Agile Hardware at Bird Technologies

Kevin Thompson, Ph.D.
www.kevinthompsonphd.com
November 26, 2019

Abstract. Bird Technologies has found Scrum and Agile techniques useful for developing both hardware and software elements of their radio-frequency test and communications equipment. This case study describes how the author guided the company's migration to an Agile development process, with an emphasis on challenges encountered and lessons learned. The central conclusion—that Scrum and Agile techniques apply as well to mechanical and electrical engineering as they do to software development—is confirmed by the smooth adoption of these concepts at Bird Technologies.

Contents

1 Introduction 2
2 Background 3
3 Consulting work 4
3.1 Challenge: Who Should be Scrum Master? 4
3.2 Challenge: Writing Stories 4
3.3 Challenge: Estimating Work 5
3.4 Challenge: Forecasting Velocity 6
3.5 Challenge: Scheduling Stories when Lead Times are Long and Parts are not Reliable 6
3.6 Challenge: Sprint Planning 7
3.7 Challenge: Finishing Planned Work 7
4 Additional Insights 8
5 Agile Hardware Myths vs. Reality 8
5.1 The Problem of Long Lead Times 8
5.2 The Problem that Development of Useful Deliverables Takes more than One Sprint 8
5.3 The Problem of Implementing Functionality by Vertical Slices across Technologies 9
5.4 The Problem of High Skill Specialization 9
6 Conclusions 10
7 Acknowledgments 10

1 Introduction

Bird Technologies is a telecommunications company headquartered in Solon, Ohio. Founded in 1942 by J. Raymond Bird, and still family-owned, the company specializes in high-accuracy and high-reliability radio-frequency (RF) testing and communication equipment. As of 2018, the company had approximately 300 employees across multiple locations.

The company's current president, Terrance (Terry) Grant, was promoted to this role in September 2017, and moved swiftly to address improve the company's ability to develop and ship products. John Winter, then Interim Manager of Engineering and Innovation, took this mandate and began the search for solutions.

When training in Lean Business Systems led to improvements in Bird's manufacturing process, they next looked to find ways to improve the company's ability to develop products. John's search led quickly to Cprime, and my papers on the application of Agile concepts to hardware development. We agreed on a Statement of Work to transform the engineering organization to Agile development and kicked off the first of two consulting engagements in January 2018. I recruited Abid Akhtar, a consultant based in Houston, TX, to assist.

Compared to most Agile transformations, the novelty with Bird Technologies was the application of Scrum and Agile techniques to work that involved Mechanical Engineering and Electrical Engineering, along with firmware and software development. This work differs from software development in key ways, and this engagement highlighted those differences.

2 Background

Several Scrum myths cause many to believe Scrum and other Agile techniques are inappropriate for hardware development. Some of these myths include:

- Long lead times for component orders make finishing a Story in a two-week Sprint impossible.
- The amount of work required to produce functioning device components is too great to fit into a two-week Sprint, which is incompatible with Scrum.
- Scrum requires developing capabilities as vertical slices across layers of technology, which is incompatible with hardware development.
- Scrum depends on Story-Point estimation and collaboration via swarming, but these are not possible in hardware development.

Many of these myths emerge from the way Scrum is often taught and coached as a software-development methodology. The reality is Scrum is a way to organize people to get a job done and can work well with hardware-oriented product development. In fact, in Takeuchi and Nonaka's "The New New Product Development Game," [1] which influenced Jeff Sutherland's early work on Scrum, all the example projects are hardware projects such as Canon's PC-10 personal photocopier.

My research and conclusions on how to apply Scrum and other Agile concepts to development of hardware products are documented in my paper, "Agile Processes for Hardware Development," [2] and in my book, *Solutions for Agile Governance in the Enterprise (Sage)* [3].

The Agile transformation for Bird Technologies would provide the largest-scale test yet of these discoveries, ultimately spanning nine teams. The work was spread across two consulting engagements, each with its own focus. The first engagement brought Scrum to two engineering teams, while the second scaled Agile collaboration to span all groups involved in product development, including engineering, manufacturing, sales, and support.

