ANNAFORSS.SPACES.LIVE.COM/

Confessions of a serial product owner

Based on a true story

Anna Forss

2009

ANNA.FORSS@GMAIL.COM

Preface

When starting up the software project, where one of the architects sent the CEO some reads on agile and domain driven design. These were areas of interest for the architect and we'd discussed the benefits of working this way. Working at a company solidly into waterfall development, we had no idea what would happen next.

Without clearing it with us developers, the CEO said in front of the whole company that we were starting using scrum as from then.

But what did that mean? How were we supposed to work? Who was going to take on the different roles? I got started reading. First, I was scrum master and product owner proxy (yes, this is not a role which can be found in hard core scrum, but I'll get back to that later). A couple of months later, I became a product owner.

What I found was that most of the literature and examples are described from a scrum master or developer perspective. Since I'm a business person I rather got the sense that I was the "bad guy", someone developers and scrum masters thought was "never available" and the primary one to blame when the project failed.

Is it true, and if so: why? I definitely do not have the answers, but these are my experiences and tips for a product owner in the making.

This is a book for you who are getting familiar with scrum. Perhaps you've read some blogs or a shorter book on the subject. Perhaps you've used scrum for a while and is asking yourself: is this really supposed to be like this?

Since I'm a business person and not a software developer, this book is written more for business people than developers.

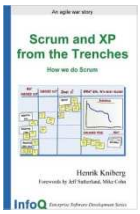
I would like to thank Bram Lauwers, Franco Martinig, Claudio Brancher Kerber and Siraj Sirajuddin for their help reviewing and commenting on this text. Also, a special thanks to Johan Andersson, Morten Nielsen, Carina Bergström and Wilhelm Svenselius. These are people I've had the honor of working with during a time when I learned so much about scrum and the complexity of software development. It was an honor working with you. I would also like to give a big hug to my husband Håkan for putting up with me during this time.

Tallkrogen, February 2009

Table of Contents

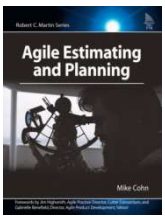
What is scrum?

There is so much good literature which describes scrum on the Internet, so I won't go into details. Just search online and you'll get tons of descriptions. Some of my suggestions are:



Scrum from the Trenches, by Henrik Kniberg at Crisp. A free PDF copy can be downloaded from Internet. You can also buy a printed copy. Agile software Development with Scrum by Ken Schwaber and Mike Beedle.

You can also benefit from reading books on agile software development in general. Often some examples are taken from scrum and this can also give you an idea on how scrum is positioned with other development methodologies.



Agile Estimating and Planning by Mike Cohn is probably the book which has given me most inspiration from a product owner point of view. Agile and Iterative Development by CraigLarman is a really good guide to agile software development and if you're looking at combining scrum with other methodologies, this book includes lots of tips.

But to give you a short description, scrum tries to answer the following questions:

- How to decide what to develop?
- Who decides what to build?
- How deliveries are presented?
- How team member keeps each other updated?

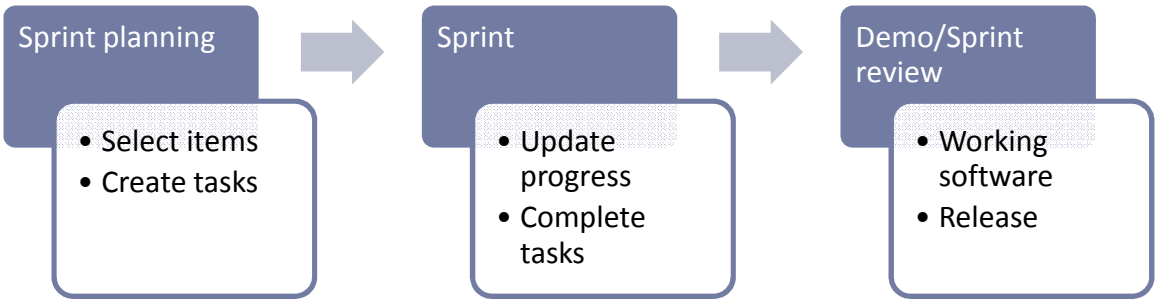
Scrum does not specify how you develop and if you look at the list above you can see that scrum specifies rules for communication during development.

This is the reason why scrum is most often combined with a more detailed software development methodology like Extreme Programming: since it does not help the developer during the actual programming you might want to add other rules or methods.

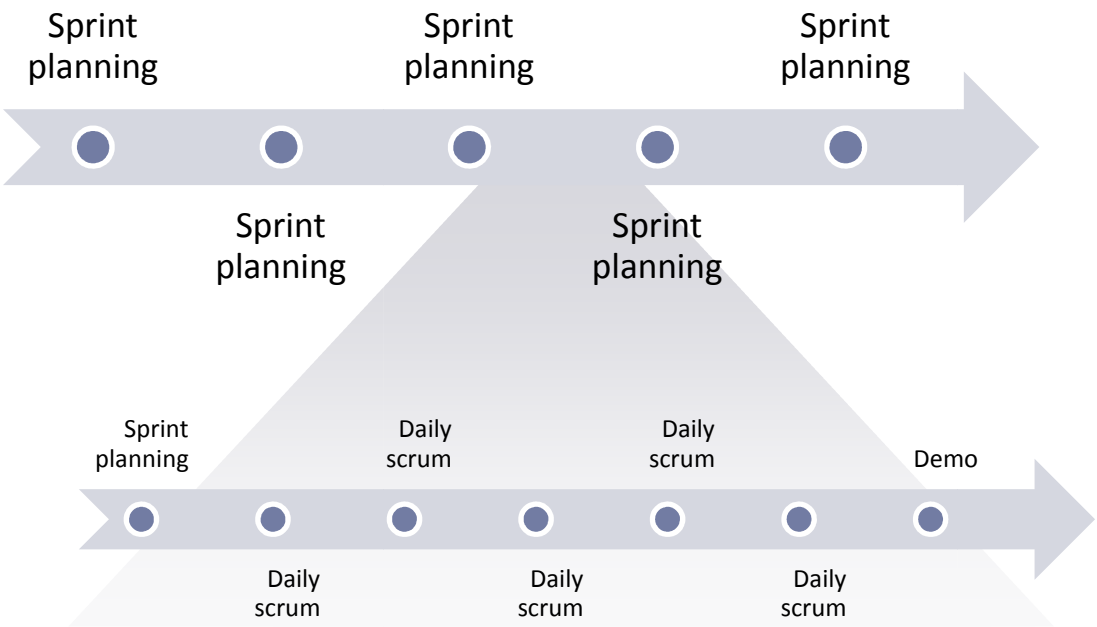
General processes

The scrum process can be described as the following:

You take the highest prioritized requirements on a regular basis and during a meeting (called the sprint planning meeting) specify which tasks are needed to fill the requirements. After this meeting, during a time boxed period (called a sprint), the team finishes the tasks and keep each other updated during daily 15 minutes meetings (called Daily scrum). When the sprint comes to the end, the team releases working software and present the result to the stakeholders during a meeting (called the demo or sprint review).



Process steps on a scrum project. The project is divided into smaller sub projects which all start with a sprint planning meeting, where the team specifies what objectives are to be met. During the sprint, the tasks needed to meet the objectives are completed. The result is then demonstrated during a sprint review in the form of a demo.



Example of sprints. Each sprint is divided into working days. On each working day, the team members keep themselves updated by having a daily meeting called Daily scrum.

During the *sprint planning meeting* it is decided what is to be accomplished during the sprint. The person who presents the options is the *product owner*. The product owner has collected a list of the requirements of the system(s). This list is prioritized so that each requirement has a unique priority. This list is called a *product backlog*.

During the sprint planning meeting, the team takes the highest prioritized feature(s). The team commits to which objectives they are to meet during the sprint. They should select the highest prioritized item(s).

During the *sprint*, the objectives of the current sprint cannot change without the product owner and the team members accept this. This means that the product owner cannot add or change features without the team accepting the consequences. The product owner can add and change details within the scope of the feature.

The principle is that the scrum team works undisturbed by external distractions during the sprint. This does not mean that they cannot communicate with others. Quite the opposite, team members are welcome to ask stakeholders and users about details, usability and preferred behavior. But outsiders should not themselves confront and disturb individual team members with questions and input. Instead one person is selected to be the one to which an outsider addresses questions. This person is called the *scrum master*. The scrum master is the guardian of the process and protects the team from distractions.

How do you get an understanding?

Not everyone involved will read a line of writing about agile or scrum. Many won't listen to the podcasts which you recommend or take the classes you want them to. But everyone involved in a scrum project need to understand the basic principle. So, you must help them. Some will never understand, some will understand but think the rules do not apply to them. But besides this, you can offer a basic understanding of the process using a simple exercise.

Remember that this exercise is just one example. Depending on the participants, you can use another example. Please observe that the different parts of the exercise require reading the chapters on the different subjects, for example project vision.

Start by saying that this is an example of how they can view a scrum project from a situation, well known to them. And the situation is that the wife comes in and says "We're having a party!". What does this mean? Depending on who you are you will think different things when you visualize what the party means. Alas, the need for a project vision. You then decide on a product vision, for example that you're having a family party at home, directed to your parents and the kids in your family and you're not having anything like the Christmas party.

You then move towards the other steps in the planning phase. You can use the different headlines in the chapter "Getting started" and just exemplify the need of these steps by using your example.

Then, when this is done you can move to the actual project. You can say that there are five days to the actual party. Consider what would happen if you made all the plans today and just expect everything to be done on Friday. Therefore, you make every day a sprint. And this can exemplify some of the issues on one of the sprint planning meetings:

On day two, you get together and see what will be done next. Getting the party hats proved easy but buying the stuff for the case was not completed since the icing was not available at the store. Can we do without the icing or will we remove something else? Doug bought the wrong type of chicken, so here is the question if we should change recipes or if should take the time to go back to the store.

You can discuss the importance of quality, of communication, the ease with which confusion is built and how you can become agile.

The exercise probably takes about an hour and time for discussions afterwards is needed.

Conclusions and summery

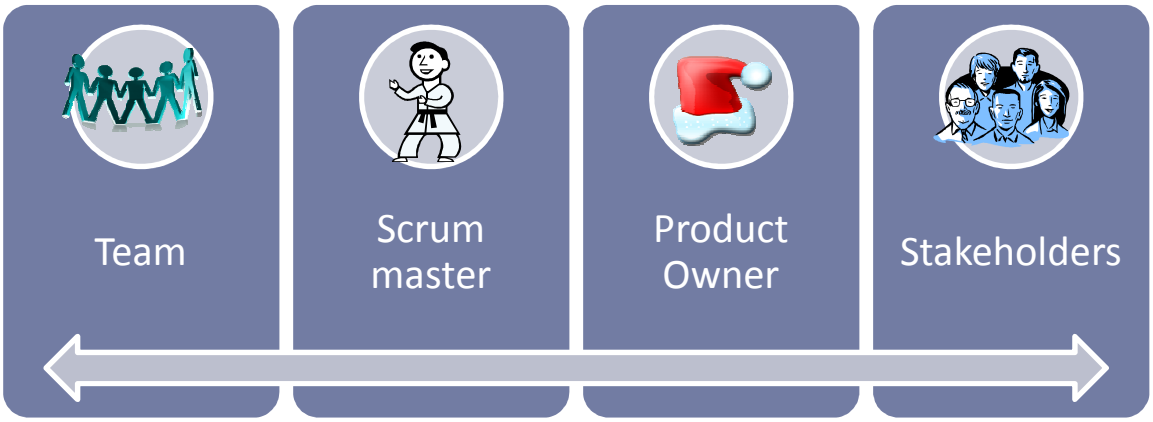
Scrum is a method which you can use to make clear the pulse which you have when you develop. It also specifies the rules for who makes the decisions what is to be done and when changes can be made. The principle is that the timeline is divided into time boxed sprints during which you don't change the priorities.

Try	Avoid
Time box everything from meetings to sprints. When people know the time frames, they most often work more effectively.	Skipping daily scrum, sprint planning meetings or sprint review, even if you don't understand why. At least in the beginning. When you've tried using the artifacts, you can perhaps understand better the implications of including or not including.
Enable that everyone involved can build an understanding of the general process, for example using an exercise as described in this chapter.	Having a time box, but not sticking to it.
Let the scrum team decide on the sprint length. Just state how often you expect a public release and let them decide if the time between public releases is divided into separate sprints	Having just one scrum sponsor on the project. You need multiple people involved, and multiple opinions on how scrum is to be implemented. If only one person have knowledge and an interest, this becomes an issue for this person
Have the knowledge exercise for managers for both team members and other stakeholders. They will need an understanding of how their teams are affected.	Reading just one book on scrum and agile. Different books have so different approach and perspective.

Scrum roles

The product owner gathers and prioritizes the requirements from all the stakeholders. The scrum master protects and motivates the team to complete the tasks and by doing so, he has a close dialog with the product owner.

But this does not mean that team members should not communicate directly with stakeholders! If requirements are changed the product owner must make the final decision and the team should all be in on the changes. The scrum master is the team representative in these discussions and if he's not up for a change, the rest of the team should not be involved. If the scrum master thinks it's an issue worth bringing up to the team, he takes responsibility for the effect of the diversion.



The team members

The team members of a scrum team should be able to take a high prioritized task from the sprint backlog and complete it. He should ask for help when he gets stuck and report on his progress. A team member is active during formal and informal meetings so he keeps himself updated.

A team member focuses on the team result, not the individual result. This means that he takes pride in his job but he's even more pride of the team effort. This means that he's not the guy who just grabs the fun tasks and ignores helping others. He the guy who drops his task if he's working on a low priority task and someone is stuck on something with a higher priority.

The product owner

It has often been stated that the product owner is the most difficult role to fill and the single most important reason for project failure.

So who is the product owner? The product owner is

- the one person who ultimately decides what is on the product backlog and
- the one person who sets the priority of all the items on that list.

The product owner is responsible for making sure that all the items on the product backlog can be understood by the team and all the stakeholders. The team needs to know what is to be built and the stakeholders must know where on the list their "stuff" can be found.

The product owner needs to take responsibility for the priority. All the items have their own priority and it's important that you get the most value for the least cost. It is all about understanding all the requirements. This to understand

- how important an item is
- the difficulty of each item
- the dependencies between items

The product owner needs to be able to explain the needs and requirements to the developer and must be able to give answers to their questions about details. This does not mean that they need to know everything but a good product owner knows who knows.

The product owner needs to be able to make decisions on the spot if this is necessary. And a product owner needs to take responsibility for these decisions.

The product owner should also give some fire and emotions to the team – to make them want to build the stuff!

The product owner should not prioritize the things he likes or which he can benefit from: a product owner must look at the whole picture and all stakeholders and be able to know which brings the most business value to the product.

Perhaps the biggest challenges in the Product Owner role just come from the fact that the many different interest and stakeholders are concentrated into a single role. This is of course not impossible, but challenging! If you are going to act as a product owner, prepare to be involved in conflicts. There is a good chance you'll be in the middle of lots of conflicts.

The scrum master

The scrum master is not the team member who does the most work. It's the person who can inspire everyone to do the right job. It's the person who can communicate to both team members and stakeholders. As with the product owner, the communicative skills are crucial.

The scrum master should also be the kind of person who has a good "feeling" for how the status is; without having to look at dashboards and backlogs. He should be able to know how things are going.

The scrum master needs also to be a person who can say yes and no. He is the person who takes responsibility when it comes to delivery and inspires the team to deliver expected value in the form of working software. All team members should feel a sense of responsibility but the scrum master is responsible.

So, why do I depict the scrum master as a warrior? Well, he should be an example to his team and work with them! A scrum master who uses all his time administrating is not the best scrum master. A scrum master takes responsibility for his team and for the end result and the delivery.

