Tracking, forecasting and learning with charts

Allan Kelly, March 2015

Just about every Agile team uses some form of chart to track progress and create forecasts. Burn down charts are the most common but cumulative flow charts are common too. Yet looking at these humble graphs in more depth throws up a number of important lessons. In this essay Allan Kelly takes a closer look at these graphs and what we can learn from them. . .

In keeping with the visual nature of Agile teams usually use some form of graph to keep track of progress. Such charts can also serve to forecast when a team will be complete or highlight when problems arise.

In general forecast of future work - start dates, end dates, progress, etc. - have a lot in common with weather forecasts. They describe what we think will happen based on the information and tools we have available. Like weather forecasts the further into the future one looks the unreliable the forecast. A forecast for tomorrow carries a lot of certainty, a forecast for next week can be quite reliable too, but next month? Or six months out?

In keeping with a visual approach to management Agile teams generally use some kind of chart to track progress and forecast into the future. These charts have two purposes, one is to forecast the other is to identify problems and opportunities for improvement.

The burn down chart is the most popular of tracking tools but while very intuitive in nature it suffers from several problems. The alternative cumulative flow diagram (sometimes called a burn-up chart) offers a more informed view of progress and is more accommodating but at the price of being more difficult to construct and read.

Most, if not all, electronic Agile management and tracking tools will produce graphs and charts for a team. However if individuals don't spend some time learning to read these charts they may well misread the charts.

Lets start by examining a burn down chart and then discuss some of the problems they throw up. After that we can look at cumulative flow diagrams.

Burn down charts

Figure 1 is a burn down chart, the intuitive nature of the burn down chart is the great advantage they have over other tools. A few moments spent

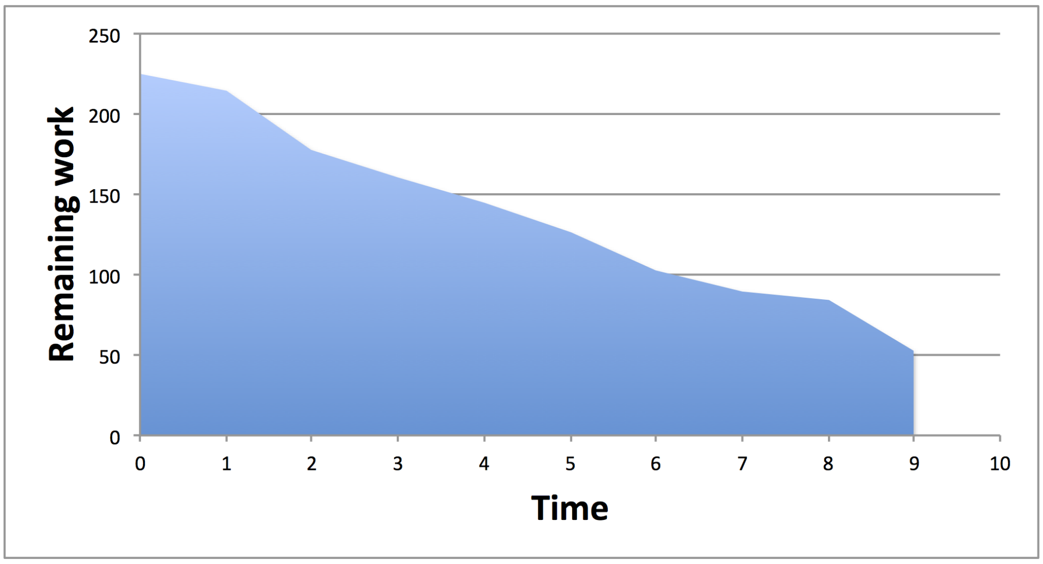

Figure 1: Simple burn-down chart

studying this diagram and you can probably understand what it is telling you.

Simply, the work a team needs to do is shown on the Y axis. Bit by bit the work is reduced. The team will be done - in the traditional sense of having undertaken all the work they were tasked with - when the area shown on the graph is reduced to zero, i.e. the line interests the X axis.

Of course the team might finish before they reach zero - figure 1 might show a team's historic progress to completion in iteration nine with 50 units of work remaining. Alternatively the team shown may still be working and aiming to do all requested work. In this case figure 2 used a simple extrapolation to forecasts completion in iteration 12.

The unit of measurement for the work doesn't make much difference to the chart itself. Many teams use story points, others use nebulous units of time, abstract points, drupals, teabags, ideal hours or even hours. Whatever unit of work is used the team starts with a lot of it and gradually reduces it. The speed at which the team reduces the work is called the velocity and is measured in the same units. As long as the quantity of work and rate of change are measured in the same units the charts have integrity.

In fact there are two commonly used burn down charts. The first is used within the sprint. Intra-spring burn downs typically use days as the time interval on the X-axis, so the "Time" label on these figures would be replaced with “Days”. Each day as the team completes a bit more work the graph is updated.

