A Guide to Iteration Planning meetings

By Allan Kelly, allan@allankelly.net, http://www.allankelly.net

> This guide is intended to accompany the Iteration Planning Sheet.
>
> Visit http://www.softwarestrategy.co.uk/dlgsheets/planningsheet.html to download a sheet or to buy a printed sheet.

One of the things I enjoy about my work as an Agile Consultant is visiting teams and observing iteration planning meeting. If you've never observed another team's planning meeting, or if you've never seen one, you might expect that they are fairly similar. Indeed they are but, as is so often the case, the devil is in the detail. I am routinely surprised by the ability of teams to interpret, find and invent different ways to doing things in the meeting.

Take for example deciding how much work a team can undertake. Some teams just have the product owner propose stories and they accept stories until they feel they have "enough." Some teams are strict in only accepting stories they are sure they can get done - the Scrum idea of "commitment"; other teams will take on more work than they expect to do.

Some teams will use velocity to judge how much they can do. They determine how many points they can do and accept stories up to that limit - give or take a bit. Some teams set the upper limit by simply looking at how many points they did last time and rolling that forward. Some teams play planning poker to decide how many points they can do. Some teams think about who's on holiday, who's not, who was ill, what got in the way and a million other things.

Personally I advise teams to use a rolling average of the last 4 or 5 sprints, ignore holidays, illness and other factors that might have or will disrupt work. Then schedule slightly more work than they expect to do. Estimates velocity are only a rough mechanism for gauging about how much work will be done. The team should accept enough work so they don't run out of things to do but they should not accept more than is reasonable to expect. Preparing work which will not get done is wasteful.

Seeing these variations is often educational for me, and vice-versa, I sometimes suggest to changes to team's current practice. However it does mean that when describing a planning meeting to someone there are a lot of details which could be different without necessarily being wrong. Indeed they could be better than my suggestions.

What follow is my take on a planning meeting and how it runs. This description matches the A1 Planning Sheet I have devised for to help new teams run planning meetings. I like to call this type of Agile "Xanpan"[1] and I hope it is close enough to both XP and Scrum to make this description and the sheets useful for many teams.

After this description I detail some of the activities which might occur outside of the planning meeting but which contribute to a smooth running planning meeting.

I will also detail some variations I've seen. . . .

The Players

- **The Product Owner**: this role is usually played by a Product Manager, or a Business Analyst acting as a proxy for the real customer. On occasions the real customer might play the role. Some companies employ Subject Matter Experts who can play Product Owner at times. Sometimes someone else needs to play the role. Whoever plays the Product Owner they need to do their homework (see below) and be ready to propose stories, answer questions, prioritize and make decisions as the meeting progresses.

  While it is common practice for there to be only one Product Owner there may be more than one. If more than one Product

[1] Xanpan is pronounced "Zan-pan".

Owner (for the same product) will attend the planning meeting the should agree before hand priorities and approaches.

- **The Creators**: Software Engineers and Testers mainly, although sometimes there are others such as User Interface Designers involved. These are the people who will build the things the Product Owner asks for.
- **The Facilitator**: Sometimes there is a dedicated facilitator who is not the Product Owner or a member of the building team. They may be, for example, a Project Manager, Scrum Master or Agile Coach.

  Some teams are too small to have a dedicated facilitator and a developer steps into the role - in which case they wear two hats during the meeting. Experienced teams may not need a facilitator however inexperienced teams who lack a facilitator may find the meetings long and difficult. Whoever plays facilitator should have some experience of planning meetings and facilitation, they should also have respect and authority from the team to play the role.

  Usually it is not a good idea for the Product Owner to double up as Facilitator because someone needs to watch for and resolve disputes between the Product Owner and Developers. Plus, the Product Owner usually has more than enough to do during the planning meeting already.

I consider total of all the people in these roles, and possibly some more as well, to be: The Development Team. The Product Owner and Facilitator are, in my book, as much part of the team as the Testers and Coders.

The Artefacts

There are several artefacts, or props, which are normally used in a planning meeting. When teams rehearse planning meetings in by training courses I use dice to simulate the work in an iteration. In real planning meetings, and real iterations, dice are not used.

- **Blue Cards**: Describe bits of functionality which are useful to the business in a language the business representatives understand. These are vertical slices of business functionality which are conceivably deliverable on their own. Often User Story format (Cohn

2004) is used and the cards are usually created before the planning meeting.

- **White Cards**: Each card describe one task needed in building the thing described on the Blue card. White cards are normally written during the planning meeting and there are usually multiple White task cards for each Blue card.

- **Red Cards**: Bugs and other expedited items. Normally Reds are not broken down into tasks but they are occasionally. What colour you use for tasks related to a Red is up to you, White or Red would both be appropriate.

- **Planning board**: Usually a 1.2m by 90cm magnetic white board divided into columns. Other sizes and types of boards may be used. While many teams use electronic tracking systems too I strongly advise teams to initially adopt physical boards and cards and only progress to electronic systems when they have some physical experience. Even then both systems may be run in parallel.

- **Planning poker cards**: the description which follows assume the team are playing planning poker. Not all teams play planning poker, it isn't compulsory so feel free to use whatever method works for you. That said, whatever method you use should address some of the point discussed below.

  There are special sets of planning poker cards available - these can be surprising difficult to buy but are often freely available from Agile training and consulting companies - just ask! Different planning poker sets have slightly different sequences.

The exact colour of the cards can vary from team to team. Keeping to the colours outlined here does keep things consistent.

Some teams use additional colours to signal other types of work. One team started using **Yellow Cards** to signal unplanned work, this Xanpan convention has been described elsewhere. By their nature Yellow cards will only appear planning meeting when they are carry over work.

The Meeting Sequence

The basic format of the meeting is shown in the diagram below. The team agree how much work they will attempt, the Product Owner presents the work they would like done and the team work through each item in priority order. They discuss each item, break it down to tasks and estimate the tasks.