A scrum master should not see his role as full time: the best implemented scrum needs the least of its scrum master. This means that the scrum master will probably have to put in a lot of work in the beginning of a scrum project but this should be a task with declining size. A maximum of one hour should be spent and most of that time should be spent on communicating verbally.

Can you have rotating scrum master or should it be one person? To put it simple; it is easier not to change scrum master. But to make it easier during holidays and sick leaves, more than one should be able to do the tasks. I think it's good to have two persons who take on the role on a rotating schedule. Not that you change every sprint but perhaps four times a year, including holidays.

Should the scrum master be certified? Well, the certification does not make anyone a good or bad scrum master. Everyone involved in a scrum project should have adequate education and since the scrum master is the guardian of the process, he should have enough learning. A formal training can be beneficial both from the actual classes and from meeting the other participants.

Product Owner Proxies and distant /vacant product owners

A term often used in scrum implementation is product owner proxy. Most of the times I've seen this is when the product owner is inactive and the scrum master takes on his role as well and calls himself product owner proxy.

This is worthless. A role is a personal responsibility and by saying someone is a proxy: who is responsible? In most cases this will result in no one feeling really responsible.

Think of a Broadway play. When one of the actors is sick a replacement is called in. Someone else plays the part. Someone else is responsible for playing that role. That's not a proxy, that's a replacement.

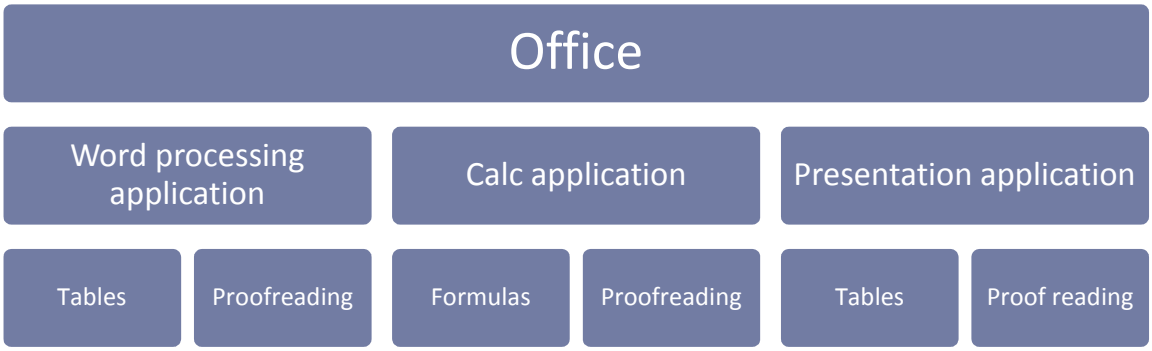
If a product owner is not part of the development, the product owner needs to be replaced. A product owner needs to be around and needs to communicate with the team members and stakeholders. A good product owner should of course have routines for involving managers and others so they are "in" on the priorities. Someone always makes the decisions. And chances are that whoever is present makes a decision, even if he's not aware of it.

Scaling the product owner role

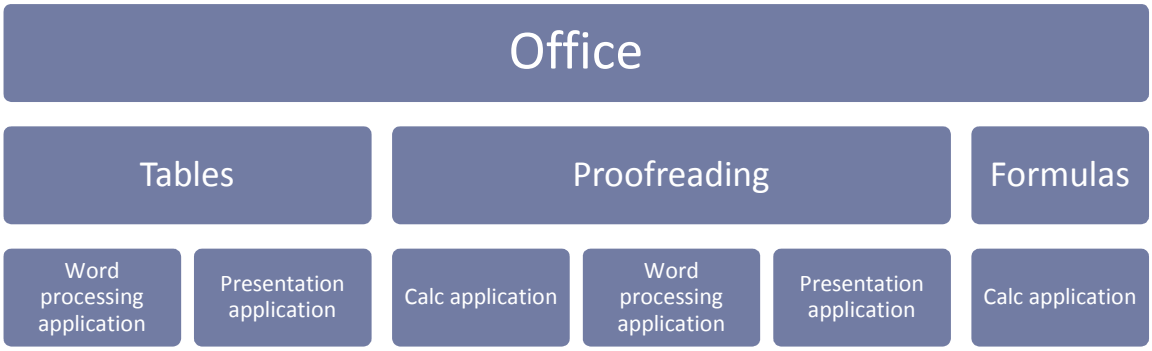
In large software developments for large organizations, a single product owner may not be realistic. Because a smooth flow in the scrum process puts demands on the product owner to the readily available for detailed questions on the prioritized issues, a single person may not be able to be all this for a large system at once.

In this case it's common to run multiple scrum teams each concentrating on certain parts of the system, where a Chief Product Owner for some large organizations fills the role of being the single source of high-level prioritization and strategic decision making. Your own organization's size, decision-making culture and readiness for scrum should guide you in choosing the right blend for you.

It is important to clarify where one product owner's responsibility starts and ends in these situations. One example is that the chief product owner is not responsible during implementation phases. It can also be a good idea to have a separate title for the chief product owner.



Example of a scaling of the product owner role in a company with an Office product suite including three application where each application has its own product owner, under each there are different product owners for the different features. Observe that this can lead to very different solutions for tables in the different applications.



Example of a scaling of the product owner role in a company with an Office product suite including three programs where each feature has its own product owner and the lowest level is responsible for the implementation in a specific application. This can lead to no one taking responsibility for one application

Combined scrum master and product owner

If you view these roles as roles, then they can be combined. But it's not for all to combine roles. A person who combines roles needs to be able to make clear when he's playing a certain role so that everyone else knows if it's the product owner or the scrum master who is talking.

There is also a risk when you combine the product owner role and the scrum master role: you get a project manager who takes all responsibility and both think and is perceived as the "boss". It is good to take responsibility but this can lessen the others sense of responsibility. To summarize, it's best to keep the roles separate.

Summery

A *scrum team* includes all individuals that are working on the team effort. Which people are included depends on your organization and which other methodologies you use while developing. Whether or not you're a member of a scrum team also depends on how you and the team view your work. This means that if for example an acceptance tester works integrated with the development team, he can be seen as a part of the team.

The *scrum master* is the guardian of the process, which means that this person makes sure that both the team members and others keep their end of the bargain. The scrum master is also responsible for making sure that obstacles are removed or handled.

The *product owner* is the person who decided how the different needs are prioritized for development.

The rest or *stakeholders* are all that have an interest in the project but do not fit the criteria of the three defined roles.

Try	Avoid
Discuss different solution for the scaling of product ownership and just not jump to one solution.	Combined scrum master and product owner, product owner proxies.
Keep the roles separate. Even if you are both scrum master and team member, use some method for making clear when you have a specific role. For example, have different e-mail addresses.	Pointing at the former project manager and without discussion deciding that this person is now the scrum master. Instead, read about the traits and select the most suitable person.
	Seeing the product owner and the scrum master as the most powerful and having managers take these roles without them having the time or the right sentiment for the tasks.

Getting started

Before you go scrum, you need to get prepared. A project doesn't become agile or scrum by itself. And if the start is rocky, chances are that people say that scrum does not work and lose faith. Losing faith is easy, regaining trust is hard!

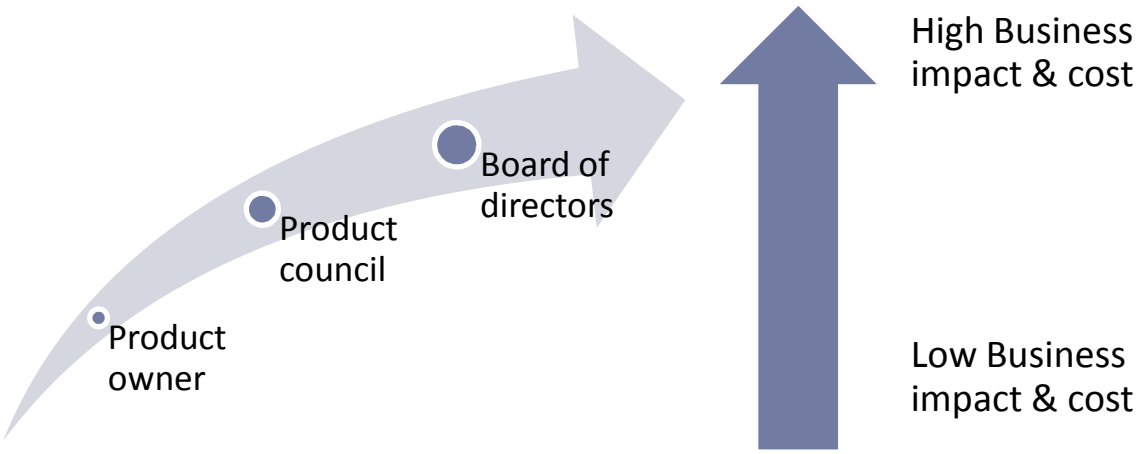
So, don't miss the preparation and think that this will fix itself eventually. It probably won't and things will just become more expensive if you wait.

Product Vision

Even if you think that this is just a project, start looking at the end result as a product. Chances are that the built system or functionality will be upgraded and changed. So, by looking at what you're building as a product, you will prepare for this.

All products need a product vision. The reason for this is that most people like working toward a goal, an objective. Also, when it comes to software development and scrum, you need to calculate business value. And how do you calculate business value if you don't know what you're building?

The product vision is nothing a product owner creates on his own: depending on which business effect the product will have on the organization, the product vision can be everything from a board decision to the decision of a sole product owner.



Who specifies the product vision? Depending how important a project is (and how costly the product will be), the decision can be made on different levels. The individuals who specifies the product vision also gives an indication on who the stakeholders are – who will be able to post requirements and who will be affected.

Geoffrey Moore wrote in his book, *Crossing the chasm*, a template for a product vision:

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

So, it need not be harder than that: just fill in the content in the parenthesis and you have a product vision. But most makes it much harder, or rather, complicated. Because by specifying a product vision you also say who you're not targeting, which businesses are not to be met. And this is not easy for business people. A CEO given the task to specify a product vision gave me a two page description.

Techniques for creating an understanding for the product vision

Including many participants when creating a product vision makes the people more involved in the basic question: why are we working? For many, it is important to know why things are done and a product vision, if used all the time, can give that sense of meaning.

Therefore, it can be a good idea to include team members and other stakeholders in a product vision workshop during which the participants discuss the product vision. The workshop should include all team members, the product owner and other stakeholders. Don't forget operations; it is often a good idea to let them know what is coming!

During the workshop the participants can come to a greater understanding of what the product vision will mean. This is also a good introduction to story writing seminars, which is discussed in a later chapter.

During a product owner training, these exercises are described in greater detail, so I'll just give you a list on examples. What you do is that the participants are divided into groups of for example five each. Cross functional groups are the best. Then the groups can describe the product using the following forms:

- A product box – if the product was to be packaged in a box, how would it look?
- A poster – draw a commercial poster for the product
- A magazine article – if the product was to be described in an industry magazine, how would it be described?

End the session by viewing all the work and letting the product owner clarify if something about the vision is unclear. This can easily be spotted during a presentation.

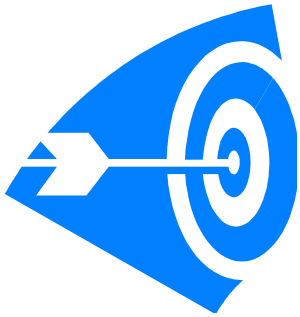
A basic timeline / road map

Do we have some important events coming up? Is there some new legislation, which requires some changes in the system? Events and dates which affect the system should be documented and communicated to team members and stakeholders so they can be discussed and planned for. This is an ongoing process and this list must be kept updated.

These events can also be used to divide the project into smaller, clearer, sub projects. This gives the team something to aim at, for example: during Q4 customer CVX will start using the system live and they will need functionality T, which has not been implemented.

It is too common that for example these dates are big and horrible surprises to the developers and by not communicating these dates and milestones, there is really room for conflicts and not met expectations from customers.

By instead communicating with the developers, milestones can instead become concrete and shared goals. It can also be a good idea to discuss what the milestones mean, and this can be accomplished by specifying a project vision for each major milestone.

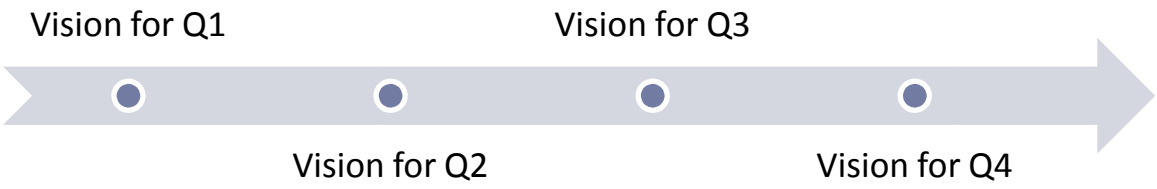


Project Vision and milestones

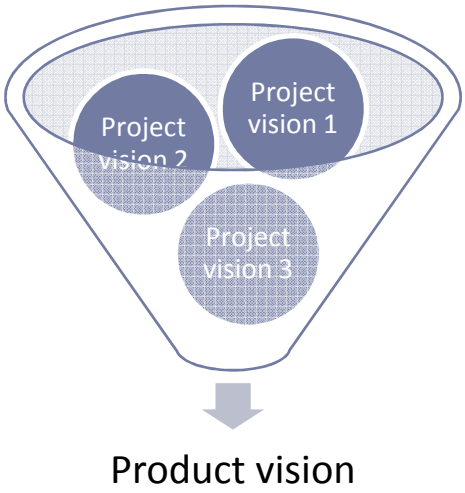
If you always look at the big picture and the long time objectives, the goal can become very abstract and does not guide you in the here and now.

And when it's abstract to the product owner, just imagine how it is for the developer. Say that you have a product vision which will be met in five years. How many of your developers will still be with the company then?

By dividing the big project/product into smaller projects were each and every one has its own vision. This can be made visual by a simple drawing:



The road to the product vision on a one year project. The project is divided into four smaller projects, were all sub projects have their own project vision



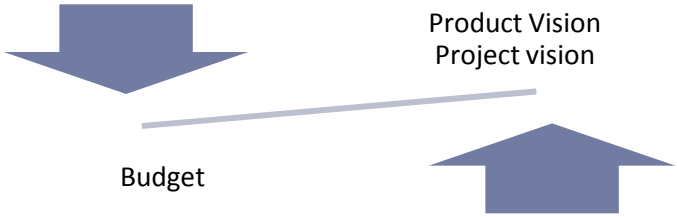
Another way to visualize the product vision and the project visions.

And for you who haven't guessed it: you create the project vision the same way you create the product vision.

Do you have the funds?

It's easy to create this big product or project vision and forget about the money. You need to know how much money you have to work with.

You should evaluate if the objectives are reachable. Sometimes you can only view the money available and compare with the visions and you know this can never become a reality. Perhaps you need to include some developers/architects to be able to see that or to give some numbers. And if you and the developers don't believe in it, the budget or the visions need some revisions. Perhaps the corporate leaders didn't understand the complexity? It might feel like a failure to cancel a project at this phase, but believe me; the failure does not feel easier just because you start spending money. Perhaps a smaller vision can be met or perhaps it's not the right time for that large project just yet?



Don't start a project where the budget and visions are not in line – if the participants don't believe in the meeting of the visions with the current budget, the project will be next to impossible to succeed.

Setting up team

In the best of worlds, you can pick and choose your team members. We all know that this will almost certainly not happen to you. If you're lucky, you might pick some team members. Perhaps you might even get them full time.

