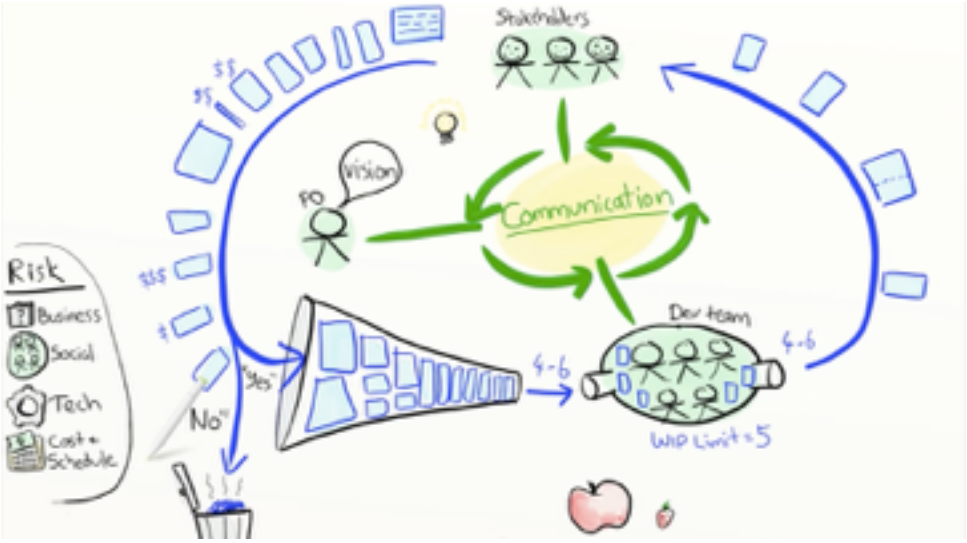

Agile Product Ownership in a Nutshell

Henrik Kniberg

http://blog.crisp.se/2012/10/25/henrikkniberg/agile-product-ownership-in-a-nutshell

Let's talk about Agile software development from the perspective of the Product Owner.

Here's Pat. She's a product owner. She has a product vision that's she's really passionate about. She doesn't know the details of what her product is going to do, but she knows why we're building the product, what problem it is going to solve, and for whom. She talks about it all the time.

Here are the stakeholders. The people who are going to use, support, or in any way be affected by the system being developed. Pat's vision is that these people will love our system and use it all the time.

The stakeholder needs are expressed as user stories. For example in a flight booking system, people need to be able to search for a flight. That would be one user story. Stories often correspond roughly to a feature, or a use case. The stakeholders have lots of ideas, Pat helps them divide these into fairly distinct user stories.

Now, somebody has to BUILD the system. Here they are, the development team.

Since this is an agile team, they don't save up for a big bang release at the end. Instead, they release early and often. They usually release 4-6 stories per week, so that is their capacity. It is easy to measure, just count the number of stories released per week. Some stories are big, so they count as two, some are small and count as a half. But all in all, it adds up to about 4-6 stories per week.

In order to maintain this pace, and not get bogged down by manual regression testing, the team invests heavily in automated testing and continuous integration. Every feature has automated acceptance tests, and most of the code has automated unit tests.

The problem is, here are bunch of stakeholders asking for all kinds of stuff, and they sure aren't going to be limited to 4-6 ideas per week. They have LOTS of ideas and LOTS of wishes. And every time we deliver something to them, they will get even more ideas and ask for even more stuff.

So what happens if we try to do everything they ask for? We'll get overflow. Suppose the team starts working on 10 new stories per week. If the input is 10, and the output is 4-6, the team will get overloaded with work. That will cause multitasking, demotivation, and ultimately lower output and lower quality. It's a lose-lose proposition.

It's like trying to shove more paper into a printer to make it print faster. Or shoving more cars onto a crammed highway system. It doesn't work, it just makes things work. It's a basic law of the universe – if Input is greater than the Output, you unavoidably get overcrowding or leaks.

The Scrum and XP way of avoiding this problem is called "yesterday's weather". The team says "well, the past few weeks we've finished 4-6 features per week. So which 4-6 features shall we build this week?.". The product owner's job is to figure out, out of all the possible stories in the whole universe, which 4-6 shall we deliver next?

The Kanban way is to limit work-in-progress. Suppose the team decides that 5 is the optimal number of stories to be working on simultaneously, just enough to keep everyone busy without causing overload. So they decide that 5 is their WIP limit. Whenever they finish one story, they will accept one new story, thereby making sure they never break the limit of 5 ongoing stories.

Both of these approaches work fine. And they will cause a queue to form in front of the team, which in Scrum is called a Product Backlog.

This queue needs to be managed. If stakeholders keep asking for 10 new stories every week, and the teams deliver 4-6, the queue will keep getting longer and longer. Before you know it, you have a 6 month long wish list in the backlog. That means that, on average, every story that the team delivers is something that somebody asked for 6 months ago. How agile is that?