Hopefully the team will complete all the work by the end of the sprint. When that starts to look unlikely the team can take action - work more hours, remove some work, bring in help or whatever.

Personally I have never found these intra-sprint burn down charts much use. I would rather look at the state of the team board and see how much work had been complete and how much was left to do. Having said that, if a team find these charts useful then they can by all means use them.

I have always found the second kind of burn down chart more useful. These inter-sprint burn down charts span multiple iterations. In these the X-axis would be labelled “Iteration” or “Sprint” and at the end of each two-week period the graph would be updated.

With such charts forecasting the end date (i.e. when all the work will be done) is simply a matter of finding a line of best fit and extrapolating it to interact with the X-axis, i.e. the point of zero remaining work, as in figure 2 This may be done mathematically, and Excel has several suitable functions, or manually by printing the chart and lying a ruler on the graph - a process most of us learned before the age of 14.

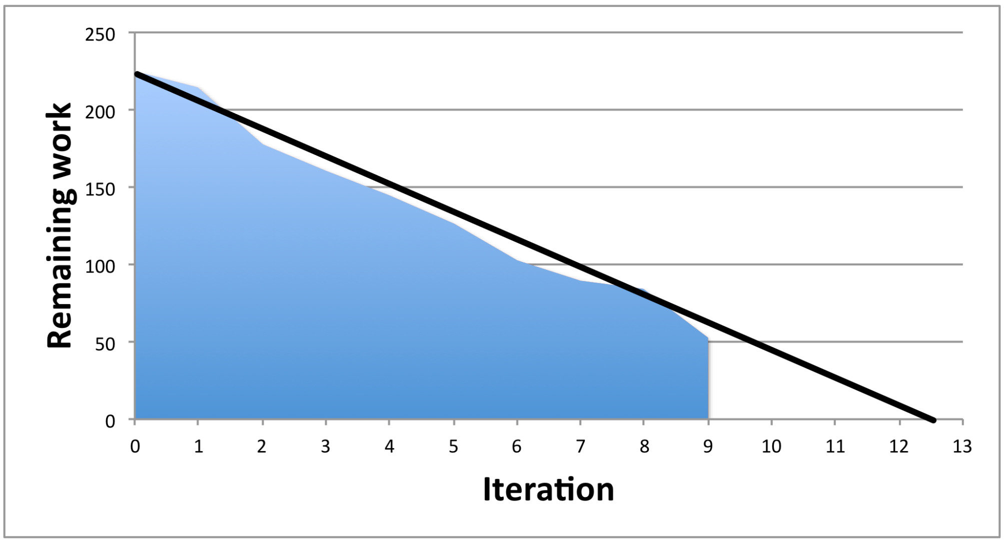

Figure 2: Burn down with a crude line of best fit to forecast end

The key to obtaining accuracy with such charts is to have more data points - this is also one of the problems but more of that in a moment. Again, the units of measurement are not important as long as they are consistent, accuracy comes from having many data points rather than highly accurate individual points.

Before discussing the problems with turndowns - and then the alternatives - let us re-iterate the advantages of burn downs:

- burn down charts are intuitive, they are easy to read and relatively easy to create
- burn down charts are very popular with development teams and therefore well know
- burn down charts can be highly accurate when there is plenty of historical data

Problems with burn down charts

Despite the intuitive nature burn down charts have a number of problems. Knowing these problems can help avoiding them and may motivate you to use the alternative, cumulative flow charts, instead.

Data

The first problem stems directly from the main strength of burn down charts: data. Without some data points a burn down chart cannot predict anything. A new team, with no record of past performance, starting a new piece of work, has no way of forecasting when it will be done.

If a team has put some high-level “ball park” estimates on the initial work then it is possible to draw a burn down chart and mark the work to do on day-zero, i.e. before any work commences. However knowing how many units of work need to done alone is not enough.

Imagine your manager comes to see you: “Would you like a new job in the company’s new office? The office is overseas and comes with a salary of 250,000 and a relocation package of another 250,000 to spend as you wish.” Without known the currency, and details such as the exchange rate, cost of housing and food, it is impossible to make a rational decision.

A new team, at the start of work faces a similar problem. If they do not know what one unit equates to in time and how many units a week they can achieve then they have no idea whether 2,000 units of work is one week or one year of work.

Even if a team has some known unit of work which they can use to estimate, say "Ideal hours" the team does not know how fast ("velocity") they will go. This is like asking someone: "How long will it take you to travel the 400miles from London to Edinburgh?". Unless the mode of transport, and therefore approximate velocity is know the question cannot be answered using data because there is none.

Less obvious is that once a team starts to undertake work and is able to start answering some of these questions the answers will have a low probability of accuracy. Forecasting by extrapolation on a burn down chart with two data points, say the initial work score and the remaining work at the end of iteration one, will provide an answer to the "when will you be done?" question but almost certainly this answer will be wrong. One data point is just not enough data.

