The Slacker's Guide to Project Tracking

or spending time on more important things…

James Davison, Tim Mackinnon, Michael Royle

ThoughtWorks UK
Berkshire House, 168-173 High Holborn
London WC1V 7AA

{jdavison, tmackinn, mlroyle} @thoughtworks.com

Abstract

As a Project Manager, your time is far too important to be wasted on mundane tasks like detailed tracking of the day-to-day activities of each of your developers. Wouldn't it be nice if you spent your time negotiating project scope and identifying and removing team impediments? Our experience has shown that consistency in card sizes and estimates allows you to perform full project planning with little effort. Additionally, it results in diagrams that accurately reflect your project's status. With this, release planning sessions take hours not days, freeing up valuable time for both you and your developers.

1. Introduction

While it's easy to convince Project Managers that some form of project tracking is important, what is difficult is to get the right degree of tracking that adequately helps them make informed decisions. This paper is based on our work with a large organisation that had a very formal process for project selection, definition and execution. However, one of their software delivery departments was open to suggestions about how to make their process more agile and better able to deliver working solutions on time and on budget. By working with several of their development teams, all of whom had little agile experience, we began to discover basic techniques for making project tracking both simple and extremely effective. In turn, these teams have developed a great track record for delivery that our tracking techniques have both facilitated and made more visible within the organisation.

We found it made sense to break down the tracking process into three areas: iteration planning, progress tracking, and release planning. Each of these areas presents specific information that helps organize and understand project status by tracking at different levels of granularity. For instance, release planning deals with the entire release at a high level by considering a unit of measure of weeks. Thus after a release planning session we can't say that we will be finished at precisely 12 p.m. on July 15, but we can say that the current scope seems to be a reasonable fit for 3 months of work. This is then supplemented by other tracking techniques which look at a finer level of detail and help confirm or contradict the initial release plan. It is important to remember that this is not a science because this process is based on estimates. Estimates are inherently inaccurate but with continued tracking and attention these risks can be mitigated, as we will show in this article.

2. Why Track Progress?

When we began looking at what tracking would be necessary for a team, we started with first principles and examined who the audiences were for the results of our tracking data. This enabled us to tailor the tracking process to meet these needs without doing unnecessary work.

In a large organisation, such as the one we were working with, one of the main stakeholders for any project is business management. For this group, development of software is an investment, and they contribute time and money to make projects happen. This also means that their reputations are on the line for delivering the value from the software, and like any investor, they want to know how that investment is proceeding. Most of the time these stakeholders will not have in-depth knowledge of how the software development process works, so they want information presented to them in such a way that they can easily digest it and can decide if there are any actions they need to take to help ensure the delivery of the solution.

Along with business management, the users are another important group that have significant interest in the progress of the project. In larger organisations, a nominated person or smaller group of people represents the entire user group on the team. This person (or smaller group) is not always technical and so they need an easy way of understanding progress in such a way that they can simply and accurately communicate it back to their peers.

Alongside management and the users, the Project Manager is accountable for the delivery of the project. Having accurate information that can be used to quickly and easily discuss progress with all of these groups is extremely important. Furthermore the Project Manager

needs to be able to interpret this information so that any corrective adjustments can be made to ensure the success of the project. For example, if progress is slow, it might be possible to hire more developers or remove barriers that are impeding progress. If there are indications of creeping scope, there needs to be a discussion with the stakeholders as early as possible to show how this will impact delivery.

Finally the development team itself needs to have an understanding of how well they are doing both for pacing and morale purposes. This can be vital to the success of a project. If the project is behind schedule then the team can adjust its approach or make any necessary changes that will help mitigate the risk of the project not delivering.

3. Iteration Planning

One of the first teams we encountered in the organisation was a collocated team of about 13 people. They had been waiting for final project approval to proceed with the execute phase of the project. They had been spending time spiking [2] different user interface libraries and persistence mechanisms in Microsoft .Net [1].

The time spent, while useful, was rather unstructured and wasn't being measured in a way that could be used for predicative purposes. Borrowing from the practices of eXtreme Programming (XP) [2], we immediately held a "Planning Game" and planned for "Short Iterations" of 1 week. We also began implementing and tightening up the other XP practices, however that work is outside the scope of this paper.

3.1 First Steps

As there was already a proposed project plan in place due to the organisation's gated acceptance process, we chose to concentrate on getting repeatable development iterations working. We also concentrated on measuring a development velocity [3], which would give an indication of how much work the team could achieve in a new development technology (.Net).

We held a planning game based on a velocity from a different team and selected some initial stories that dealt with persistence and reference data administration.

3rd Party Setups (#5.1)	1.00	100%
Sort on product (#53.16)	1.00	0%
Group by Country (#93.1)	0.25	100%
Transfer Simple Product (#62.5)	0.25	100%
Transfer Complex Product (#66.2)	2.00	100%
Totalling (#53.20)	0.25	100%
Country Product Setup (#16.2)	0.25	100%
Velocity Total	4.00	

Figure 1 – Iteration 3 results

Our planning process was similar to that described in [2], whereby cards were estimated in ideal days, and we tracked the total of how many cards were completed in an iteration. We also adopted the velocity simplification described in [3] and used a fixed iteration length that avoided using slightly more complicated "load factor" arithmetic. As an example, by the end of the third iteration, the measured results looked like Figure 1. The developers had finished 6 story cards, giving a velocity total of 4.