3 Consulting work

The goal of the first consulting engagement was to get two hardware-development teams up and running effectively with Scrum. The second focused on scaling the process to nine collaborating teams. The focus in this paper is on the hardware-related aspects, which are critical at the team level but less so at the organizational level.

We began on Jan 29, 2018, with me teaching two days of *Agile Project Management with Scrum (for Hardware Development)* training at the company's Solon, OH headquarters, followed by more time on-site to mentor and coach the organization through the first three two-week Sprints. The attendees for this "Scrum for Hardware" class included teams responsible for two products. One team was the M2M ("Machine to Machine") product, which produced hardware solutions to operate and collect data from remote sensors. The other was the RFIP team, which developed amplifiers and relays. The RFIP team was primarily based in the Angola, NY facility, with some key members in Ohio.

3.1 Challenge: Who Should be Scrum Master?

The first challenge emerged on our first coaching day, as we discussed who should fill the Scrum Master role for the RFIP team. In the training, I had emphasized that it was unwise to have one person act as both Team member and Scrum Master, since the effectiveness of the Scrum Master role suffers. Unfortunately, this was exactly what they had been intending to do, and now the challenge was to find someone else who could fill the Scrum Master role.

An effective Scrum Master must first have enough time to do all the things that are needed to enable the team to be successful. The person must also have the right mix of supportiveness and firmness (especially around protecting the team members' time and plans from disruption). A Scrum Master does not have to have enough technical knowledge to do the work of the team members but must have enough to communicate with them effectively on technical topics.

We spent much of the day discussing alternatives. In the end, one of the people present (Jim) was "volunteered" to be Scrum Master for the moment, but this was not work he could take on indefinitely. The name of another person came up frequently in the conversation, and the group eventually made the decision to ask him to take on the role. Unfortunately, as he had not been identified as someone who needed training, he had remained in Angola and had not attended the class. Abid and Jim would have to work closely with him over time to bring him up to speed on his new role.

Fortunately, everyone agreed that Ken would be the Product Owner, and he did attend the training. Ken's role had already included product definition with respect to business and customer needs, and so he fit the role naturally.

Also fortunate was that filling the roles for the M2M team was straightforward. In particular, Ann, one of Bird's project managers, was a natural for the Scrum Master role. She had the right mindset and sufficient time to fulfill the responsibilities of the role.

3.2 Challenge: Writing Stories

User Stories provide a common format for describing some aspect of a product that a team will produce. Each such Story contains a Title, a first-person Narrative that describes how a particular user role

performs an action in order to achieve a particular benefit, and a list of Acceptance Criteria to be confirmed during testing of the deliverable.

The concept of User Stories goes hand-in-hand with the standard advice to "implement software features across all levels of the technology stack," meaning to implement the functionality by making appropriate changes to the user interface, the application logic, the database, and other architectural layers or sub-systems for each such capability. To implement any one User Story usually requires touching the code for multiple layers.

Unfortunately, these patterns didn't apply to the work Bird Technologies needed to do. Much of their work required designing and modifying circuits and boards, isolating unruly RF signals, routing cables, and tuning RF transmissions, as well as developing the firmware and user software. While the devices are complex, relatively little of the work relates to specific user-oriented features of the products. Instead, the internal elements of the product must be designed from the beginning to support a range of capabilities, which only become usable later when the device has accumulated enough parts to have some kind of behavior.

In other words, most pieces of the RF devices Bird designs could not be represented as User Stories. A feature-oriented approach, which spans a technology stack, was not applicable. What did work was a component model, where each bracket, circuit, chassis, and so forth was represented by one or more *Technical Stories*, for which there are no human users.

I introduced both User Stories and Technical Stories in the training, and Abid and I coached the teams through writing Stories. Technical Stories resemble User Stories but lack a user role. The Narrative does not describe a user experience but instead describes the thing to be developed. The Title and Acceptance Criteria are the same as for User Stories.