But when the team is set together, they need to become a team; they need to understand what is expected of them. Starting with a product vision or project vision exercise is a good idea. Besides getting everyone to share a vision, team members can see other's interests, function and competence. Team building exercises are also an idea.

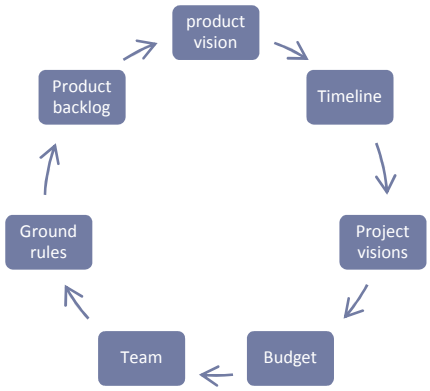
But I think the most important thing is for team members to know the outline: what are the objectives, who are the team members, is there something special during this project? And what better way to do that than discussing the product and project vision?

Setting up ground rules

Before the project start, it's a good idea to give all the stakeholders and team members some ground rules: how requirements are handled, how communication will work, how releases are to be handled. These are things which are bound to change later, but some rules must be specified from the start. Make it clear to everyone that this can change and input will be welcomed. Post the rules on the team room wall and take them up for discussion during retrospectives.

Getting real

If you viewed the chapters before like a checklist, don't! These issues should be discussed and changed during the project. When something in the chain changes, the other pieces are also affected. Of course, changing the product vision over and over again is nothing you should go into, but it's not uncommon that it takes a year to specify a long time product vision.
The most important thing is viewing these items as interlinked: if something is changed: look at the effects on the other items. There is something that is worse than a lacking product vision and that is an outdated one. So, just keep the documentation that you are willing to keep updated and a single person should own each documentation item.



Going scrum during a project

Can you go scrum half way through a project? Well, yes! The most important thing is not skipping the work with product visions, etc, as described in the chapters before. Quite the opposite, the current project has an advantage over the new project in this aspect. Already having started, these things should already be clear.

But chances are they are not. Even if the overall objective is clear to you and the business people, everyone does not have the same view or haven't been involved from the start. Also remember that new team members can more easily come into the project when these things are in place.

Are we in this together?

Software development is hard, independent of method. As Ken Schwaber put it wisely, as always; Scrum does not bring out excellence, it exposes incompetence.

Management understanding and commitment cannot be underestimated. If the CEO will not respect the ground rules and for example try changing priorities mid sprint, you're up for a rocky ride.

Start using scrum is letting go of control to enable the person nearest the problem to take responsibility. Often it will feel like the first time you let your kids ride to school along that horrible, trafficked road.

The first step towards Scrum is knowledge. All involved should have appropriate knowledge from literature or/and training. But then there are the everyday's things. How people react to a panicky situation. An important question is therefore: are we in this together? And this is a question that managers and team members should ask and respond to.

Summery

Visions, timelines, budget, team constellations, ground rules and product backlog all depend on each other. You need all the artifacts and the artifacts all change during a project. But remember that since the artifacts depend on each other, changing one thing will affect the others.

Try	Avoid
Get a product vision!	Skipping these steps because you think they are not needed or that you don't have the time.
	Believing that including developers in these phases are a cost – you will later see that the project will be cheaper if they have a clear understanding.
Use clear and open communication about the project: the needed time line, the budget and the vision.	Hiding the truth. A worse version will probably spread anyway.

Create a product backlog

When I started my first scrum project, the product owner saw no need for a product backlog: he felt that we could decide what to be done just before the sprint started. When I left the project, the first thing the new product owner did was skipping the product backlog. The result: disaster.

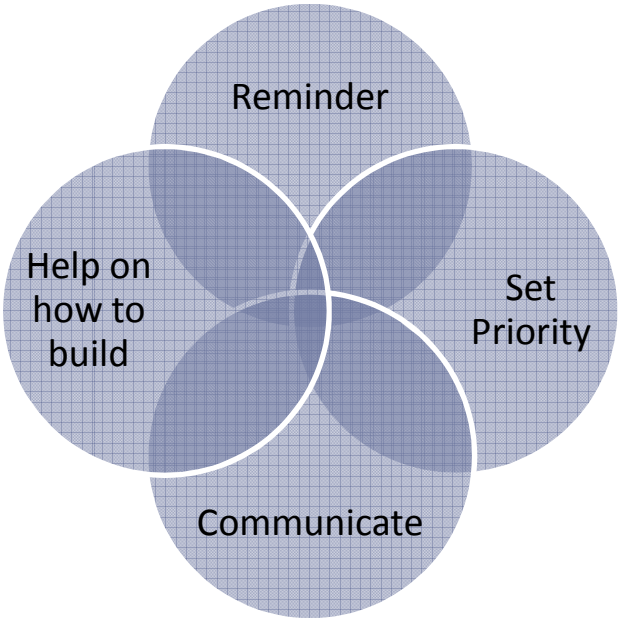
New product owners sometimes do not understand the need for a product backlog. And discussing this with other product owners, I get the same picture: it's hard to know what the product backlog is there for. The main reasons for having a product backlog are:

1. Communication to stakeholders.
   By having a product backlog, stakeholders have a picture of what is planned and what is not planned. And adding knowledge of velocity, stakeholders can also get an opinion of when stuff is to be implemented

2. Communication with developers.
   Developers can see the roadmap ahead but they can also see priorities: what is to be implemented and what can wait. This can also be a good thing during the actual development. Depending on near future plans, different solutions can be implemented.

3. Identify possible cuts in scope.
   By dividing vague functionality into more specific parts, the less important things can be cut for now. And this can be communicated to stakeholders and developers.

4. List for own memory.
   Many new product owners think that this is the most important reason for having a product backlog. But this is not the case. You can perhaps a product backlog in your head but it's of no use there. And it makes the product owner into a person and not a role.



Why do you have a product backlog?

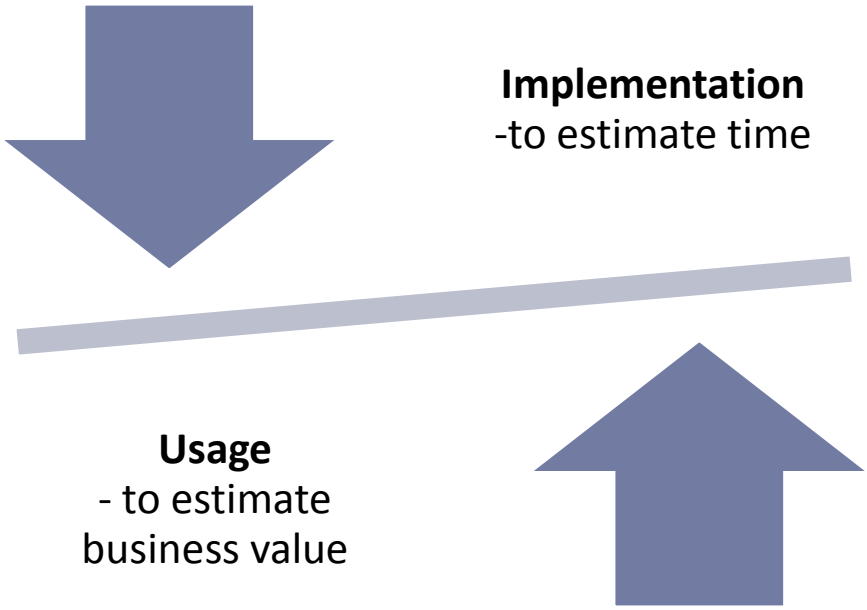
Product backlog items

What do you keep on the product backlog? All the stuff you want the scrum team to do which leads to product increment. This includes all the stuff the team can choose to do and choose not to do. This can include installation of test environment (if team members do this) and computer training. It probably will not include the team going on a corporate event. But the most important thing is the product backlog is used for prioritizing things and making it clear to stakeholders and team members what is to be done and what is not to be done.

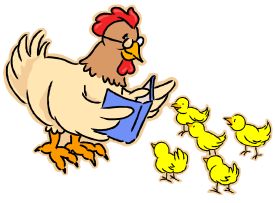
So, the things you can put on the product backlog should be:

- Something that leads to product increment
- Can be prioritized
- Can be budgeted

The form for the items is up to you, your stakeholders and the development team. Remember that their needs might be quite the opposite: the users might want to know the benefit to the users and to the business (so they can calculate business value) while the developers might want to know what technical stuff is to be built (so they can estimate time needed).



Writing product backlog items is balancing the developer's need to get enough technical details to know what needs to be built to be able to estimate the time needed, while stakeholders need to know how the functionality is to be used to be able to calculate business value



User stories

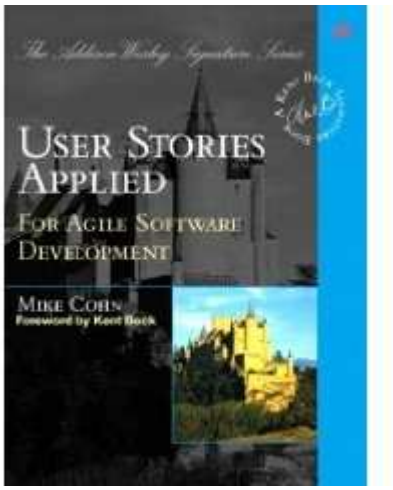
A user story is a way to write a product backlog item which focuses on the stakeholders need to understand how the functionality is to be used. By focusing on the objective of the product backlog item, the developer needs to grasp this before selecting an implementation method.

A user story is written in the form of:

As a [user], I can [functionality], so that [benefit].

Can you write everything as stories? Mike Cohn believes that 90% of all product backlog items can be written as user stories. My view is that if things get clearer if described as a user story, go right ahead. If it becomes hard to read as a story, write it in another form.

User stories Applied by Mike Cohn is an excellent book on writing user stories and a product owner class also focuses on writing stories.



I will not go into detail on writing stories: I really recommend reading the complete book. But here are some tips and advice.

User types, themes and epics

One problem is how you define different users: by type, function, client, language, user rights, computer literacy? If all your stories read: *as a user…* and the user is not specified, it is not clear for whom the functionality is built and for whom it is not built.

Let's say we have the following story on a travel site:

As a user, I can book new trips

Try estimating that. Try giving it a business value.

But if you give some more details, it becomes clearer:

As a not registered user, I can book new trips

As a booking agent, I can book new trips

And now it becomes easier to set a benefit. For example if you have these stories:

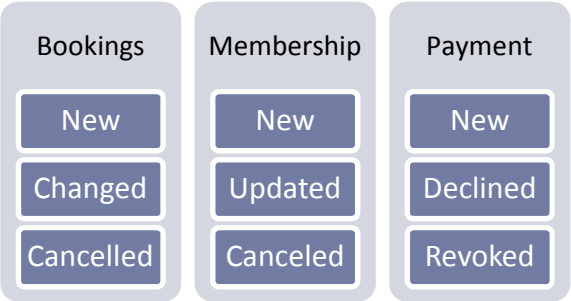
As a registered user, I can book new trips so that my new bonus is calculated

As a not registered user, I can book new trips so that I can book without being registered but so I can start being a customer without being a member

Also, it becomes easier to cut something: is it really necessary to be able to make a booking without being a member? Perhaps this is not even the business.

The problem with stories of this detail is that if you divide things into all these small chunks, the list becomes too long and complicated. And if the feature is not meant to be developed within a planable timeframe, it's unnecessary work. In this case, you might just have the story *As a user, I can book new trips* and put it somewhere in the bottom of your list. That kind of large story, meant to be divided into smaller stories, is sometimes called an epic.

A term also used by Mike Cohn is themes. These are stories that are related in the same "family" of stories. In the travel example you might have a theme for "bookings" and another for "membership handling". I look at themes as things a sales person would set on a checklist of functionality of a program and they are not written in the form of a story.

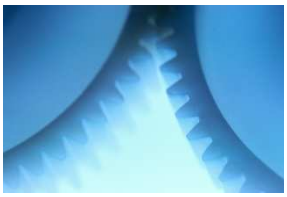


Themes with epics. Each epic can be written in the form of a story and each epic can be divided into smaller stories. Remember that a story can include many themes. If there is a new customer, this might involve both new membership and a new booking

The benefit

Sometimes the benefit is not included because this might be obvious. But remember that something might not be obvious to all and don't confuse obvious with difficult to explain. If you have problem finding the words, things might not be as clear as you think.

Remember that every time someone asks why something is as prioritized as it is or need an explanation of a story, the benefit might be missing or not accurate.



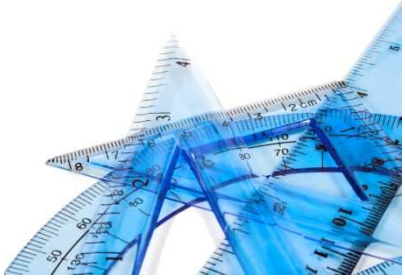
What about details?

When I described the form of a story, I of course talked about the title of a story. If you're using story cards, you print this on the front side of the card. If you're using a digital list, the story will be title of the product backlog item. The details are then added as they become clear. Mike Cohn says that stories are placeholders for a discussion. With that he means that when you present or read aloud a story for a developer, he will start asking questions. The answers of these questions are then printed on the back side of the story card or in a details field in I digital system. As I often take on a role as an acceptance tester on my team, I often ask how the story will be tested so I know what will be tested automatically and what I need to focus my manual test on.

Other examples of details can be a user who has a lot of input. Ideas and questions from stakeholders are also example of details which can be put on the back side of a card.

Often developers ask questions like "is X included in the story?" If a decision is made for a story, this is also added to the details of the story. This means that the nearer the completion of a story you are, the more details can be found on that story.

Remember that adding to details can make the estimation inaccurate. It is therefore important to look over the details now and again and don't be afraid to ask if this and that affects the estimation.



Bugs

You don't have bugs in your system, do you? Well, of course you do. So, how do you handle them? Depending on whom you ask, bugs should or should not be tracked on the product backlog.

The reasons for not having bugs on the product backlog can be many but if you're into lean software development, bugs should be handled directly and not be kept in a queue. Also, bugs according to some, should not add to velocity and should therefore not be kept on a product backlog.

My view is that there should only be one list and if you track bugs, they should be visible on the same list as everything else. If you keep a separate list you will have to have a clear cut definition of what is a formal bug and perhaps have a separate process for handling bugs. By handling bugs like everything else, there need only be one process for getting things done and it's important for stakeholders to understand that bugs are not fixed for free: they need to be prioritized to be done. And also, I don't think bugs should be hidden in the process.

One more reason to keep bugs in the same list is that the team then cannot escape from their own maintenance debt. So if you want a team aiming for quality, do not hide bugs. Some organizations solve this balancing issue by keeping part of the sprint capacity available to bug fixing or having a weekly or bi-weekly rotating role of "main-fixer" within the team. Even though this is not ideal – since you shouldn't have the bugs in the first place – it's certainly better than the alternative to hide them from the team. Also see more on this matter in a later section on product planning.

Do fixing bugs add to velocity? I tend to view bugs as not finished stories and when you calculate total cost for a feature, the cost for bugs should also be included.

Total cost for a feature with two bug fixes.

You will probably not describe bugs in the form of a story: this probably just makes it harder to read but it's important to state why the bug poses a problem. Sometimes, the problem can be obvious but often enough something can be seen as trivial to a developer but show stopping for a user. Many of these bugs fall in the usability or expected behavior category.

You might also have different attributes when describing a bug. Perhaps you need to specify steps to reproduce the defect and the wanted behavior.

Other things

What about the other stuff? Technical stories, as one of my previous team members put it. You don't want team member's dentist appointments on the backlog but you want to see all the things that can be prioritized and not prioritized.