At the end of the second iteration there are now three data points on the chart. It might now be possible to issue a best-case/worst-case prediction - as figure 3 shows.

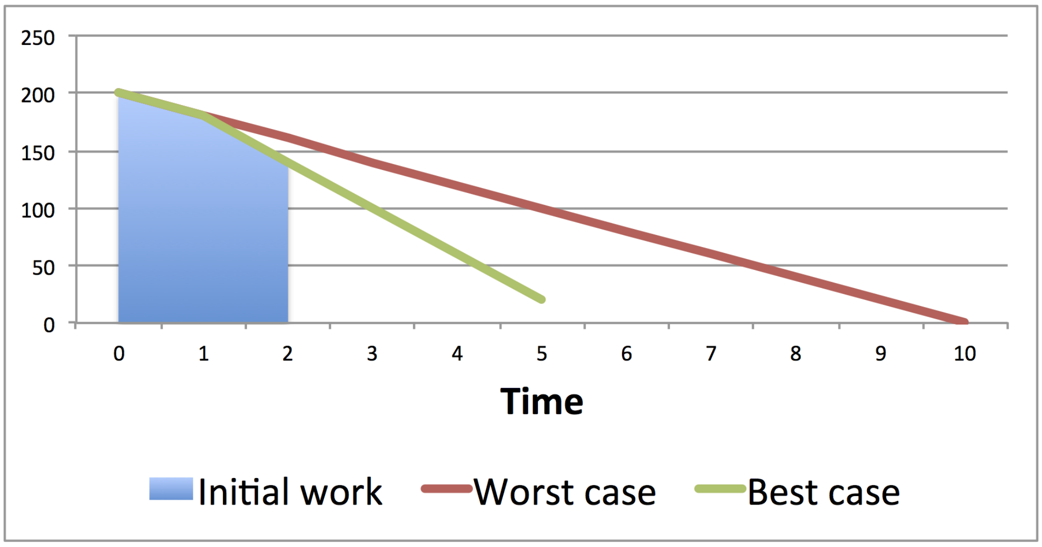

Figure 3: Burn down chart forecasting best and worst case completion

As iterations are completed and the data set becomes larger the accuracy will improve. The more data there is the more accurate the prediction.

This in itself is an argument for keeping teams together and allowing them to roll from one piece or work, or project, to the next piece of work. When a team is broken up the historic data is useless because the team no longer exists.

Reducing a team size has a fairly predictable effect on teams, e.g. if one developer leaves a team of five one might expect the velocity to fall by one fifth. But increasing a team but one might not have the same benefit because it takes time for a new member of staff to become productive. (Although in the short term the team might become more productive even before the new member of staff starts simply because they no longer spend time reviewing CVs/resumes and interviewing candidates.)

Done means. . . ?

The second problem with burn down charts is that they encourage the idea that “done” means “doing all the work which has been requested.” This assumption might hold true in a traditional project environment where a team’s success is measured on whether they deliver everything which was asked for but that does not mean it is a good assumption to make.

Suppose a team is given a large batch of work to do, say 100 story points, and asked to do it. If the team is prioritising work by business benefit then logically they would tackle the high value items first and leave the low value items till later. At some point it might be reasonable to ask: if the team are only doing low value work does it make sense to continue?

For example, imagine a team asked to undertake 30 backlog items. Imagine 10 of these items are valued at $500,000, 10 at $250,000 and the remaining 10 at £50,000. If the team delivers one item per iteration, and maintains a steady pace, then after the first 10 iteration the team have delivered $5million of value. But in the next 10 iteration they only deliver $2.5million. Does it make sense for the team to continue? If one knows that the next 10 iterations will only deliver fifth of this value the question becomes much clearer. Put it another way: in the last 10 iterations they only deliver the same value as the first one.

Given this scenario would it not make sense for the team to stop part way through an project. Abandon the remaining work and start the next project? If the next project has the same benefit profile as the first this approach will make the organization much more money.

Regular releases

Once upon a time done was important because that was the point at which the product would be released to customers. What constituted done might vary, but once done the product would be given to customers.

At that point the business benefits would start to flow - either in terms of revenue, money paid for using the product, or benefits from using the product.

But, when teams are regularly release ready - every two weeks to better - then actual releases can happen more often. Not only do some teams actually release this often some release much more often. The Guardian newspaper in the UK releases several hundred times a week. Amazon are reported to release several times a minute.

When teams release this regularly the economics of development changes. In the first instance benefits start to flow much more often. Even if a team is running behind schedule if releases are delivering value what does it matter? As long as the team are delivering more benefit than they cost why not continue?

Additional benefit comes because the team is giving the business a choice, an option, with every release: continue or change?

If there is other work the same team could undertake which would deliver a greater benefit then switching work stream will deliver more value. When a team deliver regularly they are never far from a delivery and safe place to stop. There is no need wait long to "finish this feature" or "complete the test cycle" or, heaven forbid, "fix the bugs." The sunk costs are much less of an issue in deciding what the team does.