On the positive side, understanding the Story format and writing in that format was not a major stumbling block. By and large, the new Scrum Team membership could do this. As I have seen in other hardware companies, the bulk of this Story writing was done by Team members, under the overall guidance of the Product Owner. This arrangement fell out naturally due to the highly technical nature of the Stories to be written.

Challenges lay not so much in writing in the Story format as in becoming effective at identifying deliverables of the right size, suitable for implementation in a Sprint. Also difficult was formulating research Stories ("spikes") whose outcome was not predictable, and whose results could change the direction of development work in unknown ways. While people grasped these concepts well enough in the abstract, they struggled in practice with defining the research clearly and staying within bounds of the time they chose to allocate for the work. These difficulties were not fully resolved during the course of the engagement.

3.3 Challenge: Estimating Work

The Planning-Poker technique from the training worked well for estimating Stories, as long as the units were based on effort (time), and not Story Points.

The use of Story Points requires that all members of a Scrum Team can understand a set of reference Stories in the same way, and scale their concept of size of Stories in the same way. For this to be possible there must be significant overlap in skills across the team members. That degree of overlap

does not happen when the team members are spread across the disciplines of Mechanical Engineering, Electrical Engineering, and Firmware development. It is not possible to create a set of reference Stories that all team members will understand well enough to use as a basis for Story estimation.

One response might be to re-organize as single-discipline teams, but that is no more desirable for hardware development than it is for software development. It is better to have product-oriented teams in both cases, and this means we need an alternative to Story Points that can be understood the same way across team members.

The language of time and effort *is* understood the same way across all hardware team members (and I find that software teams often prefer it to Story Points as well). The unit of measure for Stories is a "Person-Day" or "Man-Day," which means eight hours of effort applied to a piece of work. Thus if someone votes a "2" in Planning Poker, the number means that all work done on the Story, added up across all Team members who work on it, is two Person-Days, or sixteen hours. (The work for each Story's tasks is estimated in hours per task, which is a common practice in Scrum in general).

Fortunately, time-based estimation was familiar to all of the technical people at Bird Technologies, and the concept posed no difficulties. It was true that not all Team members could provide estimates for each Story, but enough could contribute to multiple Stories to get the benefits of group scrutiny.

3.4 Challenge: Forecasting Velocity

The units of Velocity must match the units of Story estimates, which were Person-Days for these teams. In this situation, the "Yesterday's weather" approach is less appealing than a forecast based on how much time the Team members can contribute to work in the Sprint. I helped them work out this forecast, using a simple spreadsheet model for how much time each Team member should be available contribute to the work in the Sprint. As a useful bonus, this approach also clarified how many hours of work were available for the different specialized skills that would be needed in the company's two-week Sprints.

3.5 Challenge: Scheduling Stories when Lead Times are Long and Parts are not Reliable

The dependence on physical components acquired from other vendors provided two complications for the schedule of work. First, the time from ordering to receiving parts could be on the order of a few weeks. Second, the parts might not be acceptable or have characteristics that would require re-design of other aspects of the product under development.

The first problem is what I call *latency*, meaning a pause in the middle of the development activity. We dealt with latency by writing Stories to encompass the work up to placing the order, and Stories about what to do once the order arrived. The point is to avoid letting significant latency occur within the scope of a Story.

The second problem is common and generates unplanned work. By definition, unplanned work cannot be planned before the need for it appears. Dealing with unplanned work effectively is managed by some combination of buffer time in the schedule and scope reduction. (If this kind of disruption is common, I recommend Velocity buffering, meaning to plan less work per Sprint than the team's ideal Velocity would predict. Planning to 70% of ideal Velocity provides a 30% buffer for unplanned work).

3.6 Challenge: Sprint Planning

I presented a common two-part Sprint Planning meeting in the training. Part 1 generates a draft Sprint Backlog based on Story estimates, the Velocity forecast, and any other useful insights. Part 2 completes planning by creating a Task Breakdown for each Story, and using the Task-level effort estimates (in hours) to create the final plan based on this finer-grained information.

These may have been the longest Sprint Planning meetings I have ever attended. No one had difficulties with the techniques, but simply getting through each meeting took about five hours.