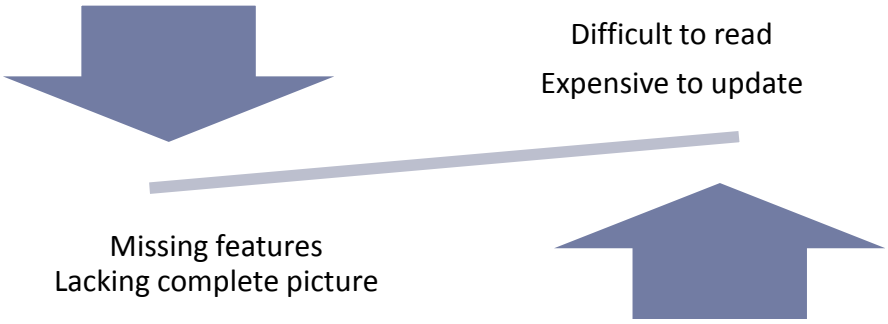
Product owner should not hide technical tasks or team member training: this only gives room for conflict and developers not feeling that "their stuff" gets prioritized. This can also lead to developers deciding to do this stuff anyway, which of course affects the sprint.

Also, by having this on the product backlog, the effect of for example paying off a technical debt and refactoring can be tracked. For example, if there is a problem with a feature which was built so that manual testing is required, the refactoring of this feature is an excellent thing for the product backlog. After implementation, the bugs in the feature as well as spent testing time can be tracked to view the effects. Training has more long time effects and effects can most often not be tracked so easily, but the principle is the same.

Should the other stuff be written as stories? As I stated before, it should if it makes the things more clear. Adding the reason clause can be a good thing independent on if the story form is used just to make stakeholders understand why this is needed and again, it forces developers to explain why something is needed. For example: *Refactor feature X so it can be tested automatically* is an example of a product backlog item which does not use the user story form but has a reason attached to it.

How much (and small) is enough?

If your product backlog does not include all the wanted features, it is hard to know what is missing and what can be prioritized. If a product backlog include all wanted features the length of the list can result in it being difficult to read and it becomes very expensive to keep updated.

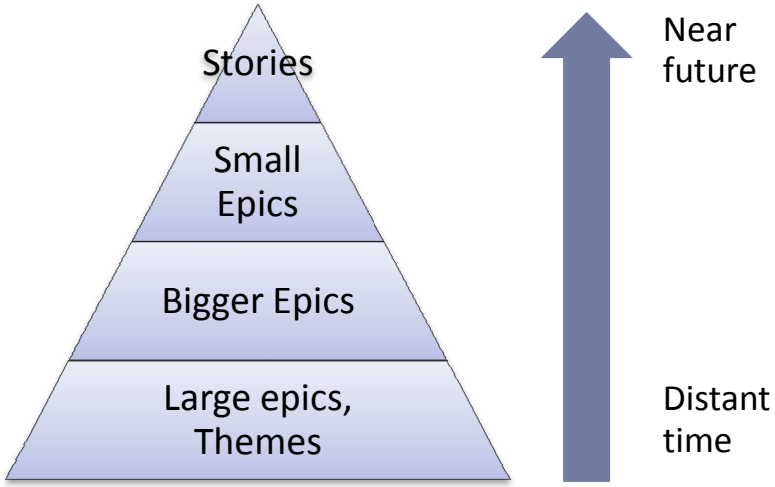


Therefore, you need to decide on how to find a middle way and have a product backlog which is readable and which still does not lack wanted details.

One solution is not getting into details when it comes to product backlog items which lie far ahead in the future. Keep these items as themes or epics and just divide them as it becomes necessary.

Here is also a possibility to scale the product owner role: a chief product owner can be responsible for forming the large epics and the themes, while the factual product owner is responsible for the stories which end up on the sprint backlogs.

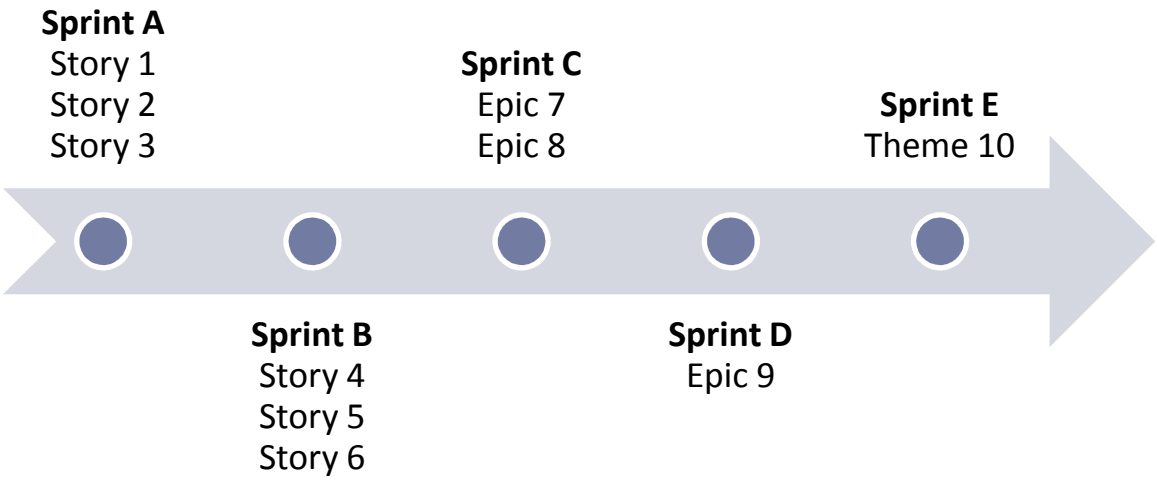
You will have some smaller stories in the lower part of the product backlog as a result of the division of stories and things being cut. An important task is then to know when to keep a cut product backlog item and when to through it away when you think it will not be implemented in the planable future or if the risk for the item becoming obsolete or very different when it comes to implementation.



So, how large should the items on your product backlog be? Well, the large stories and epics can be of any size.

The first time you need to start thinking about story size is when something needs estimation. You often need an estimation to be able to prioritize. In this case, make the story/epic/theme so small that estimation can be made.

When it comes to implementation, I try to have stories for two or three sprints fixed and doable. The reason for this is that if the product owner becomes ill or there need to be some serious re-planning, you have something to work with. Stories which are up for implementation are so small that three can be implemented during a sprint. This is to make it easier not to fail a complete sprint in the worst case scenario. Imagine having just one story on the sprint backlog and this story cannot be completed. Then you have nothing to show on the demo. The three story rule also gives room for those technical stories. If those were so big that a full sprint was spent on refactoring, again not so much to show on a demo.



Example of a rough plan. In this case we have the need to give customers and sales people an idea on what is going to happen within the four nearest sprints. The upcoming two sprints are rather well defined. One or two stories may be changed. The further into the future, the rougher the description becomes and for the sprint which lies five sprints into the future, there is just an idea which area is to be addressed.

This also means that the agility is depending on how far into the future a feature can be implemented. If the information that story 5 is to be implemented in the near future is changed and this story is moved to sprint F, how does this affect all the stakeholders? Perhaps a customer dependent on story 5 is due to go live during sprint D? Agility means the option to change but that does not take away the effects. But it's also important to make clear how and when priorities and plans can change and how this is communicated. The salesperson who promised story 5 might not be so happy to learn this on the sprint demo for sprint B.

Tools and gadgets - visualization

Mike Cohn says that of his entire near 2000 scrum projects, the need for anything else than paper story cards has been the exception.

Story cards

Mike Cohn prefers using story cards. Story cards are just plain paper cards, a bit larger than a business card. On the front page of the card, you write the story and on the back side you write the details. If the details don't fit, the story is probably too large so you divide the story into multiple stories.

As a travel agent, I can view the travel history of a client so that I can refer to previous experiences during my sales procedure

Example of Story card

Prompted to view history if such

See who handled previous booking

Automatically see if previous bookings have been made

Example of back side of story. Observe that these can be made into smaller stories, if the main story becomes too big. Also see that one of the notes is a question – every need not be answered up front but this is an issue that needs to be addressed during implementation.

Mike Cohn likes story cards since you can always create new ones on the fly and they are easy to throw away.

I don't like story cards for one reason and that they are hard to visualize for others and sometimes others need a list of the product backlog. Keeping a parallel list seems to take away the positive thing about story cards.

Post-it stickers

A version of the story cards is the post-it stickers. The difference is the sticky backside and that you can hang them on a wall a bit easier.

I've tried both story cards and post-it stickers but I came up with the same problem – often stakeholders wanted a digital product backlog for their planning and then I sat there with Excel again.

Microsoft Excel

I guess most product backlogs are kept on post-its or Microsoft Excel. The good thing about Microsoft Excel is that the problem with sending a copy of the backlog is fixed.

You can also add more attributes to a product backlog item, for example numbers (so you don't have to refer a product backlog item to the complete title but a simple ID).

The problem with a Microsoft Excel file is that it's a personal file, kept on someone's computer. Yes, you can send copies but the copy becomes outdated when the "real" product backlog becomes outdated.

Yes, you can keep the product backlog on a file server and that does makes it easier to access. But remember that even if the product backlog is owned by the product owner in the sense that he is responsible for the content and should be the sole editor of the file, a product backlog is the property of the project and the product.

Intranet sites

Keeping a product backlog on a intranet site solves some of the visualization problems if the intranet is used by the stakeholders.

I've been using SharePoint lists for handling my product backlog. By doing so, I could also add the advantages of using Microsoft Excel, since there is an Excel integration and since there are the possibilities of direct grid view editing.

If you don't have a working intranet or if the product owner does not know how to use the intranet for publication, this could cause a problem if selecting this option. Also, you have a problem if all stakeholders or team members don't use the intranet.

Team foundation server

Team System Server (TFS) is integrated in Visual Studio and besides having a work item tracking system. With integrations to SharePoint, Microsoft Project and Microsoft Excel, a web access, TFS can become a tool which takes the advantages of intranets and Excel and combining this with a tool that is already used by the developers. Since code check ins can be associated with backlog items, other advantages can also be met.

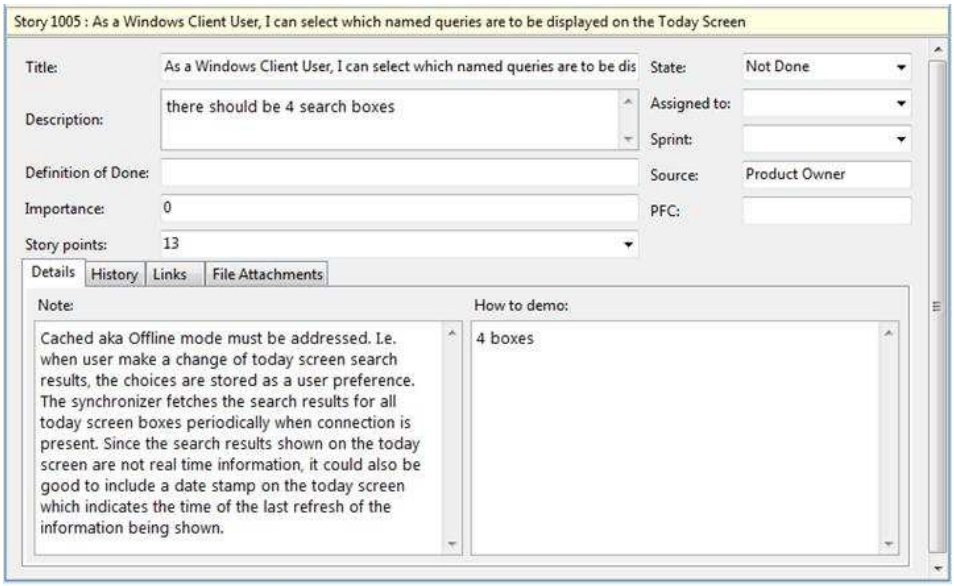
TFS has a work item tracking system and if you're keeping the product backlog in TFS, the backlog items will be stored in the form of work items.

You can define your work work item types, where each type can have its own attributes and there is also a work flow engine, which enables you to get some help while following your process. You can create these templates on your own or you can use (or modify) templates. There are templates especially made for scrum, for example escrum from Microsoft and Scrum For Team System from Conchango. On CodePlex there are always some new or updated tools and gadgets which you can use to live your own TFS dream.

The positive side to using TFS are besides the visual tools for the sprint (in the form of visual dashboards, etc) the integration with a system that the developers already use. If they/you/their manager wants to, they can associate their actual coding with things from the backlog.

The downside is that this is not a simple or easy tool. For me, having worked with TFS as a tester on the team, the learning time wasn't too big, but it's not an easy task for your average product owner. If this is a problem, the product owner can stick to the Excel integration and the product backlog items are still found in the work item tracking system of TFS.

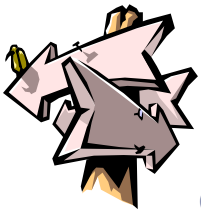
On my blog, I have some posts which address keeping the product backlog in TFS.



Example of story in TFS

Other tools

Just search online for tools and scrum and you'll find there are loads of options for you looking for the perfect tool for your product backlog. Some are free while others will cost a fortune.



Choosing a tool

There is probably no such thing as a perfect tool. Different groups, different teams, different projects and different product owners have different needs and priorities.

List all the needs you have for the product backlog. Look at what is necessary and what is desired. And start using a tool. You, the stakeholders and your team will probably change their minds later on and there is good chance you will have to move your product backlog as you go along. And this is probably a good idea: moving a product backlog is something like a physical move: you tend to throw away stuff you don't need or want anymore.

My advice to you is that you start using story cards or post-it notes, if you are unsure what to use. Since this is the simplest solution, you will either love this or soon find the downsides. But then you know what you miss and then you can choose a tool that meets the needs you and your team have.

Attributes describing product backlog items

This is just an issue for you who keep the product backlog in some kind of digital form. If you're using Post-its or story cards, this is never an issue.

The mandatory attributes of a product backlog items are:

- Title
  Unique name which describes the product backlog item. Can be printed in the form of a user story.

- Priority
  Unique numeric value which specifies the place of the item on the list.

- Estimation
  If an estimation is needed for setting the priority, this is specified. The estimation can be blank on a product backlog item. Don't estimate things you don't plan to do in the near future or which you don't need an estimate to prioritize. It is important to have an estimate before you start building the product backlog item.

Depending on your needs, the following attributes can be set:

- Description
  Notes from business people, which can help in understanding objective
- Notes
  Notes from the developers, which help with implementation
- How to demo
  A list of things that should be included in the demo
  Tests which should be performed but might not be obvious to the developer or tester
- State
  In which state the item is in. This applies if you keep the history in the product backlog and if old items are not just deleted.
- Assigned to
  Most commonly used when a story is not understood or cannot be estimated by the team.
- Sprint
  The sprint during which the story is completed.
- Source
  Who's the source of the story. The person who best should be able to answer questions or validate the solution.
- Epic
  If the story is part of a bigger epic.
- Theme
  If themes are used to categorize stories.

If you have bugs on your product backlog, the following attributes can also be of use:

- Registration date and Close date
  Can be used to enable the creation of a value stream mapping.

- Steps to reproduce
  Steps that anyone can follow to reproduce the bug. Should be mandatory.

- Severity
  How serious the bug is perceived

- Environment
  In which environment can the bug be found. Is the bug found in the environment of a specific user or customer or all, or just the development environment?

- Affects versions/releases
  Versions/Releases which are affected by the bug

When it comes to attributes: don't overdo it. Remember that you need to keep all this updated. Add as few attributes as possible from start and then add them as they are being missed.

Summery

All scrum projects need a product backlog that can be used by all stakeholders. The items highest up on the list should be described in greater detail than the items on the bottom of the list. All the items on the product backlog should have their unique priority so that all items are prioritized compared with each other.

Choose the simplest tool possible for keeping your product backlog but change tools if your requirements are not met.