In Figure 1, notice how the team didn't finish card #53.16 which is why the completed total adds up to 4 and not 5. Therefore in the next iteration, using the concept of "yesterday's weather" [2], the team signed up for another 4 units of work. The potential stories for the next iteration were laid out on a table and the team collectively discussed new estimates based on their previous experience. Note, we differed slightly from the technique presented in [2], and used team estimates. These were much simpler than having individual developers track personal velocities and also helped with motivation by avoiding any blame culture related to not finishing cards. By using this set-up, planning games were quite simple although they did require someone in the role of Iteration Manager (IM), which will be discussed later in more detail, to facilitate team discussions to keep them focused. We found that in these meetings, the best strategy for the IM was to periodically ask the team "Do you have enough information to put an estimate on that?" This reinforces the message that not all decisions have to be made collectively with the team, just that enough common strategy needs to be agreed and recorded on task cards for a later pair to pick up and work with.

Once the estimates were in place, the users were then able to select from the estimated stories up to a total of 4 as shown in Figure 2. Notice card #53.16 was given highest priority since it had "hungover" from the previous iteration.

Sort on product (#53.16)	1.00
Product Setup (#62.2)	0.25
Transfer New Product (#66.3)	0.50
Database Qualifiers	0.50
Transfer Multiples (#62.3.5)	0.50
Stock Counts (#62.4)	0.50
Warehouse Isle Setup (#62.1)	0.50
Display Type of Trade (#89.1)	0.25
Total	4.00

Figure 2 - Iteration 4

3.2 Steady Iterations

As we continued planning 1-week iterations with the team, we found that they were very effective at giving quick feedback on story progress. More importantly, were much faster to plan due to their small size. This is an important point as many developers dislike planning

meetings, and so the smaller duration makes them much more acceptable. We found that the users, who were also initially sceptical, also began to like them because any cards that were deferred due to velocity constraints were available for reconsideration in a short space of time.

Once an iteration had been created, the development team were encouraged by the Iteration Manager to take any cards larger than half a day and split them into meaningful development tasks to ensure that measurable progress could be made. We also noticed that stories that were larger than 2 days tended to run into difficulty and so we typically suggested that these stories be split into smaller but still useful chunks that could be more easily completed and tracked. While we didn't strictly enforce this, we found that the users noticed this trend as well, and so they started getting better at writing smaller stories of their own accord.

As the development iterations occurred every week, we also found that physically moving the cards to a meeting room was laborious. To overcome this, we used some simple story card wallets[1] which we created by duct taping CD protective wallets together and hanging them on a white board using bull clips (Figure 3). To ensure that we had smooth running planning games we also stuck helpful tips on the back of the wallets to indicate placeholders for Story Card format (title, text, author etc.), Acceptance tests (Action, Result), Pair History and Tasks 1 to 5.

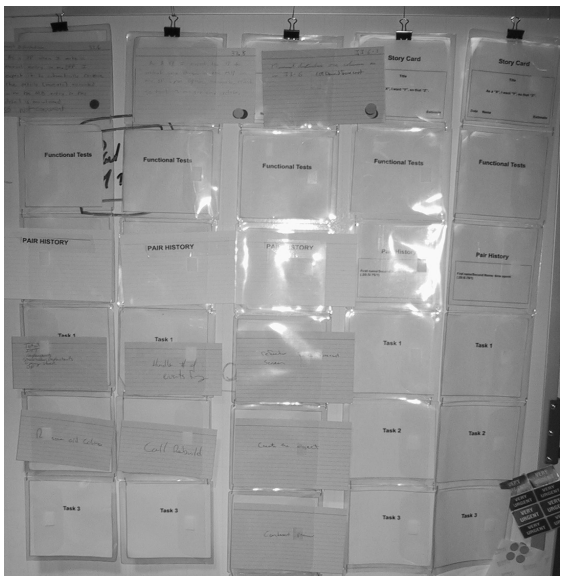

Figure 3 - Iteration Planning Wallets

In addition to the logistical advantages, it also enabled visibility of the process. Often when cards are just placed on a board by themselves, it is difficult to work out the relationships between them. The card wallets allowed us to group relevant cards and communicate their relationship in an easy and effective manor. We have even noticed that another team using the wallets has decided to put an "End" marker after that last task so that they can see if there are any missing cards.

As we continued to complete iterations we found that our users felt a bit divorced from the development process. They often noticed that cards were being considered complete when in fact they had known problems. In an attempt to increase the visibility of the card status, we instituted a coloured sticker scheme[2] to indicate the status of cards as shown in Figure 4.