The M2M team's planning session took a long time because they had a large number of small Stories (over twenty). Planning does not take long if the Velocity forecast is twenty, and each Story has an estimate of five. It takes lot longer when the estimates are one or less.

At various points, I asked if there were a way to bundle the work into larger pieces. I learned that this might have been possible to some extent, if we could have started over, but mostly they just had a lot of little things that needed to be done that were fairly independent of each other. That would not be the case for every Sprint, but is what happened with this one.

The RFIP team had a different problem. They had a unique approach to planning out the tasks for each Story, which I had never seen. I would summarize it as, "First determine all solutions that cannot work, and then put what remains into the plan."

From my perspective, they started on each Story by ruling things out, which made the discussions very long. I tried to get them to start with what was likely to work, rather than on what would not, but to little effect. The meeting went on the same way.

At least the meeting did finish with a reasonable plan. I hoped that the team would become faster with practice.

3.7 Challenge: Finishing Planned Work

My on-site time ended after the teams finished planning for Sprint 1, but Abid remained to coach them through the first three Sprints. The two of us spoke frequently, and I monitored the stream of Burndown charts that came from the two Teams over time.

We place great emphasis, in Scrum, on planning for what is feasible, and not taking on more work than we can be reasonably confident of finishing in a Sprint. There are many good reasons for this policy, ranging from morale to having good insight into progress.

New teams often plan more work into Sprints than they can actually accomplish, resulting in Burndown charts that never get down to zero hours of planned work remaining. While no team can be perfect all the time, I generally expect that teams that are diligent about the various Scrum practices will eventually be able to complete their planned work in most Sprints.

Over time, the M2M team did show a trend towards improvement, although they did not get to zero work remaining at the end of the first three Sprints. There was no single obvious reason for this difficulty, and I expected progress to continue as the team matured.

The RFIP team was a different story. In every Sprint, the Team put more work into their Sprint Backlog than they could finish and routinely moved unfinished Stories into the next Sprint from each Sprint into the next. At one point, I emailed the Scrum Master for the team, asking carefully about this issue. I learned that he and the Team members liked how things were going, and were happy that they had improved their efficiency and "found hours" to do more work over time. I would have preferred a greater focus on keeping planned work to a level they could routinely complete, but at least they were functioning, and that was a win for everyone.

4 Additional Insights

One early insight discovered during the adoption of Scrum was that the RFIP team was extremely bottlenecked by reliance on one person, who was overloaded as a result. This wasn't surprising, but the greater clarity provided by the Scrum approach made this point much clearer. The problem was not one that could be solved instantly, but the company did begin taking steps to resolve the problem over time.

Another early insight, which the M2M team addressed effectively, was that the Product Backlog contained Stories from several different products. This mix diffused the team's focus and could have pushed out completion dates on high-priority work. In response, the Product Owner was able to clarify priorities and thus focus the team's resources to deliver the highest value to the company.

The Backlog Refinement and Sprint Planning meetings made clear that people often had very different understandings of work to be done, without knowing that they differed. The emphasis on collaborative definition and planning of work paid off in generating a common understanding of work, and reduction of confusion and false starts.

5 Agile Hardware Myths vs. Reality

The myths that I listed at the beginning of this paper did not prevent Bird Technologies for adopting Scrum. The resolution of the various problems listed in those myths follows.

5.1 The Problem of Long Lead Times

Long lead times are not unique to hardware development, although they tend to occur more often than for software. A four-week lead time for ordering a part is always going to impact the development schedule. It is a common source of latency, meaning a pause in some continuing thread of activity.

However, latency has no effect on the ability to complete Stories. Stories describe deliverables to be produced by the team, so we write Stories for work that precedes the latency, and Stories for work that follow the latency. We do not write Stories that contain large latencies.

Long lead times occur in any kind of product development. The approach to dealing with them does not change based on the type of product.

