Agile Release Planning - My practical methods

One real-world approach to initial agile release planning for one team, assuming continuous release plan refactoring

Joseph Little

Agile Release Planning - My practical methods

One real-world approach to initial agile release planning for one team, assuming continuous release plan refactoring

Joseph Little

This book is for sale at
http://leanpub.com/joesagilereleaseplanning

This version was published on 2017-06-12

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Introduction

This book is mainly about initial, quick, up-front *Agile Release Planning* (ARP). We define ARP as the initial work done to create the *Product Backlog* and release plan before you start Sprinting.

ARP is about people, and getting them to work together.

In this book, we are talking about my usual Agile Release Planning approach, meaning that it is:

- For one team
- My usual way (your way may be different; even I may do it differently in some situations)
- For a relatively small set of work (I use six months as the example)

Agile Release Planning is:

- More about the people than the plan
- Assuming the centrality of knowledge creation
- Assuming continuous refactoring of the plan
- Assuming you will use common sense in applying these ideas to your situation

So, this book has a limited scope.

Planning, Not Plans

The first thing to emphasize is that the initial plan is only the initial plan. We immediately start to do what I call *Release Plan Refactoring* every *Sprint*. This is the idea that for every Sprint we refactor the release plan to reflect all new information.

Some of you have heard about *Product Backlog Grooming or Product Backlog Refinement*. To me, Release Plan Refactoring includes that idea (or those ideas) and more, as you will see.

The basic idea is that we are continuously planning instead of choosing to "plan the work upfront, and just work the plan."

To summarize: We are using planning to help us learn as fast as possible, and to deliver a

better product (at the time of delivery) and more quickly.

Emerging

A key idea is that we do not know everything upfront — of course! — and that new knowledge is continuously emerging during the course of the effort. This is one of the key reasons we must continuously re-plan.

Early Edition

This is an early edition of this book — Version 1.4.01.

We have been making improvements, as we make it better for the LeanPub format. See LeanPub.com[1].

I particularly want to thank Tom Gray, Jane Little and Kerry Lengyel for proofreading it and fixing many of my errors.

I consider this version 'ready,' but I am still open to feedback.

I urge you to send me comments at jhlittle@leanagiletraining.com

About this book

This book is for beginner to intermediate teams in typical situations. Your mileage may vary (YMMV), as they say. Advanced teams, particularly, may wish to try other approaches, although we think we have some ideas here that will challenge and improve your practice is this area.

We discuss, in later sections, *Release Plan Refactoring* — the revisions to the release plan that should happen every Sprint. Some changes will be minor and some major, but it is essential that revisions be considered and probably made every Sprint. Too many things are changing in the real world.

This proposed approach is probably not for everyone. I am not trying to propose a "one and only one" pattern. Still, this is a pattern that I think works in many or most situations, and it gives the basic ideas I try to apply in almost all situations. But it might not fit your situation.

[1] https://leanpub.com/joesagilereleaseplanning

When first doing this, I recommend a team select the next six months of work for that team. From experience, this seems about the right size. Enough work for them to learn how to do *Agile Release Planning* (ARP). And not so much work that it becomes tedious. (If it is tedious, they will not learn ARP as well.) If you have 2 years worth of work, then I recommend the first time, you only plan the first six months of work this way.

Also, obviously, this should be one of the top sets of work for your organization — a top product or project.

When learning ARP, keep it to only one team. KISS (Keep It Stupid Simple).

If the work is that size — about six months — then I want the initial release plan to be done quickly, in about one day, up-front. (It might take a bit longer.)

This book describes what I would do for ARP given the above situation.

I never believe in the initial plan. It must be improved every Sprint. It will never be perfect. It must be continually changed to adapt to new information and to a changed situation, and change is of course always happening.

Note: Despite 'not believing' in the initial plan, sometimes we are forced by circumstances to make important business decisions based on the initial information or forecast. I think we can do that to some degree, as I will discuss later, but these are risky business decisions made with 'too little' information. Still, most business decisions are made with incomplete information, and all business decisions are risky, to some degree.

If done with reasonable professionalism, I think the risk is somewhat reduced compared to a similar waterfall plan.

Warnings

Not all situations require *Agile Release Planning* (ARP). For example, if you release every *Sprint*, then release planning (or even a product roadmap) may not be needed at all.

I especially warn you against taking too long to do release planning. Many people think too much up-front, expecting to arrive at close to perfect information. In our business of new product development, I have never seen anyone have close to perfect information up-front. Usually, to use technical terms, we are lucky if we have C-level information up-front. (C-level information is below A-level or B-level information. It might be defined as 'not too crappy.') But then, we are at *Day 0*, which is the day of the effort when we are dumbest.

Some people in Agile recommend that we think very little (almost not at all) up-front. This is consistent with the 'no-estimates' thinking. I think, for virtually all of the situations I am in, this is not useful advice. For example, useful information or disagreements will not be sufficiently discussed.

So, use good judgment in balancing these concerns — both, in my experience, are legitimate.

Comments on the ‘no estimates’ idea

In a sense this book itself is an extended comment on the ‘no estimates’ idea.

But first, let’s define the ‘no estimates’ idea. The idea is that it is not useful, and maybe worse, to estimate any work. Usually, but not always, this idea is joined with the idea that all estimates are terrible and have to be terrible. And, usually, also joined with this idea, is the idea that managers are ‘bad’, at least in the sense that a manager will use any estimate in a bad way.

So, first, let’s agree on two things.

Estimates are often inaccurate. Significantly inaccurate, especially at first.

And managers can and do mis-use estimates sometimes. This is true in our opinion and it is sad. It may not be happening at your company, but it happens far too often out there.

But….

Estimates are needed. We have to make business decisions with the best available business information, even if that information is weak. A business needs this, and customers need this. Time is important, and ‘expected delivery time’ is a key factor in decisions that businesses and customers must make.

Estimates are not always terrible. And people can learn how to estimate or plan better.

More importantly, in the process of planning, much learning happens. So, we agree, do not take your estimates too seriously, and do not take any plan too seriously. Things will change. As many have said, it is not about the plan, but the planning (by the people).

We think you and others can make managers better. And managers can learn. So that managers will no longer 'blame us' when we cannot fulfill the initial plan. But this is an important and difficult education process in many cases. And it will not always be successful. You have to decide when to give up (and some of you should, and move to a different manager, or maybe a different company).

If you are a 'no estimates' person, let me ask that you not pass judgment on the ARP idea until you have done this work with me in a workshop. If you are already convinced that no-estimates is correct, it is extremely unlikely that any words will convince you otherwise. But an experience might. Try the experience. Let me add: It is not the main purpose of this book to address your concerns. We are mainly trying to help people who want to do planning and estimating, and do it better.

Why Agile Release Planning?

Here are some values, principles and practices around Agile Release Planning (ARP) that I have taught to numerous teams. In general, the teams have found them very useful. While you may not find all of them useful, I hope at least they will make you think, and at least that thinking will make your results better.

The phrase 'release planning' is apparently a loaded phrase for many people. My suggestion: When talking to others, be sure you are talking about the same thing, and expect good communication to take longer than usual.

We probably should define release planning — I find there are many different definitions out there. I can be certain that some readers are right now imagining a very different elephant than what I am trying to talk about. You may be imagining something closer to waterfall release planning — and I mean agile, adaptive release planning.

I led a discussion at the Ottawa Agile and Scrum groups some months ago.

I asked the group to come up with some reasons to do release planning. In essence, these reasons represent the meta 'purposes' of release planning, and these purposes are what we should optimize in our practical approach to doing ARP.

The group came up with many good ideas, almost all of which I agreed with.

Here are some purposes that are not always mentioned, and I think they are critical:

1. **Team building**. A set of work or exercises that allows the newly formed Team or *Scrum Team* to start to think of themselves as a team, and do some Forming, Storming, Norming and Performing to start to become a team.
2. **Realize where they disagree**. Often we think people agree what the work or product is all about. In waterfall release planning, we have no means (or only a poor means) to identify where we disagree. If ARP could only identify (better) where we disagree, then we all could learn from that.
3. **Start to get the whole team on the same page**. For example, it might be useful for the whole team to be, at a working level, on the same page about the product features and the *effort*. This might include: the *vision*, a good-sized feature list, where the *Business Value* is highest, what will cost the most to build, etc.
4. **Knowledge creation together**. By exposing different explicit and tacit knowledge, the team can create new knowledge about the Business Value, the product and the work. This, in my view, is extremely valuable. They can also share, across the team, knowledge that each individual may have.
5. **Motivation**. The team's motivation increases in at least three ways — probably more. *One*, they feel they are no longer being given the 'mushroom treatment.' So, this de-motivation is gone. *Two*, they feel they were part of creating this work. This is no longer the Project Manager's project and they are helping. It is my project

> (or our product). *Three*, they feel a sense of relative power over the work. No longer is the work the large unknown thing — it is much better known — and we (the team) have all shared and created knowledge about it. *Four*, they hopefully start to feel that 'this is good' and the customers will like it, so they feel a sense of purpose about the product, and about their work as a Team. This motivational effect is quite important. Where there is a will, there is a way.

If you see these purposes of ARP as important, then you want to do it a different way. I think #3, #4 and #5 are especially important.

The Story Begins

One day in 2011 I got an email from Mark, an attendee at a course some months prior.

He was working at a software company that builds software for railroads, and he wanted to start a new software product that would support managing the repairs of railroad cars (a big business).

He said, "Joe, I want to get a whole bunch of people in a room and figure this product out. We are in the early stages, we think we have some customers, and I think I can get them to collaborate."

This started the conversation.

He became the *Product Owner* (PO) and I became the *Scrum-Master* (SM) at least for purposes of the 'release planning' work that we did.

Later, I meet with his group that included both internal people and external customers, and we did most of what is described here.

That was the first time I did almost all of the things described in this book.

The exact way it starts for you may be similar or may be a bit different. It does not have to begin any particular way.

And eventually you identify some people — often the PO and the SM.

Usually, some sort of rough 'planning' has been done (I call this generically "Level 1" planning), and then they give it to a team, and the team (with others) does "Level 2" planning.[2]

Next steps

Mark and I discussed the people. *Who should we include?* We agreed to include the whole *Scrum Team* and about five business stakeholders. In this case, all five were real customers.

It made me worry (eg, too many people), but it seemed like the right thing to do. We did not expect all the customers to agree, and we did not expect all the customers to be equally 'useful' (in the sense of identifying features that would lead to a great product for many customers).

But it seemed best to listen to them — at least the customers who would take the time to participate.

We discussed the mechanics. We would teach them Scrum and Agile (enough), and then do a 'workshop' where we actually built out the *Product Backlog* and identified the real releases as best we could on *Day 0*. We would then be honest with them and say, "We expect it to change."

What Happened

Mostly the right people actually showed up.

The training went well, and the workshop went very well.

[2]Level 1 planning and Level 2 planning are discussed in a later section.

They did most of what is described below, and they had some of the issues or questions indicated in the discussion.

Mark felt he wanted to make some changes to the Product Backlog afterward. He felt every customer was not equal, and so, in the workshop, some of the user stories were not prioritized as he wanted (all customers were not equal, even though we treated them that way in the workshop).

Overall, Mark was happy.

The team was happy.

The customers were happy. Even though they did not fully agree with each other, they felt good about the transparency. They understood how each company (each customer) was different, and that trade-offs were necessary. They completely understood that not every feature can be #1 (in fact, only one can).

Overall, a lot of learning occurred and the stage had been set for a successful effort.

What Happened Next?

This experience was a watershed for me.

Some months after the workshop with Mark, I spoke to my friend Catherine Louis[3]. She said, "You need to add a workshop to your courses." I said, "No way. They can't take it." She said, "You must. This is the kind of thing I did in my old company, and it worked." I said, "OK, let's give it a try."

I fully expected the trial (which was ARP) to prove that a third day workshop was wrong, and that it would not work.

[3]Catherine Louis can be found on LinkedIn or at http://www.cll-group.com/

In fact, it proved the opposite.

And it made me see that people need this concrete exercise in applying many of the basic ideas of Scrum and Agile before they can really 'get' Agile and Scrum. Attendees said this, and continue to say this, to me in lots of different ways.

I have now been doing this workshop almost every week for 5 years. It works every time. I will not say that every 'team' (the 4 or 5 people at each table) does it equally well. But every team (table) does it well enough. And it always works.

So, I want to share these ideas, values, principles and practices with you. YMMV (Your Mileage May Vary), but by now many, many people have found these ideas useful. I think there is a good chance you, too, will find them useful.

By now, I have done the Agile Release Planning workshop close to 200 times. Let's assume on average 15 people in each workshop. And virtually 'every time' it has worked. Maybe five people out of the 3,000 have left the workshop feeling 'this won't work for me.' The more typical responses are: "Wow, this is great." "Now I see how to do this." "This is better than what we have done so far." "I want to replace the current Product Backlog with what we did today. [Often the 'team' in the workshop included no one else from his company.]" "We have to do this at my company."

I also get the response: "Wow. This is great. I am not sure I can get my company to do it." So, we are talking about some change. And I believe you can actually lead the change. Perhaps I believe more that good change will happen than you do. In any case, for now, how to get good change to happen is not a major topic of this book.

My Approach — Summary

The People

The people are most important.

Surprisingly, this idea is a surprise to many people.

Who are the right people for this work?

I expect this to be done by the Scrum Team (*Product Owner*, *ScrumMaster*, Implementers) (\~seven people in a team) and the business stakeholders (\~four people).

The Process

To call the things I describe below simply 'process' or process steps is somewhat misleading, but let's go with it.

Here's what I think *Agile Release Planning* (ARP) should comprise.

For a set of work for about five to seven months, most individual teams should do this in about one day. It might bleed into the next day, especially if you take long breaks. The ARP will have good enough quality to then start the first *Sprint* the next day.

