AGILE MONITORING USING THE LINE OF BALANCE

April 2009 © Eduardo Miranda, Institute for Software Research – Carnegie-Mellon University,

Pierre Bourque, École de technologie supérieure – Université du Québec

Keywords: Scrum, project management, tracking and control, line of balance, release planning, burn down charts, cumulative flow diagrams, Agile methodologies, LOB

1. ABSTRACT

There is a need to collect, measure, and present progress information in all projects, and Agile projects are no exception. In this article, the authors show how the Line of Balance, a relatively obscure indicator, can be used to gain insights into the progress of projects not provided by burn down charts or cumulative flow diagrams, two of the most common indicators used to track and report progress in agile projects. The authors also propose to replace the original plan-based control point lead-time calculations with dynamic information extracted from a version control system and introduce the concept of the ideal plan to measure progress relative to both, end of iteration milestones and project completion date.

2. INTRODUCTION

Progress monitoring and reporting is a basic function during the execution of any project. Progress needs to be monitored so potential problems can be adverted before they materialize. Besides being required to steer the project, timely and accurate reporting is essential to keep stakeholder support and funds flowing.

With the exception of earned value reporting (Project Management Institute 2004), which is almost mandatory in most large government-sponsored projects (GAO 2007), few, if any, tracking and reporting mechanisms have been standardized. Despite this, certain practices have emerged as preferred within some Agile communities and not in others. For example, burn down charts (Schwaber and Beedle 2004) are favored by the Scrum community, cumulative flow diagrams (Anderson 2004; Microsoft 2006) by Feature Driven Development (FDD) practitioners, and stories completed and tests passed (Wake 2001) by Xp adepts.

While simple to produce and easy to understand, these charts do not communicate the whole picture. Burn down charts, stories completed and tests passed report how much work is left and provide some indication of where the project ought to be, had it progressed at a constant rate. None of them report work in progress. Cumulative flow diagrams on the other hand, report work in progress but fail to relate it to how much should have been accomplished if the project is to meet its commitments.

The reason why it is important to consider work in progress versus any commitments made, is that this information allows the team to focus its resources where they are needed the most. For example, should the work in progress indicators point to a bottleneck in the testing activities the team could redirect its efforts from coding to testing. Reporting work in progress also helps communicate to outside stakeholders that the team is advancing, even if user stories are not being completed daily.

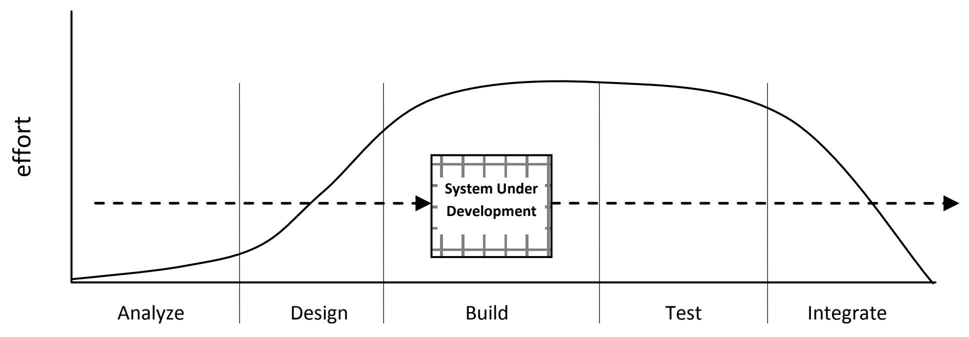
In this paper, we propose the use of the Line of Balance (LOB) (Office of Naval Material 1962) method as an alternative to overcome the deficiencies noted above. The LOB method allows the team to deliver at its maximum speed, by helping balance the different activities in the production chain. In keeping with the idea of using the team's actual performance, we also propose to derive the lead-times for the so called "control points", not from an activity network as in the original LOB formulation, but from the team's "velocity".

To illustrate the concepts presented, we have chosen artifacts and processes from Scrum (Schwaber and Beedle 2004) and FDD (Coad, Lefebvre et al. 1999) as examples. The reader, however, should have no problem extending them to other contexts.

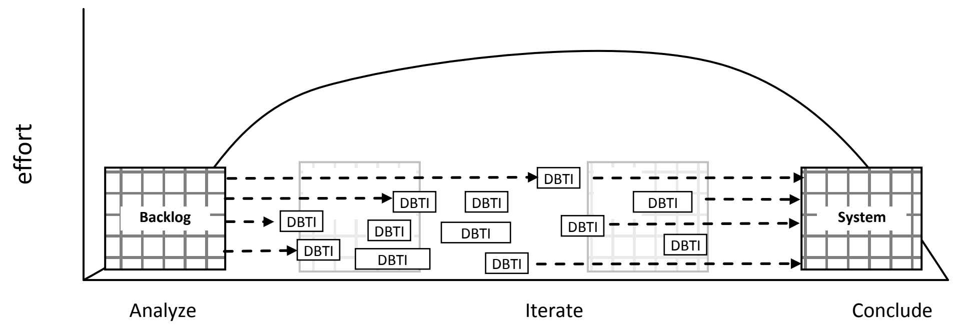
Section 2 discusses current approaches to progress monitoring and reporting in Agile projects. In Section 3 we describe the LOB method and we then pursue to explain how LOB outputs are interpreted, Section 4, and the extension of LOB methods to teams of teams, Section 5, and to portfolio management, Section 6.

3. PROGRESS MONITORING AND REPORTING IN AGILE PROJECTS

Agile methods are characterized by the recurring end-to-end development of discrete software system capabilities. That is, instead of evolving the whole software, or large chunks of it, through the stages of the development life cycle (Figure 1.a), following a brief up-front analysis phase, they break down the total effort into small self-contained pieces of value called user stories or features; and each of them is evolved through the entire life cycle, and with the exception of technical dependencies, mostly independently of the others. The sequence Design, Build, Test, Integrate (DBTI) cycle is repeated over the life of the project as many times as user stories are, generating partial versions of the complete software system along the way (Figure 1.b).



a) Coarse grain development



b) Fine grain development

Figure 1 Software life cycles: a) coarse grain development, and b) fine grain development – lightly shaded squares represent partial versions of the system.

In a project using Scrum, progress will be tracked and reported by means of a release burn down chart, an iteration burn-down chart and a task board (Cohn 2006). The charts are called “burn down” because they show what work remains to be done rather than what work has been completed.

The release burn down chart (Figure 2.a) is used to monitor and report the overall progress of the project to both sponsors and team members. The release burn down chart shows two key indicators: the overall rate of progress and the amount of work remaining. By extrapolating the rate of progress, it is possible to forecast the time of completion. If work is added to the project, the curve is adjusted upwards, if work is dropped it is adjusted downwards.