Traditional teams often have sunk costs: money which has been spent and will be written off if the work is not completed. For example, if a company spent a $1 million budget to reach "feature complete" and then find the need to spend $100,000 to undertake bug-fixing and more testing before the software can be release, then, it has $1 million of sunk costs. If it does not spend the $100,000 then the while $1million must be written off.

However, if the company spend $100,000 to fix the bugs there is no guarantee that it will not need to spend even more. At the $1.1 million mark more bugs will have been found, the software still might not be deliverable and another $100,000 is needed.

Each time the budget - and schedule - are extended in the hope of avoiding a write off the problem becomes bigger. The potential write off gets larger. Meanwhile the cost of delay increases and the forecast benefits may reduce.

Every investment book - whether about IT or anything else - ever written states that future investments should not consider sunk costs. Spending more money to recover money already spent is never a good strategy. But in truth investors, and IT managers, do consider sunk costs - all the time!

In short: once teams start delivering regularly the sunk costs become far more manageable and thus continuing to the end of the project, the end of the burn down chart, makes less sense. And consequently burn down charts become less useful. Indeed, because they appear to show an end they may distort thinking.

More work

Complicating matters further is the fact that work changes as it progresses. Performing teams are likely to uncover new opportunities for work. Traditionally project managers were trained to resist scope creep and change requests because these disrupted the schedule and made completing "all the work" harder.

Some of the new work may be oversights, things which should have been foreseen and included in the original statement. Some may also be fixes: deficits which have been uncovered either in the original statement of the work to do or errors that have slipped into the work as it is done.

And some of the new work will be genuine new work, things nobody had foreseen until work was in progress. IT work often uncovers such work because it is difficult for people to envisage the solution to their problem until they start to see the solution - "I don't know what I want until I see it."

IT work also finds such work because people often don't understand the problem they are trying to solve until they start to see the solution. How many people knew they needed e-mail synchronised between their desktop and their phone until they has e-mail on their phone?

Technology, economic, environment and other changes continue while software work is being done creating unforeseen (and unforeseeable) changes. The longer the time lapse between implementation and delivery the greater the external changes.

The traditional approach to unforeseen work used two broad strategies. Firstly significant resources and time were devoted to examining the requested work and project "scope" in detail before any construction began. The belief was that that this would prevent unforeseen problems.

The secondly strategy was to set up barriers to work entering the system when it did appear. Neither approach made for happy customers.

A third strategy - often used in tandem with barriers - was to differentiate the type of new work being requested: change requests, feature request, bug fixes and so on. Arguing over the categorisation of changes could occupy significant management time and again doesn't make for happy customers.

The general strategy for Agile teams is to embrace change and use it to their advantage. However this does not help when drawing burn down charts.

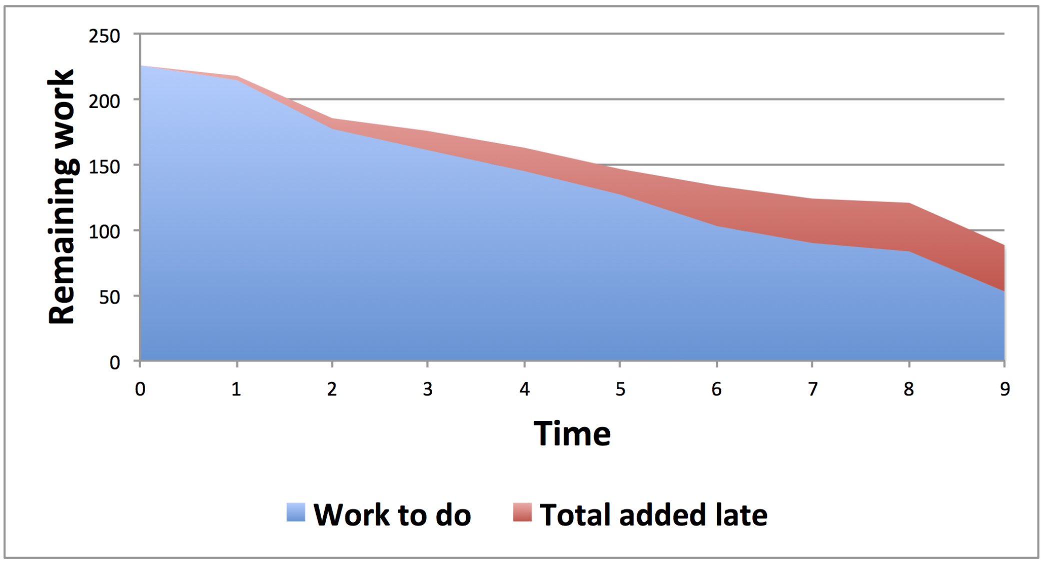

Figure 4: Burn down with added work shown on top

Figure 4 shows how one could try and layer "new work" on top but this ignores business benefit and only provides an accurate view of the world if old work is done before new - a first-in, first-out approach.