There is only one way to stop the queue from going out of the control. That is the word No. It is the most important word for a product owner, and Pat practices it every day in from of the mirror. Saying yes to a new feature request is easy. The most important job for a product owner is to decide what NOT to build, and take the consequences of that decision.

The product owner decides what goes in, and what goes out. The product owner also decides the sequencing – what do we build now, what do we build later? This is a hard job, and needs to be done in collaboration with the team and stakeholders.

———

To be able prioritize, the product owner must have some idea of the value of each story, as well as the size. Some stories are critically important, others are really just bonus features. Some stories take just a few hours to build, others take months.

Guess what the correlation is between story value and story size? That's right – None! Bigger doesn't mean better. Think of any system that you have used, and I bet you can think of at least one really simple feature that is very important, that you use every day. And I bet you can think of least one huge complicated feature that is totally unimportant – like that silly paperclip guy that was in Microsoft office a few years ago.

Value and size is what helps Pat prioritize intelligently.

These two stories are roughly the same size, but have different value. So build this one first. And over here- these two stories have roughly the same value, but different size. So build this one first. And so on.

But wait a sec – how does she know the value of a story? How does she know the size? Well, here's the bad news – she doesn't. It's a guessing game.

And it's game that everyone is involved in! Pat continuously talks to stakeholders to find out what they value. She continuously talks to the team to find out what they think is big or small, in terms of implementation effort. These are relative guesses, not absolute numbers. I don't know what this apple weighs, or that strawberry. But I'm pretty sure that the apple weighs at least five times as much, and that the strawberry tastes better (to me). And that's all Pat need to know in order to prioritize the backlog! It's pretty cool that way.

At the beginning of a new project our guesses will inevitably suck. But that's OK, the biggest value is really in the conversations rather than in the actually numbers. And every time the team delivers something to real users, we learn something and get better at guessing both value and size. That's why we continuously prioritize and estimate. Trying to get it all right from the beginning is pretty dumb, because that's when we know the least. The feedback loop is our friend.

———

Prioritization is not enough though. In order to deliver early and often, we need to break the stories down into bite-sized pieces, preferably just a few days of work per story. We want this nice funnel shape, with small, clear stories at the front and more vague stories at the back. By doing this break-down in a just-in-time fashion, we can take advantage of our latest insights about the product and the user needs.

All this stuff I've been talking about – estimating the value and size of stories, prioritizing, splitting – all that is usually called "backlog grooming". Pat runs a backlog grooming workshop every wednesday from 11:00 – 12:00. The whole team is usually there, and sometimes a few stakeholders as well. The agenda varies, sometimes the focus is on estimation, sometimes on splitting stories, and sometimes on writing acceptance criteria for a story.

I hope you're noticing the theme here. Communication! Product Ownership is really all about communication. When I ask experienced product owners what it takes to succeed, they usually emphasize passion and communication. It is no coincidence that the first principle of the agile manifesto is "Individuals and Interactions over Processes and Tools".

So the PO's job is not to spoon-feed the team with stories. That's boring and ineffective. Pat, instead, makes sure everybody understands the vision, that the team is in direct contact with stakeholders, and that there is a short feedback loop in terms of frequent deliveries to real users. That way the team learns and can make daily tradeoff decisions on their own, so Pat can focus on the big picture.

———

There are number of tradeoffs that need to be made by Pat and the team. Let's look at a few.

First of all, there's the tradeoff between different types of value. Early on in a project, uncertainty and risk is our enemy.

There's business risk: are we building the right thing? There's social risk: can these people build it? There's technical risk – will it work on the platform that we want to run it on? Will it scale? And there's cost and schedule risk. Can we finish the product in a reasonable amount of time, for a reasonable amount of money?

Knowledge is the opposite of risk. So when uncertainty is high, our focus is knowledge acquisition- we focus on things like user interface prototypes and technical spikes, or experiments. Not too exciting for the customers, but still valuable because we are reducing risk.

From a customer value perspective, the curve looks like this in the beginning.

As uncertainty is reduced, we gradually focus more and more on customer value. We know what we're going to build and how, so just do it! And by doing the highest-value stories first, we get this nice steep value curve.

And then, gradually, the value curve starts flattening out. We've built the most important stuff, now we're just adding the "bonus features", the toppings on the ice cream. This is a nice place to be, because, at any point, Pat and the team may decide to "trim the tail" and move on to another, more important project, or start on a whole new feature area within the same product. That is business agility.

So when I talk about Value here, I mean Knowledge value + Customer value. And we need to find a tradeoff between these two.

———

Another tradeoff is short term vs long term thinking. What should we build next? Should we do that urgent bug fix, or build that awesome new feature that will blow the users away, or do that difficult platform upgrade that will enable faster development in the future? We need to continuously balance between reactive work and proactive work, or fire-fighting and fire-prevention.

This is related to another tradeoff. Should we focus on "building the right thing", "building the thing right", "building it fast"? Ideally we want all three, but it's hard to find the balance. Suppose we are here – trying to building the perfect product, with the perfect architecture. If we spend too much time to trying to get it perfect, we may miss the market window or run into cash-flow problems.