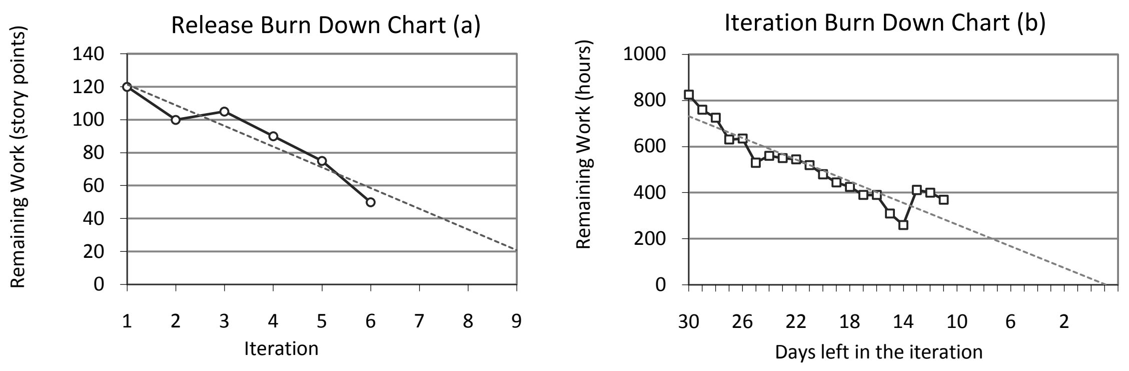


Figure 2 Burn-down charts: a) release chart; b) iteration chart.

Iteration burns down charts (Figure 2.b) are derived from the task board information (Figure 3), and its audience is the team members. The purpose of the chart is to show the number of hours of work left versus the number of days left on the current iteration. The development team uses this information to conclude whether all the work of the iteration can be completed at the current pace or whether a couple of extra hours would be needed or some work would have to be rescheduled.

The task board adds a great deal of information by showing the breakdown of a user story into tasks. Tasks are embodied in task cards. At a minimum, the task cards identify the type of task, e.g. coding, writing test cases, integrating, etc. and the number of hours estimated for its execution. As work progresses, team members move task cards from one state (pending, assigned, in progress, completed) to another. The board provides the means to coordinate work among team members and the raw data, i.e. hours of work left, to produce the iteration burn down chart. Other information on the board, such as how many user stories are being coded or how many are being tested, is not exploited - at least in a structured way, under the assumption that all that counts is work completed and work remaining.

As of 5/28/08		Iteration ends 6/12/08		Work days left: 11	
User story	To Do (1)	Assigned (2)	In Progress (3)	Completed	Hours Left (1+2+3)
User story 1	Integrate	Test	Code	Design	28
User story 2		Integrate	Test	Code Design	8
• • •	•••	•••	•••	•••	298
User story n	Code Design Test Integrate				36
				Total Hours Left	370

Figure 3 Scrum task board showing the status of all the tasks included in the iteration.

The cumulative flow chart (Jackson 1991; Anderson 2004), is constructed by counting the number of user stories that have reached a certain state of development at a given time. Compared to the burn-down chart, the cumulative flow diagram favored by FDD practitioners, offers a wealth of information: rate of flow, quantity in process and time in process (Figure 4). Unlike the line of balance chart, cumulative flow diagrams do not show target information.

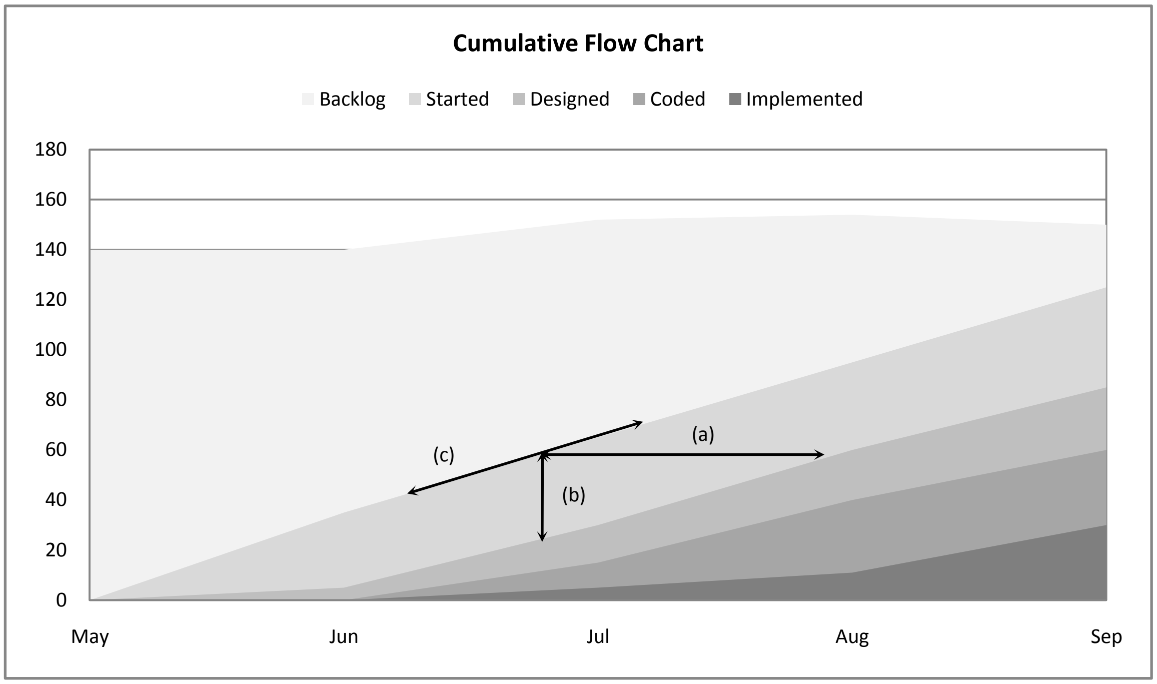


Figure 4 Cumulative Flow Chart the upper series (backlog) shows the total amount of work to be done. The ups and down correspond to work added or dropped respectively. The horizontal line (a) measures the average time in state for each user story. The vertical line (b) reports the number of user stories in a given state at a particular time. The inclined line (c) represents the rate at which user stories reach a particular state.

Of special interest is the use of earned value techniques in agile projects which has been more in response to reporting requirements (Alleman, Henderson et al. 2003; Rusk 2009) than a choice of the development teams and which has required some tweaking or reinterpretation of some fundamental concepts to be applicable (Cabri and Griffiths 2006; Sulaiman, Barton et al. 2006). When used at the project level, earned value is more a reporting mechanism between the project sponsor and the team doing the work than a diagnostic tool. It lacks the visibility required to take any decision. To be used as a diagnostic tool, earned value requires the identification of deliverables and their contributing tasks by means of a WBS or similar arrangement, the definition of their start and end dates, the allocation of a budget, and a time reporting system capable of tracking data at the same level of granularity. Because these conditions are rarely found in agile projects it usefulness is limited to that of a burn down chart with the addition of spending reporting.

4. THE LINE OF BALANCE METHOD

The purpose of the LOB method is to ensure that the many activities of a repetitive production process stay “in balance” that is, they are producing at a pace which allows an even flow of the items produced through a process and at a speed compatible with the goals set forth in a plan. The method does this by calculating how many items should have passed through a given operation or control point, and showing these figures alongside the number that actually did (Al Sarraj 1990; Arditi, Tokdemir et al. 2002) (Figure 5). In the context of Scrum an item would be user story and in the case of FDD, a feature.