Again, if the team are prioritising by business benefit then it is entirely possible that newly found work is more valuable then the original statement. Therefore why would a team continue to do the original work and foregone benefit?

The net result is that such a burn down chart does not represent the true state of play and may even encourage decisions which reduce benefits delivered.

In the extreme change may outpace velocity - like a mortgage that cannot be serviced - and the burn down head upwards.

Hours and days

Having said the unit of measurement is unimportant there is a caveat. Measuring in common time units, namely hours and days, can create problems for several reasons.

To start with estimates using units of time will not be accurate. It has long been known that humans are poor at estimating the time it takes to complete tasks. From "Vierordt's law" in 1868, through "The Planning Fallacy" (Kahneman and Tversky 1979) and "Hofstadter Law" (Hofstadter 1980) humans are known to underestimate task duration both prospectively and retrospectively.

Not only to humans consistently underestimate task duration itself they are consistently over confident in their estimate - even when shown evidence of their own previous underestimation. (Indeed there are even more issues with working in hours, I undertook a longer review of the research a few years ago ("Estimation and Retrospective Estimation" (Kelly 2011) and "More research on estimation" (Kelly 2013)).

Still hour based estimation could work on a burn down chart provided the hours are calibrated against actual deliveries rather than hours worked. But this approach creates problems other. . .

Depending on which country you live in you may be employed for 35, 37.5 or 40 hours a week - indeed you might be employed for a completely different time period. But the burn down attempts to measure work done, not effort expended. If a team estimates in hours and regularly deducts their contracted hours the chart will not be accurate.

Only when burn downs track work done (i.e. completed), not effort expended, can they provide meaningful forecasts. For similar reasons putting "target velocity" or "idealised burn" on charts serves little purpose, I prefer to keep the charts free from clutter.

Hence, if a team estimates in "hours", measures work in units called "hours" and measures velocity in "hours" of completed work only, then and only then, might the chart have some accuracy.

However, the velocity is almost guaranteed to be less than the number of hours people are contracted to work. This creates a new problem.

Although the team are using the word "hours" or "days" as their unit of measurement they are in fact working in some abstract unit. This itself may cause confusion (not to mention cognitive dissonance) for individuals and those who work with the team.

This problem can best be characterised if we imagine a senior manager visiting the team and examine the burn down chart. After a few minutes he observes that the team have a velocity of about 30 "hours" a week per employee. He turns to the team and says:

> "Can you explain something to me? - I know we employ you for 40 hours a week because I see the payroll each month, so why do you only schedule 30 hours of work? What do you do with the other 10 hours we are paying you for?"

Some developers prefer to sidestep these problems by working in "Ideal hours." As long as "Ideal hours" are calibrated against actual completed work this is fine. However, personally, I prefer to avoid any mention of hours or days.

Points obsession

Whatever units are used for measuring work they need to calibrate as the measurement units against completed work rather than some desired work or ideal fit. When teams start chasing a certain number of points, work hours or matching some ideal line on the burn down chart they corrupt their own measurement system.

Teams - and their managers - occasionally get obsessed by scoring point. I've seen teams who were encouraged to "score more points" and companies which try to sign contracts based on points. When this happens the points take on a second role which is incompatible with their measurement role.

When this happens Goodhart's Law is at work:

> "Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes." Professor Charles Goodhart

Goodhart's law was originally coined in connection with economics but it apply here too. Units of work are a currency too and when they are used for control purposes they will change their behaviour.

Cumulative flow diagrams

Figure 5 shows a cumulative flow diagram - or CFD. A simplified CFD but one I prefer. CFDs lack the simplicity of burn down charts and demand a greater understanding from their readers but they represent the state of development more accurately.