Originally we only had 3 states for the cards: Not Started (Red), Developer Complete (Yellow), and User Accepted (Green). We gave the green stickers to the users and the red and yellow stickers to the Iteration Manager. However we found that sometimes, even though a card was a place holder for a conversation, that conversation was not always happening. To overcome this, we added an extra state, Story Discussed with User (Blue) that must always happen before a story is Developer Complete.

A nice side effect of the colored stickers was that they had the effect of showing iteration status in a quick glance. As more stories were accepted, the iteration board would slowly turn green. In our experience, without this it was often difficult to determine the state of each card played in an iteration. Usually, this was because this information resided entirely within the Project Manager's head, or worse, spread out amongst the team members.

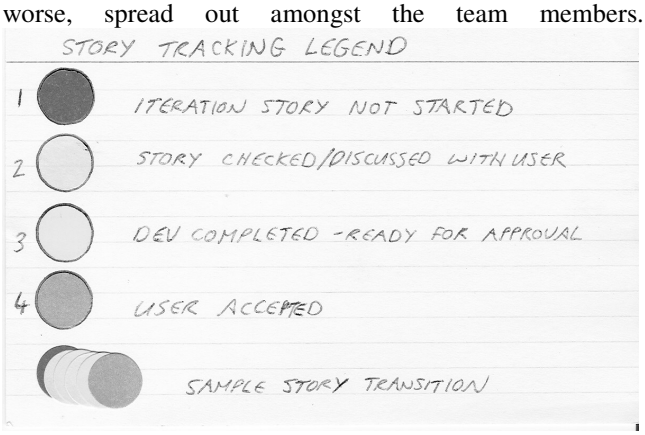

Figure 4 - Story Card Transitions

3.3 Later Refinements

Once the team was used to working on weekly iterations we noticed that we didn't always have as many pairs working as we thought. This was typically because some project members were required to help other teams or were on a short holiday. To easily account for this without getting too detailed, we decided to simply count the number of pairs available at the stand-up meeting and use the weekly average for the following iteration. This simple solution worked very well and also emphasised everyone's commitment to the project. At the beginning of each

[1] The original wallet idea came from Connextra

[2] An idea we also saw used at Connextra

planning meeting we reviewed our previous velocity and then discussed the number of pairs available to determine our velocity for the next iteration as shown in Figure 5.Although the table shows 2 decimal points, the Iteration Manager would normally round to the nearest quarter day.

Velocity To Use		
Pairs	Pessimistic Velocity	Optimistic Velocity
1	1.06	1.29
2	2.13	2.58
3	3.19	3.86
4	4.25	5.15
5	5.31	6.44

Figure 5 - Velocity Matrix

Another issue that we encountered when measuring a pure velocity was referred to as the "Friday Afternoon" syndrome. On the last day of the iteration, if there was only 1 large card left to work on that was obviously not possible to complete it in the time available, one developer came to the conclusion that he might as well do nothing and "go to the pub". This was especially the case if the work was not going to count in some way towards the velocity (admittedly this developer had a tendency to look for excuses to "go to the pub").

As the team felt strongly about this, we decided that partially completed work should be visible when measuring velocity. Rather than losing the benefits of "yesterday's weather", we introduced a second "optimistic" velocity that also counted a percentage of ideal time that had been measured on incomplete stories. In reality these percentages were very coarse grained and were normally one of 25%, 50%, 90% and 99%. At the end of each iteration the IM queried developers for the percentage on any incomplete stories, and this was added to the normal "pessimistic" velocity to form a range on the velocity matrix as shown in Figure 5. This was not meant to be used as an excuse to not finish whole cards, but it did mean that in a planning game the team could guarantee to finish the pessimistic velocity but could stretch to finish the optimistic target. This range of velocities enabled the IM to use their judgement on how much work should be attempted. Sometimes there was evidence that the optimistic target was uncharacteristic and so the team reverted to just using the pessimistic velocity. At other times the team was unlucky and they wanted to stretch to try and achieve a more optimistic figure. This helps address the complaint that strict adherence to "yesterday's weather" removed any judgement from the planning process. The IM (as well the team and customers) all have the necessary information needed to make an informed decision, and set the correct expectations in an easy to maintain manner.

This technique does come with a health warning. Teams that consistently have a wide divergence between pessimistic and optimistic velocities are exhibiting a project smell of not properly completing cards. However, if used sensitively, a range helps gain team support for using an appropriately measured velocity. We also found that our users were very supportive of this technique as they could see that attempts were being made to actively make improvements.

4. Progress Tracking

While significant progress was being made on each iteration, and we had a functionally running application to demonstrate, our steering group continuously asked about project progress. As we were using XP as a development methodology, we were able to easily tell them the number of stories finished, the number of stories remaining, the number of bugs fixed as well as other typical metrics like man days used. In practice, they could never really grasp how this related to the success of the project. These numbers were alien to them and so the Project Manager decided that we needed something that would capture their interest as well as convey an accurate picture of the project.

4.1 The PM's Time Constraints

While it's easy to promise timely information on project status, the reality was that consolidating and massaging this data into a presentable form could easily use up half a day. Our Project Manager was keen to take an agile and pragmatic approach to this, partly because he had a lot of other work to do.