The LOB method was devised by the members of a group headed by George E. Fouch during the 1940s to monitor production at the Goodyear Tire & Rubber Company, and it was also successfully applied to the production planning of the huge US Navy mobilization program of World War ll and during the Korean hostilities. Today, the LOB method is applied to a wide spectrum of scheduling activities, including research and development, construction flow planning, and tracking the progress of responses to trouble reports (Miranda 2006; Harroff 2008).

The LOB Status Chart in Figure 5 shows the project progress as of December 28th. The plan for the project is to deliver a total of 123 user stories. This is shown by the control point labeled "Backlog", which shows the total amount of work the team is currently committed to deliver. For the other control points, the chart displays two columns: the "Planned" column showing how many items should have passed through the control point according to the proposed production or release plan and the "Actual" column showing how many did actually pass through it.

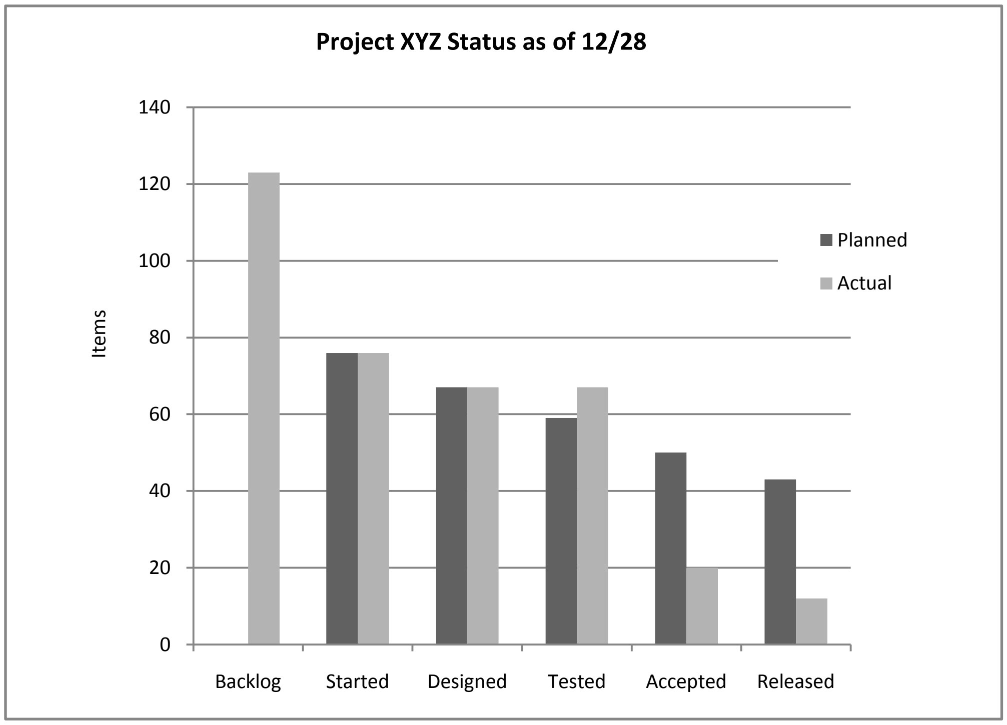


Figure 5 A Line of Balance Status Chart showing the number of items that should have passed through a given control points versus how many actually did.

By comparing the number of planned items to the number of actual items, we can see, for example, that the activities leading to the "Started" and "Designed" control points are on track with respect to the delivery plan, and that the testing activities are a little ahead of schedule. In contrast, the activities leading to the "Accepted" and "Released" control points are behind schedule. According to the plan, there should have been around 50 story points-worth of functionality accepted by this time, but in fact there are only 20, and, since the testing activities are ahead of schedule, the problem must lie with the activities leading to acceptance. The chart does not show the cause of the problem; however it is clear that whatever the reason, the slow pace of the acceptance activities is jeopardizing the next release.

The advantages of the LOB method over burn down charts and cumulative flow diagrams are that the LOB:

- Shows not only what has been achieved, but also what was supposed to be achieved in a single chart;
- Shows work in progress, permitting a more accurate assessment of the project status; and
- Exposes process bottlenecks, allowing the team and the project sponsor to focus on the points causing slippages.

Knowing how many user stories are in a given state allows us to answer the question: Where are we today? But leaves unanswered the more fundamental one of: Where are we in relation to where we planned to be? To answer this question the LOB method identifies (College 2001):

- A number of control points at which progress is to be monitored;
- A delivery plan specifying the number of user stories to be produced in each iteration; and
- A status chart, showing the number of user stories that have passed through each control point vs. the number that should have passed through according to the delivery plan.

4.1. CONTROL POINTS

In LOB terminology, a control point is a point in the development process with a well defined exit criterion at which work in progress or work completed is measured. In the context of tracking user stories, control points would correspond to all or some of the states comprising the user story's life cycle (Figure 6). Knowing the status of the project at any given time requires knowing the state of each and every user story.

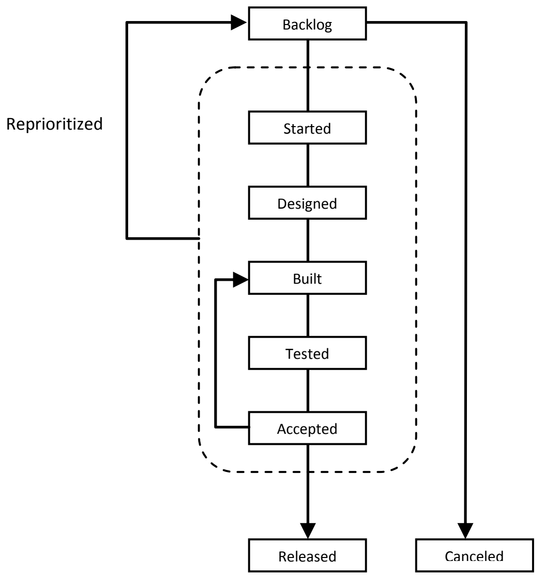


Figure 6 A user story's typical life cycle.

A control point’s lead-time (Figure 7) is the average time it takes a user story to move from that point to the point at which the user story is considered completed. In the original LOB method, these times are derived from an activity network comprising the activities involved in producing a user story while in our proposal they are calculated by measuring the average time a user story spends in each state (Figure 8).