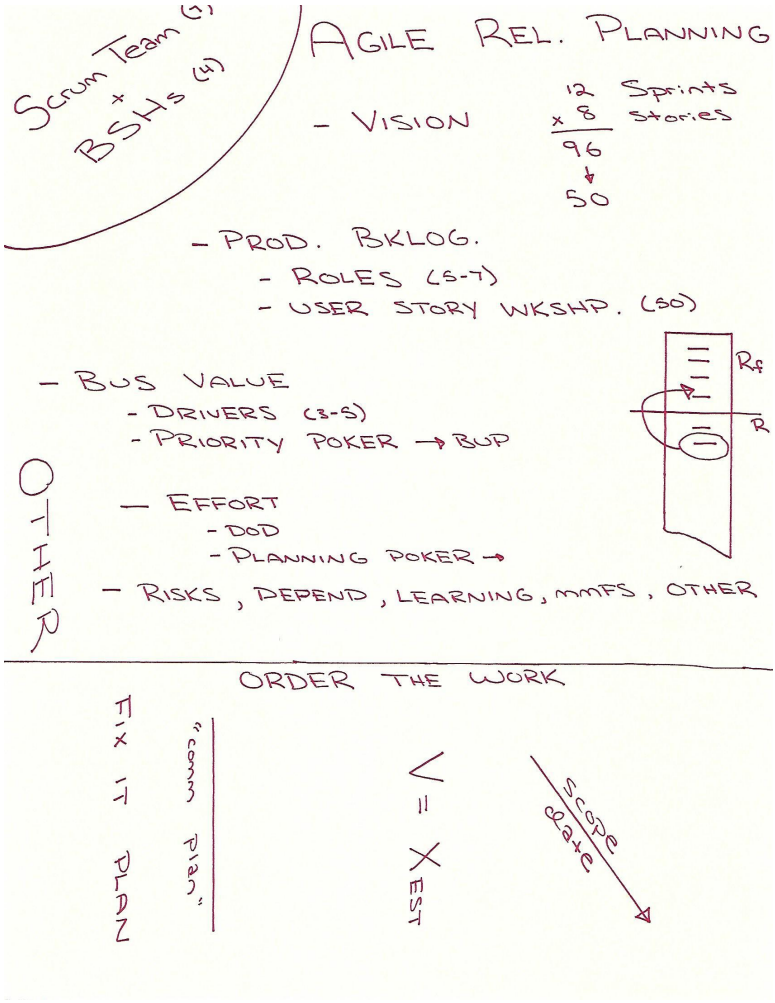
Again, we are not 'done' once we have the initial release plan. We must do Release Plan Refactoring every Sprint. This continuing adaptiveness of ARP is probably its most essential aspect.

Perhaps the two most essential ideas behind that are:

- We don't know everything up-front (by a good margin).
- Both good change and bad change will happen during our efforts, and our planning must adapt to those changes to mitigate the bad changes and to maximize the value from the good changes (e.g., learning).

Again: I do not guarantee that release planning is needed in all situations. If you just want to build a small *Product Backlog* and start the first Sprint, this book is not saying that is wrong. Still, I have done this approach now with many, many teams, and it seems to have worked with all of them. My experience is that everyone I have led through this process has liked it after they did it, they thought it was worthwhile and they actually did almost all of it.

Here's the picture of ARP that I usually draw. (next page)



Note: MMFS stands for Minimum Marketable Feature Set. See "Software by Numbers" by Mark Denne and Jane Cleland-Huang.

What do we have? For those who can't read this drawing well:

Agile Release Planning

People: *Scrum Team* (7) and *Business Stakeholders* (4)

Process:

Vision

Product Backlog

- Roles (5-7)
- User story workshop (50 stories)

Business Value

- Drivers (3-5)
- Priority Poker ⟶ BVP, Business Value points

Effort

- DOD, definition of done
- Planning Poker ⟶ Story Points (SP)

R= BVP/SP

Risks, Dependencies, Learning, *MMFS*, other

++++++++++++++++++++++++++++++++

Order the Work

Scope-Date trade-off

Velocity = X (estimated)

“Communication Plan”

Fix It Plan

OTHER

All of these short titles are explained below.

Release Plan Refactoring

After the initial ARP, we start doing *Release Plan Refactoring* every Sprint. A typical pattern might be to do three Sprints and then 'release the product to the public.' Nonetheless, in every Sprint we assume the plan can still be improved, and that doing so is useful compared to the cost of doing it (relatively low). So, RPR every Sprint until the release is complete.

The dumbest thing is not to learn. Surely we can learn at least one thing each Sprint that can make the release plan a bit better.

The purpose of RPR work is to learn and then change and improve the plan. If we learn, we need to refactor our release plan.

We certainly hope sometimes that things will not need to change very much, at least not in a bad way. Remember also that learning is good change. And we want to take advantage of good change. So, for myriad reasons, things will change, at least some.

(Of course you and your team are making every effort to learn faster and to discover change as early as possible. That should go without saying.)

Comments

As I said earlier, the real value in doing this is *not* the 'crappy' initial estimates that the team arrives at after the initial release planning. Stated too simply: The real value is that everyone is now '*on the same page*' about what the elephant is. At least

we are a whole lot more on the same page than we ever were before, which is tremendously valuable.

If the team does really bad, or no, release planning, I think it increases the chances that the stories will be too big. This means that lots of stories just can't get to 'done, done' in the Sprint. So, in that and other ways, good release planning is linked to having good Sprints! Now, the problem of bigger stories can be fixed later, but other problems could also arise. To avoid being stuck in a mess, try to do Scrum as professionally as possible from the beginning.

My Approach - The Details

Now I will describe how I typically have teams do this — how I have a decent team work through initial *Agile Release Planning* (ARP).

So, we will take the same ideas we just summarized and walk you through how to execute each step.

Note: This book only describes ARP for one team. In a later edition I may add notes about how to do this in a scaled situation (multiple teams).

The People

Why bring so many people?

We have to get them all seeing the same elephant, as we say, and they all are needed to have good knowledge creation on this subject.

If they all come, it becomes 11 people. That is really more than I like in a meeting (about seven people is much better), but who can we not invite?

What will we accomplish?

1. They will all see the same elephant.
2. They will all be more motivated.
3. They will all share their tacit knowledge.

These three things are hugely valuable.

Vision

The first step is *vision*.

We define the vision, and make sure that all the attendees see the same vision and agree with it — or at least start to align with it.

Who: The full *Scrum Team* and the business stakeholders should do this together. The Scrum Team includes of course: the implementers, the *ScrumMaster* and the *Product Owner*.

The *business stakeholders* (BSHs) provide key input about what the product should be. They are chickens; that is, they are not be full-time members of the Scrum Team. But, they will be key to success.

BSH is not a defined Scrum role — it is a concept that I use to refer to a small set of people outside of the Scrum Team.

Often what I mean by BSH is different than what 'business stakeholder' means at your firm now.

I usually think of the BSHs as being four people, but of course the number can vary. If the Scrum Team is seven and the BSHs are four, we already have 11 people in the release planning meeting. At this size, it is starting to have the communication problems of a big group, but who can we not invite? So, the size is a trade-off between efficient communication and being sure we have enough good inputs.

The BSHs may be customers or internal managers, or really anyone who will participate enough (every time) and provide

valuable input. It is mainly the PO's job, in my view, to make sure these are the best possible people, given the specific situation. Often the PO must seek help in selecting the BSHs, or in getting better ones.

In my experience, the BSHs are never perfect representatives of 'the customers' of the product. Sometimes the level of imperfection is quite impressive. Often, once the imperfection is identified and made visible, it can be corrected.

Typically the BSHs represent two main groups: (a) 'the customers,' meaning usually or mostly the external and/or internal end-users, and (b) the firm, usually personified as the widows and orphans who ultimately own the shares of the firm.

What: By *vision* we mean a relatively short statement of what the product will be, once completed. The vision includes what it will be, why we or the customers will be excited about it and a few more scoping details.

The vision establishes the 'north star' of our direction — it hints at what the business value is.

And it is like an elevator statement[4] — something short that you could explain in two minutes to your boss's boss.

We particularly like Geoffrey Moore's format, which comes from his book, "Crossing the Chasm":

- For (target customer)
- Who (statement of the need or opportunity)
- The (product name) is a (product category)
- That (key benefit, compelling reason to buy)

[4] http://en.wikipedia.org/wiki/Elevator_pitch

• Unlike (primary competitive alternative)

• Our product (statement of primary differentiation)

In addition to these nice words, we also strongly recommend you find one or two key numbers to ground the vision.

The numbers make the vision more real — the words are usually quite sweet, while the numbers ground the ideas in some reality. For example, the numbers typically make the product, in the minds of some of the attendees, either bigger or smaller than some thought from the words only. The numbers make the words more tangible.

Examples:

"We expect to make $3 million in the first year."

"Widget production will go up about 125%."

Whatever metric (or two) makes most sense.

Why:

- To clarify the direction of the product.

Yogi Berra: "You have to be very careful if you don't know where you're going, because you might not get there."

- To motivate all parties — this is quite important.
- To set a direction and a rough scope. ("This is about the iPad 3, not the iPhone, nor any other iOS gadgets.")
- To start to get everyone on one page. My experience is that this is difficult. Why? My best guess is that at this level of abstraction, it is easy to have different understandings.

We will mitigate this last issue by quickly making it less abstract by building the *Product Backlog*.

How: We find this typically takes roughly 20 minutes, often less. Occasionally more, especially if the BSHs will not agree on the vision.

We find it is useful to do 15-minute time boxes to check if there is too much aimless or circular talk, and to check for those talking too much and those talking too little.

We recommend that the 'smart guys' (those who think they already know the vision) talk, and the 'least informed' (typically, one of the implementers, who usually is just learning the vision) physically write the vision. This helps assure that the 'least informed' learn it better.

Later, the PO, who may be better with words, may improve the wording.

Remember that perhaps the most important goal is not to have knowledge (in one person's head), but to spread the knowledge into the heads of all these people.

Ending: Usually the team forgets to identify a key metric or two. Go back and do that.

Then, ask the whole team for 'the thumb metric.'

Say to them: "If the thumb is pointing straight down, this is the worst project you have ever been on. If the thumb is pointing straight up, this is the best project. Half-way, this is just an average, typical project around here. So, one-two-three, show us your thumb."

Then, the PO should lead a discussion of the results. Sometimes this means improving the vision statement. Some people are by nature skeptical at this stage, and we just learn that.

Sometimes it is just a short discussion.

Note that this is a reminder that one of the purposes of the vision exercise is to get them motivated, and we are learning now how well this exercise did that.

Roles: The PO is leading the content. The ScrumMaster is leading the facilitation. The SM checks the time boxes and tries to assure that each person talks the right amount. The PO is always assessing: Have we said the right things? Is the group appropriately motivated?

Later: The war is not won or lost in the first hour of battle. The vision is a starting point — hopefully a good start — but some projects can start with a great vision and then still get bogged down later. Some projects start with a clearer vision, some with a less clear one.

The vision is typically improved later on, in any case. For those who may be uncomfortable moving on, remind them of this.

Product Backlog

After we complete the *vision*, we must develop the *Product Backlog*.

There are two parts to this.

1. We must define the roles to use in the user stories.
2. We must write user stories. I call this second part a user story workshop.

If your team prefers to use something else than user stories to populate the Product Backlog (as the Product Backlog Items), then the wording of this section would be changed a fair amount, but I think most of the key ideas still apply. All of the teams I work with have found good success with the *user story* format.

Roles

Let's break this down.

Assumption: We have a new team that is doing Agile-Scrum for the first time.

Who: We want the whole *Scrum Team* (PO, SM and implementers) and the *BSHs*.

Usually you want the three to five best business stakeholders you can get. They are never perfect, but at least they are

the best you can get. The *Product Owner* (PO) is usually the main person driving which BSHs to bring in. Certainly the PO should have significant influence over who the BSHs should be.

The Product Owner is leading the *content*. The *ScrumMaster* (SM) is leading the *facilitation*.

What: We want about five to seven roles. These are the roles to go in the User Stories. (Not the 'roles' in the Scrum Team.)

How: Brainstorming to get five to seven roles is usually pretty easy. Sometimes it takes more time. It depends on the team and the situation. Typically five minutes, or maybe 10.

We mostly want different user roles (personas, actors), and usually they are the end users of the system or product, either external or internal. But more generally, we want any role that would want some feature in the product, even if that role does not directly use the product.

Please avoid a common misconception: These are not the roles of the Scrum Team members.

Some teams might come up with 30 roles, while some might come up with only two. We want to either synthesize or break down until the number of roles gets closer to the five to seven range. There's something magic about that range, but not worth dying for at this point. If they only have four roles, things will probably still be OK.

Whatever they come up with for roles, it will probably work the first time, but the roles will become better as they start to get more practice with stories, and appreciate, tacitly, the characteristics of a good role.

User Story Workshop

By a *user story* workshop, we simply mean that we have the 11 people start to self-organize and create user stories. (For now, a user story is a short sentence that explains one feature.) It is a workshop in part because we do recommend that they have a coach there ready to answer questions. Later, the SM can fill that role.

The first time we ask them to do a user story workshop, they may get scared. They may think this is going to take a long time. So, we may need to calm them down.

One way to do this is to tell them how many (or how few) stories we need. Often they think they will need hundreds of stories and that this will take weeks.

For 6 months of work, we tell them we need about 50 stories. And this may take about 30 to 45 minutes.

Here is the thought process:

It is typically best, the first time a team does *Agile Release Planning* (ARP), to be working with about six months of work for one team. As we said earlier, you will guess it is five to seven months worth of work when you start. This is not too little work (where they don't learn enough), nor too much work (where this initial planning starts to be too tedious for some tastes).

I won't explain all of the reasons, but that is enough work without being too much work or too little. (Later, after they are more experienced, they can handle ARP with more or less work — or you may just have more or less work. So, later, it is OK if the project is only three months or is bigger — say, 16 months.)

If we have about six months of work, let's do some math.

My rule of thumb is: In a good normal Sprint of two weeks, it is best to have eight or more stories. Six months of work is 13 Sprints. 13 Sprints x eight stories = 104 stories. Now, those 104 stories are all small enough to fit in a Sprint. I call them 'Sprint-sized stories.' (Stories too big for a Sprint I call 'epics.')

In the initial Agile Release Planning, we can be pretty happy with somewhat less granularity than what we will need in a Sprint. So, we divide 104 by two. That gives us about 50 stories, and they start to be at about the right level of granularity. This is what experience shows. So, 50 stories covering the same amount of work would be good (i.e., on average each story (epic) is about twice as big as a Sprint-sized story).

This level of granularity will give us a rough idea of what we are shooting for now (i.e., 50 stories).

Clearly, we are not shooting for 500 stories, nor for five stories. We are shooting for a manageable 50 stories. This is the key idea. This knowledge enables some of them to relax; this session building user stories will not last forever!

So, we only need about 50 stories that cover about six months worth of work.

Experience shows they can often do a decent job in less than 30 minutes. Some groups will take a full 30 minutes. And some will take 45 minutes. And the quality is good enough for today. (They can also improve the quality of a few stories later today.)

If we ask the group (the Scrum Team and BSHs) to produce 50 stories and they only produce 42, we are probably still at a good level of granularity for initial release planning. But, if

we only ask them for 42, then often they will only produce 32, which is not enough granularity (the work is described at too high a level).