Keep everything that increase product value on the product backlog, including technical items and bugs. Use the user story format when appropriate but even if you don't use that format on some items, don't forget to include the benefits of all items so people can understand the priorities.

Try	Avoid
Create a product backlog, even if you at start don't see the benefit. Only after five sprints can you make yourself an opinion if it's worth it.	Decide on a cool tool first. Use the easiest tool that suites you.
Relative estimates. Even if you think people will not understand it, perhaps they will and if so, the time spent on estimates will decrease	Having a product backlog which only you can access.
Story writing seminars	Having separate lists with things that also need to be done or different backlogs. Confusion!!!!

Gathering requirements

But how do things end up on the product backlog?

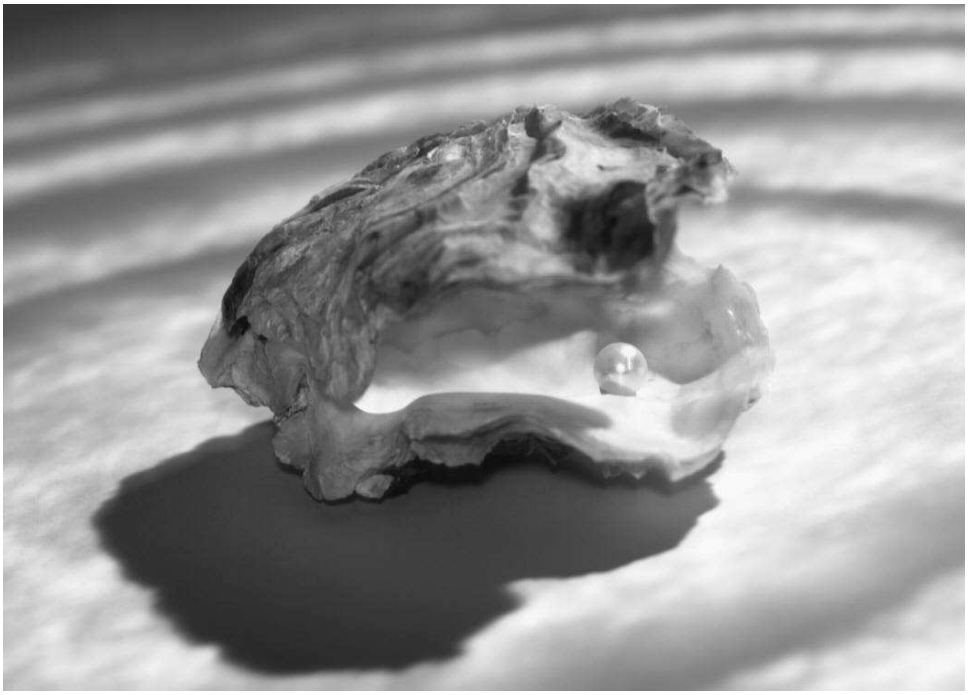
One way is to write everything yourself. Then, you have your own product backlog. But no one else will feel ownership of the backlog items and chances are they do not understand what you mean and if their items are included.

And be sure, even if you think you know best, you don't. And even if you think you have the best stories in the world, what better way to test that by throwing everything away and see if the stakeholders, users and team members come up with the same ideas?

I had a number of rather good stories for a feature but without showing them to the participants I held a story writing seminar on the feature. My boss thought me mad since he liked the stories. Why throwing away all that time from developers and users when my stories were so good?

As it turned out, the story writing seminar ended up with roughly the same stories as I had. Many of them were slightly better and almost none were worse.

But the most important thing was that the participants knew the stories and when it came to implementation, I ended up on vacation. But since many of the participants were on site, they had no need for my input.

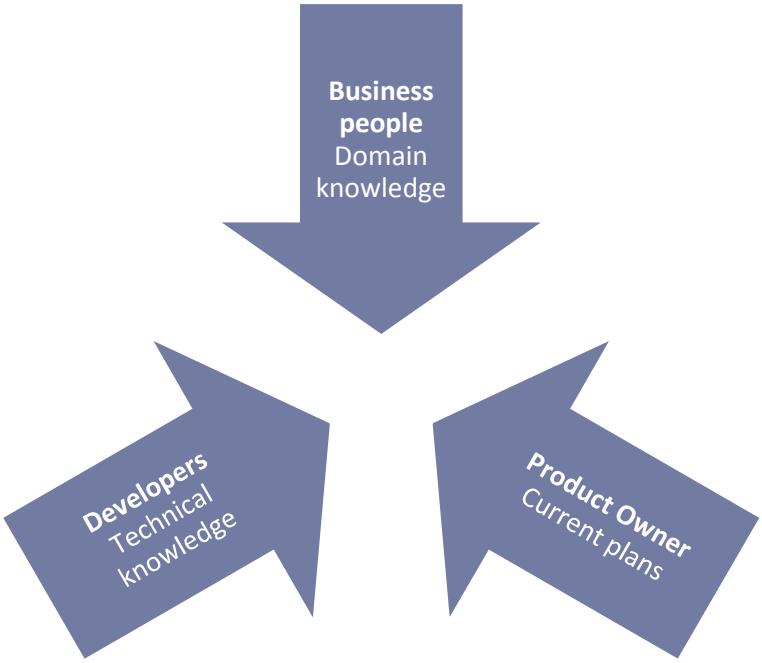


Even if you think you know best, you don't. Ideas are like hidden treasures in users, developers and other stakeholders.

Story writing seminars

So, I'll start with story writing seminars. This method is my favorite choice, when it comes to requirements gathering.

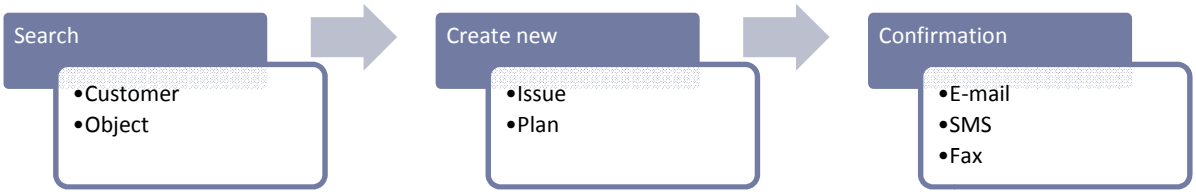
What you do is that you gather users, stakeholders and other business people around a problem in the form of a theme or an epic.



The main groups during a story writing seminars. What the different participants bring to the meeting. If one group is missing, this input will be lacking during the seminar, which will affect the end result.

They then try to define which stories are needed. As a product owner, I try not to be too active in the actual story writing. But it is important the product owner is present. To listen in on the discussions and also give input on current plans and priorities.

Business people describe the problem domain. I try to use personas and example of situations when the epic or theme is an issue to the actual users. There is also a good thing to in advance involve one or two users and let them describe their process in this respect in the form of a storyboard.



Example of a simple storyboard. The different steps are described very short and special questions and alternative paths are listed. Can also be drawn as images/ui mockups.

The use of storyboards and personas tend to get the participants to use a common language so that everyone knows which problem is to be discussed. It is also important to make clear which issues are not to be addressed by the seminar. It is better to have many and short seminars instead of long ones with a vague content. A two hour session is about what people can take without losing focus.

Another thing to consider is the number of participants. I say "the more the merrier", but this is of course a monetary issue. And if the groups are too large, some participants tend to become inactive. I think groups of between four and nine are the best and if there are more participants – divide the meeting into separate groups.

The output of the story writing seminars is product backlog items. This means that the participants do not only write stories. As explained earlier in the chapter on the product backlog, the product backlog also includes things that are not stories and these items can also be presented on the story writing seminars.

Besides the product backlog being filled with new stories, the story writing seminar should be viewed as an excellent opportunity to exchange knowledge.

What do I get from the story writing seminar? Business people are updated on technical details and plans. Developers and product owner gain domain knowledge.

Even if the story writing seminar was being carried out in separate groups, I always end by gathering all the participants and then I go through all the stories. I then ask about their opinion on which of the stories are mandatory to meet the lowest of requirements of the epic or the theme. I want all to hear in on this and I prefer if this becomes a moment of discussion.

Remember that the story writing seminar is an excellent opportunity for developers to get domain knowledge and to get a feeling for the users. Don't just send the most verbal and interested developers to the seminars. The basic rule should be that all developers participate. If you think it's too expensive, think of the expense of a developer not knowing why he's building the feature or what the user finds crucial.

Meeting users

Story writing seminars are wonderful when it comes to formal meetings where you can meet users. But users in a conference room are not the same as users in the field. In a conference room you hear users talking about things they know they want (or think they know they want). But in the field, you can see how a user works and get a feeling for what is lacking, what could run more smoothly and what is too complicated.

While meeting customers in the field (and with this I don't mean that they have to be out on a field…), it's important that they don't feel like you're spying on them or that you're there to make them look stupid. You are there for them. They should feel confident in sharing every sneak hole they are using to carry out their job.

Remember to get back to the users! Imagine how it feels for a user when the development person comes over, listens, and then get back to you and say that they are working on or have fixed something they talked about. Also, the meeting of users in the field adds much to story writing seminars. Instead of just using personas, actual live experiences can be discussed.

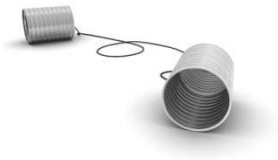
Informal meetings

Not everything in life has to be formalized, least of all, meetings. When we talk about stupid, worthless meetings, how often do we mean the informal meeting in the corridor or the talking around the coffee machine?

Not all people come to life in a conference room. That does not mean that they don't have something interesting to say. When working in facility management, I got myself updated on certain sports which I knew the users liked. By saying something about the game last night, we could start an informal discussion, which often ended up in something useful for my product backlog.

Bug lists and product feedback centers

Since you don't want a too long product backlog, there will always be things that people say or report that doesn't make it to the product backlog. How do you handle those? Outside the role as a product owner, you or someone else can keep a separate wishing list or some kind of forum where anyone can post any kind of question or input. This can be a great source for your product backlog. The important things here are that there is no formal form for the items, that outdated stuff doesn't remain too long and that some of the items become addressed sometimes. I try to comment on items which you could sense that some thought was put into, just to get the writer to feel that someone was listening.



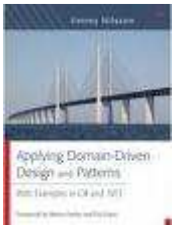
Visualizing processes and connections

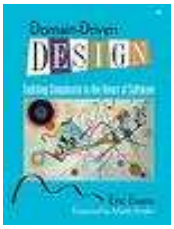
A consultant was given the task to gather all the requirements of a new system. She presented the team with a product backlog in the form of an Excel list with requirements. The problem was that you had a hard time penetrating the list and understand how the different components worked together. You had also problems understanding how the different processes worked and were connected to each other. The product backlog is a glorious thing in certain instances, but it often a bad tool for visualization.

I use different techniques and tools. Some become part of the system documentation. Some are just temporary for a specific meeting or event. But they are separate from the product backlog, since they might describe processes and functions not to be implemented or things that are already present in the system.

Conceptual, domain and information models

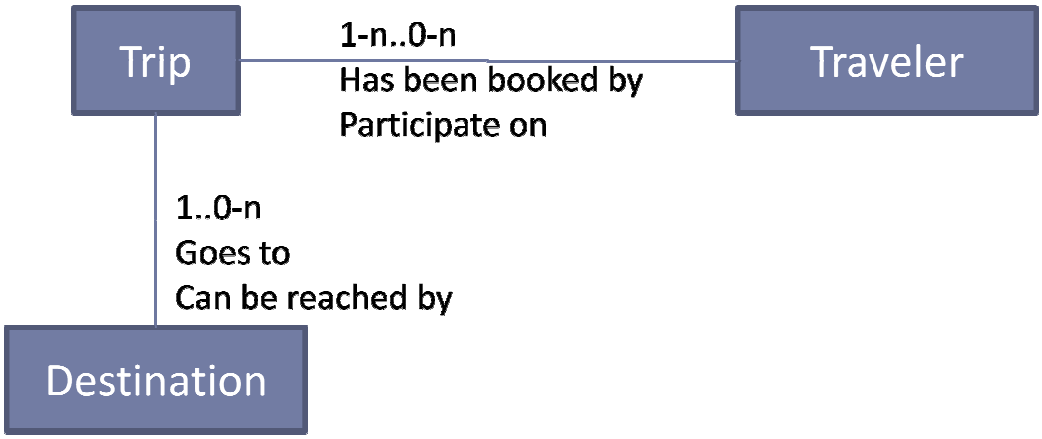
When it comes to defining relations between concepts and definition of concepts, the usage of domain, conceptual and information models are very useful. These are also powerful tools for documenting the system. If you are unfamiliar with domain models, you can get all the information you need in books by for example Eric Evans or Jimmy Nilsson. A shortened version of Eric Evans book can be downloaded from Info-Q.





Apply Domain driven design and patterns by Jimmy Nilsson and Domain Driven design by Eric Evans

But for you who have never heard these terms before, diving into these books can be a utopia, especially if you're already reading books on scrum. Creating a simple model can there be a solution. The easiest method for creating a conceptual model is using a simple drawing tool like the one in PowerPoint. UML notation or something like that can be used.



Each concept is then described in the information model, where the definition of each concept is described. The reason for this is to create a common language for all participants. If correctly applied, a domain model is applied in the actual code and both stakeholders and developer use the same terms. Since different users use different words it is though important to include used synonyms in the information model so differences in usage of terms are spotted.

Below, you can find a very simplifies example which show what you can include in the information model:

Trip: The process which brings travelers to a destination.

Relates to: Destination, Traveler

Attributes: Time (Date and time), Aircraft (Alfa numeric)

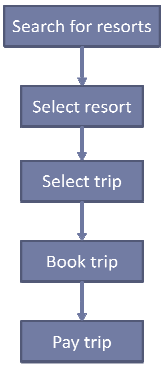
Synonyms (used by different customers or users): Journey, Event, Voyage

Observe that the model is nothing you need to have finished before you start your project. Instead see it as a continuous work in progress which evolves with the system. By including an updated model in the delivery folder of each sprint, it also becomes easier to see how the concepts evolve and change between releases.

Having an updated model takes more time than you might expect and it's important to include the updating of the model in the planning processes. Once you start slipping and don't update the model, chances are great that the model ends up on a shelf and is never used again.

Story boards

When it comes to describing processes, writing a simple story board can be of great usage. To create a simple storyboard you can use programs like PowerPoint. Keep every step of the process on an individual slide. In its simplest form, a single sentence is used to describe each step. You can also use simple flow chart diagrams.



The reason I like printing each step on a slide is that I can then print that on paper and post the process on the wall during meetings. Participants can then make notations on each step and we can freely walk around the process.

I've always used story boards as temporary documentation, since I've never prioritized keeping them updated but they are excellent basis for test protocols.

Summery

Don't let the product owner write the product backlog by himself. The product owner has full ownership of the priority of the product backlog items, but the description of the items should be set by many.

Story writing seminars, user interactions and collecting data from different user contributions lists are different sources which should be used.

Don't forget the developers and operational staff when forming product backlog items! They have valuable input and by involving them earlier they gain domain knowledge and an understanding for what is needed.

Use storyboards and models to visualize processes and connections between concepts. Decide if these artifacts are used temporary or if they are to be part of the documentation.

Try	Avoid
Discuss different solution for the scaling of product ownership and just not jump to one solution.	Combined scrum master and product owner, product owner proxies.
Use simple models for visualizing connections and processes	Introducing complex documentation which you don't have time to keep updated or is not used by anyone. Focus on the stuff which brings the most used during and after development.