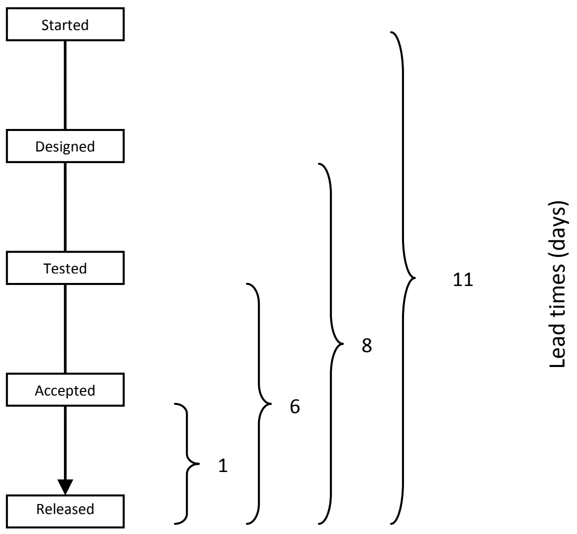
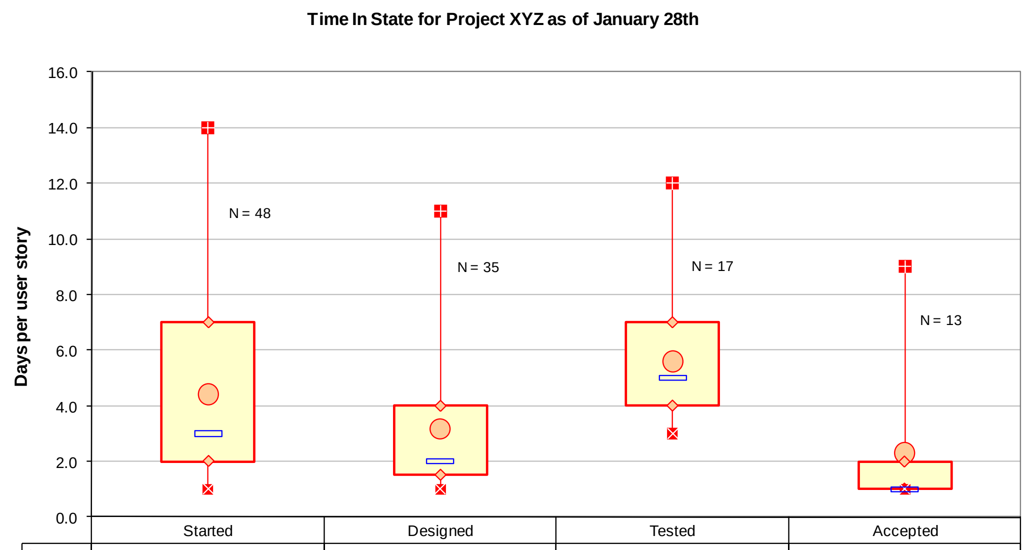


Figure 7 Control points and lead-times. Note that not all the states mentioned in the user story’s life cycle have been included for control. The decision to do this was based on the amount of visibility desired or requested. After the user story is released, it is not longer tracked.



	Started	Designed	Tested	Accepted
◇ 25%	2.00	1.50	4.00	1.00
☒ Min	1.00	1.00	3.00	1.00
○ Mean	4.42	3.17	5.59	2.31
− Median	3.00	2.00	5.00	1.00
⊞ Max	14.00	11.00	12.00	9.00
◇ 75%	7.00	4.00	7.00	2.00

Figure 8 Box-plot chart showing the distribution of times spent in each state by the user stories developed so far. The 25 and 75% quartiles delimit the range within which the middle 50% of the values are included. N denotes the number of user stories included in the statistic.

The time each user story spends in each state is calculated using expression <1>. The average time in a state for a given state could be calculated as either the median, the arithmetic mean or the mode of the individual times in state <2>. The data required by these calculations is readily available from most version control systems.

<1>

$$TimeInState_{qi} = \begin{cases} \text{if } TransitionDate_{q+1\ i} \ exists \text{ then} & TransitionDate_{q+1\ i} - TransitionDate_{qi} \\ \\ \text{otherwise} & CurrentDate - TransitionDate_{qi} \end{cases}$$

$q = 1, 2, ..., n$ is a control point mapped to one of the user story's lifecycle states

$i = 1,2,...,v$ is used to denote an individual user story

<2> $$\overline{TimeInState_q} = \begin{cases} Median\ \ TimeInState_{q1,} TimeInState_{q2,} ..., TimeInState_{qv,} \\ or \\ ArithmeticMean\ \ TimeInState_{q1,} TimeInState_{q2,} ..., TimeInState_{qv,} \\ or \\ Mode\ \ TimeInState_{q1,} TimeInState_{q2,} ..., TimeInState_{qv,} \end{cases}$$

The median is preferred over the arithmetic mean to prevent rare but very complex, or very simple, user stories from skewing the value of the statistic. This can be observed, for example, in Figure 8 by noting that the arithmetic mean for “Accepted” lies outside its interquartile range, meaning that, while most user stories were accepted in one day, a few of them took longer, driving its value up to 2.31 days. The use of this value in the calculations instead of the far more common one-day median, will inflate the number of user stories reported in the planned column of the “Accepted” control point signaling that a higher number of them should passed through it than they actually needed to.

Moving from times in a state to lead-times <3> is straightforward. Since the mean time spent in the “Accepted” state (Figure 7) is the typical time it takes a user story to go from the “Accepted” to the “Released” control point, the lead-time for “Accepted” is 1 day. In the case of the lead-time for the “Tested” control point, a user story typically spends 5 days in the “Tested” state and then 1 more in the “Accepted” state. This is equivalent to saying that it takes 5 days to move from “Tested” to “Accepted” plus 1 day in moving from “Accepted” to “Released”. Consequently, the lead-time for the “Tested” control point is 6 days. Table 1 shows the lead-time calculations for the control points in Figure 6.

<3>

$$
\begin{aligned}
&LeadTime_n = 0 \\
&LeadTime_{q \in n-1, n-2, \ldots, 1} = LeadTime_{q+1} + \overline{TimeInState_q}
\end{aligned}
$$

Table 1 Lead-time calculations.

Control Point	Time in a State (days)	Lead Time (days)
Released		0
Accepted	1	0 + 1 = 1
Tested	5	1 + 5 = 6
Designed	2	6 + 2 = 8
Started	3	8 + 3 = 11

Although different user stories will require different efforts to implement them, we will treat them as being of equal size, as most teams strive to achieve this goal when breaking down the total effort, and that it is very unlikely that in planning an iteration, a team will include all the large stories in one and all the small ones in another. Should the assumption of equal size prove inappropriate, user stories should be normalized, i.e. reduced to a common denominator, using story points, function points, man-hours, or any other measure that accounts for the relative development effort required by them.

4.2. THE DELIVERY PLAN

The delivery plan (Figure 9) comprises two sub plans, the Release plan (RP) and the Ideal Plan (IP). The RP specifies how much capability will be delivered at the end of each iteration, as agreed between the team and the project sponsor while the IP shows the proportion of capability that should have been delivered at any given time, assuming constant work progress throughout the project.

The RP is prepared by the project team based on its own estimation of productivity, the team's velocity and their resource availability. The plan can be adjusted later, with the agreement of the sponsor, to reflect the team's actual performance and changes to the product backlog. The RP in Figure 9 shows that the software system to be delivered consists of 150 user stories that must be delivered by March. Based on their previous experience and business needs, the team breaks down the total delivery into five releases. The first release, due by the beginning of November, consists of 25 user stories; the second release, due by the beginning of December, includes 35 user stories, the increase in velocity accounts for the learning effects, i.e. the team becoming more proficient with the application. In December, because of the holiday period, the team's availability is reduced and the team only commits to the delivery of 15 user stories. For the last two months, the team expects to bring some additional experienced resources on board, which will help them increase the delivery rate to 35 and 40 user stories respectively.