If you already know the work is more or less than six months (for one team), then you have to adjust this ballpark number (adjust the '50' up or down).

Typically at about this point, we give the participants markers with 'blunt' points (eg, 'fine point' Sharpies) and index cards or Post-It notes (4x4 inches or 4x6 inches). This enables everyone to write and requires them to be succinct. When a 'card' (story) is written, anyone in the group can read it from some reasonable distance.

Having the roles, the whole group starts to write stories. We let them self-organize. We give them 15-minute time boxes. At each 15-minute break, they "check in" and they inspect progress and see if they want to adjust anything. Usually they see that they want to write stories faster and worry less (e.g., edit less).

Sometimes the ScrumMaster has to say: "It is ok if the stories are not perfect. Let's just get something on the table we can learn from."

Typical productivity is: one person can write one story per minute. But, given the group and with more people, they slow down. So, it takes the group 30-45 minutes to produce 50 stories. No one will die if the group takes an hour (although I do not recommend that). If your team is faster or slower it is usually not a big deal, but that gives you a ballpark for the timing.

I have seen a team of four people produce 54 pretty good stories in 15 minutes. This was a small team, so they had an

advantage.

When they feel finished, we recommend having the whole team gather around all the stories (imagine the 50 Post-It sheets on a wall). They should look at them and ask: What needs improving the most?

Maybe a few aren't worded well. Maybe a few stories need the INVEST criteria a bit tighter. Maybe two team members now identify three missing stories.

What were the INVEST criteria?

- **I***ndependent* (we try to maximize this, but we never can make all stories independent)
- **N***egotiable*
- **V***aluable*
- **E***stimable* (clear enough that we feel *effort* can be estimated)
- **S***ized Appropriately*
- **T***estable*

Note: I believe that Bill Wake at xp123.com[5] deserves credit for the INVEST criteria.

The INVEST criteria represent a way of thinking about the stories, and a way to check to make them better.

It is sometimes useful to get more sophisticated than what we have described above. I don't like to do that the first time - I prefer KISS (Keep It Stupid Simple). They need to have at least one cycle of just doing the basics, probably several cycles.

[5] http://xp123.com/articles/

We recommend that the PO answer questions and talk about the product that he envisions. The PO should let others also identify the specific stories. Particularly at first, the PO should not try to be prolific — he should let others contribute. Writing stories gets them more engaged. They become more creative and, being more engaged, their motivation improves and their learning improves. The PO can always add or modify later.

Is the Product Backlog perfect? Of course not. In waterfall and in agile, we never have all the stories on Day 0. Never have, never will. Like 12 lawyers at the bottom of the ocean, it's a good start.

Why?

Why do we have the whole Scrum Team and the business stakeholders?

For several reasons:

1. We want all of them to see the same elephant.

   We like the metaphor of the **6 Blind Men and the Elephant**.
2. We want the group to create knowledge together.

   A bunch of things happen as they create knowledge. One is that they all start to form a similar mental picture (or pictures) of the product and the effort and of things related to it (e.g., the architecture).
3. We want the team to develop motivation together.

   Having been involved in the creation, the whole thing becomes their 'baby.' This is far better motivation than we get from 'the mushroom treatment.'

(The mushroom treatment is where the team is kept in the dark and fed manure. This is great for growing mushrooms, but not so great for growing teams.)

4. We want the whole group to share (most of) the tacit knowledge they have with the rest of the group.

   It turns out that everyone has some knowledge or ideas to share, or at least some good questions. Almost everyone does share if you get them engaged and talking.

Yes, it is somewhat expensive to have everyone involved, but the payback that comes during the project is very high — very high — and the time to market is better.

Time Box: As indicated, I like to have a series of 15-minute time boxes where the team checks in. *Are we going too fast or too slow? Anything we should change? Who is talking too much, who too little?*

Often the initial user stories can be written in 30 to 45 minutes, sometimes less, but it is not a big problem if it goes to 60 minutes. The main thing is, as the SM, if one member of the team gets talking and worrying and nothing is getting done, you can't let the team just spin. Someone must fix it and get the team productive again.

Also, in each team there are often some people who want to go faster and some people who want to go slower (often these are 'the perfectionists'). The SM must guide the team in deciding the right speed to go. If anyone starts to get too uncomfortable it can make the session less productive. So, sometimes the SM must use those facilitation skills.

Common Issues: I like brainstorming rules. Meaning: Anyone can create anything, and for this time box (say 15 minutes), no critique is allowed. Then a small editing or 'fixing'

time box where the team reviews and improves the stories. Then maybe another 'create anything' time box, and so on.

I find creating one story and then criticizing each one is too slow, and that process inhibits the team too much — especially beginning teams.

Another issue is having only one person write the stories. If a team really wants to do this it is not terrible, but things usually go faster if everyone writes, or at least multiple people write.

What about sharing the stories? This will happen for sure later; everyone will read and discuss the stories when we talk about *Business Value* and *effort*. Still, I think it is useful now if, as a person writes a story, he or she shares it with the group (says the words of the story, in the story format, out loud). That way, no one writes the same story a second time.

Using the computer: I do *not* recommend using a computer for this work, unless it is unavoidable.

I strongly recommend that the whole group be collocated, and that the stories are written together in the same space. As we said, probably on index cards or Post-It notes (4x6 in. or 4x4 in.). The knowledge creation and learning is much better when everyone is engaged, and using physical things, rather than distracted on various devices.

Top Down/Bottom Up: I find some people are top-down thinkers and others are bottom-up thinkers. Both kinds are useful. Let them be who they are, especially while they create. Fairly obviously, top-down thinkers will tend to write epics that need to be broken down further. Even at this point we can often see that some epics are just too big, so we can identify the bigger epics so that we can break them down enough to get close to 50.

Bottom-up thinkers will write stories that sometimes are too small. (We might group some small stories into an 'epic' or theme, if it is useful. Typically not.) The more typical problem is that the small stories leave out some key features.

The User Story Format: I like the user story format, and I encourage it.

The format is:

As a [role x]

I can [action y]

So that [explanation z].

The people especially tend to forget the "so that" clause. So, I encourage them to include it. But, if they just can't think of the feature in a user story format, I say, "OK, just write something as a *PBI* (Product Backlog item). Maybe later we will convert that into user story format."

Go easy on the beginners — they are just learning to ride the bike.

Results: Usually the team ends up with 42 to 55 stories that represent five to seven months of work. At this point, the duration is simply a rough 'ballpark' estimate. (It is just a gut feel.) Still, it is useful as a good start for the release planning. A bit later in ARP we will improve that estimate.

These user stories (or PBIs) are just the right middle level of 'features' for everyone to have a clear enough picture of what the product will be. It embodies the vision. It makes things concrete for people without getting mired in details. Wonderful. And they can do this the first time.

Is it perfect? No. Will it ever be perfect? No. Did we spend a reasonable amount of time to get a level of quality (in all

aspects) that is useful? Yes!

Will we write more stories, as we discover them, later in release planning? Yes. It always happens! And will we add more stories as we are doing the Sprints? Yes, always (although not necessarily every Sprint).

Business Value

Now we move on to *Business Value* in *Agile Release Planning* (ARP).

As Yogi Berra said: "You have to be very careful if you don't know where you're going, because you might not get there."

This quote might be funny, but more importantly, I think it reflects our fundamental problem. Which is: How do we identify what the customer really wants?

The work the group will do now on Business Value is one answer.

When we did the *vision*, we directly or indirectly probably talked about Business Value.

Now we want to talk about Business Value for each individual *PBI* or *user story*.

Again, we want to have two actions within this section:

1. Identify the drivers of Business Value
2. Do *Priority Poker*

People: Again, to do this work, we want the whole *Scrum Team* (PO, SM, implementers) and the BSHs (business stakeholders). Again, the BSHs are the best people we can get to attend release planning to represent the customers and the firm well. We also expect these BSHs to be the the people giving useful feedback in the Sprint Reviews.

Business Drivers

It is my contention that the key thoughts and phrases or concepts we want to use about Business Value vary by the situation. They are not usually consistent from product to product, or effort to effort. We need to make them specific to the situation, and specific to the product, the customers and the people involved.

The two biggest ideas about Business Value are fairly consistent, at least usually. They are (a) satisfy the customers and (b) maximize shareholder wealth.

We are now trying to make these BV ideas more concrete and specific.

Business drivers may involve things like making money, risk, customer satisfaction and many other things. In any given occasion, BV could be very different things.

So, the whole group gets together and then discusses and agrees on the top three to five business drivers that apply to this specific set of work (the product), and the work represented by the existing Product Backlog (its PBIs). Sometimes this will lead to the identification of some new stories (or PBIs).

Priority Poker

Now that we have the three to five drivers, we can start Priority Poker.

You may be familiar with *Planning Poker*, which is where we use the Planning Poker(r) cards or Agile estimation Fibonacci

cards to vote as a team on the relative *Story Points* of a to-be-estimated story compared to the one Story Point on the *reference story*.

Priority Poker is very similar to Planning Poker(r), except that it is about Business Value, not *effort*. Also, we assume there is no correlation between effort and value.

So, some similarities are:

- use a team of 'experts'
- vote (and re-vote)
- use Fibonacci cards
- discuss assumptions
- share knowledge and ideas
- average the numbers after reaching some relative consensus
- the resulting number (an integer) is comparative or relative
- done fairly quickly
- somewhat imprecise for each individual story, but fairly accurate over a set of stories

Some differences:

- a different set of voters than Planning Poker (our best 'experts' on Business Value)
- the reference story is the LARGEST Business Value story rather than (usually) the SMALLEST effort story
- prefer the real Fibonacci sequence at the top end (with effort the modified Fibonacci is not a bad idea)
- done in the presence of the implementers (so they can learn)

How we do it

So, first the top 'experts' (usually between four and seven people) on *Business Value* huddle around the user stories (or PBIs) and determine which single card (story) has the most Business Value. This biggest BV story is arbitrarily given a value of 100. (It is minor whether you use 90 or 100 or 144. I've come to prefer 100.)

The Business Value 'experts' are typically the PO and the *BSHs*, and maybe one of the implementers who has a lot of experience with customers. Or for some other reason, knows BV pretty well.

Let's be honest — the so-called experts on Business Value are not always experts. In fact, it might be said that no one is an expert on Business Value, in part because it is a very complex subject. So, when we say 'experts,' we mean the best people you can find to do Priority Poker.

OK, now we have the *reference story* for Business Value. Then the panel starts to vote on relative BVPs (Business Value points) for all of the other user stories (or *PBIs*).

Here are the rules:

- pick a story randomly (we do *not* want to anchor the panel by already putting stories in BV order)
- discuss the story
- identify drivers of BV that are relevant for that story (but don't get worried about exactly how big each driver is)
- do not let anyone say (now) whether they think the BV is high or low (it will become apparent as soon as they vote)

- ask each person to pick a Fibonacci card, but to not reveal the vote to anyone (the card represents the net size, in overall BV, of this story compared to the reference story)
- once everyone has selected a card, 1-2-3, they all reveal their cards simultaneously
- the persons with the highest and lowest cards discuss why each was high or low. Others too can comment
- the panel does not have to agree with any of the comments
- the panel re-votes until the results are within about three consecutive Fibonacci cards of each other (say: 13, 21, 34)
- then they average the numbers, to the nearest integer

Example: In the second round, the vote is 21, 34, 34, 55. The value for that story is 36.

Meaning: If the reference story is 100, and the vote averages 36, that means that they feel that the new story has 36% of the Business Value of the reference story, considering all of the different drivers of Business Value.

The other people listening (the non-voters) may ask questions after a story has been BV pointed. The purpose for them being there is to pick up the tacit knowledge about Business Value, which stories are more important and why.

Final review: Once all cards have been assigned BVPs (Business Value points), the experts then gather around all of the cards on the wall and look for the 'stupidest' numbers. Often in 50 cards they will identify three or four that seem stupid now. Maybe this 30 card seems a lot bigger than the other 30s. Maybe that 70 seems like it should clearly be lower. The

person identifying a 'problem' card can ask the panel to re-vote. In general, we would expect them to re-vote on a few. After all, now that they have discussed them all, they are probably smarter now about Business Value than when they started.

Timing:

Priority poker takes some time, but is worth it. Understanding Business Value is very important.

Of course, one person in your group may talk too much. So, the facilitator (the SM) must monitor that and address it.

We find a time period of 45 minutes to 75 minutes is usually quite sufficient for 50 cards. But again, if the discussion is deemed valuable by most of the participants, maybe going a bit longer is useful.

Do check-ins with the Team every 15 minutes. Assess the progress, and make minor adjustments. Typically, some members are talking too much, and some too little. Make the appropriate adjustments.

Be careful if anyone starts to feel 'we are wasting too much time here.' At least try to help them see the value, or perhaps try to make the rest of the group go faster. The SM must balance between the people who feel we are going too quickly, and those who feel we are going too slowly.

Sometimes one of the voters will be upset that the BVPs are 'wrong.' One suspects it may be the ego of that voter, but possibly that voter may be right. Remind him or her that with new evidence, we can re-vote the BVPs at any time.

Comments:

I personally feel that BVPs are best when they represent how

much an end user will get excited or happy. Remember that we will take care of dependencies and 'essential' features (that may be un-exciting) by another method later.

When we break down an epic, we do **not** assume all 'child' stories should have an equal BV number.

For example, we may have to do five steps in a business process, but only Steps 2 and 5 (when automated) are exciting to the users. The others are just chores that must happen. Steps 2 and 5, when we deliver them to the business and deliver them well, that's when the big smiles come out — or that's when the big money is made. So, we don't say that all five steps have equal value; we say that Steps 2 and 5 are big and the other steps are smaller in BV terms.

If an epic is worth 100 BVP, does that mean that when it is broken into five stories that each story is worth 20 BVP? No; as we just said, we don't assume that. In fact, along with Pareto, we assume that there are 'more vital' parts of the 100 story, as well as less vital parts. In the simplest world, one of the smaller stories would be worth 80 BVP and the other four stories would total 20 BVP together (the 80-20 rule).

Sometimes, perhaps usually, when we break down an epic, we find that the total Business Value of the child cards is not equal to the BVP of the parent. This is normal. By breaking it down, we become smarter.

Often as we do Priority Poker, new stories are identified, or a few existing stories are deemed so big, that they must be sliced and diced into smaller stories immediately. This is normal.

Sometimes an expert may feel he does not know how to vote on a specific story. He must take his best guess anyway, learn from how the others vote and learn later from how customers

react when they get that feature.