After each item is done (i.e. discussion and breakdown has ended) the team count up how much work they have accepted and compare this with what they expect to do. If they have spare capacity the next highest priority item is pulled from the Product Owner.

When the team have accepted work to their capacity they review what they have with the Product Owner and agree any changes.

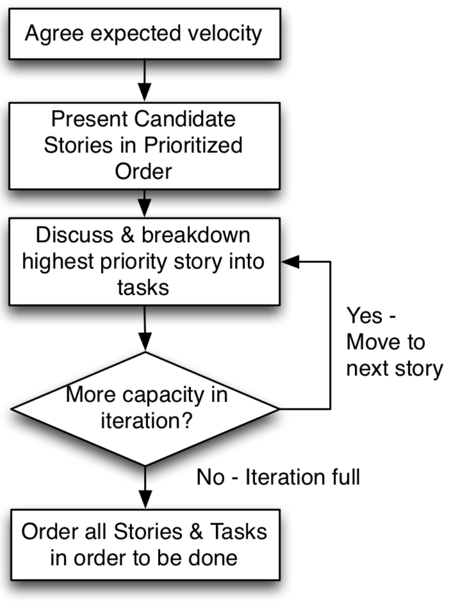
Agree expected velocity
Present Candidate Stories in Prioritized Order
Discuss & breakdown highest priority story into tasks
More capacity in iteration?
Yes - Move to next story
No - Iteration full
Order all Stories & Tasks in order to be done

Figure 1: Basic meeting sequence

First meeting

The first planning meeting a team hold is always the hardest. This is when their experience is least and the unknowns greatest, naturally it will take longer to navigate the meeting. In the worst cases the teams lack of experience can derail the meeting altogether should they encounter difficulties.

The first meeting is also the most difficult for another reason: the team have no reference points. They have little idea of what they need from a story, how long it will take to do a story, or quite what acceptance criteria should look like. Even if they have practiced for these question during training doing it for real life will be harder.

Significantly the team will also have no data on how fast they can go - they will only have a vague idea of how big "one point" is or how many points they should accept into an iteration. In my experience teams tend to accept far more work into the first sprint than they will ever get done, they are inherently optimistic.

Second and subsequent

Because the first meeting takes so long and over plans the work the second meeting, two week later, tends to be one of the shortest. So much work is carried over in one form or another the second meeting finds little to plan. After than the meetings start to settle down. The team have two meetings under their belt and two data points.

The format of the meeting also changes after the first meeting. At first the meeting is entirely forward looking, subsequent meetings have a backward looking element. Prior to the start of the meeting, or right at the start the team will demonstrate the work done in the previous iteration. They will then review the work done - usually count the points and update any charts.

Formally the demonstration, review and retrospective might be separate meetings. But they are likely to be arranged back-to-back, perhaps with a short break between each. So whether one regards them as one long meeting or several shorter meetings is debatable.

The demonstration used to be an essential part of the end-of-iteration/start-of-next when teams only delivered at the end of iteration - or after several iteration. As more teams move to continuous delivery it is worth questioning whether the demo adds anything if people can already use the software for real.

Teams may also hold a retrospective as part of the iteration end routine. Although not all teams hold retrospectives and even those who do may not hold them every iteration.

The schedule of the second and subsequent meetings is something like the diagram below.

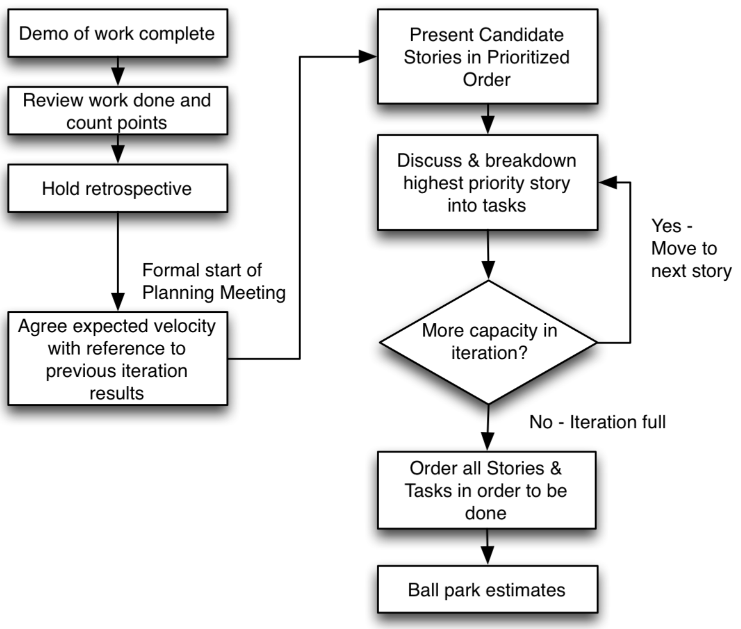
Demo of work complete
Review work done and count points
Hold retrospective
Formal start of Planning Meeting
Agree expected velocity with reference to previous iteration results
Present Candidate Stories in Prioritized Order
Discuss & breakdown highest priority story into tasks
Yes - Move to next story
More capacity in iteration?
No - Iteration full
Order all Stories & Tasks in order to be done
Ball park estimates

Figure 2: Subsequent meeting sequence

In the second meeting the team have a rough idea of how many points of work they can accept because they can sum up the points completed in the previous iteration. In the third meeting they have a better idea because they can average points from two iterations. By the fourth meeting the average is fairly accurate plus they have reasonable high water (best case) and low water (worse case) marks to guide their capacity thinking.

These meetings should happen regularly at the end/start of every iteration - typically every two weeks. They can be schedule in everyone's calendar and room bookings made months in advance. Indeed if you are using an electronic scheduling system (e.g. Microsoft Outlook) you can set up a reoccurring meeting with no end date.