5.2 The Problem that Development of Useful Deliverables Takes more than One Sprint

This reality can lead people to believe that Stories cannot be completed in two-week Sprints. However, the work can always be partitioned into smaller deliverables and tested to ensure that they are as desired. It may be the case that eight weeks are required to produce something that resembles a functioning part of a larger device, but that is a different topic.

There is a common Agile mindset that can cause problems in this area. The mindset is that every Sprint must produce something new and useful. That is not always possible in software development, and it is rarely possible in hardware development. While the goal of getting usable things done as early as possible is a good one, it is a mistake to try to require that a usable thing be completed in every Sprint. The attempt to mandate this behavior leads to long Sprints that demonstrate the typical problems of long Sprints, low morale being among them.

Big things take substantial time to develop, and cannot be implemented in one Sprint. This is true for software and hardware products, and does not limit our ability to work in two-week Sprints effectively.

5.3 The Problem of Implementing Functionality by Vertical Slices across Technologies

User Stories describe an aspect of the product that is to be built and tested by the team, and which provides a new user experience in the product. User Stories are common in software development, but there are always some deliverables that teams will need to develop that do not provide any user experience. Examples of these include research ("Spikes"), infrastructure (upgrade or replace a third-party product on which our product depends), and so forth.

I have long documented such non-user-oriented deliverables as "Technical Stories." In terms of format, these differ from User Stories in lacking a role but contain all of the other standard Story elements.

Development of hardware products does not follow the software pattern of implementing functionality as a vertical slice across some set of technologies. Hardware products are more component-oriented than layer-oriented, and components cannot be modified incrementally with affordable cost. Thus, Stories become more component-oriented and less usage-oriented.

The result is that hardware products have a much higher ratio of Technical Stories to User Stories than software products. In my experience, for software and hardware products both, Product Owners mostly write User Stories, and the Team members write Technical Stories. This pattern means that Team members write most of the Stories for hardware products, which is the reverse of what we expect for Story authorship in software products.

Thus, the inability to develop functionality in the form of vertical slices across technologies does not imply that Agile techniques cannot be used, although it does shift the balance of Story types and authorship.

5.4 The Problem of High Skill Specialization

The "generalizing specialist" concept, where most team members can operate in multiple skill areas, is not usually possible for hardware-development teams. Here again, dealing with this issue is not necessarily difficult, but the patterns differ from what we see in software development.

Specialization impacts Story Estimation and Sprint Planning in two ways.

- Story estimation must rely on time-based units (person-days) rather than Story Points.
- Sprint Planning often has to pre-assign work at the task level to Team members in the Sprint Planning meeting, as specialization means that Team members cannot work interchangeably on

different Stories. This differs from the common “swarming” technique where most people can work on most Stories and can move from one Story to the next without pre-assigning work in the planning meeting.

As long as we take into account the impact of skill specialization on how work is estimated and planned, it does not prevent us from using techniques that are common in Scrum.

6 Conclusions

The biggest conclusion I draw from this experience at Bird Technologies is that Scrum and Agile techniques apply as well to hardware-product and integrated-product development as they do to software development.

The second-biggest conclusion is that success requires understanding the different characteristics of hardware and software development. This is not so much hard to do as it is a challenge to learn for Agile experts who come from a software background.

7 Acknowledgments

I am grateful to many people for their contribution to this paper and the work at Bird Technologies.

Abid Akhtar, for his participation and contributions throughout the engagements with Bird Technologies.

John Winter, Terry Grant, and Ann Brooks at Bird Technologies, for making everything happen.

All the people at Cprime whose effort makes it possible for me to go to clients and help them.

Steve Adolph, my official “shepherd” for the Agile Alliance conference, whose patience and feedback for my various drafts is much appreciated.

REFERENCES

[1] Takeuchi, H., The new new product development game, Harvard Business Review 1986, Vol 64, Issue 1

[2] Thompson, Kevin W. “Agile Processes for Hardware Development”, 2015

[3] Thompson, Kevin W. *Solutions for Agile Governance in the Enterprise (Sage): Agile Project, Program, and Portfolio Management for Development of Hardware and Software Products*. Sophont Press, 2019