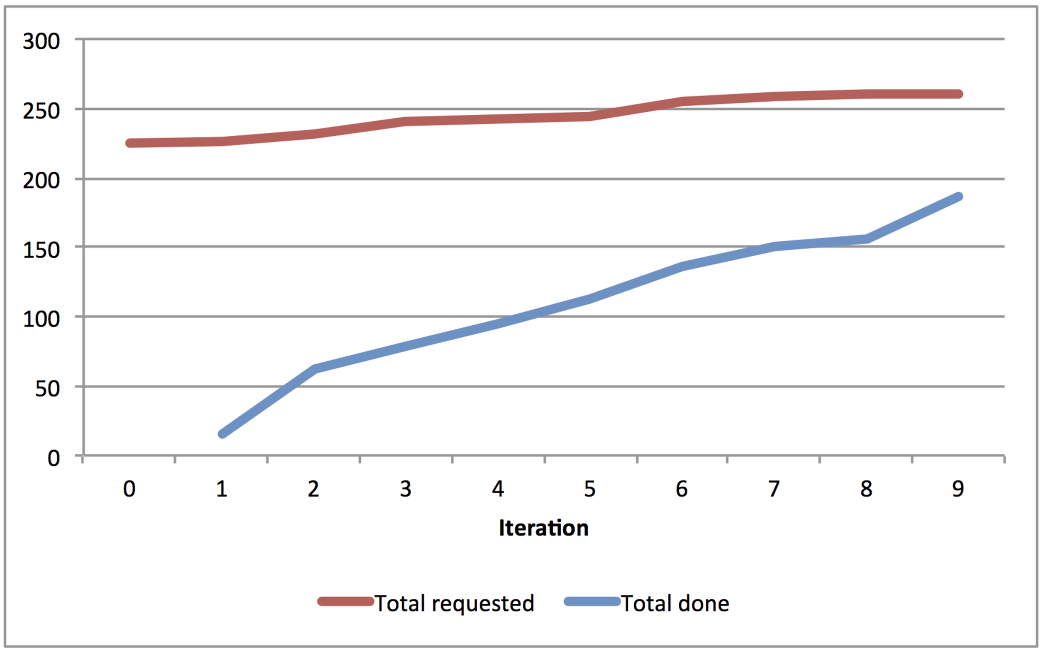

Figure 5: Simple cumulative flow diagram

Lets go through this example: this chart shows nine iterations of work. In fact the data shown in this chart is the same data used to construct the previous (figure 4) burn down chart with added "late work."

The top, red, "Total requested" line on this chart begins on day-zero of the effort, before any development work has been done. Right at the beginning the team believe they have 225 units of work to do. This is typical of work, like a project, which begins with a lot of predefined requirements

Again the actual units used to measure work are largely irrelevant. Although again, if teams measure work in hours the same problems as previously discussed will also be present. I tend to stick with abstract points but some teams count story cards and a few teams - despite my advice! - use hours.

Unlike a burn down chart the "work to do" line is never reduced - unless that is work is explicitly removed form the effort. In a CFD the "work to do" line represents all the work which has been requested of the team - whether this work has been done or is yet to be done. Contrast this with a burn down chart where the lines represents “the work remaining to be done.”

As a result when work is added the “to do” line increases. This more accurately represents what happens to a software development team as work progresses, new things are requested.

The second, blue, “total done” line, represents the work which has been done. At the very beginning, day-zero, no work has been done so there is no line to draw. At the end of the first iteration some work, 15 units in this case, has been done and this point can be plotted. As each iteration completes this line increases by the amount delivered in that iteration.

In effect the CFD splits the single line from the burn down chart into two component parts. While a burn down, at each data point, shows the net work yet to be done the CFD shows the two component elements: work requested and work done. The area between these lines is equivalent to the areas under the line shown on a burn down.

Prediction

Predicting the end of work - where the end means the traditional “all the work is done” - requires extrapolating both lines: one for the total requested and one for the total done. Where the lines meet shows when all the work requested is done.

In figure 6 the total to do and the total done are extrapolated by dashed lines to the point where they intersect. So assuming that after week nine work continues to arrive at the same pace as so far - namely four units of work per week on average. And also assuming that work is completed at the same pace - namely 20 units of work per iteration on average - then all the requested work will be complete in iteration 14.

Obviously there are a lot of assumptions in this prediction but these assumptions can be stated. Furthermore using this approach different assumptions can me modelled. For example, what if after iteration nine no more additional requested were accepted? In that case work should complete one iteration earlier.

Alternatively, what if, work was forced to complete in iteration 10? In that case 60 units of work would be left undone immediately. However what might happen about the work arising? There are another 16 units of work which would have been been found during the eight weeks it would take to reach