Averaging: Apparently there are many who have been taught *Planning Poker* and told to force the team to consensus on one Fibonacci card. This is incorrect for Planning Poker. The research shows that averaging is more accurate and a bit faster, and the same applies to Priority Poker.

Accuracy:

Are the BVPs on each and every story perfectly accurate? No!

Have we learned a lot more about our different ideas about Business Value and shared that throughout the group? Yes!

Did the numbers help? Yes!

Can we change the BVPs later, once we become smarter? Of course, by team vote.

Will we change some? I absolutely expect it. You will want to get smarter and smarter about Business Value in multiple ways. This should in part be reflected in changes to the BV points.

Results/Outcomes: One clear result is that we have a BVP number on every PBI or user story. This is remarkable, and no one in the group thinks the numbers totally suck, although individuals may disagree about some specific BVP numbers.

Note: As we said earlier, if the panel feels some of the BVPs are 'wrong,' they can re-vote them at any time.

More importantly, the whole team has shared a bunch of ideas about how the BV is distributed cross the features. Every feature is not equal — this is important learning.

Typically new stories are identified, and typically the MMFS (minimum marketable feature set) is starting to emerge.

Often, differences in views about Business Value between the different BSHs are starting to get resolved without the PO having to use political capital to resolve it. This is often a huge win for the PO.

But the main thing, to me, is that the whole group starts to share relatively specific tacit knowledge about each story — specific down to the user story — not hand waving up at the higher vision level. They share some relative specifics about what the story (feature) really is (and why it is), and about the Business Value of that story in context.

Doing the BVPs changes motivation.

And it changes the behavior later.

It changes these for the panel as well as for the other Scrum Team members who are listening. The business guys start to respect that the geeks understand them a little. The geeks now understand better why business-side work is so hard.

Another important value in doing the BVPs is getting closer, and will be revealed soon. Just a few more steps…

Effort

Now we move on to effort in release planning.

What we want to do is estimate the relative effort of each *user story* that we have so far.

So, imagine that we have 50 user stories representing roughly (based on gut feel only, at this point) about 6 months worth of effort (ballpark, 5 to 7 months).

Now we have to do two things:

- Establish a *Definition of Done* for the team.
- Do *Planning Poker*, which gives us a "*Story Point*" estimate (number) for each user story.

Definition of Done

[Some of you may be wondering: What is the difference between the acceptance criteria for each story and the DOD? If so, please check the Glossary.]

Each Team typically has one DOD, that defines when a story is 'done, done' in a Sprint (then it is considered 'complete' or finished in the sprint). When a story is done, then the Team earns the story points toward their velocity for the sprint.

Now, how to create a DOD and what could one look like?

Borrowing from Taiichi Ohno[6], I suggest we do the *DOD* differently than many people do.

What others do is a list that describes what 'done, done' means once we get there.

Maybe they say:

- Requirements described
- Coded
- Unit tested
- Basic documentation written and reviewed
- Functionally tested
- Regression tested (small test)
- No bugs (all identified bugs fixed)
- Product Owner review (any issues fixed)
- No increased technical debt
- Promoted to the QA2 Server

What I would prefer is greater clarity about *how* we got to this state.

As one benefit, this often reveals that 'no increased technical debt' is very difficult to actually do. (Don't get me wrong; I am strongly in favor of minimizing technical debt. I just want us to be honest that that goal is not easy to achieve.)

What Taiichi Ohno proposed is that we ask the workers to write down the process that they currently use.

Once the workers do that, they themselves can see that it has weaknesses, and once the process is more visible, then everyone can help improve it. Specifically, the workers themselves

[6] http://en.wikipedia.org/wiki/Taiichi_Ohno

will improve it, and they start to 'own' the process. This makes for better motivation and better results.

Some in Agile are concerned that, by writing down the process, we have locked the process in stone. And thus, we treat people like machines.

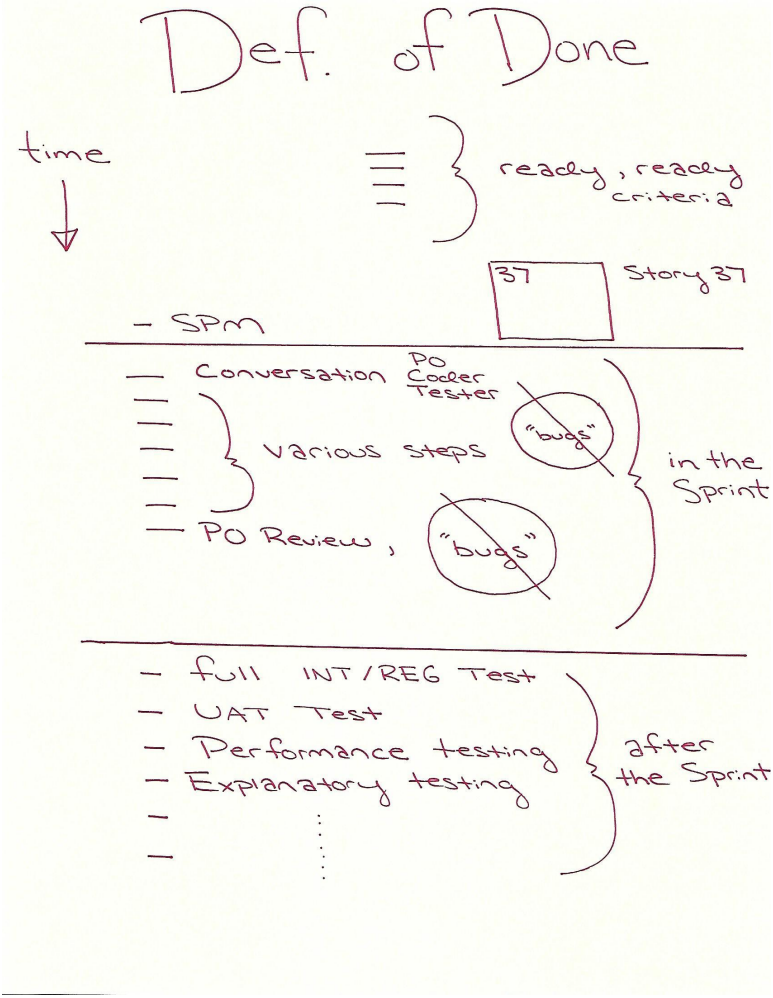
But this is actually the *opposite* of what we are doing. We are making the process visible so that it can be improved. So that it can be changed, and not so it will remain the same.

By making the process visible, we enable anyone who sees the process to have a somewhat educated opinion about it.

If the team is not strong enough, then a 'bad' manager could try to force them into his process. If you have a weak team and a bad manager, this is a real concern, but this seems unduly negative in the general case.

The process describes how we get a typical story or PBI to 'done, done.' The process should apply to most stories, although probably not to all. The DOD does not apply to special kinds of stories or PBIs (e.g., preparing a specific document if one must be done as a separate piece of work).

Let's make our suggestion more concrete in this picture. (next page)



So, before or during the Sprint Planning Meeting, the team can do many things to "get the stories 'ready, ready.'" The team makes visible what these things will be in that list.

Then they have the Sprint Planning Meeting.

Then the question is: What kind of process are we as a team

going to agree on to get one normal story (story 37 as an example) from here to 'done, done' during the Sprint?

The team agrees on its own process. The first step might be a (presumably very short) conversation among (in my example) three people about that specific story: The PO, the coder and the tester.

Then the team defines other steps in a reasonable amount of detail.

As the last step (or at least a latter step), I recommend they include 'PO Review.' This means that the PO reviews the story and gives a thumbs up or thumbs down. If there are problems, then the team has to fix them. I call those problems "bugs" (in quotes), but they are not bugs in the usual sense. They are things that need to be done to get the story to where the customers will like it (in the opinion of the PO).

In any case, whatever they agree to as a team, that's the Definition of Done. There may be some tough conversation to come to agreement.

Purpose: One purpose of expressing the DOD this way is to make clear what is being done in the Sprint, and what is being done before and after the Sprint.

What we estimate (in the SPs) is what is done in the Sprint only. The work before or after must be managed another way (possibly with another card).

Anything that is done after the Sprint — to make that story into something that can go in the live production — all that work is listed as well, below the line. Meaning: It is work after the Sprint, and hence not in the Story Pointing. That work represents all the bad news getting better with age, but we

have to accept that we can't always get to "live, in production, in use by the customer" within the Sprint. At least at first.

We think this approach to the DOD gives much more clarity or transparency.

It starts the team on the road to becoming more professional. It enables the team to improve their own process. This is key. They agree to one process and make it visible. This is *not* to ossify the process, but rather to make it easier to change and improve. And to enable the team to know, better, on a more consistent basis, how far 'done' they are — how much progress they have made.

Planning Poker

Now we come to the point of describing *Planning Poker*. This is a continuation of the discussion of *effort*.

Planning Poker has been described before in the Agile-sphere, so this will be brief. It is described at greater length many other places.

Some basic characteristics:

From the people we have, we select the five best people to vote on effort for this work. (About five, maybe three to seven.) In practice this means the implementers in the team. Often, to call them individually 'experts' in estimating effort is a stretch. The idea is that they are the people who will do the work, and probably understand their own strengths and weaknesses better than anyone else.

So, for motivational and other reasons they must do the estimating.

In practice, they may be bad at estimating at first. But with maybe a bit of time, they typically are the best at estimating the effort for the work of their team.

Usually the business stakeholders (as I call them) will not stay around for this work. This is OK; not ideal, but OK. We will bring the *BSHs* back in later. You have to decide when later is.

We use the Agile estimation cards with the Fibonacci scale.

The approach is wide-band delphi. *Delphi* means we use the best 'experts' we can find. *Wide-band* means that we allow the experts to talk and learn from each other. This bears strong resemblance to the knowledge creation ideas from Takeuchi and Nonaka.

The basic technique: We want about five 'experts' on effort from the team.

The experts are the people in the team who will be building the product. This affects motivation. This affects their knowledge. This will affect their later behavior. We try hard to get all of them, so that none of them feels left out.

The Product Owner is not one of the voters, because the PO does not know how to do 'real work,' or because the PO will not do any real work in the team this time, or because the PO is biased (or perceived to be biased). (One can imagine an exception to this rule.)

But the PO is there.

The PO has to make sure that the team is understanding each story correctly. If there are questions about the story, the PO must make the final decisions. Typically the PO is a business person, and in any case is representing the firm

and the customers. So, when those kinds of questions come up, the PO must be there to give the team the best possible assumptions to work with.

The *ScrumMaster* is there to facilitate the meeting or the work (e.g., to try to get the quiet ones to talk more and the talkative ones to talk less). The SM helps them use the time box effectively. The SM makes the estimators get all the cards estimated in some reasonable time-box. (One story is on each card.)

In this discussion, we will assume all the *PBIs* are in the user story format, so we will call them stories.

The experts must now choose the *reference story*.

Imagine that the team has 50 stories in the *Product Backlog*, covering about 6 months worth of work — the experts choose from that set the smallest effort story.

Note: The reference story should not be too small. Ideally it is about one ideal person day (8-10 ideal hours, after adding together all of the ideal hours from all of the required skill sets). Someone knowledgeable outside the Team should check that the story is about the right size. The SM might arrange this, but the SM should not tell anyone exactly how that person determined that the reference story was 'not too small, not too big, just right'.

Do not let the voters start thinking in units of time (days, hours). Still, if the story is too big or too small, it starts to distort the voting, in my experience.

The reference story is arbitrarily given a one — this means it has an effort value of one Story Point. (Doing things this way makes the numbers become somewhat more useful…trust me.)

Note: An earlier section talked about how the team must have a strong *Definition of Done* (DOD). The DOD should make clear what things are being done in the Sprint to get the story 'done, done' as we often say. So, it is the effort of the things in the DOD during the Sprint that we are estimating.

OK, now the team estimates the relative size of all the other stories in comparison to this reference story.

It happens in this way:

Select the highest value story. Compare it to the reference story (the one SP story).

Discuss it briefly to assure everyone understands the new story the same way. Perhaps the PO describes the story. Perhaps the experts ask the PO questions.

No one should say anything that suggests whether the new story is big or small. We are only discussing 'what is it' and 'what is involved in building it'.

Someone asks, "Any more questions?"

If the answer is no, then each expert privately selects an estimation card that best represents his feeling of how much bigger the new story is in comparison to the reference story. For example, if about seven times bigger, the choice is between a 5-card and an 8-card. Likely, the 8-card will be chosen.

If all experts are within three consecutive cards of each other (e.g., all have either a 5, 8 or 13), then average the numbers. Example: 5, 5, 5, 8, 13 averages to 7. The average is rounded to the nearest integer. The average does not have to be a Fibonacci number.

If the experts' votes are more dispersed (e.g., 3, 5, 8, 13, 20),

then the people who voted the two extremes talk (mostly). In this example, the person with the 3 and the person with the 20 both talk about their assumptions or reasons, etc. If the assumptions are business-related, then the PO can decide which assumption is correct or more correct.

With the new information, the experts can do a new round of voting. Again, they vote privately, and reveal their votes all at once.

Voting and discussing can go on a couple of rounds. The final answer for a card is the average after the experts get within three consecutive cards of each other (e.g., all votes are 2, 3 or 5).

If, after X number of rounds, the experts have not reached some degree of consensus (within three consecutive cards), then the team should just take the average. Each team can decide how big X should be. My guess is three or four rounds.

Once the number is decided, it is written on the card (story) in such a way that everyone knows that it represents the Story Points. Color-coding and placement on the card usually make that clear. For example, in the lower right corner in red ink.

Comments: The discussion is actually the most valuable thing. The numbers drive a better conversation about more important issues.

Sometimes important assumptions are identified. In that case, each assumption should probably be recorded (perhaps on the back of the card). Later, if we discover that the assumption is incorrect, we can refactor the Story Points for the related story or stories.

Multiple cards can have the same number of Story Points (e.g., two cards can have 23 Story Points).

Sometimes the experts will want to address certain general issues. The issues might be architecture or design related. This is where the SM has to use good judgment. Typically you should let them discuss for awhile, but not too long. Maybe make one or two drawings. Then get back to Planning Poker. Sometimes one or two people on the team want to discuss these issues too long. In this case, the SM must use good judgment and get the experts back to Planning Poker.

You will hear of people suggesting that the experts must agree on one Fibonacci number (e.g., 8) for a given story. This is *not* recommended. First, it gives a strong incentive for people to too quickly start to shade their numbers rather than vote what they really think. They find that George, the senior guy, is stubborn, and to avoid conflict and to keep things going they decide, almost subconsciously, to start voting as closely to what George says as fast as they can.