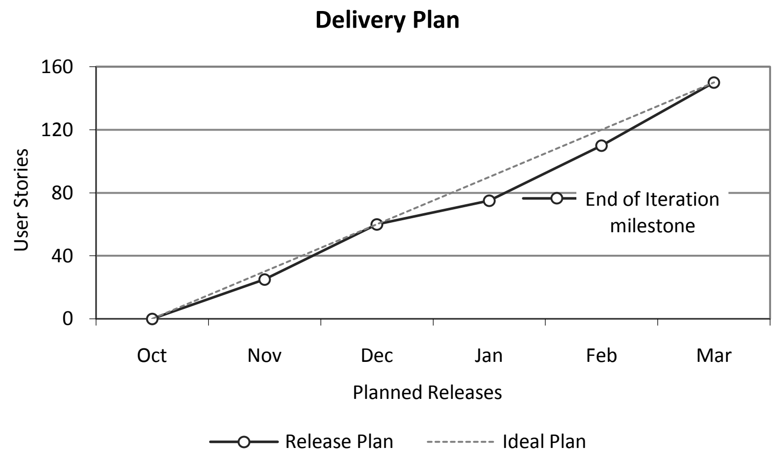


Figure 9 Delivery plan proposed by the development team. The RP curve is used to calculate progress relative to the end of the iterations while the IP provides a baseline against which to compare the promised deliveries to an average productivity over the life of the project.

The IP is built by joining the start and the expected end of the project with a straight line. The slope of the line is the average productivity that the team will need to exhibit to deliver what has been agreed with the project sponsor in the time the team has committed to. The IP helps keep the plan honest by raising the following questions: Is the average productivity reasonable, e.g. has it been observed before in like projects? Are there any periods in which the team is counting on heroic productivity levels? Are there any periods in which productivity is well below par? Why?

While the RP plan enables tracking progress against current and past iterations, the IP facilitates the assessment of delays against the total project schedule.

4.3. THE STATUS CHART

In the original formulation of the LOB, the Status Chart (SC) only provided quantitative information about the work in progress and the work completed relative to the end of iteration milestones in the RP. To these, the authors added a third indicator showing the progress to be achieved relative to the IP, in order to provide a strategic view that could prevent overreactions to missed deadlines and provide early warnings to unfavorable trends. The use of this indicator will be exemplified in the section of Interpreting the Status Chart.

To calculate the number of user stories that should have been ready at a time t relative to the RP, we need first to mathematically express it <4> as a series of straight lines, each valid in the t_i, t_{i+1} range.

$$RP_t = \begin{cases} a_1 + b_1 t & t_0 \le t < t_1 \\ a_2 + b_2 t & t_1 \le t < t_2 \\ . \\ . \\ . \\ a_r + b_r t & t_{r-1} \le t < t_r \end{cases}$$

<4>

$$b_{i=1,2,\dots,r} = \frac{UserStories_i - UserStories_{i-1}}{t_i - t_{i-1}}$$
$$a_{i=1,2,\dots,r} = UserStories_{i-1}$$
$$r = \text{number of planned releases}$$

Similarly, to calculate the planned quantities with respect to the IP we need first to find the equation <5> of the line that passes through $(0,0)$, $(t_r, UserStories_r)$.

<5>

$$IP_t = bt$$
$$b = \frac{UserStories_r}{t_r}$$

To calculate RP_{t_q} or IP_{t_q}, the number of user stories that should have passed through control point q at time t, we simply look ahead by q's lead time <6> and apply either <4> or <5>.

<6>

$$t_q = LeadTime_q + t$$

The idea behind the procedure is simple. If it takes 11 days on average for a user story to go through its complete development cycle, on any given day we should have started at least as many user stories as we are supposed to be delivering 11 days from now. Figure 10 provides a graphical exemplifies this. The same idea applies to any other control point.

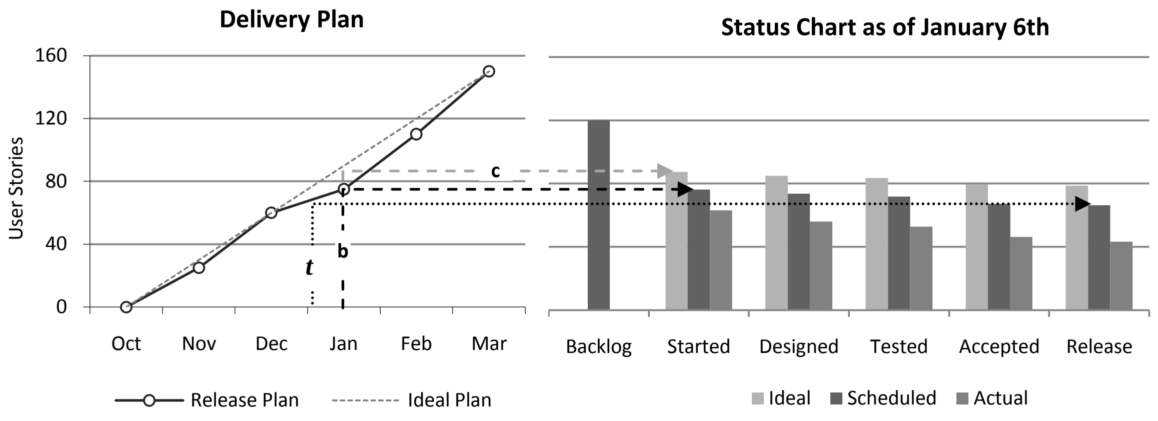


Figure 10 The intersection between the time now $t = January, 6^{th}$ line and the release plan yields the $RP_{Released}$ value, that is, the number of user stories that should have passed through the "Released" control point as of that date, to meet the target for the 4th release. The intersection between the release plan and the line (b) at $t + LeadTime_{Started}$ yields the $RP_{Started}$ value for the "Started" control point. The intersection of the line (c) at $t + LeadTime_{Started}$ with the ideal plan yields the $IP_{Started}$.

5. INTERPRETING THE STATUS CHART

In this section, we discus three examples and give some advice on what to look for in the Status Chart. The examples have been purposely designed to highlight the characteristic that we want to show.

The example in Figure 11 shows a project with a very aggressive delivery – twice the amount of user stories – targeted for March. Note the upward deviation of the RP line from the IP line. The Status Chart shows that all activities are in balance, since there are no abrupt falls from one state to the other. The chart also shows that the project is slightly behind with respect to its RP. They were probably too optimistic about their ability to deliver in March, but a little bit ahead of schedule with respect to the IP plan. If the team can keep the pace, it is probable that they will finish on time.