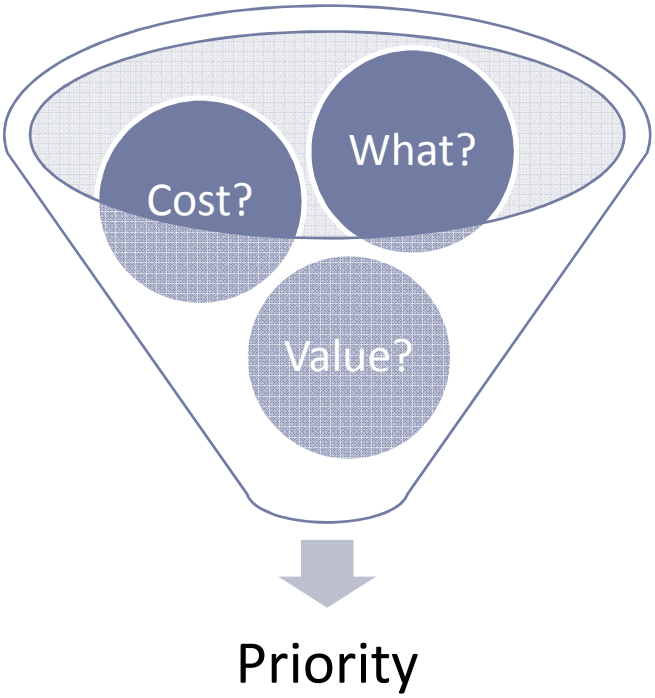
Product planning

Product planning is about planning for deploying to customer and to sales people:

- When can we start pitching feature X?
- When can we start installing a version which fixes bug Y at our customers?

In other words, how people outside the development group can plan their life and work. A feature which has been built but not sold or is in use is such waste of time. Therefore, this is one of the most important tasks for a product owner.

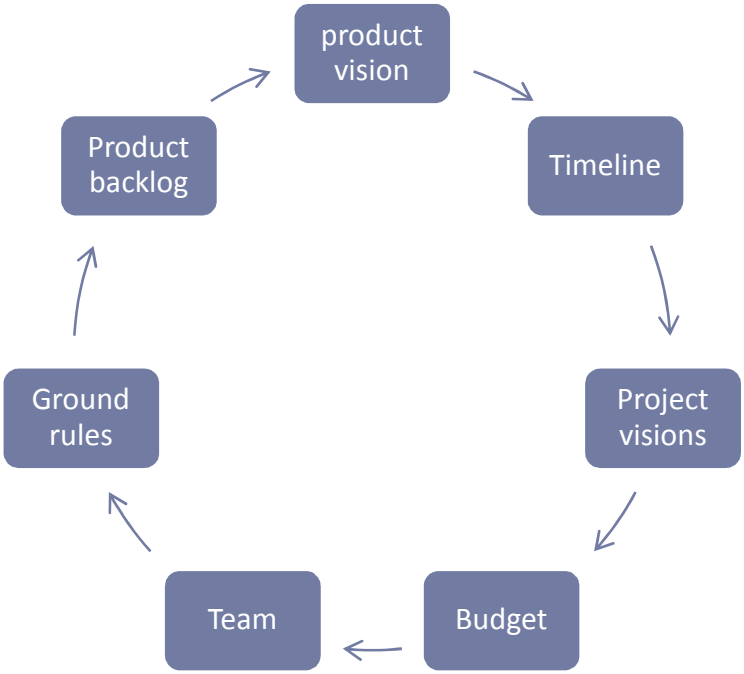
To be able to plan for something you need to know when something is to be done. To know when something is done you need to know how it's prioritized. And to prioritize, you need to know what to build and how expensive it is.



How do you set a priority on a product backlog item? By specifying stories and calculating business value and cost you can set a priority.

Another factor which you need is to know the speed of the development. Even if you know how big something is, you need to know how much the actual team can do when it's time to do the stuff. This is called Velocity and it's a key value in Scrum and agile software development.

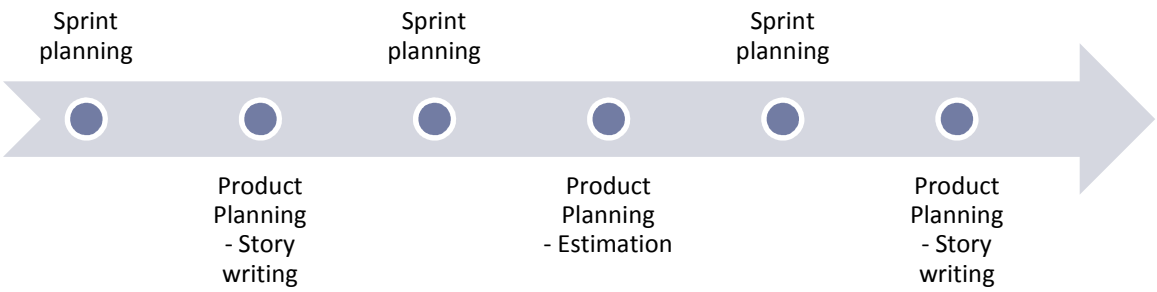
Do you remember this from one of the first chapters in the book:



This is also a way of describing the different elements of the product planning.

Product planning should be an integrated part of the software development process and the time spent should also be planned.

One way to make room for the product planning is to plan sessions for this the same way you do sprint planning. Let's say you have sprint planning meetings every other Wednesday. You can then have product planning meetings the other Wednesdays. You don't have to have fixed content for each session: sometimes the meeting is a story writing session; sometimes it's a budget meeting, etc. But the important thing is that there is a pulse to the product planning so that everyone knows that "Wednesdays are planning days".



Giving the product planning a pulse. Every other Wednesday there's a sprint planning session. The other Wednesdays, a product planning session is held. The actual content of each session varies, but each session should have a specified objective so the right participants are present.

Hot fixes and handling of bugs

One of the most crucial elements of scrum is keeping the pulse. We deliver on the delivery day.

But then things happen. Buggy software gets deployed and suddenly a customer is losing money and the next sprint planning meeting is in 10 days.

An option presented to me by Mike Cohn is using delivery sprints. This means that parallel to the scrum project, one or a few persons work on a separate "team" or branch helping out with bug fixes and other type of help to outsiders. The person on the release sprint is available for hot fixes and if an urgent problem arises, the product owner can decide to make a hot fix with the specified content. The problem to be addressed does not have to be a formal bug, but since this is costly, the decision should be well based. When the person(s) working on the release sprint does not work doing hot fixes, he's available as a resource for the rest of the team. But they need to take into consideration that the resource can disappear any time.

This should not be the common scenario: if you have one or a number of guys out there making hot fixes all the time, this is caused by a much bigger quality issue, which in itself needs solving, but in time for major releases or occasions when for example a big customer goes live or is upgraded, this is a good option.

The person(s) working on the release sprint must remember that his work is not just fixing the hot fix in the release; it is also making sure that the problem is also fixed in the branch that the rest of the developers are working on. This means implementing the same solution or another, better one, if a quick fix was set in place to fix the acute problem.

Having release sprints with this type of hot fixes is expensive and can be a source of conflict within the team. A clear, open and constant dialog with the scrum master is needed. It is also important to remember that the purpose of the release sprint is fixing problems, not causing them. In one case, five bugs were addressed during a release sprint. But the fixes all fell in the category "quickies" and there were regressions on all bugs. None of the fixes were accepted by the team members when it came to merging the fixes. So, don't bail on quality and make sure that the person(s) working on the release sprint discusses the solutions before merging.

Every third sprint has a parallel release sprint with possibility of hot fixes.

But cannot a scrum team act on a normal bug? Well, this is up to the scrum team and the product owner. If all feel that a bug can be added to the sprint backlog during the sprint, this is of course OK.

Estimations

How many days does it take to build a new house?

When you ask a developer how many days something takes to build, chances are that this question is as easy to answer as the question above. The question leaves room for so many questions? Which type of house? Who is going to build it? Is all the things needed bought or do I need to fix that too? Will I be working full time or is there something else needed to be done?

To give an accurate answer to that question will take a lot of time. Is it really worth that? When it comes to estimation, ask yourself why an estimate is needed. Estimates are often needed to be able to set priority and to be able to plan. This often means that we are tempted to believe that we need estimates in the form of “how many days” but ask yourself: are the needs met if the estimate is incorrect and how big is the chance that the estimate is wrong if someone answers “150 days”. If you got that answer to the question “How many days does it take to build a new house”, wouldn’t you be a little suspicious of that answer if it was given just like that? And still, that kind of answer is too often accepted and believed to be accurate within software development. And we are surprised when there are “delays” in the development.

So, is everything lost? Either we spend too much time on estimates or we get inaccurate estimates? Well, it's more about asking the right question to enable people to give an answer they believe in. Look at these two buildings:

Which one will take the longest to build?

If you say that the house to the right would take the longest to build, do you think it would take twice as much time or three times as long?

If you have no building experience, you can perhaps not give a better answer than that, but the point is that when making relative estimates, you can give a more accurate answer.

Another example, often used during seminars is size of animals. How much does a lion weigh? Difficult question to answer, wouldn't you say? But which animal weighs the most: the lion or the puma?

Relative estimates on large things are hard to make. Which takes the longest to build; the Buckingham Palace or the Palace in Vienna? But as with non relative estimates, this only means that the story is too big and you need to divide into smaller chunks to be able to make an estimate.

If you're with me on the statement that giving a relative estimate is easier and gives room for a more accurate answer to the question, we still have some issues with using relative estimates. And that is; do they meet our needs to be able to plan and prioritize?

Not alone, they do. But when you add the term of velocity, they do. Velocity is the speed of the development. I'll get back to that concept later on, and focus on how to make relative estimates.

How do you estimate a product backlog item? There are many methods, for example you could get an expert opinion from one or two developers. Chances are that the right developers can give a good estimate here and if your sole purpose is getting a rough sense if A is harder than B, this is an option.

The problem with the expert opinion is that the developer giving the estimate makes a lot of assumptions which are not shared by the other developers. No one knows what he believes is the task. Perhaps he asks you a few questions but only he hears the answers which are now stuck with that sole developer. When it then comes to the implementation, how big are the chances that the discussion from estimation is forgotten or not relayed to the development team?

Involving the whole team in estimation gives room for exchanging views on what the tasks are and how big they are. Just like story writing seminars they are excellent occasions to get updated on the current system and the domain. Even if there is a collective code ownership on your team, chances are that some team members have not worked on a particular part of the system for a while and do not know the current status.

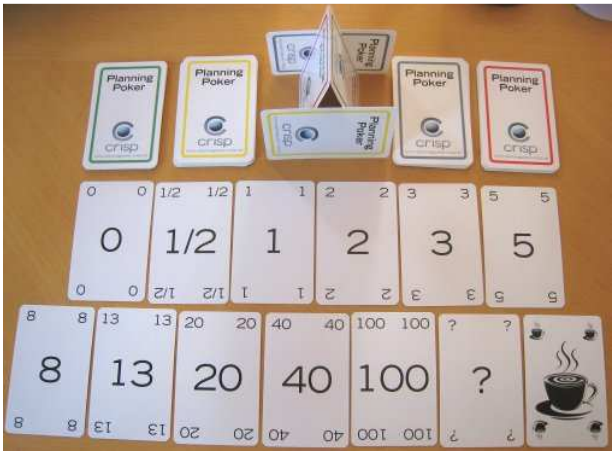
The problem with involving the whole team often boils down to costs and loss of focus. This is why it's important to choose an estimation technique which does not take too long and have a pulse in the estimation process. If you go back to the chapter on product planning you can see that product planning can include estimation sessions and that these can be planned for example once every other sprint. All depending on how much new things show up on the product backlog.

Planning poker

When it comes to finding a technique for estimating, a commonly used method in agile software development is estimation poker.

Estimation poker is a way to include a whole team in making fast and good estimates.

What you do is that you present all participants with an individual deck of cards. This is to make it clear that everyone is a part of the estimation process. Ordinary decks of cards can be used but many agile consultant firms have decks of cards, especially made for planning poker.



Planning poker deck of cards from Swedish consultant firm Crisp (www.crisp.se). They also have nice articles on Planning Poker on their web site.

Planning poker deck of cards from Mountain goat software. As one of a previous team members put it: the furry mountain goats became his poker buddies. Mountain Goat software also have an only planning poker site on the Internet for distributed teams.

Why use special decks of cards? Well, one way to make estimation more complicated is to have too many options. Is that 20 times or 21 times harder? The following numbers are therefore included in the deck:

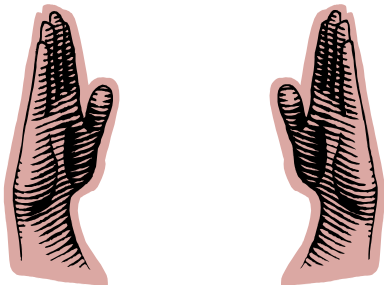
0,5	1	2	3	5
8	13	20	40	100

Some decks also has a card for “I don’t know” (?) or “I need a coffee break”.

What you do is that you start each estimation session with specifying an estimation golden list. You should have the same golden list on all sessions and it’s never a bad thing to have the same golden list on different projects.

An estimation golden list is a list with examples of that estimate. What you do is that you take a list of already accomplished product backlog items (if you’re starting a new project you can take items from a previous project. You could also use fake stories which all the participants understand, but this is the least effective solution.)

You pick an item and set this the “2 story point story”. This is a small concrete story. It really does not matter how big it is, just remember that you will need examples of much larger stories as you go along.



Then you try to find something that is about half that story’s size. That will be your 1. And you’ll just pick out items which match the other numbers.

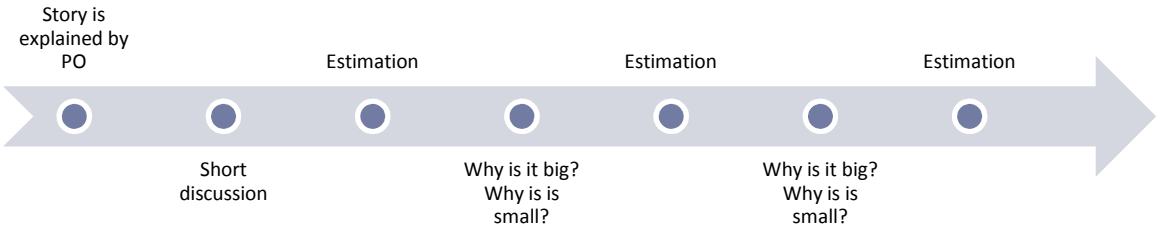
The creation of the golden list is something that one or two of the developers can do but before you start using the golden list, all the items should be read up aloud so everyone can accept the list. The list is then posted on the room used when estimating. It is an important tool while estimating.

With the golden list posted on the wall, the real estimation can start.

The product owner reads aloud the product backlog item. Participants can ask a few questions to clarify what they are estimating. The participants then look at the golden list and try to find an item which matches the new story. If a story is roughly in the middle of two estimates, the higher number is chosen.

All participants pick the card matching their estimate from their own deck of cards. Their choice should be kept hidden from the others until all have made their choice. This is to make everyone choosing on their own and not just taking someone else's estimate.

All participants show their cards. If all have the same number, you have an estimate. If not, the person with the lowest number gets to debate the person with the highest number to explain why it's a small or a large task. After a short discussion, a new estimation round is played. And so it continues until all can agree on one number. Most of the times it takes less than four rounds before the team can commit to a number.



Typical estimation using planning poker. The product owner gives a short description. After three rounds of poker, the team can commit to a number.

Sometimes the team cannot commit to a number, perhaps the product backlog item needs to be rewritten or divided into smaller stories.

On an average, a scrum team can estimate 9-20 stories in an hour. Most of the time, if the product backlog items are the result of a story writing session where the developers were included, more stories can be estimated. An estimation session should be kept rather short and focused: between one and two hours is often appropriate.

