# Agile, Scrum, and “Hitting the Date”

Kevin Thompson, Ph.D.
www.kevinthompsonphd.com

May 2, 2018

## Contents

1 The Date Misconception ... 1

2 Agile Myths ... 2

3 Date-Oriented Delivery ... 2

4 How Long-Term Planning is Done in an Agile Way ... 3

5 Managing Scope and Dates ... 6

6 Conclusion ... 7

## 1 The Date Misconception

Our clients develop hardware and software products. Their development work happens over time, and always involves some concept of a date by which the work needs to be accomplished. The idea of a date-driven schedule is not new, of course. I imagine the pyramid builders in ancient Egypt had project schedules, too.

Scrum is highly date oriented. Time is divided into two-week Sprints, which have firm start and end dates. Longer time horizons are constructed to contain a particular number of Sprints. The concept of time is present everywhere, and all planning is very much focused on what can be done over some period of time.

Why, then, do I hear prospects objecting that, “We can’t use Agile here because we have to hit dates”?

This statement is ironic in two ways. The first irony is that these clients are not hitting their dates anyway. Their work commonly blows past planned completion dates, which leads to strife and finger-pointing.

The second irony is the misconception that Agile development is oblivious to dates, which is very far from the truth.

## 2 Agile Myths

I have long been troubled by myths that float around in the Agile world. One of the big myths is that, "At the end of each Sprint, a Scrum Team must be able to ship a product that has a new feature."

I believe much of this myth originates with the common statement, "Every Sprint should produce a potentially-shippable increment of product functionality." This is a good statement, but I believe it is often misinterpreted in the above way.

The reality is that neither software nor hardware products can be developed in a way that guarantees a new and usable capability at the end of every Sprint. I have seen people struggle hard to meet this impossible goal, and observed the burnout and cynicism that follows the inevitable failure.

What actually happens in a Sprint is that a Scrum Team completes and tests a particular set of deliverables. The deliverables selected for that Sprint are chosen based on their value, and dependencies on other deliverables. The overall conception of product value in a fast-moving market does drive product strategy to complete new and useful features as quickly as practical, to get useful products in the market as soon as possible. Since one Scrum Team can only develop product capabilities in a sequence, the overall sequencing is chosen to optimize the value produced in each Sprint. So while it may be the case that a particular Sprint wraps up a new and useful capability, this is itself not a business goal. Most useful things of interest to users of the product require more than one Sprint to develop.

I do not see a conflict between this reality and the "potentially shippable increment" phrase. I interpret that statement as being about quality, not delivery of capabilities. I read it this way: The quality of what a Scrum Team produces should be high enough at all times so that if the company decided suddenly to ship the product at the end of a particular Sprint, the users would be happy with the quality. This I do believe is a valuable point, and there are various approaches to maintaining high quality at all times. (I won't cover them here, but I do recommend them.)

## 3 Date-Oriented Delivery

In reality, product shipments are commonly scheduled weeks to months in advance, so the "let's ship it now" model rarely happens.

Organizations that produce software most commonly ship product upgrades on a cadence, such as every three months, every six months, and so forth. A small minority of organizations implements a continuous-delivery model, and may push out upgrades daily, but that, too, may be a cadence.

Organizations that develop hardware products also schedule product shipment well in advance of the big day. Hardware product completion does not typically follow a cadence model, but instead bases the ship date on the desired scope of the product, and various other business needs. One product might require nine months of development, and another only six.

The cadence and scope-oriented models arise from different drivers.

In software development, scope is easy to change. If we have a scope-oriented shipping model, and the ship date is six months out, the tendency arises for people to try to get their favorite features into the next release. The resulting scope increase ends up pushing the completion date out, which alarms other people and pushes them to get their next-favorite features into the release. The resulting ship date then moves further out again.

I call this phenomenon “Death by Scope Creep,” and it has been a bane of the software world for decades. If we move to a cadence-oriented model, the problem is largely solved. Only so much scope can be fit into the next delivery cycle, and only the highest-value items that can fit into the cycle should be implemented. If my favorite feature does not make the grade in this cycle, then it may go into the next.

In hardware development, we do not have recurring product upgrades in the same sense. Instead, we design a product that will be manufactured later, and ultimately shipped to customers. With respect to product development, the work is done when the manufacturing specifications can be handed off to the organization that will manufacture the product.

While feature creep can be a problem in hardware development as well, it comes at a much higher cost in time and expense than is the case in the software world. Hardware developers have to live within design constraints that are defined early in the development cycle, which tends to limit the amount of feature creep that can even be contemplated. Thus, a hardware product is “finished” in a much stronger sense than are software products. The scope truly is finite, and so the delivery moments are based on completion of planned scope, which may not align with any cadence.

# 4 How Long-Term Planning is Done in an Agile Way

All concepts of planning work over time have to deal with the basic tradeoff between scope, schedule, and resources. If we have a certain number of people, and a particular definition of scope, then the work will take a certain amount of time to do. This relationship is called the “Iron Triangle,” or “Triple Constraint.” If we change one of those three parameters, then at least one of the others must change as well.