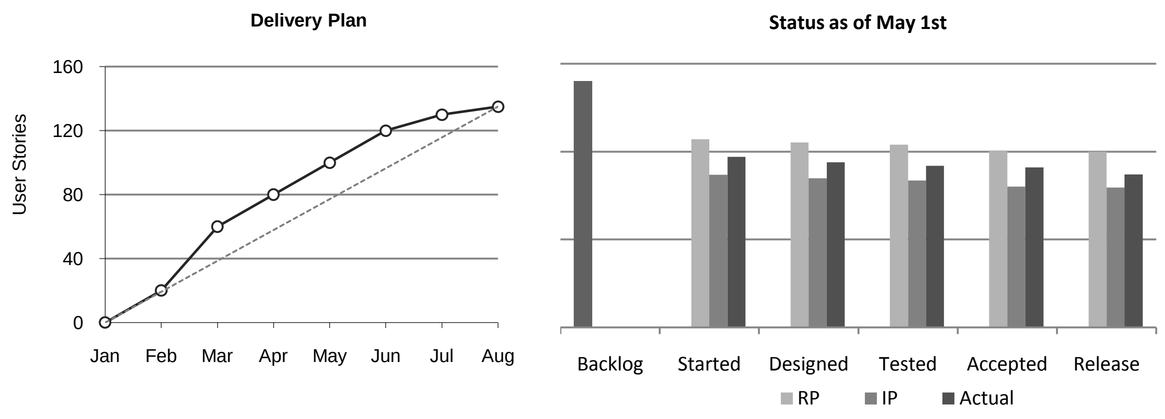


Figure 11 The team has committed to a very ambitious target for the second release. Note that, except for the month of March when the team proposes to deliver 40 user stories, the velocity is estimated at 20 or less user stories per month

The chart in Figure 12 shows the same project as in Figure 11, but this time the project is well behind with respect to both the RP and the IP. Note that there are as many user stories designed as started, but there are half as many that have gone through the “Tested” control point, which points to a bottleneck in the testing activities that seems to be starving the rest of the production chain. Should the team decide to act on this information, they would need to focus their effort on testing.

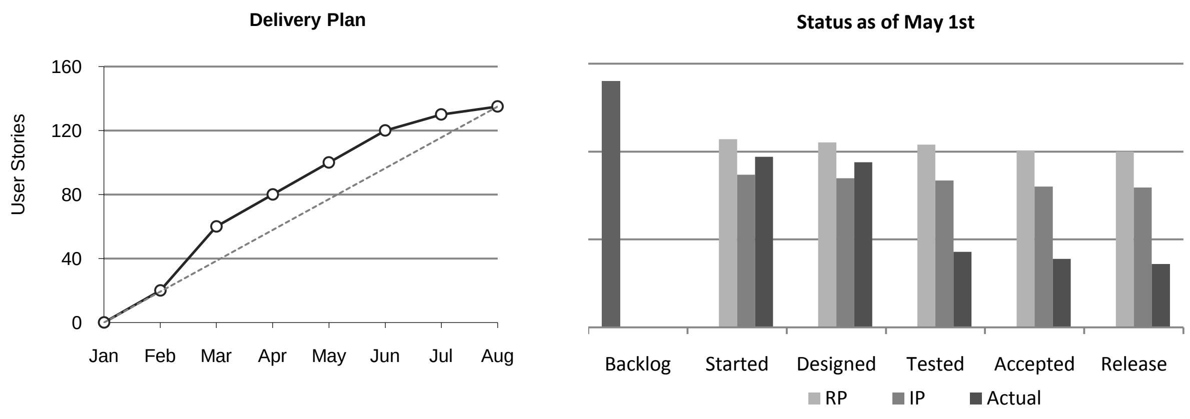


Figure 12 Chart showing a problem with testing.

In the last example (Figure 13), the team has been very cautious and provided some allowance for the team to climb the learning curve. Note the concave shape of the *RP* curve. According to the Status Chart, the project is ahead of schedule with respect to both the *RP* and the *IP*.

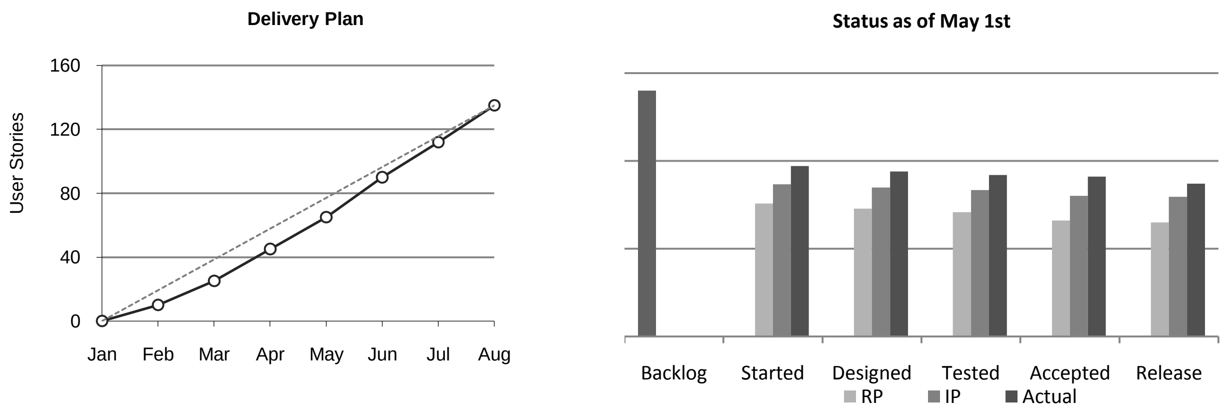


Figure 13 This team has included an allowance for learning in their release planning. Note how the estimated velocity increases towards the end.

6. DEALING WITH CHANGES IN SCOPE

So far we have not discussed what happens when user stories are added, changed or dropped from the backlog, which as every experienced developer knows, is a constant in all projects. Adding new user stories or changing already developed ones, pose no problem. For each new or changes user story we will just add one, or in its defect, the corresponding number of user points to the backlog. If there were a need to distinguish between what has been initially agreed and any subsequent changes in the scope of the project, a "Baseline" column could be displayed beside the "Backlog" column to highlight the requirements churn.

For abandoned user stories one must distinguish between those that have been started and those which have not. In the later case we will just subtract the corresponding quantity from the backlog while in the case where the user story is abandoned after being started, one would have to subtract the corresponding quantity from the backlog and all the columns to which it was added to prevent the LOB chart from reporting progress on things that are not longer desired. This could be easily done, since we know what control points the user story went through.

Simultaneously with this the delivery plan would need to be adjusted to reflect the new workload.

7. THE LOB IN LARGE AGILE PROJECTS

Most Agile projects are organized around small teams of 7 to 15 people, with larger teams organized in hierarchical fashion as teams of teams. An example of this type of arrangement is the Scrum of Scrums organization proposed in (Schwaber and Beedle 2004). One of the challenges for large, distributed, or mixed in-house outsourced teams is to keep a synchronized pace of work.

Although this could be achieved by comparing each teams' burn down charts, the problem is that, in addition to the previously discussed limitations of this type of chart, doing so requires a great deal of housekeeping. A better approach is provided by the LOB method.

Using multiple teams requires not only more forward planning than a small Agile project, but also replacing the practice of team members signing for work with a centralized assignment to ensure that cross interdependencies in the development are exploited. Parallel development also needs, at a minimum, a two-stage integration approach. At the first integration point, each team integrates its software into a local build and tests it with the software produced by the other groups. Once the software is verified by its producer, it is released for inclusion in the development baseline and made available to the other teams. The development baseline is independently tested and the delivered functionality accepted by the user. This process suggests the control points illustrated in Figure 14.