The Planning Game

Product Owner(s) presents to the team the Blue (business facing) stories they wish to have developed. These are presented in absolute priority order – 1, 2, 3, 4, etc. No duplicate priorities are allowed, i.e. there can be only one priority one, one priority two and so on.

If two items are deemed to be of equal priority (e.g. two cards are both assigned priority three) by the Product Owner then the development team are allowed to decide the ordering. If the Product Owner disagrees with the ordering then they have, by their disagreement, determined the ordering. In general it is considered an abdication of responsibility if a Product Owner does not guide the team in prioritisation.

Each Blue card is broken down by the development team to a set of White technical tasks. Blues are the domain of the Product Owner, Whites of the technical team. The break down is partly an act of design, partly an act of requirements elaboration and partly an act designed to produce the smallest practical work items. (I will discuss the breakdown in more detail later.)

Of course sometimes only one White is needed to create one Blue, in these cases the White is omitted and the team can work directly at the Blue level. In some ways this represents the idea scenario, however, for this to work the Blue must be no bigger than one White, i.e. if the Blue can be broken down to multiple Whites then it should be so.

The breakdown is a co-operative process between development team and Product Owner – both should be present. There should be conversation between both sides: developers should ask about the requirements in detail, Product Owners may promote white cards to blue cards if they think the task itself has business value, product owners can remove technical tasks if they want – even against the advice of developers although in general the two sides should strive for consensus.

When White task cards have been broken out those who will be responsible for undertaking the work - i.e. development team, all developers and testers but not the Product Owner – estimate the work in terms of Abstract Points using Planning Poker. (See discussion below).

Teams track velocity on a rolling-average over the few iterations. Unlike in financial services, past performance is considered a good indicator of future performance, or at least of the next iteration. (This is the concept XP called "Yesterday's Weather.")

Once the White task cards are estimated and enough points of work are accepted into the iteration up to the slightly more than the average velocity, i.e. the more work is schedule to be attempted than is excepted to be achieved.

This does not mean a team would stop breaking a Blue down part way through. Once breakdown has started it makes sense to see it through to the end; although the Product Owner could pull the Blue out (and perhaps substitute another) when it became clear there was a lot of work. There is no hard and fast rule but it makes little sense to plan out the first tasks for a Blue but the order in which tasks are done does not necessarily correspond to the order in which they were identified and written on Whites.

The team may, or may not, achieve all the scheduled work, they may perform below of above velocity in any given iteration. If the team do more then expected than the work is available and over time the average will rise. Conversely, if the team find the iteration harder than expected then less will be achieved and the average will fall.

If a particular task or feature must be achieved within the iteration it should be scheduled first and within the minimum recent velocity, low-water mark. This does not guarantee the work will be done but provides a very high probability. Teams are advised to track the time it takes for cards to traverse the tracking board and develop statistically reliable averages and deviations to replace the planning poker estimation process.

Testing

Different teams handle testing in different ways. Some teams have professional Testers and formal test processes while some teams have neither. And the level of automated testing is widely different between teams.

White cards are generally not testable by professional Testers. They should be tested by developers using Automated Unit Tests and other tools to ensure they are acceptable. If there is something a Tester could test they may well be involved.

Generally professional Testers work at the Blue card level. In work breakdown a team might write a White task card to test the Blue. This would only be done once all the Whites were done and the whole Blue was ready to test.

Although Testers may prefer to write two task cards: one to write the test script and one to execute the test. If the former is fully automated the latter need not exist or will happen automatically.

Other teams forgo tasks associated with testing and instead model tests via their task board. As cards move across the board they will need to pass through test columns were the testing will happen. Thus completetion of all the White cards associated with a Blue would trigger the move of the Blue into a testing column and testing to commence.

Trivia and Spikes

Truly trivial tasks, or work to be undertaken by people outside the teams may be assigned zero points. Such zero point cards represent work that the team needs to keep track of but does not represent noticeable work for the team. For example, a "Buy domain name" task would probably merit a zero point score as it would take about 10 minutes. But a task reading "Obtain quote for domain name, seek approval to spend money, buy domain name, file expenses claim" may warrant an estimate larger than zero.

Spike cards are written when the development team feel they do not have enough knowledge 0 usually technical knowledge - to begin a breakdown and estimate. Here a "Spike" card will be written to attempt the work but once the work is done it will be thrown away, "spiked."

The objective of a spike card is gain an understanding of what needs to be done. Typically the output from a spike will be a set of (White) cards describing the tasks which need to be done. Ideally these cards will be held until the next planning meeting where they can be discussed, estimated and scheduled, or deferred.

Sometimes when time is pressing the resulting cards might be estimated and scheduled into the iteration immediately. While this is entirely practical is does mean forecasts for what the iteration will produce are difficult or impossible.

Spikes are not estimated the same way other cards are estimated. Rather they are hard time-boxed. An amount of time is decided on and written on the card. This time, no more, no less, is the time allowed for this card. When the time is up research work must stop and the task cards written using the knowledge gained.

Working in time - as opposed for points - for spikes makes velocity calculations more difficult. Some approximate, rule-of-thumb, back-of-the-envelope calculations need to be done.

Counting Done

In the review part of the meeting the work completed in the previous iteration is removed from the board and reviewed. The main part of this review is simply counting the points done and updating any charts or other tracking systems. The review may also take time to examine any cards left on the board and decide whether they should be left as carry over.

As already mentioned the estimated points on completed Whites are counted as part of the teams iteration velocity. Only 100% completed Whites are counted.

Anyone who has worked with software teams for more than a few years will have seen the "80% done" scenario where a piece work remains 80% done for 80% of the time allowed. Therefore no matter how much a Developer begs "It is 95% done" incomplete cards are not counted.