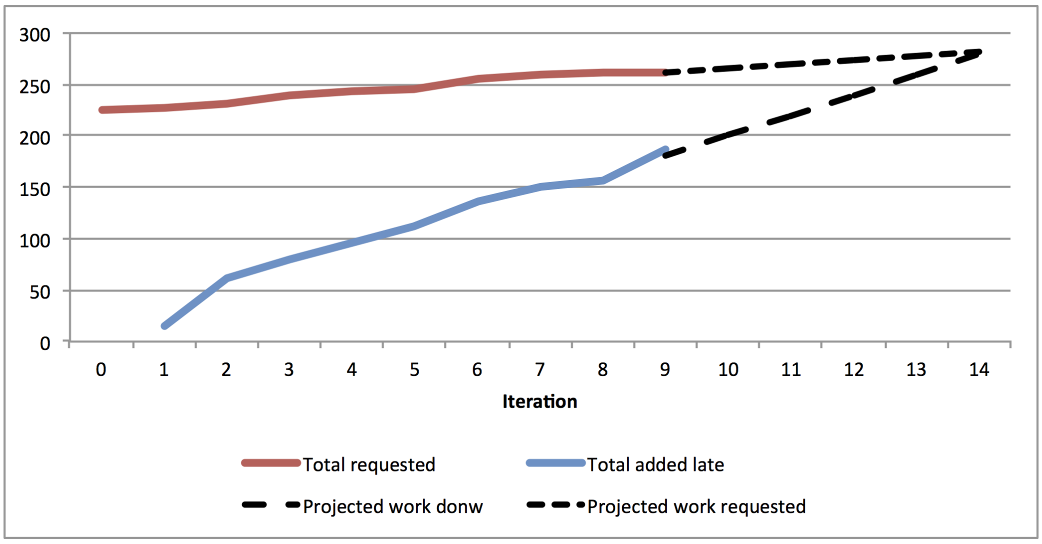

Figure 6: Predicted completion

iteration 14. Would this work have still been requested? Even if there was nobody there to take the request would it be valuable?

And for that matter, even when work is closed in iteration 14 there is no indication of whether more work continued to become necessary. Did more work appear in iteration 15 and 16?

The dashed lines have been produced using one of the simplest methods: calculating the historic average (mean) and applying this from the start of the work for both total done and total requested. There are many alternative, and probably better, mechanisms for extrapolating from past data. Which model you choose to use is entirely up to you.

However, no matter what tool you use for forecasting all will depend on the historic data. Without some historic data all models are little better than guesses. A team with no past record of working together has no data. As more data becomes available more and more statistical techniques open up.

A second example

Now consider a different CFD. This one makes two changes to the format. The first one is largely cosmetic: rather than showing liens the graphs show stacked areas.

The second change is to introduce another metric: released. This area/line tracks the work which has actually been released to customers, i.e. work which has been requested, built and made available for use. This is important in and off itself as work may be completed but sit waiting to be released to users.

In fact this design of chart can be used to track any well define stage in the process and may well track the columns on a team's board. For example: work to do (product backlog), work in development, work in testing, work awaiting sign-off and work released (done) may all be worth of their own line.

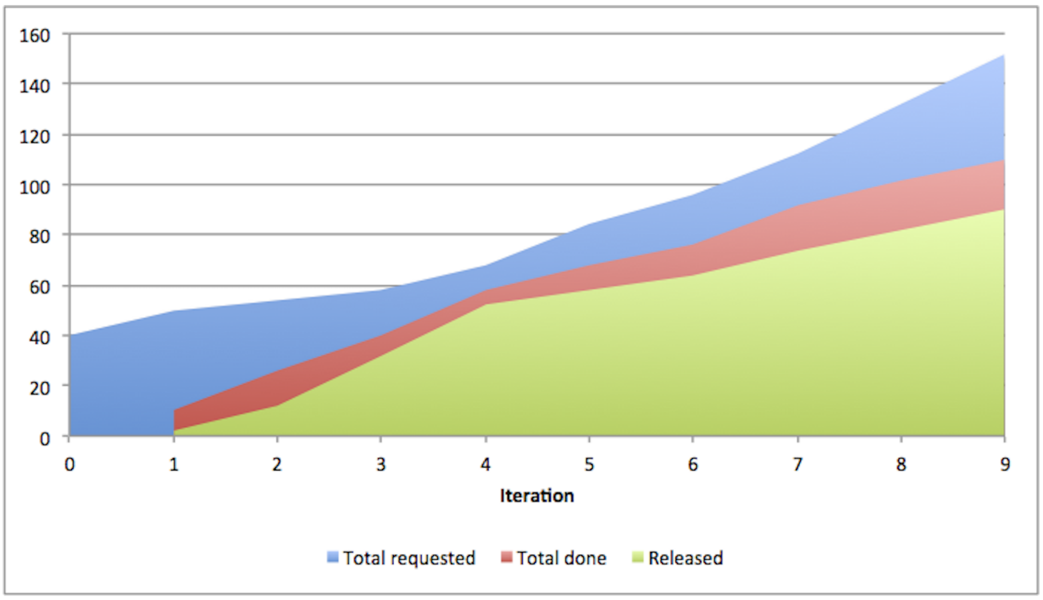

Figure 7: A CFD showing released work

The resulting graph would show a snapshot of progress at regular intervals in the past.

A closer look at the example also highlights some other issues which are easier to see on a CFD than a burn down. Look closely at the initial work: 40 units. Now look at where the total done reached 40 units.

Approximately midway through iteration three all 40 units of work were complete. Lets assume the team operated a first-in-first-out system, while this is a big assumption it keeps things simple and could ultimately be relaxed. Now it is clear that from iteration four onwards the team were working on requests made after the start of the work.

In traditional project management terms this would be "scope creep" and probably considered a problem. Indeed effort may be exerted to prevent such work entering the system at all. However for a team working in an agile fashion this is not necessarily a problem.

Furthermore the graph shows the initial 40 units of work were released to customers by iteration four. So anyone requesting work on day-zero would have waited, at most, eight weeks (assuming a two week iteration.)