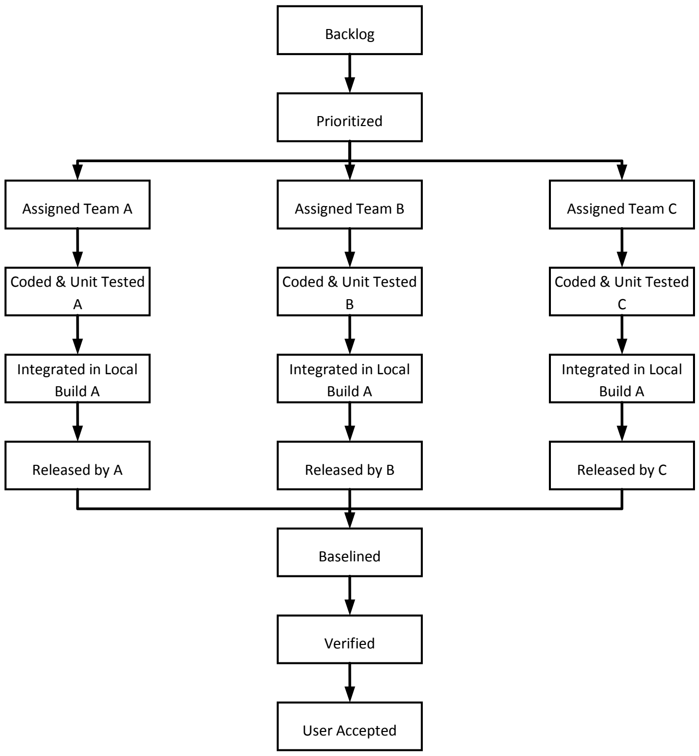


Figure 14 Control points for a project to be executed using a team of teams.

Lead times for each team's control points need to be measured independently from one another, because what we want to do is balance their production rates (velocity). The lead times for the common control points are averaged over all user stories. Each team will have its own delivery plan derived from the project goals. The project delivery plan is the aggregate of the individual plans.

Figure 15 shows a Status Chart for a team of teams presented sideways to accommodate the longer list of control points. The main observations as of June 18th, are as follows:

- Team A is slightly ahead of schedule, i.e. the actuals are slightly greater than the planned values.

- Team B is ahead in their design and coding, as indicated by its “Coded & Unit Tested” control point, but behind in integrating and releasing to the common design base (Integrated in Local Build B & Released by B). The problem might be internal to the team or it might be caused by the lack of progress by Team C should B need some deliveries from them.
- Team C is behind in its commitments by almost 50%.
- Overall, the project is behind, having delivered about 75% of what it was supposed to deliver according to the plan.

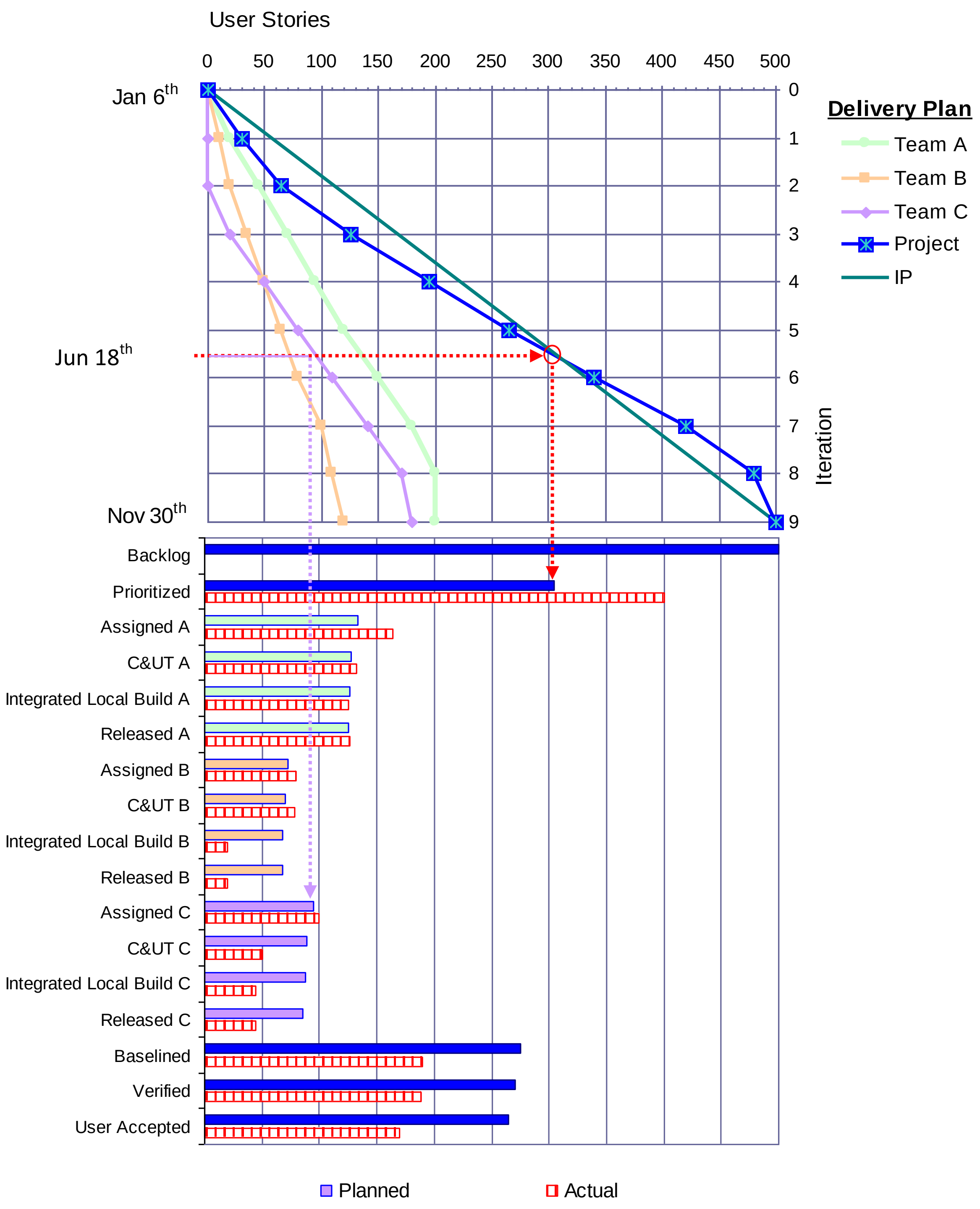


Figure 15 Delivery plan and Status Chart for a team of teams.

8. EXTENDING THE USE OF THE LOB CONCEPTS TO PORTFOLIO MANAGEMENT

Much on the same way as the LOB concepts were extended for use with a team of teams, they could be extended to be used at the program and portfolio levels. (Scotland 2003). Managing user stories at these two levels is done by creating a hierarchy of backlogs (Tengshe and Noble 2007) in addition to the release and iteration backlogs. User stories cascade from the higher to the lower level backlogs. Within each backlog each user story will be in a given state, for example one of the authors (Miranda 2003) used the following categories: in execution, committed, planned and envisioned to manage a large portfolio of telecom applications. While planned and envisioned will likely be elemental states, execution and committed will be supersets of the states identified for the team of teams approach (Figure 16).