Or suppose we are here, rushing to turn a prototype into a usable product. Great for the short term, perhaps, but in the long term we'll be drowning in technical debt and our velocity will approach zero.

Or suppose we are here, building a beautiful cathedral in record time. Except that the users didn't need a cathedral, they needed a camper van.

There is a healthy tension here between the Scrum roles. POs tend to focus on building the right thing. Teams tend to focus on building the thing right. And Scrum masters, or agile coaches, tend to focus on shortening the feedback loop.

Speed is worth emphasizing. Because a short feedback loop will accelerate learning, so you will more quickly learn what the right thing is, and how to build it right. However, all three perspectives are important so, keep trying to find the balance.

Finally, there is the tradeoff between new product development and old product improvement. Product Backlog is a confusing term, because it implies that there is only one product. And project is confusing term, because it implies that product development "ends". A product is never really "finished" there's always maintenance and improvements to be done, all the way until the product reaches end of life and is shut down.

So when a team starts developing a new product, what happens to their last one? Handing off a product from one team to another is expensive and risky. A more common scenario is that the team continues maintaining the old product while developing the new one. So it's not really a product backlog, it's more like a team backlog – a list of stuff that the product owner wants this team to build, and it can be a mix of stuff for different products. And the product owner needs to continuously make tradeoffs between these.

———

Once in a while, a stakeholder will call Pat and say "Hey, when will my stuff be done?" or "How much of my stuff will be done by christmas?". As PO, Pat is responsible for expectations management! Or, more importantly, realistic expectations management. That means no lying. I know, it's tough, but who said Agile was easy?

It's not really that hard to make a forecast, as long as it doesn't have to be exact. If you measure the velocity of your team, or the combined velocity of all your teams, you can draw a story burnup chart. The chart shows the cumulative number of stories delivered over time (or story points if you prefer).

Note the difference. This curve shows output. That curve shows outcome. That's the output, and that's the outcome that we hope it will achieve. Our goal is not to produce as much output as possible. Our goal is reach the desired outcome, using the least possible output. Less is more.

Look at the burn up chart and draw an optimistic and pessimistic trend line. You can do it using fancy statistics voodoo, or you can just draw it visually. The gap between these lines is of course related to how wavy and unpredictable your velocity is. That tends to stabilize over time, so our cone of uncertainty should get tighter and tighter.

OK, so back to expectations management.

Suppose the stakeholders ask Pat "When will all of THIS stuff be done?". "When will we be here?". That's a fixed-scope, variable time question. Pat uses the two trend lines to answer. "Most likely sometime between April and mid-May".

Suppose the stakeholders ask Pat "How much will be done by christmas?". That's a fixed time, variable scope question. The trend lines tell us "We'll most likely finish all these, some of these, and none of these."

Suppose the stakeholders say "Can we THESE features by christmas". That's a fixed time, fixed scope question. Looking at trend lines, Pat says "Nope, sorry, it ain't gonna happen", followed by "here's how much we can get done by christmas" or "here's how much more time we need". It's generally better to reduce scope than to extend time, because if reduce scope first, we still have the option to extend the time later and add the rest of the stories. Vice versa doesn't work, because, darnit, we can't turn the clock backwards! Time is rather annoying that way, isn't it. Pat

makes the choice easy by saying “we could deliver something here, and the rest later. Or we could deliver nothing here and the rest later. What do you prefer?”

The calculations are simple to do, so Pat updates the forecast every week.

The important thing here is that Pat is using empirical data to make the forecast, rather than wishful thinking. And she is being honest about the uncertainty. I said no lying right? This is a very honest way of communicating with stakeholders, and they usually appreciate that a lot. If your organization doesn’t like truth and honesty, it won’t like agile.

A word of warning though. If the team is accumulating technical debt – if they’re not writing tests, and not continuously improving the architecture – then the they will get slower and slower over time, and the story burnup curve will flatten out. That makes forecasting almost impossible. So the team is responsible for maintaining a sustainable pace, and Pat avoids pressuring them into taking shortcuts.

———

What if we have a larger project with multiple teams? Instead of one team with 5 people here, we might have 3 teams. And we might have several product owners, each with their own backlog for a different part of the product.

Overall, the model is really the same. We still need capacity management, we still need stakeholder communication, we still need product owners who can say No, we still need backlog grooming. Velocity is the sum of all output, and can be used for forecasting. Or make a separate forecast for each team, if that makes more sense.

In a multi team scenario, however, the POs have an important additional responsibility – to talk to each other! We should organize the teams and backlogs to minimize dependencies. But there will always be SOME dependencies. So there needs to be some kind of sync between the product owners so that we build things in a sensible order, and avoid sub optimization. In large projects, this usually calls for some kind of Chief Product Owner role to keep the product owners synchronized.

OK, that’s it! Agile product ownership in a nutshell.