In fact it's very easy to lose sight of the full extent of the Project Manager's job and become too focused on simply collecting project metrics. In the case of this particular team, spending time with both customers and project sponsors was an important aspect of the job. As with any relationship, open communication needs to be nurtured, especially as previous projects for this particular user group had failed. To make things even more challenging, the customer base was geographically diverse and so couldn't be meet all at once.

As well as meetings, fiscal reporting was also another important activity that took a significant amount of time. This tracking is detailed and must be accurate as it can affect departmental performance; therefore, it required adequate time to get right.

We also noticed that "other" activities were not unique to this particular team. When we examined the results from retrospectives [4] held by four other similar teams at the end of their release deliveries (roughly every 2 months), the Project Manager's were all faced with other common items that "didn't go so well" and needed to be addressed:

- Team communication
- Getting customer feedback
- Lack of appropriate documentation
- Difficulty of achieving adequate testing

We were concerned that we didn't want a new way of reporting to divert the Project Manager from these other more important responsibilities.

4.2 First Steps

The original template for project reports (Figure 6) detailed project performance by reporting Costs, Man Days and Milestone Targets with columns for planned, actual, forecast and variance. While complete, this data seemed complicated and not in keeping with the agile approach being used for development.

Man Days				
Resource Type	Planned days	Actual days	Forecast to End	% Variance
Project Manager	88	21	67	0%
Technical Architect	88	22	66	0%
Business Analysts	141	19	105	(12.1%)
Lead Developer	88	22	66	0%
TOTAL	405	84	304	(4.2%)

Figure 6 - Original reporting format

In designing a new report, the first obvious question to answer was "what data would be easy to obtain but still show progress". As data from several XP iterations was available, the Project Manager tried to show something graphical that gave a sense of progress (Figure 7.). These graphs were intended to show what percentage of stories had been completed and how much time remained. Graphs are particularly useful because they can convey a lot of information that would be difficult to follow in textual form, however we found this first attempt was still difficult to interpret.

What we really wanted the stakeholders to understand was the trend towards an on-time completion. Therefore after some brainstorming, we decided to try a stacked bar graph approach as shown in Figure 8.

The lower segment of each bar indicates the number of stories completed, while the top portion represents the total number of story cards remaining (split into those defined, and those estimated to be defined).

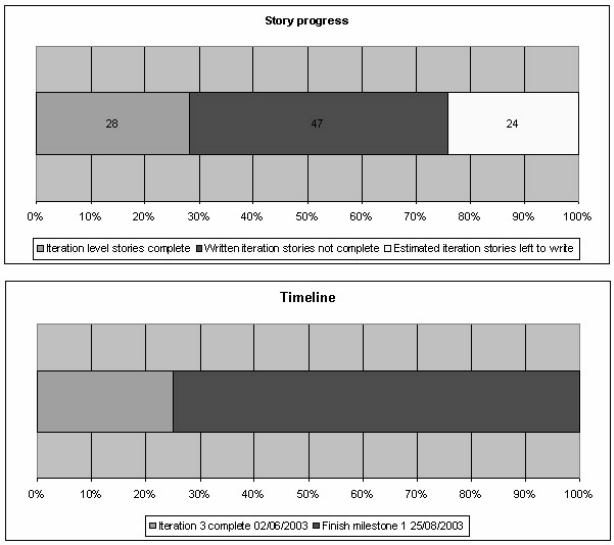

Figure 7 - First reporting attempt

As you can see this is a powerful yet simple technique for presenting this information. The format is similar in concept to a "Profit Graph", where increasing profits and decreasing expenses are viewed as a good indication of success. In our case, the reader's eye will fill in the trend line of completed stories, which is what the Project Manager wanted readers to notice. Additionally, it also shows other information about possible scope creep and other issues that may impact the delivery. Our graphs are opposite to the "Burn Down Charts" [5] used in Scrum, which show a decrease in remaining hours. We felt that showing an increase in completed work was psychologically more pleasing.

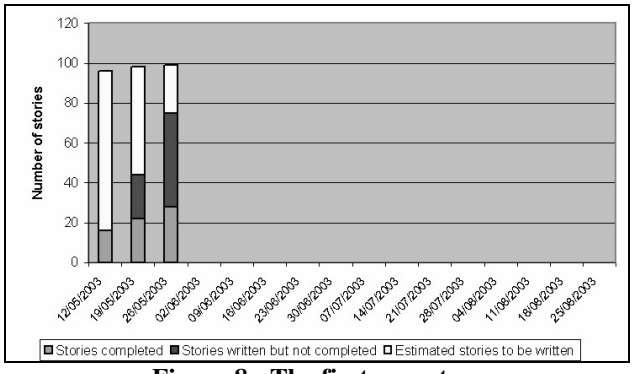

Figure 8 - The first report

In Figure 8, the first iteration bar in this graph shows only "stories completed" and "estimated stories to be written". This reflected that the users had not yet been able to translate their high level requirements into concrete user stories. Based on progress in the first iteration, the Project Manager took the high level functional areas of the system and estimated that these would translate into approximately 90 user stories which he indicated on the graph. With this graph in place it was clearer how much work there was for the users to finish writing the story cards. In fact, as the

users were now seeing progress they became better at writing more stories.