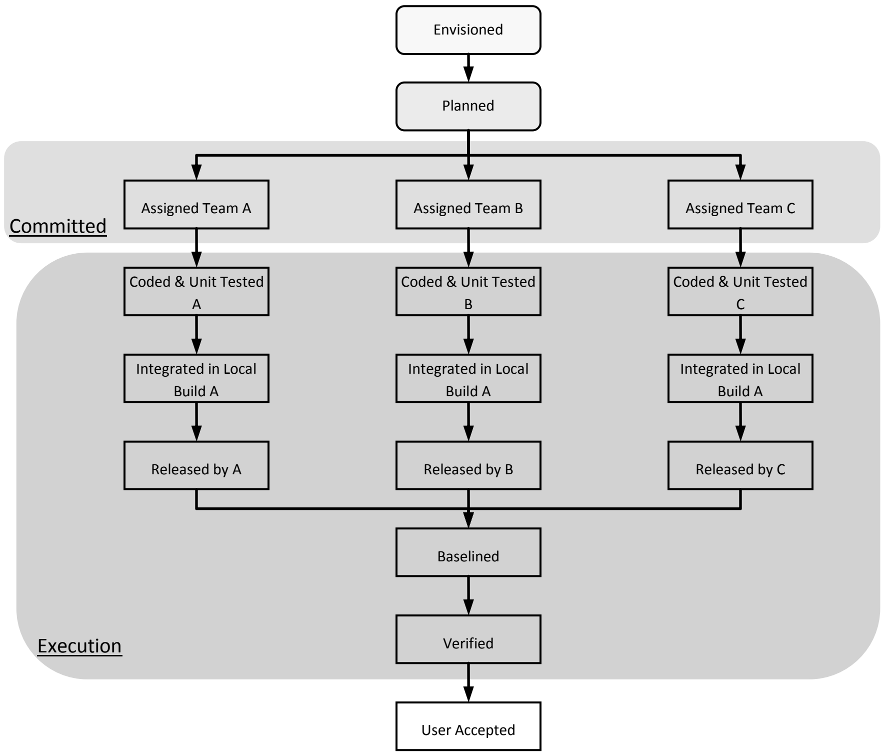


Figure 16 Applying the LOB at the portfolio level.

9. SUMMARY

While easy to produce and simple to understand, the cumulative flow diagram, the burn down charts, and the stories completed chart routinely recommended in the Agile literature tell little about what is going on inside the project. Moreover, although many practitioners will claim that all management should care about is completed work, many managers and sponsors will beg to differ.

The line of balance (LOB) chart proposed by the authors offers a practical alternative, which, while aligned with the minimalist approach appreciated by Agile practitioners, provides management with visibility and actionable information on the status of the project. The calculations required by the method are easily implemented in a spreadsheet. Using the information available from a trouble report tracking system, one of the authors implemented a limited version of the method at a large telecommunication supplier in two weeks. The information was deemed valuable by developers and managers as it pinpointed the areas where resources needed to be concentrating to clear a sizeable defect backlog.

While we have demonstrated the LOB with typical scenarios from Agile projects, its use is not limited to this type of project. The concepts presented here could be equally well applied to tracking the progress of maintenance activities, the correction of errors during testing, or the installation of software in large deployment projects.

10. ACKNOWLEDGEMENTS

The authors thankfully acknowledge the comments and suggestions received from the editor and the anonymous reviewers.

11. REFERENCES

Al Sarraj, Z. (1990). "Formal development of line-of-balance." Journal of Construction Engineering and Management **116**(4): 689-704.

Alleman, G., M. Henderson, et al. (2003). Making Agile Development Work in a Government Contracting Environment - Measuring velocity with Earned Value. Proceedings of the Conference on Agile Development, IEEE Computer Society**:** 114.

Anderson, D. (2004). Agile Management For Software Engineering, Applying the Theory of Constraints for Business Results, Prentice-Hall.

Anderson, D. (2004) "Using Cumulative Flow Diagrams." The Coad Letter – Agile Management.

Arditi, D., O. Tokdemir, et al. (2002). "Challenges in Line-of-Balance Scheduling." Journal of Construction Engineering and Management **128**(6).

Cabri, A. and M. Griffiths (2006). Earned Value and Agile Reporting. Proceedings of the conference on AGILE 2006, IEEE Computer Society**:** 17-22.

Coad, P., E. Lefebvre, et al. (1999). Java Modeling In Color With UML: Enterprise Components and Process, Prentice Hall.

Cohn, M. (2006). Agile Estimating and Planning. Upper Saddle River, NJ, Prentice Hall.

College, D. S. M. (2001). Scheduling Guide for Program Managers. http://www.dau.mil/pubs/gdbks/scheduling_guide.asp. W. Bahnmaier. Fort Belvoir, VA, Defense Acquisition University.

GAO (2007). Cost Assessment Guide - Best Practices for Estimating and Managing Program Costs. United States Government Accountability Office. Washington, US Government. **GAO-07-1134SP**.

Harroff, N. (2008). "Line of Balance."   Retrieved January 27th, 2008, 2008, from http://www.valuation-opinions.com/ev/lob.lasso.

Jackson, P. (1991). "The Cumulative Flow Plot : Understanding Basic Concepts in Material Flow." Tijdschrift voor Econornie en Management **XXXVI**(3).

Microsoft. (2006). "Microsoft Solutions Framework for Agile Software Development Process Guidance 4.1." Visual Studio Retrieved 7/14/2008, 2008.

Miranda, E. (2003). Running the Successful Hi-Tech Project Office. Boston, Artech House.

Miranda, E. (2006). Using Line of Balance to Track the Progress of Fixing Trouble Reports. Cross Talk, STSC. **19**.

Office of Naval Material (1962). Line of Balance Technology. US Navy. Washington DC.

Project Management Institute (2004). Practice Standard for Earned Value Management, PMI.

Rusk, J. (2009). "Earned Value for Agile Development."   Retrieved Oct. 31st., 2009, from http://www.agilekiwi.com/EarnedValueForAgileProjects.pdf.

Schwaber, K. and M. Beedle (2004). Agile Project Management with Scrum. Redmond, Microsoft Press.

Scotland, K. (2003). Agile Planning with a Multi-customer, Multi-project, Multi-discipline Team. XP/Agile Universe 2003, New Orleans, Springer Berlin / Heidelberg.

Sulaiman, T., B. Barton, et al. (2006). AgileEVM - Earned Value Management in Scrum Projects. Proceedings of the conference on AGILE 2006, IEEE Computer Society**:** 7-16.

Tengshe, A. and S. Noble (2007). Establishing the Agile PMO: Managing variability across Projects and Portfolios. Agile 2007, Washington, IEEE.

Wake, W. (2001). Extreme Programming Explored, Addison-Wesley Professional.