The mechanics of planning work over time vary greatly between classic, plan-driven models and Agile models.

Plan-driven models (such as the “Waterfall process” for software development) start by defining the scope of a development project. This then leads to some concept of a Work Breakdown Structure (WBS), and an associated set of tasks that become the line items in a planned schedule. The tasks have associated work estimates, and the schedule is built by sequencing tasks based on dependencies and resource constraints. The completion date is then determined by the project duration and the start date.

The Scrum approach has conceptual similarities to what I just described, but differs substantially in practical details. We do create an analog to the WBS, but the language is product-oriented rather than

project-oriented. "Epics" describe major aspects or features of the product at a high level, and are decomposed eventually into "Stories" for the finer-grained deliverables that are developed and tested in Sprints. The work for each Story and Epic is estimated as part of the planning exercise. The number of Stories and Epics scheduled into a Sprint is constrained by the team's Sprint Velocity, which is the total estimated amount of work the team can do in that Sprint.

In Agile language, the planning horizon beyond the Sprint is usually called the Release cycle. Unfortunately, the language is misleading. The term originated with the idea that a product would be released to customers at the end of the Release cycle, but the reality is that there is no necessary link between the end of a Release cycle and any concept of a product being released. Actual product releases to customers or manufacturing may occur more or less often than Release cycle boundaries.

Practical Release-cycle lengths generally range from one to three months, with three months (a quarter) being the most common. If the product's development period exceeds three months from start to finish, then that period is usually broken into multiple Release cycles for planning purposes.

The meeting at which planning is done for one to a few Release cycles is known as the Release Planning meeting. The mechanics of Release planning are too complex to present in this article, so I will simply state that the Release plans produced in these meetings map Stories and Epics to specific Sprints and Scrum Teams, over the period of interest. A key element of each Release plan is the identification and planning of cross-team dependencies, which must be managed throughout development in order to prevent work from being blocked.

A Release plan differs from a classic "MS Project" schedule or Gantt chart in a few ways:

- The items in the plan are deliverables, not tasks.
- The format is tabular, with rows for teams and columns for Sprints, rather than a list form or Gantt chart.
- Cross-team dependencies are shown by links (for a physical planning board, these are ribbon, yarn, or hand-drawn lines) between predecessor Stories and Epics and their successors.