In software development we have no way of objectively telling what is 95% done and what is 9% done. We have no way of knowing if an unexpected problem lurks in the final 5% or if someone will go ill before the day is out.

Teams are encouraged to adopt a Definition of Done to help with defining what is *Done* and what is *Not Done*.

Some see it as odd that I allow Whites to be counted even when Blues are not. "But the business functionality is incomplete" they say, "and its the business need that counts." This is reasonable - and certainly follows the rules of Scrum - but I find it leads to less predicability in the process and improves flow.

Another reason for applying these rules it to set up a small incentive for people to complete work before the end of iteration review and thus score more points. This adds a little extra motivation.

Velocity and currency

Velocity is a measure of how fast a team are working, or rather, how much work they are getting through. It is calculated by counting the points a team scored (i.e. completed) in the previous iteration.

Over a series of several iteration, say four, a team should be able to come up with a rolling average, a high and a low water-mark which can be used form planning purposes. For example, consider the team shown in this graph:

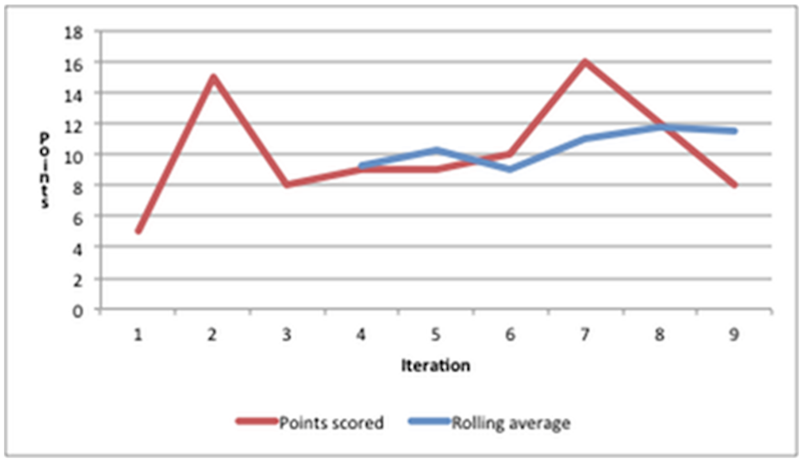

Figure 3: Velocity and rolling average

This team scored: 5, 15, 8, 9, 9, 10, 16, 12 and 8 points in the nine iterations shown here. At the end of iteration 4 the team could calculate a average of the last 4 iterations, this could be rolled forward at the end of each iteration giving rolling averages of: 9.25, 10.25, 9, 11, 11.75 and 11.5.

I would advise the team to plan for 12 points of work (because their recent average is 11.5) but accept 16 or 17 points worth - up to their recent high-water mark. Those in the planning meeting will be aware of the possible outcomes but for those outside the meeting there needs to be some management of expectations.

It is pretty much certain the team will achieve the first 8 points worth of work. The team might get points 9 to 12 done, or they might not. If they are very lucky they will do points 13 to 16.

If someone needs to know how much time the team spent on a particular task then it is simply a question of maths. Assume there five developers employed for 40 hours a week in the previous example. That is 200 hours of work producing on average slightly more than 11 points of work, so each point took on average a little over 18 hours, thus a two point card took about 36 hours.

I would prefer not to make this calculation too well know because once it entered general knowledge it would undermine the points system. I would also prefer that this calculation was regularly updated since, as shown in the table, the averaging changes.

It is vital to note that points float. Like the US Dollar, Euro or Pound Sterling, points are a currency and change their value over time. Each team has its own currency which is not directly transferable to another team.

As with currencies and other economic indicates setting targets for velocity can create problems. Goodhart's Law applies, if a team try and target a certain number of points they will meet their goal but they may not do any more work. Such teams exhibit inflation in estimates, exactly as with financial inflation the numbers are bigger but the value less.

Carry over work

For a strict Scrum team there is no issue of work carry over because teams only commit to work they guarantee will be done and thus all work committed to is done. While many Scrum teams find carrying work over from sprint-to-sprint and anathema I advise teams to carry over work. Indeed, carrying over work to improve flow is a central feature of Xanpan and is discussed in my writing on Xanpan process.

For Xanpan and other teams work carry over is a fact of life. As part of the review process preceding the planning meeting the team should look at the work remaining on the board from the closing iteration and decide which, if any, work will be carried over to the next iteration.

When work has not started on a Blue and associated Whites the Product Owner may decide to pull the card completely or roll the whole thing over. Assuming they roll it over it will need to be prioritised against the new Blues being added. That is to say: just because a Blue is rolled over does not give it special priority.

When some task associated with a Blue have been done and some tasks have not the situation is more complicated. While the Product Owner may still pull the Blue or assign it a low priority it probably makes more sense to finish work which has been started, and finish it soon, rather than leaving it in a partially done state.

There may also be engineering reasons why the Blue should be taken to completion before anything else. For example, some of the new Blues may involve the same areas of code.

In a few cases work is incomplete because after it began more tasks came to light. While I do not allow teams to change estimates on Whites once they are accepted into an iteration the team may write new Whites for additional work which emerged. They may even estimate and begin work on these Whites during the iteration if needed be. Although I prefer it if new work can be held until the planning meeting where it can be discussed, prioritised and scheduled by the team.

Obviously this approach raises the possibility of never ending work, Blues which are never done. Senior Team members need to be watchful for this and work to diagnose the underlying issues causing it.

Ball-park estimates

When the team has finished breaking down Blues to Whites, and when the work for the next iteration has been agree, and if time allows, then the team may undertake some longer range estimates. I usually call these estimates "Ball park" estimates but perhaps that is too American a term, "Rough estimates" might be a better term but this itself has been abused over the years.