The research also shows that the average is more likely to be the more correct estimate, and it actually takes less time to reach that number, usually. (OK, yes, the division is annoying.)

Often one or two *new* stories are identified while we are doing Planning Poker. This is normal. Occasionally, the new stories turn out to be very important.

At the end

It usually takes 50 to 75 minutes to estimate 50 cards decently — sometimes a bit less.

Don't let them take a lot longer in this first pass. They will be able to revise the estimates later, any time they think they have more information. Many of the stories will be broken down later, and then re-estimated. And always for each story

we will collect more information later, and that could cause a re-estmate.

The first few story cards take longer, then things speed up on average.

Sometimes, after the panel is well into the process, they will get stuck on one or two cards. As long as it is only a few, this is reasonable and not a problem.

Once all story cards have been estimated (for effort), all of the cards should be put on the wall, and then the whole team should gather around to see if they can identify one or a few badly estimated cards. By 'badly estimated' we mean that they just feel that the SPs on these few story cards are notably inconsistent compared to the other cards.

It is very common that two or three stories will need to be re-estimated, in the team's opinion. Often, these are cards that were estimated early on, when the panel was dumber. This is fine that they re-estimate them — good even — and normal. They now, as a team, are much smarter than when they started Planning Poker.

The R Factor

Now that we have a number for *Business Value* and a number for *effort*, we can calculate the *R factor*, or ratio.

One or two people in the team calculate the R ratio for each story — R = BVP / SP — and they write the number on the story card.

The R should mainly be expressed as an integer, i.e., rounded to the nearest integer. If the integer is 0 or 1 or maybe 2, then use one decimal place (e.g., 1.4 or 0.8).

Depending on which words you prefer, the R factor represents:

- return on investment
- bang for the buck
- cost-benefit analysis, or
- low hanging fruit

We next suggest that the group organize the story cards (the Product Backlog) based on the R number — highest R first, lowest R last.

The PO's main goal is to maximize the Business Value delivered from the team to the Customer (or the Firm, depending on your philosophy). *Ceteris Paribus* (other things equal), this is how the work should be done to achieve that goal.

Now ask the group: What do you think? Is this the right way to do the work?

Usually they say: "Well, it is close, a good start, but we need to make some changes." Which is always the right answer in my experience.

What ideas do we use to re-order the work? More on that in the next section.

This R factor is a huge improvement. We never had a way before to compare benefits to costs. The R factor is the best guess by our best 'experts.' In business, that's what we act on until we have a better guess.

Is return on investment important? Absolutely!

Were we ever able to see it usefully in our projects? No, not the way we did things in the past.

Is the R factor perfect? Of course not. Both the BVP and the SP numbers are estimated by humans. But, anytime they can see that the numbers are wrong, they can re-estimate them and make them better. So, the R factor represents our best understanding at any moment — our best guess — therefore, a huge improvement.

And we will use the current information to enable ourselves to actively get smarter. Only a truly stupid team will not be able to get smarter.

Risks, Dependencies, Learning, MMFS, Other

Now we come to the point of (re)ordering the *Product Backlog*. It was just (see above) ordered based on *R factor* alone. This is never (in my experience) the final or best ordering of the work.

You will recall that the *Product Owner*'s main goal is to maximize the *Business Value* from the Team. In some time period (shorter or longer, as makes best business sense in your specific situation), and deliver that to the customer.

So, in theory the R factor (see the previous section) should be the way to organize the Product Backlog, *ceteris paribus* (other things equal).

But of course, other things are never exactly equal.

So, here we use common sense — that most uncommon element — and we re-order the work based on these factors: Risks, dependencies, learning, *MMFS* and other factors. (Each of these factors is discussed below. MMFS stands for Minimum Marketable Feature Set.)

Note: James Coplien has an excellent article titled "It's Ordered —Not Prioritized!" that discusses ordering more. It is on ScrumAlliance.org[7]. Coplien also relates this to patterns at ScrumPLOP.org[8]. I highly recommend the patterns at Scrum-

[7] https://www.scrumalliance.org/community/articles/2011/august/it%E2%80%99s-ordered-%E2%80%94-not-prioritized!

[8] http://www.scrumplop.org/

PLOP.

How should we do the re-ordering?

We recommend that anyone in the group can propose to move a user story card earlier or later, and he or she must explain (justify) the change with the team (especially the PO) and get reasonable consensus. Anyone can say 'I don't agree because…' The PO has final decision authority on the ordering.

How long should this take?

Normally this does not take very long. Again, six months of work for one team is typically expressed as about 50 user stories. So, re-ordering 50 user stories, where most do not move, does not take long. It might be done in five minutes, or it might take 20 minutes. The longer time might be needed only because explaining why a story is being moved, or debating the merits of that, might take some time.

A very typical situation is that five cards move, and it takes five to 10 minutes. When they take longer than 30 minutes, it is likely that the SM should have been reining them in.

So, let's discuss each factor in turn.

Risks. There are potentially many types of risk. Business risk is often a big one. For example, we need to get a feature out before a competitor, or we have a weak understanding of the specific detailed features needed in area X. Technology risk is another common factor. We are about to use new technology and we are not sure how it will work. There are also other types of risk. In Scrum, we tend to want to attack risk early by doing one or more stories in that area. For example, to see if the risk is just a worry, or a real roadblock.

Dependencies. Again, these can be of several types. In the past, we often organized the work mainly by technical dependencies. Since the goal (we now recognize) is to maximize Business Value, we sometimes must sacrifice the efficiency of the team to a degree. But if technical dependencies will seriously reduce the efficiency of the team, then we must deal with that. There can be business dependencies, as well. It makes more sense to develop Step 1 in a process before Step 2, for example.

Learning. We are knowledge workers, and knowledge workers learn. It can be useful to learn certain things earlier, and sometimes we can organize the work to make that happen. For example, we need to learn what the customers really want. We need to learn some technical things to become more effective. If we learn certain things earlier, it can give us a great advantage.

MMFS. Minimum Marketable Feature Set. This phrase is from "Software By Numbers" by Mark Denne and Jane Cleland-Huang. The idea is that we work hard to discover some minimal set of features that must be put together before a customer can realize the value of the whole set. Sometimes this minimum is quite small, quite small indeed. In other circumstances it is much larger. In general, too many of us (producers and customers) have been brainwashed into believing the 100%-100% rule, so that we think the MMFS is much larger than it really is.

In general, Pareto's 80-20 rules is closer to what we should do. We should do 20% of the work to get 80% of the business value.

In any case, low value features sometimes must be moved up to add the 'missing something' to make the next release truly

usable — to get to the MMFS.

Other. This is a catchall for all the other reasons we have to change the order of the work (the user stories).

My favorite example is this: A committee is going to meet in three weeks to decide on the funding for our project. George is on the committee. In our opinion as PO and in the opinion of everyone else on the team, George is much too excited about user story 87, which currently would not be built until the second release. But, George is on the committee and user story 87 is only four Story Points (our Velocity is 24). So, we ask the team to go ahead and get the story done in the next Sprint so that George is happy, and George will therefore give a positive vote to funding the project. Not rational, not 'the right thing to do,' but sometimes you have to deal with real people and irrational things have to happen.

In our experience, risks and learning should be used more often to re-order the Product Backlog and dependencies less often. But, in any case, using the R factor solely is almost never the right answer.

Comments:

We recommend that the Product Backlog first (already) be ordered by the R factor.

We recommend that the whole team be there (PO, SM and implementers) and the business stakeholders.

As suggested earlier, anyone in the group can start to suggest re-ordering the Product Backlog based on any of the ideas above (Risks, Dependencies, etc.). Any move has to be explained to the whole group. If there are disagreements, the PO makes the final decision.

Again, let me emphasize that sharing (tacit) knowledge with the whole team is at least as important as any other outcome we are trying to achieve, so doing this without the team is not recommended.

Completing the Plan

As discussed in the previous section, the user stories are now ordered.

Now we must complete the release plan.

So, we must make the trade-off between scope and date.

There are three ways to do this:

1. Fixed release date: We will release every X months or Y *Sprints*.

Some teams or firms prefer to have a fixed release date. It makes things simple. It makes managers and others realize that we will release. The only question is exactly what will be in the release.

1. Fixed scope.

We will release when all of the scope is (all stories or PBIs in that scope are) completed.

1. Trade-off.

We understand our Velocity and go down the Product Backlog one Sprint at a time, trying to see (a) do we have enough features and (b) can we release as early as possible. And

we speak to ourselves: "OK, two Sprints, how many features are done? OK, three Sprints, are enough features now done? No! OK, four Sprints, the customers would really like to see another release soon, the market needs it now, but do we have enough? OK, five Sprints, I think we now have enough. Let's shoot for that." It is that kind of trade-off.

Any of these three methods eventually require that we know our Velocity. Depending upon your preferred method, you might argue that estimating Velocity should come first.

Estimating Velocity

With an existing team, you might already know their *Velocity*. With a new team, you must guess.

Here is the calculated 'guess' for a new team.

It is a fancy calculation, and you can make a bit more elaborate. Nonetheless, it remains a SWAG[9] (you can google that). It seems to give a better guess, on average, than other ways of guessing.

Imagine the team has six implementers (people who do 'the real work').

Imagine the team will do 2-week Sprints.

Let's assume the focus factor is 60%.

What does 'focus factor' mean? It means, out of an 8-hour day, roughly 60% of the minutes are usable for the project. The other minutes are used talking about the ball game yesterday, taking breaks, eating lunch, getting interrupted, answering

[9] https://en.wikipedia.org/wiki/Scientific_wild-ass_guess

questions, reading emails, going to company meetings, filling out important company forms, etc. Maybe some work, maybe even very useful work, but not work on this project.

Lastly, for the one Story Point *reference story* (for effort), how many ideal person days would it take? The team huddles around that story and reaches a decision. Imagine the number is 1 SP = 1.25 ideal days. The ratio could be anything (e.g., 1 : 1 or 1 : 0.5 or 1 : 2, etc.).

Calculation for *Sprint 1*:

2 weeks = 10 business days

6 people x 10 business days = 60 days

60 days x 60% = 36 ideal days

36 ideal days x (1 - 40%) = 21.6 ideal days (See below.)

21.6 ideal days / 1.25 (ratio) = 17.3 *Story Points* for the first Sprint

We subtract the 40% as a start-up 'cost' for the first Sprint.

The 40% is based on experience and includes these factors mainly:

- the team is learning Scrum
- the team is "Forming, Storming, Norming, Performing"
- the team always wants to over-estimate what they can do in a Sprint

For the second Sprint, we recommend you subtract a 20% start-up cost. For the third Sprint, we recommend you subtract nothing.

You may find that these rule-of-thumb numbers need to be adjusted for your situation. For example, perhaps your team members all know Scrum, but the factors seem to work for most teams that are starting up.

The calculations for the next two Sprints:

Sprint 2:

36 ideal days x (1 - 20%) = 28.8 ideal days

28.8 ideal days / 1.25 (ratio) = 23.0 Story Points for second Sprint

Sprint 3:

36 ideal days x (1 - 0%) = 36 ideal days

36 ideal days / 1.25 (ratio) = 28.8 Story Points for third Sprint

Typically we would round these to: 17, 23, and maybe 29 story points for the first three sprints..

The velocity for the fourth Sprint is assumed to be the same as the third Sprint.

Finishing the Plan

Laying out the work in Sprints

The group now lays out the work in Sprints. These 4 stories for Sprint 1. We can do these 3 stories in Sprint 2. And so on…

You are 'buying' stories (story points) with the expected velocity of each sprint.

Then you decide how many sprints for the first release.

I find a typical situation is a first release in 3 or 4 sprints. But it can of course be more or less.

Number of Releases

I tend to put pressure on new Product Owners to release earlier and release more often.

It is remarkable how important speedy delivery of something is to the customer, and is to the business, in most situations.

I notice that most POs are too mentally tied to the concept of the 100%-100% rule. (We must do all the work before we can get any of the value, and that nothing can be released until everything is done.) This is almost always wrong. Hence, I always ask for an earlier release.

The POs usually say they understand the 80-20 rule, but in doing the work they tend to execute much closer to the 100%-100% rule. There are many reasons for this. (This topic, important as it is, is beyond the scope of this book).

Communicating the Plan

Once the PO and team agree on the scope and date, we then have to talk about the 'communications plan' as I call it. It covers the Who, What, When, How, Where of communicating.

First, the communication plan is how we will discuss with the right people everthing that they need to know. And the main topics are the obvious ones, often phrased as scope, date, and budget.

The communications plan is an on-going approach to explaining how the team will deliver on X date and, despite the inevitable changes later, will be perceived as winners (we

hope). It includes expectations management over some span of time.

Also, if the team works with a manager who truly understands adaptive planning (meaning that the current plan will be revised and improved every *Sprint* — that this initial plan is only the first guess at the plan), then tell that manager the truth.

The team talks about: Here's what we guess, this is what we are worried about, this is our feeling about how it is likely to change, and then we talk about any contradictory feelings we may have.

Often enough some key managers do *not* understand adaptive planning. These 'tough' managers (or customers) want you to give a fixed date on Day 0, and then deliver to that date.

In that case, you are stuck in a hard place. So, we have to do what we used to do in waterfall — add some 'contingency' (padding, buffer, etc.) to the date. This is to account for all the change that we know from experience will always happen later. It accounts also for problems, for people time being 'stolen' from us, for errors in the estimates, for new stories that will be identified later, etc.

It is difficult to guess how much buffer to add. We have no additional magic.

But, we do strongly suggest that you protect yourself and your team. Do not get them in a Death March trying to meet an impossible date. More about this later.

Now we discuss how the date will be communicated. (Almost always, the key issue is the date. The other two classic parts are 'the scope' (which features will be delivered) and the budget.)

The usual best thing to do is communicate nothing today or tomorrow, but rather to improve the plan over a few weeks or a few sprints, and then start communicating.

You have to start setting expectations that the date will change if other things change (substantially), and that the other things are likely to change substantially. This is not one conversation by the PO with the 'project sponsor,' but a series of conversations by the PO and the team with all the different groups of people who care.

The conversations span some time — from now until some time close to the delivery date. We want the people (e.g., managers and customers) over time to start to see and feel the power of adaptive planning.