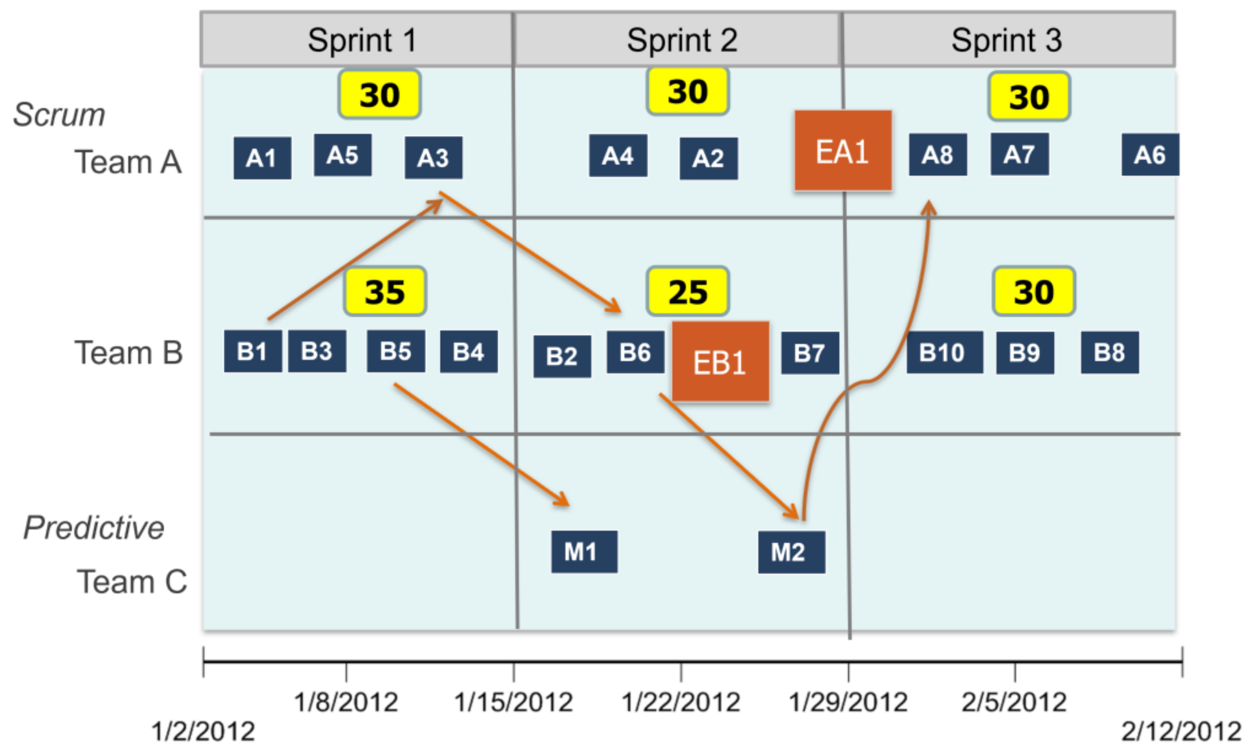


*Figure 1 Hybrid Release Plan*

Figure 1 shows an example of a "hybrid release plan," meaning one that incorporates not only Scrum Teams, but also a team that uses classic project scheduling (labeled "Predictive" in the figure). The Scrum-team rows show Stories (A1, A2, etc.) and Epics (EA1, EA2). The yellow boxes containing numbers show the estimated velocity per team for each Sprint. The predictive team row shows milestones associated with dates. The arrows show dependencies, and point from predecessor items to successor items.

The exercise of Release planning differs from more classic approaches in key ways:

- Product Owners and Team members write the Stories and Epics in advance of the meeting.
- Work estimates are done by entire Scrum Teams, not a subset.

A major difference in Agile Release planning compared to classic project planning has to do with the authoring of specifications. The Product Owners and Team members for Scrum Teams write the specifications for deliverables, as opposed to lead engineers or some other subset of the organization.

Another difference is that all members of each Scrum Team collaborate to generate work estimates, using techniques such as Planning Poker and Affinity Estimation. The rule is that the people who do the work are the ones who estimate the work, and the entire team must participate in the estimation process. In classic project-management approaches, estimation is often done by technical leads, as opposed to all of the hands-on people.

Finally, and reprising the team theme, the plan is produced by the in-person collaboration of all Team members across all participating Teams. This, too, differs from common project-planning techniques that may have a project manager creating the project schedule.

# 5 Managing Scope and Dates

The schedule generated in Release planning either shows the work to be done for a particular cadence (e.g., every three months), or the work and schedule required to complete a fixed-scope conception of the product. Either way, the plan is as reliable as a plan is likely to be, but it is also understood that the plan will not survive contact with reality, and we will have to adjust to unexpected developments.

One strategy for dealing with unknowns is some form of buffering. Classic project schedules do this by adding time buffers, while Agile processes do it by reserving some capacity for unknowns. (I suggest planning the Release cycle to 70% of the Teams' forecasted capacity, leaving a 30% reserve.)

Other strategies for hitting dates require changing dates or changing scope. Depending on the details, either or both may be acceptable.

I want to focus on scope management here, because this is where Agile approaches shine. If a completion date is fixed (for example, we must exhibit at a trade show), then we have to manage scope in order to hit the date.

But what does "hit the date" mean? For many people I've met, it means, "get all of the work done on the promised schedule." However, that is a demand, not a solution, and it is often impossible to meet. The interesting cases are the ones where that demand cannot be met, so I want to consider those now.

The key is to recognize that products should be built to achieve business goals, not to check off all of the line items on a detailed wish list. It is common to define business goals in terms such as "support a load of at least five times the current limit," or "support a frequency range from 50 Hz to 22 kHz."

A business goal does not normally map to any fixed conception of product scope to be developed. Different technical solution may yield very different types of work to be done. In addition, and crucially, for my focus, business goals can often be achieved with solutions that are less ambitious than desired.

For example, consider a case of frequency analysis for radio-frequency signals, for which two approaches are possible. The elegant approach provides greater stability and precision, but pushes the current state of the art. The less-elegant approach has less stability and precision, but is cheaper and faster to implement. Both approaches will meet the business goals, but use of the more-elegant solution will provide additional "bragging rights" and also serve as a foundation for next-generation products.

We select the more elegant solution, and work commences on the product. The plan appears achievable at first, but then reality sets in. Various unplanned events occur, to the point where it will no longer be possible implement all of the planned scope. In this case, we might elect to implement the less-elegant, but faster and cheaper, solution, so that we can still achieve our business goals on the planned date. We

can revise our plan and get to work on the new direction quickly, no later than the start of the next two-week Sprint.

# 6 Conclusion

Scrum and related Agile-framework concepts are very time-oriented, and provide effective solutions for achieving business goals on required dates. The combination of whole-team participation in scope definition, work estimation, and planning improves the reliability of development plans relative to alternate approaches. The deliverable-oriented nature of the long-term plans, combined with the flexibility to change plans every two weeks with a Scrum process, enables rapid change in direction when needed. These capabilities enable us to achieve business goals on planned dates even in environments characterized by substantial uncertainty. We achieve these goals by monitoring progress daily, and modifying scope in response to challenges as needed.

Never make the mistake of equating business goals to a specific scope definition. Goals are important, but a scope definition is simply one path to achieving the goals. If one scope definition will not get us there, we can often change the scope so that we can achieve our goals.