Once all the work is decided the Product Owner may present some Blue cards for Ball-park estimation. These may be cards which have recently been added to the backlog or ideas which have been suggested recently. Or they may be existing Blues the PO is considering for a future iteration.

Ball-park estimates are made on Blue cards without any breakdown. Discussion is kept general, specifics are, as far as possible, ignored, and generally these estimates are significantly higher than those found on Whites.

It is vital to remember that Blue ball-park estimates are exactly that: estimates. They are for guidance only.

Ball-parks are not commitments, they are not accurate, they are not specific. They are subject to change and bind nobody. When the time comes for the Blue to be developed it will be broken down and the associated Whites estimated exactly as above.

The sum of these task estimate supersedes any ball-park estimate. The sum might be larger or smaller than the ball-park estimate. Indeed, if the sum of the tasks is regularly the same as the ball-park estimates then something is probably wrong and deserves more investigation.

How long is a planning meeting?

I would expect a well practiced team to complete a planning meeting in half a day, my preference is for afternoon meetings. Obviously there is some variability depending on how big the team is, how much work is being planned and whether the team is carrying over any work, but half a day should be enough.

The exception is the first meeting which frequently take much longer, perhaps as much as a day. Meetings can also stretch on when Product Owners are poorly prepared for the meeting or take issue with estimates. Design questions can also derail meetings but on the whole most design issues can followed up later.

A team holding a retrospective before the meeting should allow 60 to 90 minutes depending on the techniques and exercises being used. I find a Dialogue Sheet retrospective takes 60 minutes for the sheet plus up to 30 minutes for post sheet discussion and action items.

Pre-Planning Meeting

Some teams move some of the activities to a pre-planing meeting, or in the case of a the demo and retrospective even a post-planning meeting. The difficulty with holding any meeting after the planning meeting is that any action arising can't be included in the iteration until the next planning meeting. For some teams this is acceptable but in general most teams avoid the problem by holding these meetings before the planning meeting.

Software demos are frequently held before the formal planning meeting. However as the planning meeting represents the end of one iteration and the start of the next holding the demo before the planning meetings creates a gap. During this gap the team might continue working on the software in which case the demo is not complete. More worryingly the software might get broken in this gap. Neither issue need be too problematic provided the demo is not held too far in advance.

A pre-planning meeting is usually an opportunity for the Product Owner to flag up some of the stories they plan to request and get the teams feedback before the formal planning meeting. Teams may even use the pre-planning meeting to make ball-park estimates on Blues. As a result of the feedback the Product Owner might rethink what they actually request in the planning meeting a few days later.

Again there is a danger that if the pre-planning meeting is held too far in advance then changes may render it pointless. A second danger is that teams do too much preparation work in the pre-planning meeting and this work is invalidated by the end of the iteration (e.g. they break down stories which are then de-prioritised).

Since all pre-planning meeting activities eat into the time available during an iteration to actually do work it is preferable to avoid them. However pre-planning can sometimes be useful.

Planning Poker

Readers who have played planning poker may care to skip this section. For those who have not come across planning poker a quick description is appropriate.

Planning poker can be played with a normal set of playing cards but is more normally played with a specially printed set numbers with an approximately Fibonacci sequence in order to spread estimates out. I like to use a set numbered: 0, ½, 1, 2, 3, 5, 8, 13, 21, 40, 65, 100, infinity and question mark. The last two of these are used to flag up problems such as "That is truly massive" or "I don't even know where to start".

Many training and consulting firms get planning poker card sets printed and given away as marketing materials. In addition there are now applications for mobile phones which can be used too. I am not a fan of such application because I believe in keeping planning sessions both physical and interactive.

A the team may make their own planning poker cards. To do this each team member should take several index cards (any colour) and write the following sequence on the cards - one number per card: Half, 0, 1, 2, 3, 5, 8, 13, 21, 40, 65, 100 and question mark "?".

To estimate a piece of work the team place the task card in the centre - easily done when work is on physical card; if working virtually this needs to be done virtually. If the team are unfamiliar with this card there may need to be some discussion about what the card means. For example, if this is a Blue business card the Product Owner may describe what is wanted. If it is a White card then - hopefully - the card was only written a few minutes before as a collective team effort.

Ideally during this pre-estimation discussion team members will avoid saying "O its easy" or "That should be a six" but pointing this out is a good way to ensure it does happen.

When the team are ready they each select a card from their playing cards, e.g. a 5 card or an 8 card, and keep it hidden. Somebody, Project Manager, Scrum Master, Senior Developer or just a team member, gives some kind of lead in (e.g. "3, 2, 1" or "Ready… Steady… Vote") and on que all team members reveal their card.

If all the cards agree, e.g. everyone plays 3, then the estimate is accepted and recorded on the card. If the estimates do not agree there needs to be a second round of voting. Before re-voting someone will give an argument for the highest estimate and someone else for the lowest estimate. Normally only two people need to speak, one person speaking for the high vote and one speaking for the low. Those who voted in-between remain silent, any duplicate high (or low) voters don't need to speak.

Normally this argument is a short statement of position. There is seldom a need to engage in lengthy debate. After the two positions are given the voting process can repeat exactly the same as before.

Hopefully on the second vote the estimates are all in agreement however this is seldom the case. At this point different teams do different things.

After a second some teams vote will engage in a negotiation. They would ask the low voter what it would take for them to go up and ask high voters to go down.

Some teams I have heard of will again take positions and play a third and even more rounds until convergence is achieved.

I advise teams to go with the majority vote (e.g. if four developers vote 8 and one votes 3 I would accept 8, in effect going with the median) or I would go with an approximate mean average (e.g. 7). At this point I am not concerned about the Fibonacci series, it has served its purpose and to keep things moving fast I use all numbers. However some teams stick with the series and will not take numbers not on the cards.

Once one story or task card is estimated the attention shifts to the next. The intention is to keep estimation moving forward in order to estimate a lot of work quickly.