As the project progressed, the next monthly report (Figure 9) continued to show development progress as well as the completion of all of the story cards (show in iteration 7, the last bar on the graph). This indication of story completion removed a large amount of risk from the project.

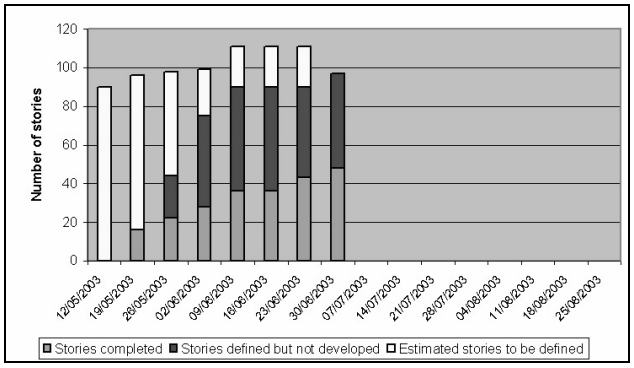

Figure 9 - The second report

At the end of the first project release (Figure 10), you can see that development progressed at a relatively constant pace (shown on the graph as the bottom segment of each bar). However, we can also see that the Project Manager had to manage scope creep very carefully. These scope changes were partially due to:

- The lack of completely defined stories at the beginning of the project
- The inexperience of users, new to writing story cards
- Some stories that were intended for the next release being "accidentally" moved earlier

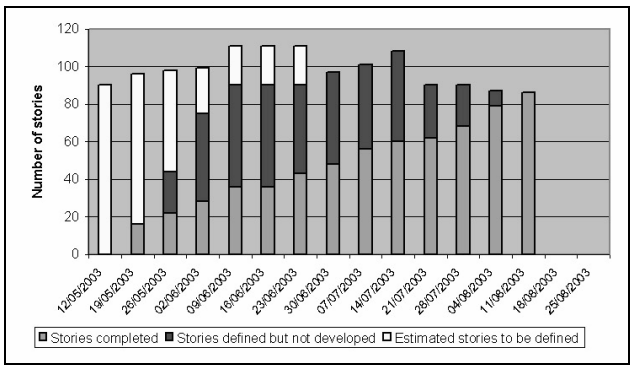

Figure 10 - The final report for release one

For the next release (Figure 11), the situation was quite different as all of the stories were defined up front. In fact in this release it was even possible to add more features than were initially specified. Again it's interesting to notice that the development progress of stories completed increased at a steady rate just as in the previous release.

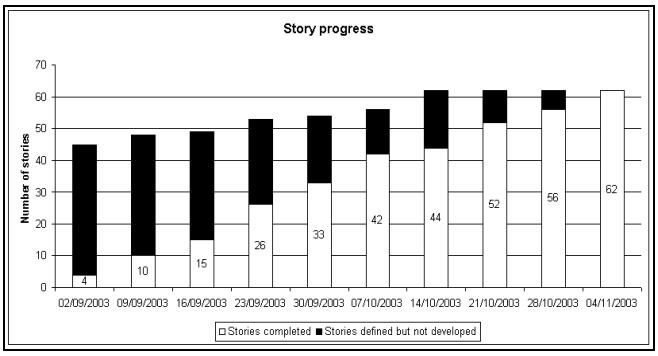

Figure 11 - Progress in the second release

4.3 More Detailed Tracking

As good as this tracking was, the Iteration Manager was worried about the inaccuracy of this method of reporting progress. The problem was that not every story was of equal size, and he feared that by treating them as if they were, the reported progress would be skewed. Instead, his idea was to plot the number of completed ideal days as a function of the iteration. Figure 12 shows an example of this graph. Again this shows a similar overall trend of increasing development as well as how many ideal days of work were left (the line).

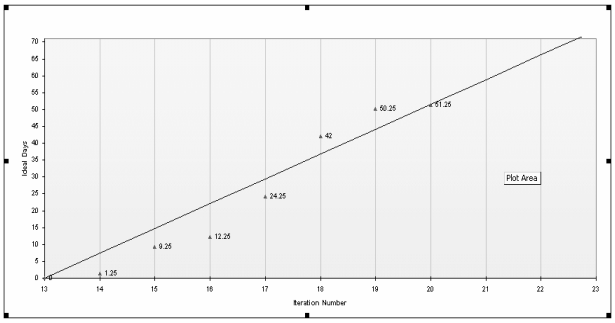

Figure 12 - A more detailed measure of story progress

Unfortunately, as we progressed through several iterations, this graph became more problematic. The difficulty was in dealing with story cards that spanned multiple iterations. For example, if card #x was initially started in iteration 3 with an estimate of 1 ideal day, it required additional effort to track the history of the card if it wasn't completed. If the card was finished in the next iteration, we had to take the original estimate and add any additional estimated time and put this on the graph. While not terribly difficult, the additional tracking overhead began to add extra time to creating the graph. We also noticed that this method of tracking didn't easily lend itself to showing scope creep.