For some of you, the issue is not so much "the communications plan," but, "What do we put in the contract?" A full discussion of how to use Agile Release Planning to address contract issues is beyond the scope of this book. Still, we can say that in our experience, this approach offers two major advantages for contract estimates compared to our prior approaches. One, it is done more quickly (in one or two days). Two, the quality is slightly higher than how we used to do it. Even though this is the first time the team has ever done it this way. And after the team gets more experienced with this approach, the quality (e.g., the relative accuracy of the date) is even better. But, honestly, it is still a *Day 0* SWAG. You still must add a significant 'contingency,' and it is risky.

I am not particular how you formalize the communications plan (you may write it down), but I do strongly advise that the plan be done with the whole team and with the *BSHs*.

In any case, I call all of this work the 'communications plan,' even though it is not about a plan per se.

The Fix-It Plan

Either before or after we do the communications plan, we also must identify a 'fix-it plan.'

The fix-it plan is the list of the top three to five things we want to do to improve the release plan that we currently have — the top three to five areas of focus for improvement.

Usually, one is: "We need to know the team *Velocity* more accurately."

Typically, one is: "Gosh, I know we are missing some features, some stories." So, the question becomes, how do we discover those features, those missing stories, sooner?

Sometimes, one is: "We are using this new technology, and it is risky." So, maybe a key issue is determining whether that risk is real or not, or how big it will be.

The fix-it plan is the work we focus on in the first couple of days after the Agile Release Planning day. (We assume that first Sprint will start in a few days, and when that starts, it also includes Release Plan Refactoring.)

One idea is to communicate nothing to 'anyone outside the room' until the basic fix-it plan has been done. At that point, the plan is much better — much more accurate — so that the amount of contingency can be smaller. Hence, we have more confidence in communicating.

'Finalizing' the Plan

The plan is in fact never final. It can be revised every day, although perhaps often the changes at some point can become fairly minimal.

Assuming you have BSHs (as I described them earlier), then in addition you typically also have to review the release plan you have with the Level 1 people (maybe called a 'Steering Committee'), a business sponsors and some (other) stakeholders, and maybe other groups. Perhaps also key customers. Get their buy-in or comments or adjustments. This, of course, may affect the communications plan.

Still, depending on what you say to whom, it can be perceived as 'final.' So, be careful. Do not get the team in trouble by the way you communicate. You must use common sense.

The Budget

In many cases, someone has already calculated the team's cost per *Sprint*.

So, let us use as an example: if the team costs $30,000 per Sprint, and the first release will take 10 Sprints, then the total cost of the first release is about $300,000. (This assumes no other external costs.)

In any case, you can see that the calculation of the budget is straight-forward once we have a cost-per-Sprint and a number of Sprints.

We have now completed most of the basics.

There are a bunch of issues we have not addressed, some of which I will address in the next section.

The ‘OTHER’

At the beginning I talked about the OTHER.

It is composed of mainly two groups of things.

1. Other steps or activities that you may want to add to the process (the agenda for the day).
2. Addressing *Infrastructure, Architecture and Design.*

Other steps or activities

There are lots of other things from which any given team might choose a few that it feels it needs to discuss during the Agile Release Planning day.

These might include architecture, design, scope issues (in scope, out of scope), organizational issues, defining the customer sets, identifying the legacy systems involved, key *impediments*, key risks, etc. Some teams like to identify key assumptions. Some teams like to identify key issues, etc.

In general, I think discussion of these ‘other things’ is good. Discussion of these other things must occur in some sort of time box, each in a time box.

I strongly support using a whiteboard or flip chart to draw and discuss these issues in a time box.

If the team finds this valuable, or your group has done certain things in the past that were very valuable, by all means add them to the process described here.

Be sure to time box the work and the discussion, otherwise you can spend a lot of time for relatively little value.

I-A-D

It doesn't always happen, but often someone says, "Great, we have most of the new features now, but we have a lot of foundational work to do before we can build those features!"

And when this comment comes, it is almost always correct to some degree.

I call this foundational work *I-A-D*: Infrastructure, Architecture and Design.

We are using the house metaphor. The idea is that we must lay the foundation before we can build the house on top of it. We must do the foundational work before we can build user features.

Now, again, in some situations virtually all the IAD is already done or is given to us. So, our IAD work is minimal or maybe very close to none.

In other situations (e.g., with some big companies) there is an Infrastructure department, an Architecture Dept., and a Design Department, and they must do 'all' of this kind of work. Even if that is the case, sometimes we still must do some work to integrate what IAD groups do with our own 'product.' So, typically we do the initial identification of all the work, and then work with the other groups.

Here is what I recommend doing with beginning teams:

Do not address IAD at first. When a team first starts doing Agile Release Planning this way, it is complicated enough. Do not ask them to create IAD 'stories' at the very beginning.

Create IAD stories. After the initial release plan has been settled, or mostly settled, then add in the IAD stories.

We often think of them as technical stories or IAD stories. Ideally we write them from a user's point of view, but it is OK if we write them from the point of view of a technology person.

We just want to be careful not to build too much IAD before we start building user stories. (In some places, this is an issue.)

Divide the IAD stories. A typical situation is that some of the IAD work will be done by the team, and some will be done by people or groups outside of the team. This varies a lot from situation to situation and even from product to product (or project to project).

For all the IAD stories that the team must complete, these stories must be added to the team's *Product Backlog* and included in release planning.

All the stories (work) done by people outside the team must be 'accepted' (by whomever will do it) and then tracked and done. So, for those 'cards,' the team may wish to use them to track the work of the outside people on a board (or part of a board).

Including the IAD stories in the Product Backlog. As we said, all of the IAD stories that the team will do must be included in its Product Backlog. This means those stories need BV points and *Story Points*. The BV should be what the customer would consider value, the 'wow factor' to the customer, not how important our technology people think it is. Typically to the customer, foundational work has little 'wow factor.'

We also calculate the R factor on each IAD story.

Often, based on R factor, these items would be placed low on the Product Backlog, but because of dependencies, we often move many or most IAD stories much higher on the Product Backlog. As the metaphor suggests, we can't build certain things without the proper foundation.

Sometimes the IAD has a significant impact on the delivery date for the first release. Often this puts us under pressure to figure out how to do the least amount of IAD and still get the a few key new features out in a first Release. This is a typical problem for us, and it is a problem that can be solved professionally or unprofessionally. It is a difficult problem; I don't like to give advice from a distance.

Closing Up

When is the Initial Release Planning completed?

To me, the team is balancing quality versus time. You want to fit into a rough time box, and do the best job you can in that timebox. You want the output to be of reasonably high high quality given the time-box.

That means:

- a good *vision*
- a *Product Backlog* that reflects, and will realize, that vision
- small stories at the top of the Product Backlog
- fairly accurate BV estimates
- fairly accurate *effort* estimates
- a reasonable ordering of the work
- each release is sufficient (MMFS; each release has enough features)
- early (or quick) releases
- a plan we can win with

This is what you want. Often, due to time constraints, you must live for now with lower quality, but which I mean we understand the plan will need to be changed.

The PO and SM must balance quality and time and decide which way the team wants to go for now. It is an art, not a science.

For example, if the team in general is very impatient to begin, you probably have to go faster than you 'should' for now, and then catch up on the quality in Release Plan Refactoring.

Sometimes the team wants to go very slowly — they want to think through 'everything.' If they start to get stuck in analysis paralysis, want too much perfection, then sometimes you must make them move on.

Very typically, if you have both types of people on the team, one person wants to go faster while another person wants to go slower. So, you must use common sense and balance the needs of the team. In some situations, it does not make much difference. If the quality is too low now, you can catch it up in Release Plan Refactoring later.

Obviously, this approach (and probably any approach) will not work well if managers or customers are demanding a 'perfectly accurate' plan on *Day 0*.

The bigger problem is the desire for perfection. The truth is that the Day 0 plan is always far from perfect. Many things will change. Do not let the team (or others) drag this out beyond two days elapsed time — at least that is the maximum I can imagine for six months of work with the teams I have seen.

If they only want to extend it one to two hours longer, then you should let them continue — this is probably a reasonable compromise.

One Key Thing

You really want the top of the *Product Backlog* 'groomed' before the end of initial release planning.

What are the minimal requirements for this?

This is a common and open question in the Scrum community, and I also do not have a 'hard and fast' answer, although a strong opinion.

Opinion: At the least, the top stories in the first sprint should be small. I want whatever work we think we can do in the first *Sprint* (let's say 20 *Story Points* worth) to be divided into eight small stories, ideally about the same size. (I am assuming a 2-week Sprint length.) Those assumptions mean every story is about 2.5 Story Points…. more concretely, almost all stories have either 2 SPs or 3 SPs.

To me, getting the stories in the first sprint to the right size is the key before closing the 'initial release planning.'

What do you have 'in hand' at the end?

The main artifact is a *Product Backlog*.

It has the user stories in the order we plan to do the work. On average the user stories are too big to go into real Sprints, but they are also not terribly big.

Each *user story* has BVPs and SPs and an *R factor*.

No two user stories are tied; one story after the other in the list.

If the Product Backlog is on the wall, then lines are drawn where the first two releases are currently expected to occur, and dates are associated with those lines. (If not on the wall, at least this is done in concept.)

You also have estimated the velocity for the first several Sprints, and you could have done a budget.

The Product Backlog includes consideration of risks, dependencies, learning, MMFS, and other factors.

The dates include contingency, and we have estimated what we think is reasonable contingency given all the factors.

We have started the discussion on communication, and have taken some notes.

And we have a short list (3-5 items) of 'work items' to make the plan better starting tomorrow morning.

Some details

It is relatively unimportant how the Product Backlog is instantiated. We do recommend having the Product Backlog as physical slips of paper 'on the wall' if at all possible. Alternately, the PB can be shown as 'items on the wall' via a Scrum tool such as Jira or something similar. Some of these Scrum tools allow 'all' the cards to be seen at the same time on a BIG screen. And the cards can be moved quickly with a mouse or some similar interface.

I strongly recommend doing this work in person, collocated, or that you get as many people as possible collocated.

As suggested, if you are collocated, I strongly prefer to use Post-It notes (4x6 inches) or index cards on a wall or white-

board. The ideal is index cards with the magnetic stick pins on a magnetic whiteboard (most whiteboards are magnetic).

See these magnetic pins in Amazon, as an example: ASIN B01GDSCAE6

I would also recommend some large easel-sized Post-It notes, as useful in many ways, including 'holding' the stories that are the Product Backlog.

Soon you will want to put the Product Backlog into a 'system' or 'Scrum tool.' I always 'back-up' the PB in a spreadsheet if we are not using a more formal Scrum tool. Examples include: Rally, Version One, Pivotal Tracker, etc, etc.

The Real Value

Now we have completed the initial release plan.

What were the key outcomes?

Well, the vision has now some sort of scope, date and budget. It is likely that we now have a revised scope, date and budget compared to the one they gave us before we started ARP.

We have a set of stories in a Product Backlog, and releases are indicated. We have an estimate of the Velocity of the team. We may have calculated a budget for the first release. We might have a communications plan or some first notes on that. We have estimated the contingency. And we have pulled together a 'fix-it plan.'

Usually on *Day 0* the quality of the release plan is 'crappy,' to use the technical term. As it must be, given how much we don't know on Day 0. And given all the change that will occur.

To be fair, we also have set up what I call the 'early warning system.'

By this I mean we have the basic release planning information, so that as time moves forward, we can see, in a useful way, whether we are on track or not.

It is like 12 lawyers at the bottom of the ocean: a good start.

We also have a release plan that is easy to refactor, or at least, much easier to change than when I used to update the Microsoft Project (release) plan. That was hard! And we can

track against the (refactored) release plan, and give ourselves (as a team) an early warning if we start to get into trouble.

The Real Value

Still, what is the *real value*? What is the most important outcome of the day?

If you ask me and most of the people who have done this, it is not the outputs or deliverables I have just mentioned.

The real win is that we now have 'everyone' on the same page.

You can say it many ways.

They are on the same page at a useful middle level of detail. What the work is that we are about to do. This group includes the full Scrum Team and the business stakeholders. This is huge.

We did not stay at too high of a level (e.g., just agreeing on the vision or the charter), and we did not get lost in the weeds (at too low a level).

We have changed the people in three important ways.

They now all see the same elephant. The same vision, and the same Product Backlog.

We have also affected their motivation. To each of them, this is now 'my' project. They have their fingerprints on it. They built it together. They own it. They have buy-in.

We have started to affect their behavior. They know the order they will do the work. They have a sense of, "I want to put more work in these high value stories, and I want to put less work in the low value stories." And it all makes more sense to them.

Another key win is that the team has shared the tacit knowledge. They have shared almost all the key tacit knowledge they have about this work and themselves and the situation. And this is so hard to make happen.

Again, let's summarize the key benefits:

- **they see the same elephant at a nice middle level of detail**
- **they are much more motivated (buy-in)**
- **they have shared the tacit knowledge and created knowledge together**

These three are huge.

If you don't recognize these people factors as the main goal, then you can easily be talked out of following the approach proposed here.

Someone will say: "It is more efficient if we have Vivek and Kara stay in a conference room for 3 weeks, and then they can tell us the answer later." And that will sound convincing. But it is wrong.

After you do this agile release planning this way, again and again, you will appreciate more and more the importance and value of these people factors.

These effects on the people are far more important than the initial 'plan' per se, and much more important than the 'scope,' 'date' and 'budget,' which will all change.

Refactoring the Release Plan

What's Next?

Now we are about to start the first *Sprint*, or more specifically, we can start the first Sprint Planning Meeting.

So, in some sense, release planning does not end. We now begin *RPR*, Release Plan Refactoring.

RPR is mostly new words to express a set of ideas that have gone under a bunch of names, such as 'pre-planning meeting' or 'product backlog refinement' or 'product backlog grooming'. The main problem with those terms is that there are many many different and non-agreeing definitions of them out there in ScrumLand. So, I hesitate to use them because you might easily get confused by those disparate definitions.

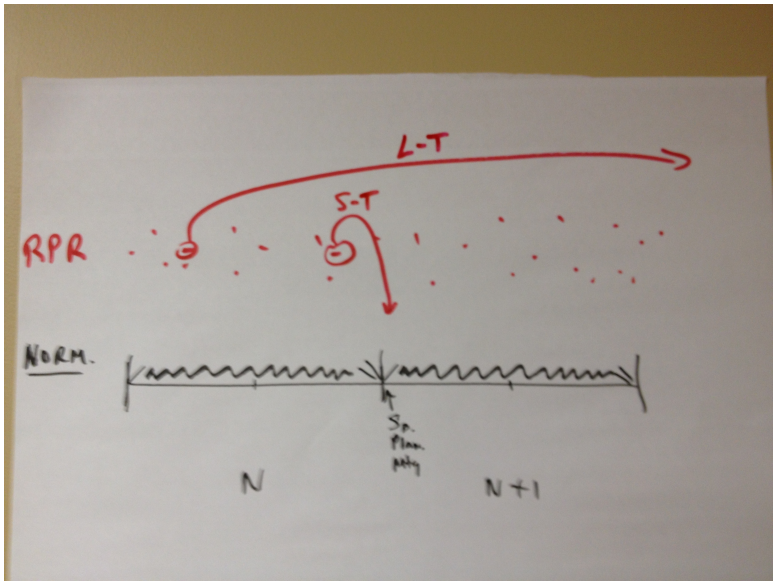
Almost all of the ideas expressed here have been stated, at least at a high level, by Jeff Sutherland and others. But never under one term. (Or under different terms.) In some cases I have taken a high-level idea and made it a bit more specific and/or concrete.