You can also use something called Ideal days instead of story points while estimating. An ideal day is a full focus day. The problem with this approach is that people not involved in details misunderstand the concept and that it makes it hard to keep the estimation session focusing on relative estimates.

Calculating business value

During the product owner course I took, a lot of time was spent on this question. Also, the Agile Estimating and Planning book by Mike Cohn discusses this in detail. So, I recommend you getting yourself educated on the best methods for you and your organization.

Here, I'll just focus on describing what business value is, why you need to calculate it and an easy method you can use just to get started.

Business value is how much a feature is worth from a business perspective. The estimation described in the former chapter is used to calculate relative cost but here we're talking about how much its worth.

To calculate business value, you need to know the business and here you need to bring forth the product vision. If you don't know the objective of the system, how could you possibly estimate how much something is worth? How big is the business value of a SMS ticket service? Well, if we're building a system for a travel agency this might have a big business value but if you're building a warehouse solution for a hospital?

When you have the product vision you can list different types of stakeholders when it comes to business value. Perhaps you have different industries which your organization is targeting. Perhaps you have both old and new customers. What is important with this documentation is realizing that something might have a high business value for one group and none for another. Sales people tend not going into details when it comes to using functions. For them a fast demo flow has a high business value, while bugs and usability issues which are not spotted on a sales demo have high business value for a current user and the person training users how to use a system.

Don't forget IT departments, helpdesk and operations. How much value lies in removing a problem which causes ten support calls every day? How much value lies in changing the installation so that most users can install without a guy from the IT department?

Technical features and refactoring also have business value? If a function is poorly implemented so that it increases the development time, things that could otherwise be built cannot be done. Code which is not automatically tested causes increased test time, which can increase overall manual testing time or that other areas of the product is not tested.

When the business value of a feature is to be calculated, these different aspects need to be addressed and the same aspects must be addressed when calculating all features. If you leave out some aspects when calculating business values for some features and not for others, you will compare apples with pears and the business value will not be valid and useful.

As I started with, there a multiple methods for calculating business value but one easy way to begin is to have a representative for each identified aspect and then ask about the business value of having the feature and the lost business value if the feature is not implemented.

Business value is calculated numerically. If you have a method for calculating this in money, this can be done. Another method is using the numeric series from planning poker.

Aspect	Business value if implemented	Lost business value if not implemented
Sales	100	20
Operations & helpdesk	0	100
Administrative users	40	100
Normal users	13	0
Novice users	0	0
Development	0	0

Example of calculation of business value for a feature with a number of aspects. In this case, the numeric series from planning poker has been used.

Remember that when planning for a release, a product owner can choose to focus on a specific aspect. For example if we in a release want to focus on increased sales, we can add a factor to the business value derived from increased sales:

Aspect	Business value if implemented	Lost business value if not implemented
Sales	100*1,5=250	20*1,5=30
Operations & helpdesk	0	100
Administrative users	40	100
Normal users	13	0
Novice users	0	0
Development	0	0

Adding a factor for a release where the focus lies on increased sales. Another release, another aspect can be the focal point.

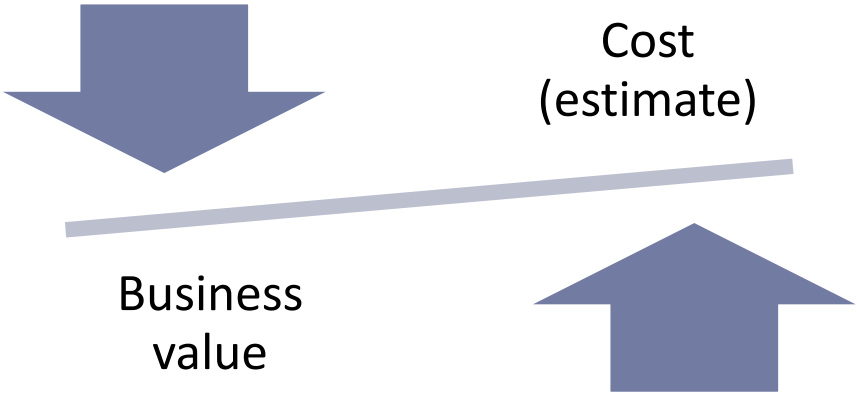
Remember that different themes within a product can also be seen as aspects so you can think in the terms of “this feature is important to support billing but not to issue handling”.

The business value can be set by having product planning meetings or just a form sent to different stakeholders. Again, it is important that you find a proper solution for you and your organization.

Prioritizing

Priority is the only attribute on the product backlog item which the product owner has sole ownership over. He and no one else set the priority on an item.

So, how do you set priority? The easy answer is that you look at costs and business value and then you find the items which give the most value to the least cost.



Aspects which affect the priority of a product backlog item

This does not take in other things like risk and that some things become less expensive if done together. Some things have to be done due to for example changes in legislation or that the software licenses become invalid.

Priorities also tend to change all the time. New needs surface, others become obsolete. Business values and costs change with every sale and every sprint.

To be able to set the priority, the product owner need to have a good feeling for the changing reality of the organization.

It's also about being smart: if there is a large risky product backlog item which might not give any visual effects on a demo, you can increase the priority of a low cost item which will be very visible during a demo.

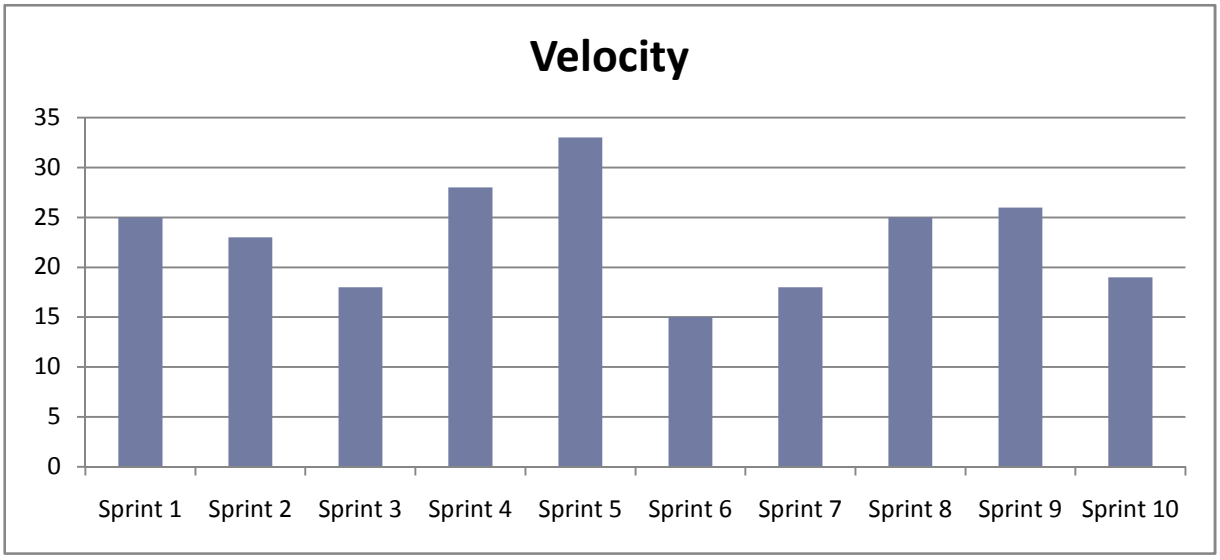
It is also important to remember to target different groups of stakeholders on a regular basis so that everyone can feel that their needs are also met.

Also, remember that a change in priority affect others. Late changes must therefore be communicated and it's important that people can get hold of the current product backlog so they can keep themselves updated.

Calculating and using velocity

Velocity is a key issue in all agile software development since it specifies the speed. Before you have calculated velocity, estimations of story points cannot be used for planning. The reason for this is that if a product backlog item has the estimate of 8 and you don't know how many story points can be accomplished during a sprint, how can you possibly know how long it will take to complete the task?

Velocity is best calculated from historic values. That is, you look at the previous sprints and see how much has the team accomplished in the past. Here is an example:



Here we can see that the number of story points accomplished during the sprints have varied over time. So which velocity does the team have?

You take the lowest three values, in this case 15, 18 and 18 and calculate the medium of these three values. You get in this case 17. This is your lowest or worst velocity.

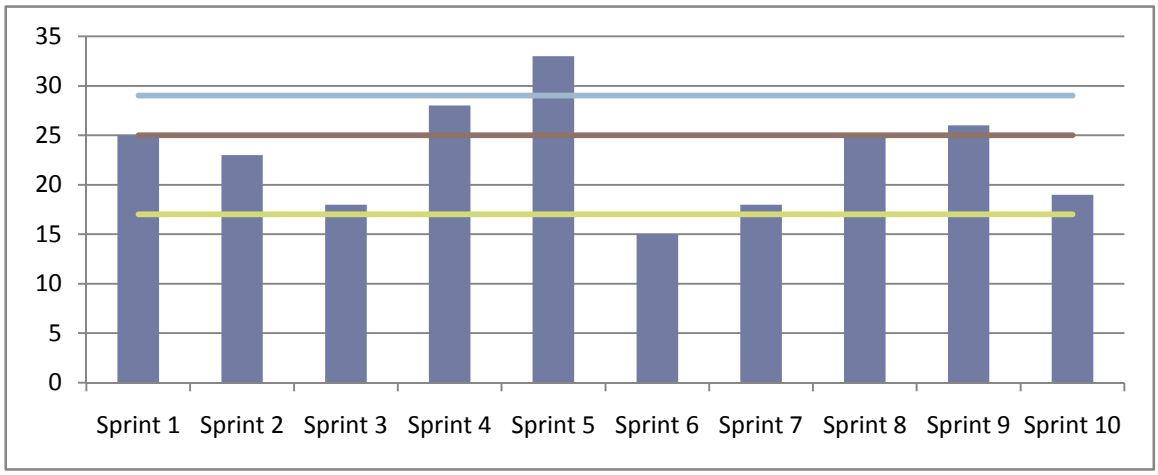
You then take your middle three values. In this case 23, 25 and 26 and calculate the medium of these three values. In this case you get 25. This is your average velocity.

And finally you do the same with your three highest values. In this case the average of 33, 28 and 26 becomes 29. Now you have your top velocity.

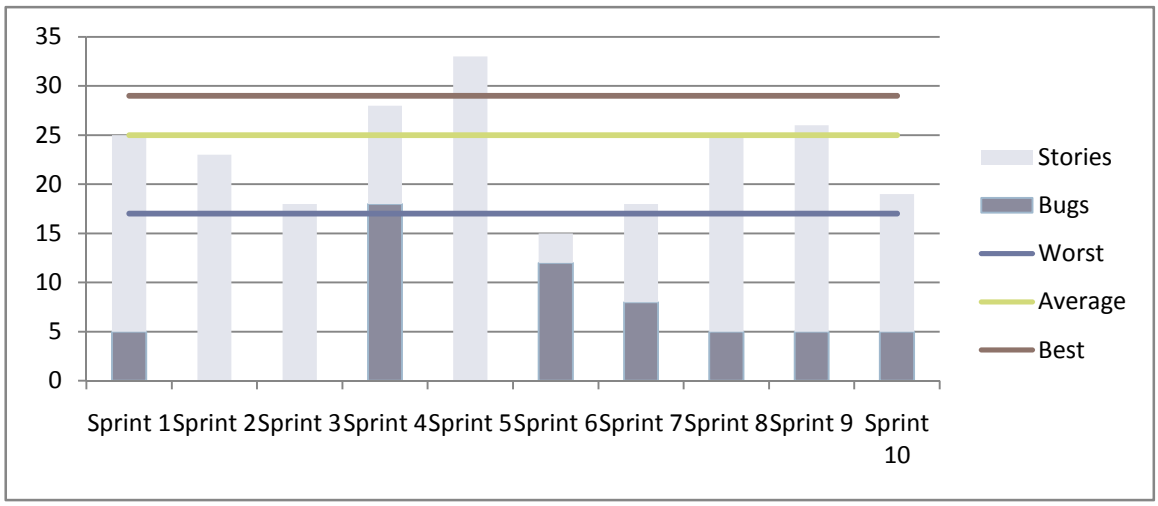
The way you use this is that you now know how much the team should be able to do and what they can do if all goes well:

- You can count on the team coping with 17 story points per sprint.
- There is no use planning for more than 29 story points.

You can also use the values to evaluate the different sprints. Look at sprint 5. At first glance it might look like this was the best sprint ever. But what happened after? The velocity fell dramatically. Where there quality problems or did the team lose a team member?



Don't take for granted that the best sprint is the sprint with the highest velocity: look at the sprints before and after. And it can also be a good thing to divide story points for stories and story points for bugs. And now we can see the effects a little bit better. Sprint 5 was a sprint where no time was spent on fixing bugs but this work was then spent on the coming sprints.



You can also add data with available man hours so changes in available resources can be seen but remember it is often better to start with a simple chart and show the details if needed. And don't try to show all the details at the same time.

So, how do you do from the beginning, before you have historic values? Well, now you can use your old golden list from the estimation exercises. Try to make an educated guess how many of those stories could be done with the current setup and state of the system. Involve the developers and ask for their opinion. And then as the project moves along, you can start using historic values instead.

Don't make velocity into a contest. Mike Cohn worked with two groups who were constantly compared. Why was the velocity so much lower in one of the teams? The effect of this was that the estimations were doubled. And so the velocity rose. Developers are not stupid and math is often not a problem to these guys.

But how do you use velocity while planning?

Let's go back to our example and now we add a nice product backlog:

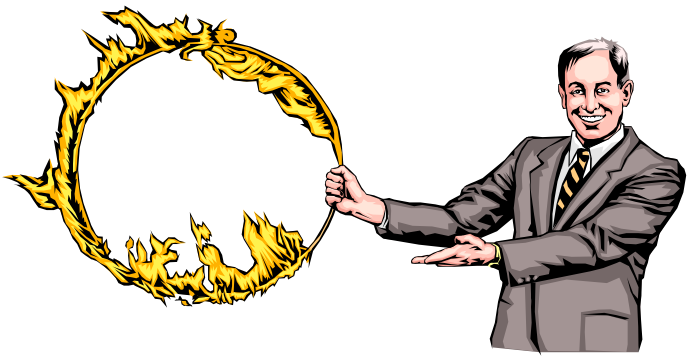
Story	Story points	Priority	Needed velocity
Story 1	5	1	5
Story 2	8	2	13
Story 3	5	3	18
Story 4	2	4	20
Story 5	5	5	25
Story 6	20	6	45
Story 7	5	7	50
Story 8	8	8	58
Story 9	1	9	59
Story 10	3	10	62

As you can see, if we were nearing a sprint planning meeting we know that can count on stories 1-3 being done. Stories 4 and 5 are a little bit unsure but since story 6 is a huge thing, we have no chance in the world to get it done during a sprint.

So, what are our options? One solution is going into the sprint and if all goes very well and we finish stories 1-5 and have spare time, we can skip story 6 and go for a story lower down on the list. You don't have to take the stories straight from the list. If for example we just have a little time to spare, story 9 might be a good idea.

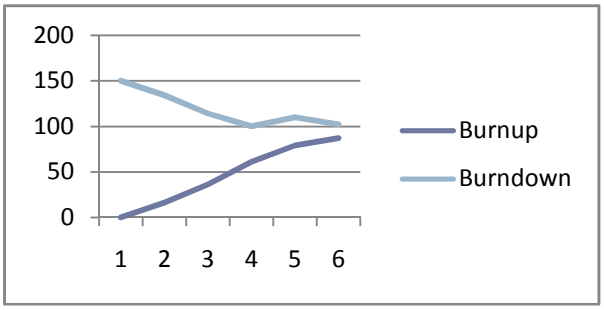
Another option is having a story writing seminar around story 6 and dividing it into smaller stories. Then we can start with story 6 if all goes well. Story 6 should anyway be divided (if this is possible) before coming to a sprint planning meeting: such a large story is a high risk project.