It is important to remember these estimates are just that, estimates. The objective of planning poker is to be roughly right rather than precisely wrong. Coupled with velocity measures and rolling averages accuracy is gained through metrics (averages and aggregates) over multiple estimates and work items. Not through individually accurate estimates.

Both new teams and existing teams adopting planning poker face one troublesome problem: what is one?

In order to find baseline teams should take one Blue which has been broken down into multiple White tasks. They should examine the Whites and find the one which looks like the least effort. This process doesn't need to be exact or have complete agreement. This task then becomes *One*. One point that is.

The team then take the next task, possible the one that looked second least effort or perhaps another. They play planning poker on this and obtain an effort value. The team now have two reference points and can continue.

When a team starts playing planning poker the value of one is very variable. At first the team might feel a need to refer back to the baseline one. Over time the team will gain an intrinsic understanding of what one point is, the value will become more stable and reference back to the baseline should cease. As the team actually do the work this understanding solidifies. Over time the value of one point will change, the currency floats. This is perfectly acceptable.

To draw an analogy: when someone goes to live in another country which uses another currency they regularly calculate back to their home country, currency and prices. This works but the same products costs different amounts in different currencies.

When I left the UK for the USA in 2000 the exchange rate was about £1 to $1.50. A gallon of gas (petrol) cost about $1.80 - a little over £1.20 - high by US standards at the time but unbelievably cheap by UK standards. To complicate matters the UK sold petrol by litres not gallons, and in a final twist a US gallon was about 3.7L and a UK gallon about 4.5L.

After a few weeks the expatriate stops translating back and starts to think in the new currency and reference local prices. (Holidays are seldom long enough to fully demonstrate this effect.)

While the value of one changes teams which stay together continue to use the same baseline as they move from story to story, iteration to iteration and even project to project. Only if the team member change significantly do they need to rebase one.

Finally, as a team decides what one point is, and decides what two, five, 13 and others are the team take these values as shared values. Occasionally individuals try to use their own value system even in parallel.

Some Planning Poker theory

The theory underpinning planning poker is "Wide band Delphi" (see http://en.wikipedia.org/wiki/Wide_band_delphi for more about the history of this method.) You don't need to know that to play planning poker but it does help to know a little of why this silly looking technique is useful:

- A psychological phenomenon known as "anchoring" leads people to rely on the first piece of information that received more than others. Coupled with social pressure to confirm this means that “O that is easy, that should be 4 hours” people are likely to cluster their estimates around this statement. Thus reducing the effectiveness of multiple independent estimates.

- It been established shown that experts are no better at giving time estimates than non-experts. What does improve estimation is to have multiple independent estimates. (see Makridakis, Hogarth, and Gaba 2010; and Surowiecki 2004)
- Some other studies (e.g. Weick and Guinote 2010) report that the greater an individual’s power and authority the more optimistic their estimates. Thus we might expect a Project Manager or Architect to provide lower estimates than Junior Developers.
- According to some researchers humans produce better estimates when they estimate on behalf of others (Buehler, Griffin, and Ross 1994). Rather than ask “How long do you think it will take to do X?” we should ask “How long do you think it will take another team member to do X?”

Why breakdown Blues?

I always advise teams to break down Blue business facing cards into White tasks - although I often called Whites “Developer tasks” this is slightly misleading because they could be tasks for Testers, Analysts or anyone else. Breaking cards cards down has several benefits.

Stand-alone Blue cards should have business value themselves, they should also be small enough to be doable in the near future, e.g. within this iteration. These two aims can be in conflict: something which has stand alone business value needs to bigger which means it cannot be accomplished soon. Breaking it down thus serves two purposes: it provides another opportunity to find small nuggets of business value and provides a way of tracking progress on large pieces of work.

Breaking cards down also serves as an requirements elicitation processes. The team need to discuss the Blue between themselves and with the Product Owner - hence why I like the Product Owner to be in the room during this process. This dialogue serves to flush out details, additional requirements, assumptions and mis-understandings.

The breakdown is also a **design activity** because it causes the team members to discuss how they will approach the coding of the story. For some teams this design activity is major part of the breakdown while for others it is trivial.

Taken together the design and requirements elicitation serve to build a **shared understanding** between all teams members: coders, testers and product people. In doing so it helps identify the key value elements of the stories and provides a forum for trade-offs to be made.

By breaking Blue stories separately before estimating of Whites the team engage in planning the tasks free from estimation. When estimation happens it is based on a scenario - the Whites. This forces people to think what is involved in achieving the goal rather than the goal itself (Buehler, Griffin, and Ross 1994; Wiseman 2009). Part of the reason for optimistic estimates is that people focus on the goal and the desirability of achieving the goal leads to wishful thinking.

Indeed, Wiseman advices that in setting a goal one should share it with friends and family, break the goal down into a series of sub-tasks, and reward yourself as your progress. This is advice match the process described here: Blues selected and publicly stated, they are broken down collectively to sub-tasks and as each sub-task is completed it is moved across the board and the points scored. Because the breakdown is collective it is open and shared, moving completed cards across the board is a public statement of success.

While Blue business cards are normally written in User Story format (As a… I can… So that…) the same is not true of Whites. These are written in whatever language and format makes sense to the team. Similarly while Blues will typically have some acceptance criteria associated with them Whites do not. It is up to a Developer to set their own criteria for Whites, usually using Unit Testing.

(Because each Blue represents business functionality they may be tested by professional testers individually. Whites are not usually testable alone by professional testers.)

If in breaking cards down the Product Owner sees a White which they deem to have business value in and of itself they may "upgrade" the card to a Blue.

If this card then need breaking down itself then it is broken down exactly the same way.