Now repeat the process for work which entered the system in iteration one. This work too was also completed and released by iteration four. Thus anyone who injected work two weeks later only had to wait six weeks to see the result.

The same was also true for work infected during iteration two - although only just. These people only needed to wait four weeks to see the result. Work injected in iteration three was coded in the same iteration but delivery was delayed, possibly until iteration five. This took five or six week to be delivered, lets call it six to be safe (figure 8.)

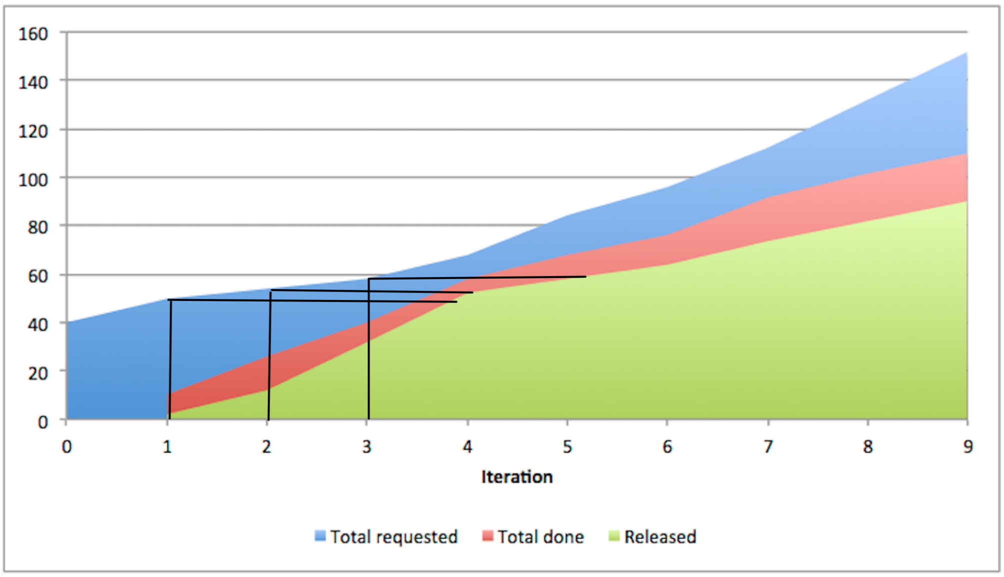

Figure 8: Sampling data points on the flow diagram

By sampling historical data points one can build a model of how long work takes - this is called cycle-time. The data examined so far suggest an average cycle-time of six weeks. Thus even without estimating a single piece of work it is now possible to forecast how long a piece of work will take.

A more detailed analysis could example the progress of each piece of work individually. When this is done the first-in-first-out assumption can be relaxed because the data is not being aggravated into a graph.

Whether a team wants to abandon estimation altogether is another question. Here I only to highlight the possibility, full discussion will need to wait for another day. Having shown the possibility I will also warn against overly simplistic models. Achieving accuracy with statistical models requires a more detailed examination of the data, the nature of the work and the distributions involved.

Finally, this example chart shows another curious feature. Close examination of the final few iterations shows that the lines are diverging. The amount of total requested (but not done) work is increasing faster than the total done line. Unfortunately this situation is all to common.

Conclusion

All the charts shown here serve only to track the work and highlight good and bad news. The chart itself cannot change anything. When charts start to show problems in the system - like divergent lines - then it is unto the chart readers, and the team, to take action to rectify the problems.

Cumulative flow diagrams are less intuitive than burn down charts but they convey a lot more information. They are less susceptible to the problems that burn down charts have - although not totally immune.

However these advantages come at a price: they require the reader to spend time and effort understanding what they are seeing. Once one really starts to understand CFDs other opportunities emerge.

One problem that exists with both types of charts concerns the units to use in drawing the charts: hours, story points, ideal hours, abstract points, cards or some other unit. Time, whether in hours or days, is the obvious unit to use but is unsuitable for several reasons. In fact almost any unit except time may be used but unfortunately because so many organizations and individuals are accustomed to managing work through hours not using hours can be hard for some people to accept.

References

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

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

Kelly, A. 2011. "Estimation and Retrospective Estimation." http://www.allankelly.net/static/writing/webonly/EstimationAndRetrospecitveEstimation.pdf.

———. 2013. "More research on estimation." http://www.allankelly.net/static/writing/webonly/MoreEstimationResearch.pdf.

About the author

Allan Kelly is Chief Consultant at Software Strategy where he helps teams adopt and deepen Agile practices, advises companies on development in general and writes far too much. He specialises in working with software product companies and aligning products and processes with company strategy.

He is the author of three books: "Xanpan - team centric Agile Software Development", "Business Patterns for Software Developers" and "Changing Software Development: Learning to be Agile"; the originator of Retrospective Dialogue Sheets, a regular conference speaker and frequent contributor to journals. He can be found on Twitter as @allankellynet.