We now return to the size of stories. As I stated before, a sprint should cover about three product backlog items. And the reason for this is this example. It will be much easier to pick stories to the sprint backlog and taking on stories does not feel so risky. There is no chance that a sane scrum team commits to stories 1-6 during a sprint if things have not changed dramatically in some way. But if story 6 was divided into a number of three point's stories, they might take on one or two if they feel lucky. It is no use pushing developers into committing to a plan they don't believe in.



Burndown charts

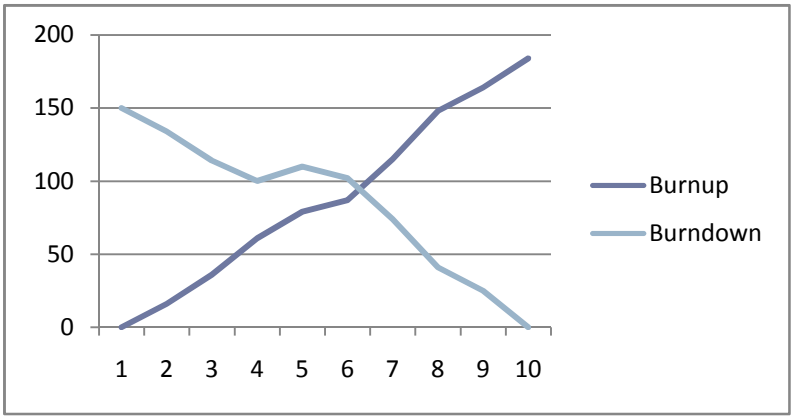
People familiar with scrum vocabulary have probably heard of sprint burndown chart. This is used to visualize the progress of the sprint. What you do is that you track how much of the planned work remains on a day to day basis. You can also have a product Burnup chart which tracks the work set as done. Combining these in own diagram is quite confusing but here is a combined diagram just for you to understand the differences.



Sprint burndown chart and Sprint burnup chart. The Y-axis is the number of manhours remaining/spent while the X-axis is the day of the sprint.

The sprint burndown chart is tracked by the scrum master and the product owner does not get involved in the details. Following the burndown can though be interesting if cutting or adding features during a sprint is being discussed.

The product owner can create product burndown charts and project burndown charts to visualize progress against an objective. The chart should show the number of story points accomplished and remaining until a project objective is met.



Project burndown and burnup chart. Y-axis is number of story points and X-axis is sprint. The Burnup curve shows the accomplished story points and the Burndown shows the story points remaining until objective met. Observe that the stories needed to meet objective was increased during two sprints in the middle.

Public and internal releases

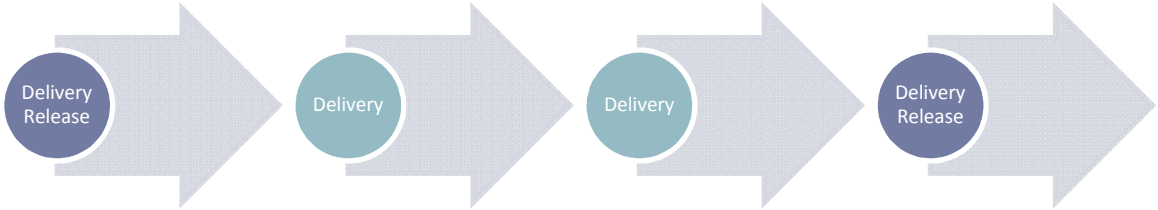
Can you successfully deploy your product every other week if you have 1000 customers with integrations and a mobile work force?

One of the criticisms I hear from people not buying into agile and scrum is that they don't believe in them being able to deploy to customer as often as every two weeks.

This might very well be true, but release does not mean deployed to customer. It means that the quality is so high that you COULD deploy to customer.

One of the developers I've had the honor to have worked with came up with the idea of separating the terms and made me starting to use the terms Delivery and Release. A scrum team makes a delivery of working software at the end of each sprint.

If the delivery is to be deployed on a customer, the delivery becomes a release.



A project where the delivery of every other sprint is deployed at customer sites

As a product owner you can also use this during sprint planning meetings. Let's go back to our example from the previous sprint. Let's say they release every other sprint. So, in the coming release, at least 2*17 story points can be counted on and you know that 2*29 is not possible.

Story	Story points	Priority	Needed velocity
Story 1	5	1	5
Story 2	8	2	13
Story 3	5	3	18
Story 4	2	4	20
Story 5	5	5	25
Story 6a	5	6	30
Story 6b	5	7	35
Story 6c	8	8	43
Story 6d	3	9	46
Story 6e	5	10	51
Story 7	5	11	56
Story 8	8	12	64
Story 9	1	13	65
Story 10	3	14	68

This means that the team can choose from stories 1-6a and if they will do stories 2,3,4 and 5 first and for some reason wait with story 1 to the second sprint, this is a risk they should be able to take.

The most important thing for the product owner is looking at stories 6a-6e: the lower down in the list, the smaller the chance of it being done until the next release. And isn't possible that 6a is enough for this release and that 6b-6e can get a lower priority?

Another thing that might strike you is when I divided the story 6; the sum did not add up to 20. This is more often than not the probable scenario. When splitting things, new questions surface and new ideas are presented. Sometimes splitting a story means that the total sum becomes bigger and sometimes it becomes smaller.

Another important task for product owners is the communication of how this method for planning works. Making clear that stories 8-10 will not make it in this release if something else is not removed is also very important.

If you have a public product backlog which stakeholders can view and follow themselves, you can make your life (and probably theirs as well) much easier since they can get hold on the current plan.

Still, understanding story points, velocity and all the scrum terms is not easy and counting on people understanding this on their own or by just reading a How-To on a intranet does not work in most cases. If you have a corporate event, a verbal presentation or even better, some kind of training including an example exercise is often needed.

Summery

Release planning is about planning for the actual implementation for a new version or release. To be able to make such predictions you need to estimate the product backlog and calculate your velocity. You also need to calculate the business value and from these values calculate the priority of all the items on the product backlog.

Planning poker is an excellent and fun way way to estimate the product backlog. Burndown charts are used to communicate progress.

Try	Avoid
Decide the need for calculating velocity. How this value is used should decide on how much effort you put in the calculation.	Visualizing velocity and business value like something absolute and correct.
Planning poker for estimations of both size and business value	Using absolute days when estimating.
Calculate best, worst and probable velocity. This is important to make clear that the situation can affect the outcome. It becomes easier for everyone to make their own plans.	Skipping the calculation of velocity. This brings uncertainty and stakeholders making their own guesses.
If you want to have one pulse for planning and one for development, introduce delivery sprints	

Sprint planning meetings

The product owner participates in the first part of the sprint planning meeting by presenting the top stories of the product backlog. I print out the stories on paper and present them one at the time. I print the estimate on the paper but I don't mention this during my presentation. The reason for this is that I don't want the estimations done during sprint planning meetings should be affected by the estimation of the original story.

After this, the product owner becomes a source and an inspiration. The scrum master leads the meeting from this point by coaching the team into dividing the stories into tasks, which are estimated. Estimations during the sprint planning meetings should be done using man hours since the stories should be divided into greater detail. A task should not be bigger than 4-6 hours. But as a product owner, this is not your choice. The scrum master and the team is responsible for the sprint from the moment you presented the product backlog.

The product owner answers questions and gives other input. Sometimes a product owner can remove a story from the board. This has most often happened when the original estimate was small but when the participants realize that it's more difficult than anticipated. After the team has committed to the stories, a product owner can help defining a sprint objective. This is not mandatory but can be a good inspiration during the sprint. After the sprint planning meeting, the scrum master or the product owner (or both) communicate things like time for demo, sprint content, etc, to stakeholders and others.

The hardest thing for a product owner is probably letting go of the responsibility. But this does not mean not participating. A product owner should take the time to visit the complete sprint planning meeting. Perhaps he does not listen in on all the technical debates which might take place and work with something else in the mean time, but being in the room is more important and valuable than you could imagine. Another possibility as a product owner is that you can take on tasks which are presented during the sprint planning meeting. Being a part of the team is often appriciated and makes you a better product owner. But remember that when you take on tasks as a team member you are playing a different role, the one as team member. When carrying out these tasks you are not acting product owner.

Summery

A product owner participates during the sprint planning meeting to bring inspiration and to gain knowledge.

Try	Avoid
Use storyboards for visualizing the stories	Trying to make the team commit to more or question their estimates

During the sprint

Acting product owner

One of the problems you face as a product owner is that you need to be present in the current sprint while you at the same time are moving into the future. You need to plan the upcoming sprints, work with product planning, etc. This means that the better part of your work will probably be outside the scope of the current sprint.

But this does not mean that you should leave the current sprint. Being available is very important since decisions are made all the time by all team members. Perhaps you need to make instant decisions; perhaps you need to point to someone who can.

Mike Cohn said that to be an effective product owner, you need to sit with the team at least 2-3 days a week. You probably have other tasks but if you're printing out a report, why not do that in the team room? If there are problems with you being disturbed, set up rules for that. You are probably not the only one with that problem on the team. Being a product owner during a sprint can also mean that you need to make some decisions. If something is to be cut from the sprint, this is a decision made by the team and the product owner. In most cases the team is represented during these discussions by the scrum master.

Sometimes things are added to the sprint after the sprint planning meeting. This is of course nothing to strive for, but sometimes it's inevitable. These are the most common scenarios:

- The team missed a task during the sprint planning meeting.
  The scrum master should inform the product owner on which effect this will have on the sprint. If the effect is minor, this might just be an e-mail describing the change. If the added task poses new risks to the sprint, a discussion might be a good thing. Perhaps the complete story should be cut or changed to minimize the risk.

- A bug is found.
  If a critical bug is found during a sprint the team and the product owner can decide to add the bug to the sprint backlog. As with added tasks, if the bug is minor and the scrum team feels the risk is minor, an e-mail can be used but since all changed and added features affects testing, adding bugs to the sprint backlog should also be discussed if they are risky or costly from a testing or developing perspective.

- The sprint objectives are met in advance and there is room for more to do.
  The decision what to add to the sprint is made by the product owner. This is a wonderful opportunity to give developers little more free choices and do things they've wanted to do before. If the added things add to the risk of the sprint, it can be a good idea not to include the added features to the delivery but keep it in a separate branch.

Acting team member

One way to get involved is being an active team member. One possible role as a team member is a tester of some kind. Becoming a good tester is not a simple task; get educated!

But don't take on tasks which you don't have time for. If you're missing out on a task result in the failure of a sprint, this will not be good for team morale.

Summery

To be an effective product owner, you need take part of the team's every day routines. Not every day, but often. A good product owner also gets involved as a team member. Product owners are often good acceptance testers.

Try	Avoid
Volunteer on tasks which you can carry out.	Taking on tasks which you don't have time to complete.
Prioritize being on daily scrum.	Making team members feel like you're guarding them when you're in the room.
Plan time to just hang around in the team room. Sit with one or two of the developers, giving input on how you view the requirements.	Selling out your team or back talking them. You are also part of the team.
Bring a stakeholder to the team room and make them see the real work. Let them have input on work in progress and make the developer's see that there are actual people behind the requirements	Just being there on sprint planning meeting and sprint review. It's during the time between you can be really useful.

The sprint review and delivery

Finally, it's time for the show!

The demo and the delivery should be a time of celebration if the team has done well. This does not mean that critique should be hidden. I once saw a product owner who prepped the stakeholders before a sprint demo, asking them not to say something negative. He knew that the end result wasn't what the stakeholders wanted but this was really a lousy idea. Especially since the error didn't lie in the coding but in the handling of requirements. And this was the product owner's responsibility. He failed both his team and his stakeholders.

The Sprint review

The team demonstrates the result of the sprint to the product owner. The team must decide on their own how they want to hold their demo (as long as they present working software and not PowerPoint presentations).

Sometimes stakeholders are not fond of developer held demonstrations and want something user friendlier. Well, there is nothing that hinders you from making other presentations in the forms that applies to specific user groups.

It is always important to listen in to the stakeholders which materials they need after a delivery. Also, remember that this will often be dependent on if the release is public or internal.

Summery

The product owner should use the sprint review as a time to gather input and to celebrate. Hiding critique is never a good solution.

Try	Avoid
If a participant has lots of input or critique, book a meeting with him and one of the developers after the review so this can be discussed in full without disturbing the flow of the review.	Missing the sprint review.
Making the review feel like something that is to be celebrated. Homemade cookies are often welcome.	Telling stakeholder's to hold back on critique.

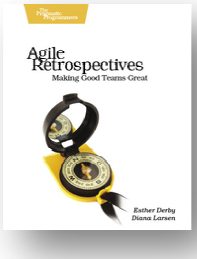
Retrospectives

If you do not constantly strive at getting better, you will become worse. And what better way is there to get better than looking back at your experience and try to improve?

After a sprint, it is recommended that a sprint retrospective is being held. This is the responsibility of the scrum master. Should the product owner participate? I think that this is up to the team so a good idea is to let the scrum master address this question to the team members and let them decide this.

As a product owner, you should also hold formal retrospectives. I try to have a very short retrospective at the end of each formal meeting. It can be just asking everyone what they thought and if something should change next time. I can also ask people to put ideas in a box before leaving After major releases and deliveries, a longer retrospective should be held.

My main source for ideas for retrospectives are found in Agile Retrospectives – making good teams great by Derby and Larsen.



Mark Levison also has some great ideas on his blog:

http://www.notesfromatooluser.com/2008/10/agile-games-for-making-retrospectives-interesting.html

The principles I try to keep during a longer retrospective meeting (exercises are described in Derby's and Larson's book) are:

1. Initial emotions – just get a word from each participant to catch their feelings for the meeting
2. Timeline – let participants describe events which affected them during the time frame which the retrospective covers
3. Prioritize using scores – let the participants choose individually what they want to focus on
4. Solutions – The by the group highest prioritized suggestions should be addressed, but how and by whom?
5. Appreciations – I love this exercise which enables participants to give credits to other participants. It's a wonderful way to end a meeting, especially if it's been turbulent.

Summery

Retrospectives are good ways to improve the process. They should be active and the meetings should be well planned and have a useful output.

But what if we get stuck?

Going agile is hard. Implementing scrum is hard. Scrum won't solve all your problems. Most of the problems you had before turning to scrum will probably remain, if you don't handle the problems.

Retrospectives and acting on the problems discussed during retrospectives is important, independent on which method you use. But this requires that problems are open and clear; what use is a retrospective if the real problem is never discussed?

Getting some training is always a good idea. Often you send the already interested and informed on scrum classes but you should focus on the people who are not interested or does not see the benefit. By hearing the details from an outsider, the process can sometimes be better accepted.

Get involved in agile communities and read blogs. Sometimes it can feel like you're all alone with your problems. You're probably not and participating in communities, on conferences and by reading blogs you can hear about others solving or handling the same problems. Some problems you will have to live with.

You can also bring in some expert consultants, from other internal teams or from a consultant firm. Outsiders can often spot those hidden problem or does not have the same problems discussing internal issues. Having experience implementing scrum really give you an edge and by contracting an agile consultant you can miss out on some of the most common problems.

ANNAFORSS.SPACES.LIVE.COM/

Working as a product owner or inspiring to be?

Thinking about using SCRUM but don't know what that means to business people?

Are you a business person and your development department uses SCRUM and you don't understand how you fit in or how it works?

Confessions of a serial product owner is a short description of how SCRUM can work, as described from a business person's point of view.

2009

ANNA.FORSS@GMAIL.COM