Some teams find they do not need to break down Blues. The Blues are themselves small enough to be worked on. This is perfectly acceptable. Similarly some teams find that with experience they can dispense with the breakdown to Whites. Generally I tend to find such teams work with more modern technologies, e.g. they are building websites in PHP or Ruby. Teams which benefit more from breakdown are working with older technologies or are further from the user, e.g. telecom and server systems written in C++ or Java.

Estimate in Points Not Hours

I always advise teams to estimate in points not hours. I usually use the term "Abstract points" although some people called these "Story Points" (Cohn 2004) and several other terms are used, e.g. Nebulous Units of Time. The important point is: Points are not hours. A point is an *Unit of effort* - it is not, specifically, the amount of time it will take to do something, nor is it a measure of complexity. It is the measure of the effort required

Some people estimate stories (Blues) in some kind of points but switch to hours for tasks. I do not recommend this approach. As I will describe in a moment I don't believe humans can accurately estimate in hours. Secondly, I see little point in using one unit of measurement for stories and another for tasks. To my mind points are a team's currency and using two different units is equivalent to having two currencies in circulation.

That said I expect estimates on White tasks to be smaller than estimates on Blue stories. When playing planning poker I expect Whites to be estimated using the smaller numbers (1, 2, 5, etc.) and ball-park Blue estimates to use the bigger numbers (21, 40, 65).

In 1979 two Pentagon researched published a paper describing "The Planning Fallacy" (Kahneman and Tversky 1979) (the same authors went on to win the Nobel prize for Economics for not entirely unrelated other work). The planning fallacy states:

- people tend to underestimate the amount of time that work will take to get done. This isn't occasional or random, its systematic

- people are overconfident in their own predictions
- even when shown with evidence of past optimism

These findings have been upheld by multiple subsequent studies, (e.g. Zackay and Block 2004; Buehler, Griffin, and Ross 1994). The Zachay study is particularly interesting in that it extends the Planning Fallacy to *Retrospective Estimates*, i.e. asked to state how long it took to undertake a task they have already done an individual will still make an underestimate of the time spent. Other studies (e.g. Buehler, Griffin, and Peetz 2010) support finding that the past is remembered optimistically.

Around the same time Douglas Hofstadter coined Hofstadter's Law:

> "It always takes longer than you expect, even when you take into account Hofstadter's Law." (Hofstadter 1980)

When asked to estimate in hours a number of additional forces come into play: nobody wants to be considered a slacker at work, people may actively want to disguise how much time they spend doing a task, or they may consciously change their estimates in an effort to be assigned, or to avoid, a particular piece of work.

But estimating in points is only half the story, to complete the story, we need to consider the past performance of the team, their *velocity*. Only when both are known can accurate time based estimate be made.

The primary reason for moving teams away from hours as an estimation unit is to help compensate for the the planning fallacy. By estimating in points and comparing the points to past performance uses historical data which individuals do not.

Ideal hours

Some teams prefer to estimate in "Ideal Hours" unfortunately an ideal hour rarely exists but using this expression itself creates misunderstanding. To team members it is some abstract measuring unit which vaguely resembles an hour; an ideal hour is an hour where everything went well: no interruptions, no distractions, no unexpected surprised.

To those outside of the team it is an hour.

The question is: when planning work what do the team benchmark themselves against?

If a team benchmark themselves on a standard 40 hour work week then ideal hours need to add up to 40. if they do not then management may well wonder what the team are doing with the rest of their contracted hours.

If instead the team benchmark against their past performance then the unit of measurement is floating and is effectively an abstract point - by whatever name we choose to call it.

And "Actuals"

As mentioned before humans underestimate how long it takes to do a piece work even in retrospect. Thus the thing that many companies call "Actuals", i.e. how long it took to actually undertake a piece of work, is nothing more than another estimate. Although usually this retrospective estimate is generate by an individual rather than by a team.

To my mind the fact that an retrospective "actuals" estimate has not benefitted from multiple independent estimates, is the product of one person and may reflect biases in the way they work - or how often they go the toilet - renders it a different currency. Consequently I ignore "actuals."

The time-tracking systems used by many corporations compound the problems with actuals. One friend of mine reports the American bank he worked for did not allow more than the contracted 37 hours a week to be entered. And Capers Jones points out that few systems allow "slack time" or unpaid overtime to be entered (Jones 2008). Jones reports normal software measurement practices seldom collect more than 80% of the true effort. For MIS projects can omit 60% of the total effort according to Jones.

In some cases managers may also encourage team members to change actuals to match estimates. One company I saw awarded Project Managers bonuses for the accuracy of estimates against actuals recorded. This incentived Project Managers to ensure all team members recorded a number of hours equal to the estimated hours.

As a consequence traditional time tracking systems can be a major source of project risk. If time-tracking data from one development initiative is used to forecast and plan a new one then the new work may immediately start with a 20% schedule slippage, and possibly a 20% budget overrun simple because the benchmark was inaccurate.

While I would like teams to shun traditional time-tracking systems altogether this may not be very acceptable in some environments. I advise people in this situation to double-think.

For work planning and estimation use points and the tools described here. For filling in time-tracking systems forget all that and use whatever mechanism you like. Such systems exist for the benefit of corporate accounting and have nothing to do with sizing work.

The golden rule is: do not let the data from the time-tracking system be used for planning purposes. These are two different currencies. Using them together would like offering to pay a restaurant bill for $70 with a $20 and €50 note.

Deadlines

The research mentioned above also throws up another interesting finding: People are very good at working to deadlines. In one study (Buehler, Griffin, and Ross 1994) the (external) deadline was met 80% of the time. This finding had nothing to do with estimates, estimates were still too low, but it seems people meet deadlines.

When I explain this to people I often ask a group: "How many of you did courses at school or college which involved doing an assignment to be handed in to a deadline set by the teach?" This is a familiar experience to most people. I then ask: "When did you do it? How many of you did the work in plenty of time? And how many left it to the day or night before?" Overwhelmingly most people leave assignments until as late as possible.