About halfway through the release, we decided to compare results from the two graphing techniques to see if either of them was better able to predict an end date for the project. We were initially surprised to find that both methods predicted the same end date. At the same time we were quite relieved as the story count method of tracking

was much easier to maintain and so we decided to abandon the second tracking technique.

After considering the results of our experiment, the explanation we came up with was two-fold. The first centred on our philosophy for estimation consistency over accuracy. Estimates by definition are fuzzy and not particularly accurate. Therefore, how do we use them to accurately predict how long it will take to develop an application? There are two schools of thought on this. The first and most often used is to keep track of the actual time taken to finish something and then use this actual as the basis for the next set of estimates. For instance, let's say we wrote a personal contact screen and the estimate was one day but it actually took two days. When the card for a screen for business contacts is being estimated it would be given an estimate of 2 days as it is similar in size to the personal contacts screen. The biggest advantage of this method is that it incorporates past experience when producing new estimates. This means that over time the difference between the estimated and actual time will tend towards zero. However, it also means that you need to accurately keep track of the actual time taken for each card and present that information in such a way that the team can quickly recall the amount of time taken for any one card.

Because of the extra effort involved in the first approach, we chose to use the second method, which is to keep estimates consistent. To use the example from above, rather than giving the business contacts screen a two day estimate we would give it a one day estimate. The reason this works is because if the estimates are consistent then we should have a nearly constant load factor [2] which can then be used for any estimates to determine relatively accurately how long they will take. This simplifies the tracking process because we don't need to record the actual time taken for each card. Additionally, we found that we didn't even need to record the original estimate because we used a simple guideline for estimation. For any estimate, we asked that it should be based on the amount of time the developers thought it would take to "Hack" the solution. Therefore, we didn't include testing, refactoring [6], or non-development time in estimates and simply tracked all of these within our development velocity. Thus the time taken to "spike" a possible solution can be used as an ideal estimate, as re-implementing a production worthy version incorporates refactoring and proper unit testing in the measured velocity.

The second factor in the success of the simple tracking model was our tendency to keep cards to a manageable size. Our goal for every card was to keep it between 0.25 and 2 ideal days. As we estimated cards, if we found cards larger than 2 ideal days we generally broke them down into meaningful pieces that could be easily completed. As previously mentioned, while this wasn't strongly enforced we found that users began to notice that smaller cards were much easier for them to test and accept as finished. Thus they also began to write cards of this size.

The combination of these two simple factors: estimation consistency and small card size, created an environment where simply counting cards was enough to track progress.

4.4 Exciting Observations

After the introduction of the new graphs, we began to notice that at steering group meetings, the attendees would immediately turn to the page with the graph on it to see what progress had been made. Although they were used to a common format for project reports, it was clear that the new graph was conveying to them more than what they had seen in previous reports. Furthermore, because a lot of information was captured in one picture it became a good catalyst for some of the best project discussions.

The last refinement we made to the graph was adding a predictive aspect to it. This was just a simple average weighted by the anticipated number of pairs available. Figure 13 shows a graph with these projections added. You can see that from the seventh iteration onwards, the bars are a different colour. This is the predictive section of the graph. It shows the predicted number of cards completed based on a weighted average and the anticipated total number of cards. In this example, the predicted total number of cards remains constant. We tried to be more accurate by predicting how many cards would be added and split-up. However, at that point we decided that the extra information would only confuse the matter and make the results more ambiguous.

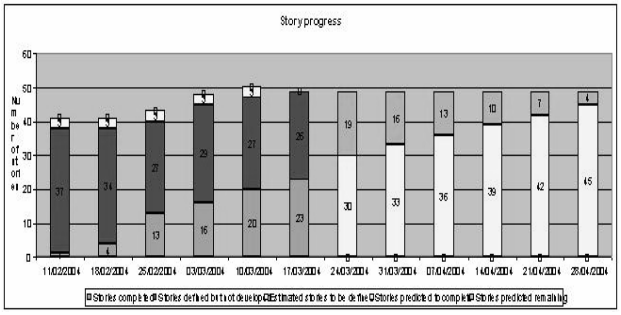

Figure 13 - Progress tracking with predictions

One downside of the predictions is that it adds a level of precision that in reality isn't there. For instance, at one steering group meeting about halfway though a release we had to fend off questions about why it was predicting that there would be two cards remaining that couldn't be finished. As with most things, people will believe numbers even if they know they are just a guess or estimate. Therefore, you should try to judge your audience and their propensity for believing numbers before showing them a simplistic predictive model.

In retrospect, there is a lot to be said for not even using a predicative model and simply showing current progress.

With a single end bar that shows the expected number of completed stories, you would then rely on the reader to imagine the trend curve themselves. When you have only completed 2 or 3 iterations, the line they imagine will be (in their minds) fairly inaccurate as they understand that they don't have enough data points. As you get more iterations, the visualisation of a trend line becomes much easier and it becomes much more obvious if you are on track or not. We haven't tried reverting to this model but it would definitely be in line with our tendency to question the appearance of too much accuracy and look for simplifications wherever possible.