It is absolutely fundamental to Agile Release Planning that we refactor the release plan frequently. In fact, we must refactor the release plan every Sprint — every Sprint. Mechanically speaking, this is not a lot of work. Still, it is hard and important work.

Why must we refactor? Because the whole Team (and others) are actively trying to learn, in multiple domains, to help deliver a better product for the customer. Given the amount of learning by these knowledge workers, it almost insures that the plan must be updated every Sprint.

Perhaps sometimes there will be so little change and so little learning that we don't have to change the release plan at all. Yes, maybe this occasionally happens, but at least we must evaluate that no change is needed to the plan.

The following picture is what I usually draw to explain RPR (Release Plan Refactoring).



In case your print out is not in color (but only B&W), the RPR area (to the right of those letters) is supposed to be in red. And the Norm area (to the right) is supposed to be in black.

The black 'zone' represents the normal work of a Sprint.

(Hence 'Norm'.) We show two Sprints (N and N+1), and each Sprint is two weeks (in this example).

(I will be talking of the black zone and the red zone, which makes it a bit more fun for me, since it reminds me of hockey.)

The three big meetings are shown (Sprint Planning, Sprint Review and Retrospective) as slashes at the beginning and end of each Sprint.

And the squiggle line (in black) represents the team doing the so-called 'real work' of building the stories. And it represents doing the Daily Scrums.

In parallel, the PO leads the RPR — the red zone.

In the red zone, the PO may have many one-off meetings (indicated by the random dots). In addition, we recommend the PO pull together one meeting each week to do the RPR work.

The meeting length can vary as needed, but about one hour for each meeting is common. The attendees can vary, but generally we want the whole Scrum Team and we would like some *BSHs (Business Stakeholders)*. I usually can't get the BSHs to show up regularly — this is not great, but we can live with it.

In the first week, we have the Long-Term RPR meeting ('L-T'). This is not strictly long-term only, but the focus tends to be more long-term. In the second week, we have the Short-Term RPR meeting. Again, not strictly S-T.

The key to the S-T meeting is to see if all 8+ stories proposed for Sprint N+1 are ready-ready. This S-T meeting happens roughly 1.5 to 2 days before the Sprint Planning Meeting (for Sprint N+1).

Long-Term Release Plan Refactoring

In the long-term Release Plan Refactoring meeting, we do all of the following, but only 'as needed':

- changing the *vision*
- improving the *Product Backlog Items* (adding, subtracting, smaller, better)
- re-estimating *business value* (very key and very hard)
- re-estimating the effort (e.g., where Story Points are wrong, adding SPs for new stories or newly sliced stories, etc.)
- taking a new look at the R Factor (ROI per story)
- discussing new learning about risks, dependencies, learning, *MMFS*, etc.
- re-ordering the work
- doing a new scope-date trade-off, usually to make the initial release smaller (fewer aggregate Story Points) and sooner
- including the newest *Velocity* average into the projection (based on recent trends)
- adjusting a (revised?) contingency and communicating the new, new release plan (scope, date, budget) to the right people

Short-Term Release Plan Refactoring

For S-T Release Plan Refactoring, as we said above, the main goal is to assure that all 8+ stories are ready-ready, or at least have the implementers judge whether each story is or is not ready-ready. And they use the Ready, Ready Criteria for this (aka Definition of Ready).

The team uses the S-T meeting to assure it has the information it needs to be successful in the next Sprint. In effect, the Team prepares so that they will have a successful Sprint Planning Meeting.

More specifically, the team reviews the information that the PO has had prepared for each Story, and reviews that against their needs and against the ready-ready criteria to decide whether each story might go into the next Sprint.

I like to use Thumb voting. And any Team member can black-ball a story with a Thumbs down. If anyone has a thumb sideways, then he or she must identify what information is lacking and the PO has a day or so to get that information (either the PO or someone else will get it). A vote of thumbs down implies that additional information needed is too great to gather in 1.5-2 days.

The purpose here is based on Garbage In-Garbage Out. The Stories must be defined enough so that the Team has a good chance to 'get them done' in the Sprint.

As mentioned earlier, some people use phrases such as *'product backlog grooming' or 'product backlog refinement'* for this. (Note: I find 'product backlog grooming' is defined many ways, depending on who you ask.)

Comments

Other activities that could happen in the L-T meeting or the S-T meeting include:

- improving the stories or PBIs (the wording, the INVEST criteria, etc.)

- asking or answering questions from the team about the upcoming Sprint stories
- reviewing the *acceptance criteria* for each story
- reviewing and adding detail (notes, acceptance criteria, etc.)
- identifying key content for the *Enabling Spec* (defined below)
- reviewing the *Enabling Spec* (defined below) for each story or PBI that is expected to be in the next Sprint [This is mainly in the S-T meeting.]
- other reviews or activities (as the team may decide)

Some of these topics can be discussed or worked on earlier, within reason.

The Enabling Spec is just enough, just-in-time documentation. (We are not worried now exactly how the 'documentation' is instantiated, e.g., maybe on paper or maybe electronically, etc., etc.)

To start, think of one 'mini-spec' for each user story. Imagine that we quickly drew some pictures and made a few lists. And that these notes are stapled together as a mini-spec. An Enabling Spec.

The documentation is in the most useful form, that is, pictures and lists. The amount of the information varies based on many factors, including the knowledge and experience of the team. Typically a more experienced team needs less documentation, and a less experienced team needs more.

So, how much? Hard to say from afar. But often it is a half page or a couple of pages of pictures or lists or formulas, etc. that enable the team, and that give that team all the information it needs to be successful quickly.

Nothing the Team does not use. And enough so that they think they have no further questions. To be fair, always the team has more questions later, at least on one story or another.

One could argue that L-T Release Plan Refactoring and S-T RPR are completely different. I don't agree. I think it is more helpful to view them as coherent, that they fit together and that they both are key to success.

One of the purposes of release planning is to make the work in the Sprints more successful. S-T RPR is not really completed until, for each Sprint Planning Meeting, we have the information we need ready-ready for a successful Sprint Planning Meeting (and a successful Sprint).

Among the key success criteria for a Sprint Planning Meeting is that we use the time of the business stakeholders well.

Let me pause.

Why is using the time of the business stakeholders an important issue?

The business stakeholders are the key independent check on the (limited) thinking of the *Product Owner*. We know from long experience in our industry that 'knowing what the customer wants' is extremely hard. Extremely hard. The business stakeholders are people whose time is scarce and yet these people are essential to us in getting a much better fix on what the customer really wants (or what the firm wants).

We want the business stakeholders in the Sprint Planning Meeting agreeing with what the team takes in (which stories the team commits to do). We want them at the Sprint Review giving us feedback. Because these independent input and

output checks are so vital, we must use the time of the business stakeholders efficiently. Or, very often, they will not come (or start skipping meetings). And their presence is crucial.

Now, back to the broader issues.

Yes, to make it easier to understand, we talk about short-term RPR and long-term RPR, and then we talk about the 'black zone,' where the stories are actually built. But all three of these things work together and support each other.

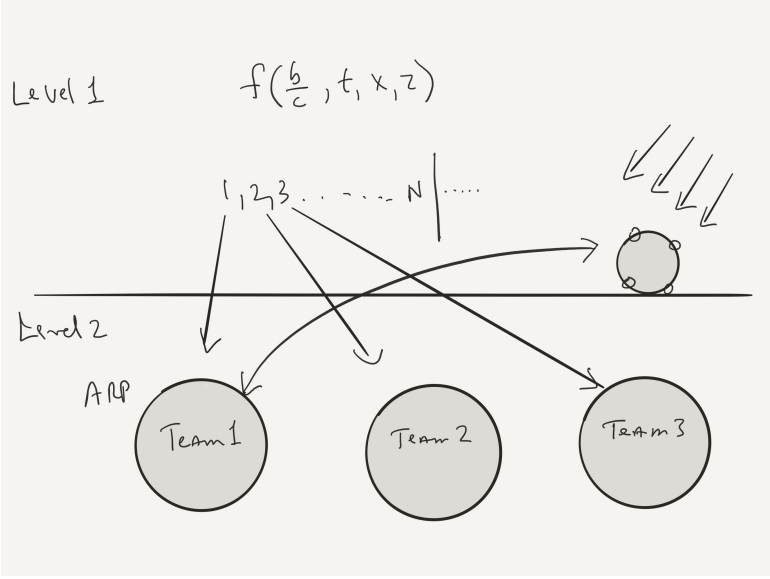
Thus, for me it is better to think of higher level Release Plan Refactoring and lower level Release Plan Refactoring. When people say 'Product Backlog Grooming,' they often only mean lower level Release Plan Refactoring of those stories about to go into the next Sprint.

Again, the key idea here is that we *always* refactor the release plan every Sprint. We improve it. This is absolutely key to adaptive planning.

Level 1 Planning and Level 2 Planning

I find that all companies I work with have at least two levels of planning. Often a larger firm will have 5 levels of planning.

The two levels are shown in the following picture. (next page)



Each company is different and uses different words, but the basic concepts are usually very similar. At least that is what I have experienced. So, something like the picture above is fairly common across firms.

At Level 1, we have a whole bunch of ideas coming in on the right (they may call these initiatives, or projects or product ideas). Maybe the ideas come in over the year, or in a brainstorming session. Then the firm has one or several people to evaluate each idea.

Usually these people or this committee (many different names are used) is rational, and describes its decision-making criteria; how they will decide what to prioritize. Often this includes benefits over costs, time, and specific local factors (X and Z in my example). Hence the function in the top middle.

Then ideas are reviewed and compared. And the 'committee' (circle) prioritizes the work. It may be at the release level, at the product level, or at the project level, depending on how they think of each set of work.

The work is lined up (prioritized) usually as projects, from 1 to N, meaning that in the next year (next period), we feel we can get these N projects done.

Often they speak of these ideas now as projects, and they often speak of 'the 1 to N list'. It might also be called 'the plan' or 'the roadmap' or another name. This 1-N list can have many names.

So, the work has been prioritized (by the Level 1 people) at a macro level, and some work does not make the cut. That work won't be done 'next year' (or whatever the time period may be).

Next, one project is given to a 'team' (or some group of people) to deliver.

Always, the 'giving' includes some information. Perhaps only a discussion, perhaps one sheet of paper, perhaps a formal BRD (business requirements document), but at least the work

is said to have a 'scope,' and to some degree it does. Often the work also has an estimated due date and budget as well.

Then, at Level 2, the team (or the small group) plans the work at a lower level.

In the old days, this would often be a PM (project manager) using Microsoft Project or something similar. It was very painful.

So, instead of that old style Level 2 work, I am suggesting we do 'Agile Release Planning.'

Is Level 2 planning repeating the work done in Level 1 planning? Yes, in some sense.

What is different? We think there are 3 key differences.

1. We have the best people in the company of Project 1. The best team and the best BSHs.
2. We will plan at lower level of granularity, and always the planning is better then (as long as the level is not too low).
3. We have learned some things since Level 1 did their estimate. Time has elapsed, things have changed, we have learned. And therefore the Level 2 planning will be better because it will include this new learning.

Could Level 2 planning come up with different 'answers' than Level 1 planning? Yes, of course, and in fact, to some degree, it is certain to happen. For example, the scope, the date, the budget could all be (somewhat?) different. This should be considered a fairly normal result.

How do we cope with that different result? In my experience, usually (but not always) the Level 2 planning is better, more accurate, more meaningful.

We suggest that, whether the differences are small or large, that your team give the Level 1 people some 'feedback.' Hence, you see that line in the picture. The team and the committee should discuss and decide on the appropriate action.

Two key suggestions.

1. Draw the picture of how your firm or department does Level 1 and Level 2 planning now. First, you will notice that many people do not know how it works, or will not recognize the same picture. This learning will be helpful.
2. You will notice that your firm's picture will not be 'as good as it should be.' For example, it may have some weaknesses compared to my picture. So, over time, improve the weak areas.

My key point

My key point for now that affects what we do in Agile Release Planning… We do not start Agile Release Planning in a vacuum, with a blank slate. Always it is preceded by Level 1 planning. We always start with something, and my Level 1 idea suggests some of the things that might be.

So, I assume every company is already doing Level 1 planning in some way — perhaps even two or three levels of planning 'above' the team level (the team level is 'Level 2' as described above), and then the team must do 'Level 2' planning in more detail. This is normal, common, virtually universal in my experience.

You must devise a reasonable solution regarding how those two levels of planning (or more levels) interact.

For your company to become more fully Agile, that interaction probably needs to change.

Suggested, not Prescriptive

In this book I made many basic concepts quite concrete. To conserve space, I did not suggest alternatives.

It may have seemed that I was being prescriptive (a word generally frowned upon in the Agile community). This is not the case. I merely described what I usually recommend. I respect that you have an independent mind, you know your specific situation and you will use common sense in applying (or not applying) these specific practices (or even these specific ideas) to your situation.

I made the ideas concrete to make them more practical and more visible to some people (those of us who think more concretely). We also know that "in theory there is no difference between theory and practice, but in practice there is" (Yogi Berra). Meaning, for example, that we often have a better understanding of ideas once we can see them in practice.

Obviously, this approach at explanation has risks. Some readers will no doubt take me as prescriptive.

Again, I think these ideas and practices work in many common situations, but I am not trying to be prescriptive, and far less am I expecting you to take these words that way. Use common sense (which is very uncommon, it seems).

Final Comment

Agile Release Planning is, most importantly, about the people.

It is about helping the individuals and the team be more successful. It is about delivering better products to the customer.

I hope you will use common sense in applying these ideas, and I hope they are helpful to you, your team, and ultimately, your customers.

Please contact us with comments or questions. Or if we can help.

Joseph Little

jhlittle@kittyhawkconsulting.com

FAQ

Below are some frequently asked questions.