When estimates are made with an externally imposed deadline people change their estimating behaviour. In one experiment the researchers above set two groups the same task but with different deadlines. The ones with the later deadline provided larger estimates. Yet the extra time did not make a difference to the actual time taken to complete the task.

While subjects in the experiment denied letting the deadline influence their estimates the estimates were highly correlated with the deadline. Possibly deadlines are imposing a form of anchoring.

Imposing deadlines - external deadlines - can be demotivating for people. In my own experience when I have had an arbitrary deadline imposed on my at work I am demotivated.

However this anecdote does not seem to stand up to research. It seems that externally imposed deadlines are more effective at delivering task completion (Buehler, Griffin, and Peetz 2010) and regular evenly-spaced deadlines are more effective still (Ariely and Wertenbroch 2002)

The planning process described here exploits these factors in several ways:

- Estimates are kept free from deadlines.
- Deadlines are effective externally imposed and occur regularly
- Fairness is maintained because team members decide between themselves what they will do in the time, and, over the longer term, have a say in how regularly the deadline happens
- If a customer wants a piece of work by a particular date (and I encourage them to think like this) then there is a negotiation with the development team over what can be achieved in the time.

Product Owner Preparations (Homework)

One of the reoccurring reasons I see for planning meetings not going smoothly is a lack of preparation on the part of the Product Owner. The planning meeting is not the place for the Product Owner to decide what is required, although they may may make trade-offs and substitutions during the meeting they need to go into the meeting knowing very clearly what they want to ask for.

The Product Owner needs to be on top of their brief, they need to be able to answer developer questions and clarify what is being asked for. If they cannot they need to either bring someone who can or they need to be prepared to make changes to what they want. Thus, if the Testers and Developers ask questions about issues the Product Owner cannot answer immediately they can bring another story into play while they find out the answers. This may mean that a story is postponed until the following iteration - or even later - or it may be possible to schedule the difficult story until later in the iteration.

As you might guess from this it helps if the Product Owner is not only prepared for the iteration they are planning now but also has a rough idea of what they plan to ask for in future iterations. These plans shouldn't be too detailed - because things change both in priority and detail - but the Product Owner needs to have some ideas.

Medium term plans, about the next few iterations, were traditionally called Release Plans but I believe the name Quarterly Plans bot better describes the plan and moves away from the association with releases. I have discussed such plan elsewhere (Kelly 2010) and will do again.

It is also critical that the Product Owner has the authority from the organization and team to make decisions during the planning meeting: on priorities, on changes to priorities, on details of features and on trade-offs. Nothing is more disruptive - and morale sapping - than completing a planning meeting one day to discover the a day or two later that somebody else has overruled the Product Owner and has changed what was agreed in the planning meeting.

As mentioned before, there is sometimes a need to have more than one Product Owner in the planning meeting. When this is all Product Owners concerned should be in agreement about what is going to be asked for, what the priorities are and be prepared for problems. It may be that the Product Owners benefit from having their own small meeting prior to the full planning meeting.

Now

> Visit http://www.softwarestrategy.co.uk/dlgsheets/planningsheet.html to download a sheet or to buy a printed sheet.

References

Ariely, D., and K. Wertenbroch. 2002. "Procrastination, deadlines, and performance: self-control by precommitment." *Psychological Science* 13 (3).

Buehler, R., D. Griffin, and J. Peetz. 2010. "The Planning Fallacy: Cognitive, Motivational, and Social Origins." *Advances in Experimental Social Psychology* 43: 1–62.

Buehler, R., D. Griffin, and M. Ross. 1994. “Exploring the ‘Planning Fallacy:’ Why People Underestimate Their Task Completion Times.” *Journal of Personalty and Social Psychology* 67 (3): 366–381.

Cohn, M. 2004. *User Stories Applied.* Addison-Wesley.

Hofstadter, Douglas R. 1980. *Godel Escher Bach: An eternal golden braid.* Harmondsworth: Penguin Books.

Jones, C. 2008. *Applied Software Measurement.* McGraw Hill.

Kahneman, and Tversky. 1979. “Intuitive Prediction: Biases and Corrective Procedures.” *TIMS Studies in Management Science* (12): 313–327.

Kelly, A. 2010. “Three Plans for Agile.” Toronto: RWNG. http://www.requirementsnetwork.com/node/2663.

Makridakis, S., R. M. Hogarth, and A. Gaba. 2010. “Why Forecasts Fail. What to Do Instead.” *MIT Sloan Management Review* 51 (2).

Surowiecki, James. 2004. *The wisdom of crowds.* 1st ed.. New York: Doubleday. http://www.loc.gov/catdir/bios/random055/2003070095.html.

Weick, M., and A. Guinote. 2010. “How long will it take? Power biases time predictions.” *Journal of Experimental Social Psychology* 46.

Wiseman, R. 2009. *59 Secods.* Macmillan.

Zackay, D., and R. A. Block. 2004. “Prospective and retrospective duration judgments: an executive-control perspective.” *Acta Neurobiol Ex* (64): 319–328.

This essay is a work in progress. The author welcomes comments and feedback at the address above. April 2013

About the author

Allan Kelly has held just about every job in the software world, from system admin to development manager. Today he works as consultant, trainer and writer helping teams adopt and deepen Agile practices, and helping companies benefit from developing software. He specialises in working with software product companies and aligning products and processes with company strategy.

He is the author of two books “Business Patterns for Software Developers” and “Changing Software Development: Learning to be Agile”, the originator of Retrospective Dialogue Sheets (http://www.dialoguesheets.com), a regular conference speakers and frequent contributor to journals.

Allan lives in London and holds BSc and MBA degrees. More about Allan at http://www.allankelly.net and on Twitter as @allankellynet (http://twitter.com/allankellynet).