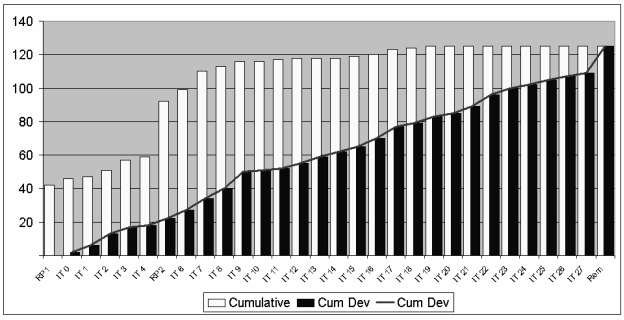

Figure 14 - Another progress tracking example

Finally, after seeing the success of our progress tracking, other teams within our department have begun using similar methods. Figure 14 is a diagram used by another team where the bars represent the same information but they have added a trend line to aid visualisation.

An interesting point about this graph is that it is obvious when the scope changed. In iteration 5, the users asked the team to deliver more functionality. This information could be presented to a steering group to show why the team may not make the deadline. In this case, they were able to get additional time and budget to cover the new scope.

5. Release Planning

The organization we were working for had insisted that no project would be approved unless it had a "defined" project plan. Thus for our project, there was already a high level plan that had been created by the Project Manager and the Technical Architect. While this plan was mainly methodology neutral, it was a more traditional plan with pieces of work shown in a Gant chart. As development was progressing in an agile, needs driven basis, we saw that the reality of the emerging iterative solution was diverging from the original plan. Therefore, we couldn't continue to rely upon it to adequately predict if any changes would endanger the team hitting the required schedule.

5.1 Reluctance

The lack of a clear plan that truly reflected the stories of work that needed to be completed made us feel rather uncomfortable. Although we didn't like moving forward with development without an adequate release plan, our fears were tempered by the desire of the team to start writing code. The thought of taking several days to wade through cards and get high level estimates was met with a large amount of resistance from the entire team. As previously mentioned, we felt that it was in the best interest of the team to actually get some real experience delivering some software. At the same time, we kept looking for opportunities to fill in a release plan.

5.2 A Simple Strategy

The breakthrough for simplified release planning occurred almost half way into the first milestone of the project (after about 6 iterations). During an iteration planning meeting, the planning had gone quickly but we were still a bit uneasy about the breadth of the project that was unfolding (a smell that a release plan was missing). As we had a little more time available in the meeting room, we suggested that our customers read out some of the higher level stories that remained. We felt that this would give us an idea about what they intended to get done for the first official release of the product.

As an experiment, one of the authors began to write on a card, "gut feel" estimates for the amount of time required to finish those high level items. After a few minutes he realised that this was a useful technique for everyone to try and so he gave all the developers a card and asked them to try the following:

- On a card write down a high level estimate for each story card
- Keep your estimate to yourself
- Try and keep conversation to a minimum, limiting it to clarification of story details or technical questions

We then proceeded to record an estimate for each requirement that was described. Sometimes we needed a bit more detail from the user, like "how many reports would need to be produced" or "what kind of response time was required". Occasionally someone would ask a technical question like "does the current database technology provide support for offline replication". Often some of the questions that were asked caused people to go back and scribble out an estimate and increase it, or sometimes even decrease it. In cases where conversation was dwelling on a decision about a particular implementations we just asked everyone to make their estimate reflect their uncertainty. After about an hour we had covered all of the stories. The results looked similar to the cards shown in Figure 15, although these cards are a later example of the technique where we asked developers to give high and low estimates.

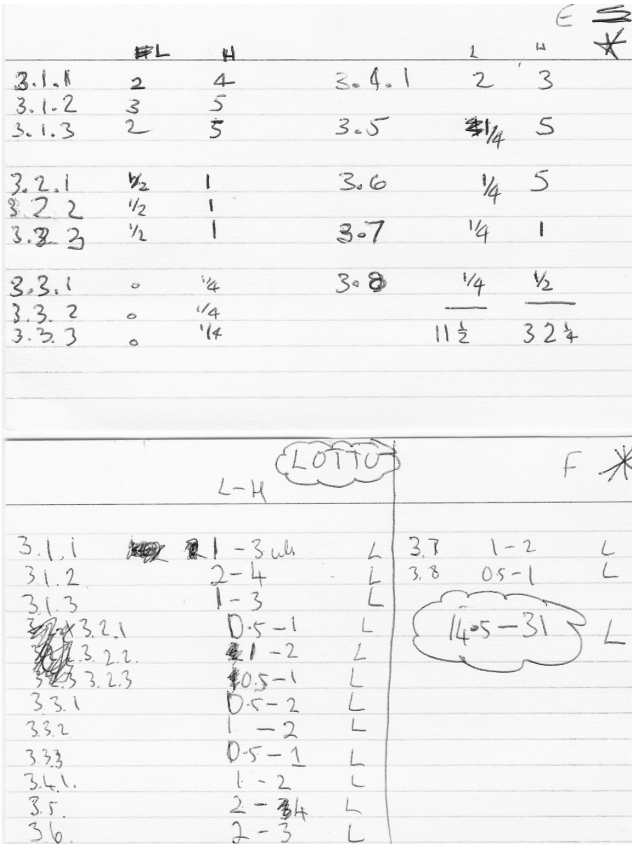

Figure 15 - Release Planning Estimates

Once we had completed estimates for each of the stories, we then went around the room and asked each developer to read out their estimate for each high level story, which we recorded on a flip chart. This proved to be quite entertaining as we quickly saw who was optimistic in the team and who had more knowledge of a particular area. We then simply averaged the estimates to get a total for the estimated time remaining.

