Thermo Fisher Scientific: Lesson #12 – Why Swarming Is Different in Hardware Development

One of the predictions from my Agile Hardware research was that swarming would be much less effective in hardware development than it is in software development.

The Thermo Fisher Scientific engagement validated that prediction.

Swarming is one of the most important concepts in Scrum. It is a strategy for assigning people to work during a Sprint that helps maximize delivered value while reducing the risk associated with uncertainty.

In a typical software Scrum Team, work is assigned dynamically throughout the Sprint. As team members complete their current tasks, they move to the highest-priority Product Backlog Items where they can contribute. Over time, small groups of people form around individual items and then disperse as work is completed.

The result is that the team's effort naturally concentrates on the highest-priority work. This strategy also reduces parallelization by encouraging team members to complete existing work before starting additional items. While traditional project-management approaches often seek to maximize parallel activity, Scrum generally favors finishing higher-priority work sooner, even if fewer items are being worked simultaneously.

This approach works because many software teams are composed of what Agile practitioners call *generalizing specialists*. Team members possess deep expertise in one area but can contribute meaningfully in several others.

The Thermo Fisher team operated differently.

Like many hardware-development teams, it consisted of highly specialized experts in disciplines such as optics, electronics, firmware, physics, and mechanical engineering.

These specialists were certainly capable of collaborating with one another, but they could not readily perform each other's work.

As a result, most Stories had only one person on the team who was qualified to perform the majority of the work.

This dramatically reduced the effectiveness of swarming. Because specialists often had to work independently on different Stories, the amount of parallel work in progress naturally increased.

Where a software Story might attract a swarm of two or three contributors, a hardware Story often had a practical swarm size of one.

The team could still collaborate, review designs, and provide feedback. However, the actual implementation work generally remained in the hands of a single specialist.

This observation has an important organizational consequence.

Suppose we decided that enabling swarming was our primary objective.

We could reorganize the teams by discipline rather than by product. One team could consist entirely of mechanical engineers, another entirely of electrical engineers, and so forth.

Within each team, skill overlap would be much greater and swarming would become easier.

Unfortunately, this solution creates a larger problem.

The teams would no longer be organized around building products. Instead, they would be organized around engineering specialties.

The coordination required to deliver a product would then move from inside the team to between teams.

The result would be more handoffs, more coordination meetings, more dependencies, and more opportunities for misunderstanding and delay.

In other words, we would reduce the cost of collaboration within disciplines only by increasing the cost of collaboration across disciplines.

For hardware development, this is usually a poor tradeoff.

The purpose of Scrum Teams is not to maximize swarming. The purpose is to develop products as quickly and efficiently as possible.

For that reason, maintaining cross-functional teams is generally more important than maximizing opportunities for swarming.

This experience reinforced another conclusion from my Agile Hardware research.

Lesson #12: In hardware development, cross-functional teams and swarming are often competing objectives. When forced to choose, organizations should favor cross-functional product teams over discipline-based teams designed to maximize swarming.

The lesson is not that collaboration becomes unimportant. Rather, it is that hardware development often requires a different balance between specialization and flexibility than software development.

In the next article, I will conclude this series by describing how the transformation unfolded at Thermo Fisher Scientific, sharing the team's results, and offering some final reflections on the experience.

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.