1. What do you recommend for multiple teams in a scaling situation?

Ans: For now, I am not trying to discuss scaling in this book much. Please contact me to discuss. To answer it here would take many more pages. I do plan to add a section later to discuss this.

The short answer is that scaling is always hard. The communications, for example, are always hard. *ARP* can be done, using the basic ideas in this book, but it remains hard. We are stuck in a contradiction:

- in simple terms, we want everyone to know 'everything,' but…
  - communicating across seven people is hard, and with (say) 30 people it is very, very hard.

I will suggest some more specific ideas later.

1. Where is your scientific proof that this is the best method?

Ans: In general, as an overall approach, it is heuristic. That means: It has no proof that I would call scientific, but I have done it many, many times with many teams, and it works.

I have had almost no situations where I could not get the team to do it (fairly special situations). Maybe that was the right decision for them.

I am not aware that any approach has been scientifically proven. However, there has been significant study on some parts of the approach (and on alternatives). I hope to write a paper soon that reviews that information.

I think these methods are 'against' some common water-fall or Agile methods, but this approach has different goals than the 'old' methods have or had. For example, suppose someone has evidence that Planning Poker were not the most accurate method for estimating – nonetheless I would likely still recommend it because of the value of the team knowledge creation with Planning Poker, and the sharing of tacit knowledge. That is a main goal for me (and, I think, for you).

In simple terms, 'success' depends on what you value.

1. What about use cases?

Ans: I generally like light-weight use cases, although I do not usually recommend them where they are not already being used.

In general, I have not found them necessary when doing the initial release planning, but they often are very useful later (for the implementers and others in doing their work). The

issues that use[10]cases[11] address will come up in estimating the *effort* (and in other places), and the issues should be discussed (and maybe documented) 'just enough' at that time so that release planning may proceed.

1. How about more discussion of the theories behind release planning?

Ans: I did discuss the ideas quite a bit. In particular, I feel that the goals I have for Agile Release Planning are different than what many people have had for release planning, but I wanted to keep this book short. Perhaps there is another book in that subject. I thought it would be more useful to many people to describe the practical application first. Some people see better when you talk practical details.

1. Should we do this if we don't have a full team and a full set of business stakeholders?

Ans: Probably yes. Try to get a full team (I am assuming seven people) and four good business stakeholders. If you only have nine people (when wanting 11), then I would probably start. At the end of the day, you can only do what you can do. Then, it is probably best to start with what you do have.

I suppose one could imagine having only one person (out of the 11). In that extreme case, I probably would delay, with the hope of gathering at least nine in a day or two or a week. If so few can show up, do you really have an important set of work to do?

[10] https://en.wikipedia.org/wiki/Use_case

[11] https://en.wikipedia.org/wiki/Use_case

1. Do you think user story cards are sufficient for Agile Release Planning? When should we add *acceptance criteria*?

Ans: I think usually a team can do decent initial release planning with only story cards. Almost always they discuss some other issues along the way. Those important details should be documented in some way. For example, for a few stories, they might write down the acceptance criteria that were identified. (But do not do that for all stories.)

But those details (the ones discussed) are never all the details that the implementers will need. Before doing work in the *Sprint*, I recommend the *enabling spec* idea; 'just enough, just in time' documentation delivered before the Sprint for those stories going in the Sprint. "Documentation of the requirements" is saying it too quickly, but it gives you the basic idea. In any case, not my main subject here.

I definitely recommend, at a minimum, that the acceptance criteria for a story be defined before that story enters the Sprint.

1. What if they did Level 1 planning before they gave us the work (project)?

Ans: Perfect. (See above for a discussion of the meaning of Level 1 and Level 2 planning.)

Usually you have a bit more clarity on scope and the wished-for date, etc.

My suggestion is: Do the work described above. Use the prior Level 1 work as partial input, but only as input. Then, after

you are done, compare what you have to what they told you, and then explain your findings to the Level 1 committee.

The Level 1 committee may agree or disagree with you, but at least your *Scrum Team* will have a basis for engaging with them.

1. What if we are given a BRD (business requirements document) before we begin?

Usually it makes very little difference to the Team.

It might be useful.

The typical experience is that, when we start the User Story Workshop, someone says, "We ought to use the BRD," but no one really looks at it. Perhaps someone says, "Let's do this first, and then look at it later."

Then we start writing user stories. Maybe one or two people at first are looking at the BRD, and then, slowly, writing a few user stories while other people are prolifically writing user stories without the BRD. Person 1 says, "Do we have everything?'" Person 2 says, "I think so." Person 1 says, "Well, I will compare all of these to the BRD." Person 2 says, "Not now, right? We have a pretty good list for now." Person 1 says, "OK."

Q1: Do we really have everything? No!

Q2: Does this BRD still have an idea or two that could help us identify one or two additional stories? Yes.

Q3: Was the BRD itself complete? No. And we now have some user stories (and some are nowhere in the BRD) that prove that the BRD was not complete.

Q4: Does anyone have the heart to read every sentence in the BRD carefully and identify those last two 'missing' user stories? No, although one or two people may make a half-hearted attempt at it. Most likely, it will be the guy who 'owns' the BRD the most. Even he will give up fairly soon.

Q5: Yikes! So, how do we identify the missing stories? First, on *Day 0*, there are always, always, always some missing stories — and not just the ones that are buried inside that 50 page BRD. Second, the PO must discover ways for 'everyone' to help identify those missing stories sooner. The first method: In each Sprint demo, ask the business stakeholders, "Did we do the most important work for you in this Sprint that we possibly could have done? With 20-20 hindsight, which features should we have built?" And they will start identifying the missing stories if they are good business stakeholders.

1. There are lots of special cases. You did not deal with them. Why?

Ans: I wished to discuss things quickly, to keep the book short. Also, you (the reader) are probably in a better position than I to address the special cases, since they often are specific to your situation. I assume you can think — you see the values and principles proposed here (or have yet better values and principles) — and can apply these values and principles for yourself in your specific situation.

But, as people raise questions about specific situations, I will address them in this FAQ or in this book.

1. How long should Agile Release Planning take? Could it take more than 1 calendar day?

Ans: Yes, it could. We do think many people have a strong bias to 'wait for perfection.' Or 'do it once, perfectly.' And this must be avoided.

But, for example, working together intensely with 11 people is tiring, and especially to introverts, typically. So, if you have a group that starts to get tired, then giving them longer breaks and having it extend into the next morning seems quite reasonable.

Just do NOT let them do individual parts of the agenda for too long. This will not help.

Another case is if you have a lot of work, let's say work for 12-18 months. Then the ARP should last more than 6-8 'ideal' hours.

[stopped]

Note on the Author

Joseph Little is an Agile Coach and Certified Scrum Trainer. He was graduated with a BA from Yale, became an international banker in NYC and then received his MBA from NYU. Around this time, he got involved in projects and technology in a more professional way. He learned a lot from co-training with Jeff Sutherland over eight courses. He was originally introduced to Agile in 2000, and has been doing Lean-Agile-Scrum exclusively since 2005.

To learn more about him, you can follow these links:

Twitter: jhlittle[12]

LinkedIn: www.linkedin.com/in/joelittle[13][14]

Website: www.LeanAgileTraining.com[15][16]

Blog: www.LeanAgileTraining.com/blog[17][18]

Please send him comments on this book at jhlittle@kittyhawkconsulting.com

Thanks!

---

[12] https://twitter.com/jhlittle

[13] www.linkedin.com/in/joelittle

[14] [images/2013-10-10_17_29_17.jpg]

[15] http://leanagiletraining.com

[16] [images/2013-10-10_17_29_17.jpg]

[17] http://leanagiletraining.com/blog

[18] [images/2013-10-10_17_29_17.jpg]

Glossary

Acceptance Criteria: A short list of criteria to determine whether one specific User Story (or PBI) is complete. Usually these items define tests at a high level. Each story typically has a different set of acceptance criteria. (You may wish to compare this to the DOD.)

Agile Release Planning (ARP): In short, the initial work done to create the Product Backlog and the release plan before you start Sprinting.

Business Stakeholders (BSHs): We have a specific definition that may be different than what you use at your company. To us, the BSHs are those people who come to each Sprint Review to give the Scrum Team feedback on whether the completed user stories will satisfy or delight the customer. So, the BSHs must be able to quickly and accurately assess whether a user story will delight the customer. And if not, can define in detail what needs to change. In addition, we want the BSHs to come to the Sprint Planning Meeting regularly to review the inputs to each Sprint, and to work closely with the PO as needed.

Business Value (BV): The value in building a product or feature or user story. The value may be in the mind of the customer or in maximizing shareholder wealth for the owners of the producing firm. There are many different ways of thinking about Business Value (e.g., reduced risk), and this concept includes them all.

Day 0: The 'first' day of the effort, when we are dumbest. Also,

the day the team does Agile Release Planning, and before the first sprint has begun.

Definition of Done (DOD): The list that describes the degree of 'doneness' of a feature or PBI or user story. One DOD list applies (usually) to all stories for the Team. We use the DOD to judge whether a story is 'done' in the Sprint and whether the team has earned the related Story Points as part of the Velocity. If a story is done according to the DOD, we often say it is 'done, done.' 'Done, done' does not mean that story has been released (e.g., if software, then released into the production environment). Every team must have a DOD.

Enabling Spec: A couple of pages of pictures, lists, formulas, etc. that enable the team to build that PBI or user story as quickly as possible. The implementers define all of the information they need to be successful quickly, i.e., they define what should be in the Enabling Spec.

I-A-D: Infrastructure, Architecture and Design.

Impediments: Anything that slows down or prevents a team from building a great product as quickly as possible. A possible impediment could be something holding back the quality of the work.

Minimum Marketable Feature Set (MMFS): The smallest piece of functionality that can be delivered that has value to the organization delivering it and/or the people using it.

Planning Poker: A team uses the Planning Poker cards or Agile estimation Fibonacci cards to vote on the relative Story Points of a to-be-estimated user story compared to the size of a reference story. So, if a story is voted as a three, then it is three times bigger than the reference story (typically assigned one Story Point). We use wide-band delphi estimation to vote

on the Story Points. The implementers vote. Note: Planning Poker is a registered trademark of Mike Cohn.

Priority Poker: Very similar to Planning Poker, except that it is about the relative Business Value of one user story compared to a reference story. It is not about the effort. Again, wide-band delphi estimation. The BSHs and the PO vote.

Product Backlog (PB): The list of the features or requirements requested for a product, expressed as a prioritized list of Product Backlog Items or user stories.

Product Backlog grooming/refinement: These words have many definitions. We will give you ours. The first idea is the metaphor that the Product Backlog is kept clean and orderly. Beyond that, the community does not clearly agree on what the terms mean. Often it includes adding detail to each story. It may include re-ordering the PB as new information arrives. Often it includes slicing and dicing the stories a Sprint or two before that story is put into a Sprint. The Scrum Guide does not define when or how Product Backlog Refinement should happen. We think it should include everything that we define as Release Plan Refactoring.

Product Backlog Items (PBI): An item in the Product Backlog. Before a PBI is put into a sprint, it should be at least small enough to be completed by a team in one Sprint. We recommend at least eight (plus) about equally-sized small PBIs per Sprint. Each PBI has Business Value, at least in the eye of the PO. PBIs are typically expressed in user story format, in which case they may be called user stories. PBIs are broken down into typically 4+ tasks in the Sprint Planning Meeting.

Product Owner (PO): The person who has final authority to decide the order of the Product Backlog. He or she represents the customer's (and firm's) interests in backlog prioritization

and requirements questions. This person is responsible for the quality of the Product Backlog, and leads the Team in defining what the new product will be.

Reference Story: We have two reference stories. The first is the largest Business Value story. We give that a value of 100, and it is used to compare all other stories to and to vote on Business Value. The second is the smallest effort story. If it is not too small and not too large, we give that a value of one Story Point. Then we compare all other stories to it and use it to vote on Story Points for each of the other stories.

Release Plan Refactoring (RPR): This is the set of work we do to refactor or improve the release plan and to get the stories 'ready, ready' before they go into a Sprint. Depending on your situation, you could choose from many different activities to make this happen. We recommend also two team meetings — one in the first week to review long-term RPR, and one in the second week to review short-term RPR. In essence, what we mean by RPR is essentially what the community calls Product Backlog grooming, or Product Backlog refinement or Pre-planning, and other names.

R factor: The ratio of Business Value Points to Story Points. It represents return on investment, bang for the buck, cost-benefit analysis or 'identifying the low hanging fruit.'

ScrumMaster (SM): The facilitator for the whole team (including the Product Owner). He or she works to assist both. The main job is impediment-remover-in-chief. That is, to drive the removal or mitigation of impediments of all types so the Velocity of the team is likely to increase significantly.

Scrum Team: Optimally comprised of seven people (possibly plus or minus two people). It self-organizes and is responsible for success. One example: to meet the Sprint goals as a team.

It includes the implementers, the Product Owner and the ScrumMaster.

Sprint: An iteration of work where an increment of product functionality is implemented (done, according to a good Definition of Done). A Sprint cannot last more than 4 weeks; we recommend 2-week Sprints usually.

Story Point (SP): Used to estimate the effort required to implement a story; a number that gives the team a relative size to each story. It is voted on in comparison to the reference story. It is related to size, complexity and relative effort.

User Story: A requirement expressed in the user-story format (As X, I want Y, so that Z). To be a Sprint-sized user story, we recommend that you can fit eight stories into the Velocity of a two-week Sprint. (Example: Velocity = 20; 20/8 = 2.5, so that most stories would be about two or three SPs). It is a short definition of a requirement, containing just enough information so the developers can produce a reasonable initial estimate of the effort to implement it.

Use Case: A diagram or list of steps usually describing interactions between a user and a system, related to achieving a particular user goal.

Vision: A future view of the solution to be developed. It is usually inspiring. It gives a conceptual feeling for how the customer's life would be better once we deliver the proposed product.

Velocity: How many Story Points of effort were 'done, done,' on average, over the last three or four Sprints. Via the Yesterday's Weather pattern, we use it to guess how many stories (really Story Points) to take off of the top of the Product Backlog and bring into the next Sprint. (The implementers get

the ultimate decision, but Velocity provides the first guess.)

RULES

1. We don't always follow the rules. We can make specific exceptions. Rarely.
2. All acronyms are 'explained' in the glossary.
3. If it is capitalized in the Scrum Guide, then we capitalize it.
4. If a term is in the Glossary, then if it is mentioned in the text…. [then what should happen…. Caps? Italics? Italics the first time??]
5. These RULES will be deleted in the final copy.