

For those interested in the full case study, the complete Thermo Fisher Scientific story is available on my website at https://kevinthompsonphd.com/storage/papers/LessonsLearned-AgileHardware-v6.pdf.