This grand total was then divided by the development velocity (that we had been measuring in our iterations) to get an indication of how many weeks would be required to complete the first release. We also did the division with our "optimistic" velocity to get a best and worst case scenario. The good news was that our expected completion date fell roughly between our best and worst case estimates and so we felt that at least the first release was attainable.

When we left the planning room, there was a sense of relief from the entire team, as the project now felt like it was achievable. Furthermore, the feeling that release planning was going to be a monster, had now been dispelled.

5.3 Refinements

We have now repeated this process of release planning on several releases and with several teams, and have been happy with the outcome.

Time and time again we have to remind ourselves that planning is not an exact science – we are making judgements based on people's estimates, and while estimates can be fairly accurate, when you add them together they do not give you an exact answer. However, we have been pleased to find that overall our results have been accurate enough to properly meet our deadlines.

We have also noticed that in these release planning sessions, developers can get concerned about details that ultimately don't appear to affect the outcome of their estimates. For this reason we introduced high-low estimate boundaries more as a way to help make estimation more efficient so that they could express "either/or" decisions. For example in Figure 15, you can see ranges of 3 to 5 weeks in some cases.

Finally, we have also used this technique further up the project chain in the organisation. During the "Select" or "Define" phases we can quickly give high level estimates for different project options that can be used when making business cases for potential projects. Not only does this help the strategy teams put meaningful estimates on the projects for governance board selection, it also allows the development teams to feel more involved in the potential projects that might come through the pipeline for eventual development.

6. The Role of an Iteration Manager

Since the premise of this paper is enabling Project Managers to spend more time on the more important aspects of their job, a brief description of the Iteration Manager role is useful.

The Iteration Manager is a role to which the Project Manager can delegate most of the inward facing team responsibilities. You can think of these roles as two sides of the same coin, one facing outward (Project Manager) and one facing inward (Iteration Manager). This means that the Iteration Manager becomes the team tracker [2], communication enabler, and potentially overall team leader.

Therefore, the Iteration Manager should have many of the same skills you would look for in a Project Manager, such as leadership, understanding of team dynamics and motivation, as well as the ability to make the tough decisions. However, Iteration Managers can also add additional value to the team rather than just being an additional Project Manager. For instance, the Iteration Managers on our project have been from developer backgrounds, and in fact, they spent most of their time as developers on the project in addition to their responsibilities of Iteration Management. This was beneficial for several reasons: it freed up the Project Manager for more important duties, being an active member of the team gave them the understanding of the technical and business aspects necessary to perform the job, and it didn't add any additional levels of pure management that can often slow down a project.

7. Conclusion

We all know that Project Managers often need to divert their attention from the more important aspects of their work to focus on the useful but more mundane tracking tasks. We've presented a strategy for allowing managers to once again spend the necessary time on building and maintaining customer and business management relationships that ensure project success. The main idea is to keep tracking simple by figuring out exactly what information is necessary for all interested parties and only focusing on these items. Additionally, splitting out the inward facing responsibilities to another team member can free up more time.

If we had it to do over again, we would create the high level release plan even if there were some unknowns. This is especially true since we know they are not time consuming to perform, therefore any changes can be accommodated quickly. What is important is to establish a pattern of successful iterations, which show progress towards completion. As the project progresses, it is also important to be conscious of the team's velocity, keeping in mind the realities of "yesterday's weather" and the pit-falls of optimistically selecting an expected velocity. Lastly, progress tracking should give a simple overview without being too precise with its predictions. Simple graphs that show story completion are very effective.

8. Acknowledgements

We would like to thank the reviewers and the following colleagues for their contributions to this paper: members of the My Supply team (in particular Brent Cryder), Rebecca Parsons, Andy Pols, Laura Waite, The Bishop of Norwich and finally the Golden-T.

9. References

[1] Microsoft online at http://www.microsoft.com/net/basics/, last visited June 2004

[2]Beck, K. Extreme programming explained: embrace change. Reading Mass. Addison-Wesley, 1999

[3] Various authors online at http://www.c2.com/cgi/wiki?VelocityVsLoadFactor, last visited June 2004

[4] Kerth N. Project Retrospectives: A Handbook for Team Reviews, Dorset-House, 2001

[5] Control Chaos online at http://www.controlchaos.com/burndown.htm

[6] Fowler M, Refactoring: Improving the Design of Existing Code, Reading Mass. Addison-Wesley, 1999