***Agile Estimating and Planning*, Mike Cohn, Prentice Hall**

I've been meaning to buy this book since its release, but Mike was kind enough to send me one for review. I've come to this book from a two year planning assignment on two large aerospace programs and prior experience as a Program Management Officer for the IT portion of a very large Department of Energy program. But these jobs are not my normal profession, which is Enterprise IT management and product development. I was pulled into these planning roles through proposal efforts that won and I was left to implement the plan. Be careful for what you wish.

How planning is done for spacecraft and construction is similar to what Mike describes in this book. Because of this background and the 6 or so years experience in managing agile software development projects, my approach to this review is to read the book through my own experience rather than assess the book as a standalone process. I've assumed all the content is well developed and vetted. What I'll try to assess is how to apply the book outside the *converted* – that is, how to put the book to work in domains outside agile software development.

First let me state my point of view. Working on DoD, NASA and DOE projects, I've been converted to the Systems Engineering paradigm. Systems Engineers see the world in an integrated manner – product development and process development are inseparable. Managing interfaces is the primary role of Systems Engineering along with product and process architecture. Yes, processes have architecture as well. Planning lives inside process architecture. Good planning is a systems engineering activity. The plan represents the processes that build the product and the architecture and technology of the product impacts how the plans are developed – driving the programmatic architecture. Planning and building are inseparable. Mike's book makes this clear in the same way a good systems engineering process does.

Quick Look

Buy this book, it's useful and informative.

It contains a few of my *hot button* errors, but that's to be expected since the agile processes, including estimating and planning are not derived from the high ceremony planning world. Mike stays pretty much on a mainstream message when describing the activities involved in planning and estimating. No weird units of measure, no unsubstantiated claims of miraculous tools and processes, no divergent philosophical excursions. Just plain, understandable suggestions, on how to approach what is a difficult and sometimes intractable problem – *how much does this cost, when will it be done, where are the risks, what are we doing about them, and what do I get when we're done*?

Summary

The concept of *agile planning* can go to one of two extremes – the *planning* of agile projects or *agile* planning of projects. Mike has straddled the middle, suggesting that the book is about the planning of agile projects, but he includes materials that can be used for general purpose planning of any project. He lays out a logical process of partitioning the planning around the *granularity* of the plan, focusing on the daily, iteration and release planning processes. Leaving the product, portfolio and strategy planning for others. This is important since those planning processes must be in place in order to provide a framework for the lower level planning activities.

My planning experience is built around the IMP/IMS (Integrated Master Plan / Integrated Master Schedule) approach (and its derivatives) as described in the Department of Defense guide. After reading Mike's book I was stuck by the similarity between the agile planning processes and IMP/IMS. Event based planning is core to IMP/IMS. The plan describes the effort necessary to deliver the increasing maturity of the program through assessable and verifiable accomplishments. These accomplishments describe progress rather than the passage of time.

Mike's approach provides measurable assessment of progress through *conditions of satisfaction*. The fine–grained iteration–based delivery process measures progress through the production of working products (features or capabilities). In IMP/IMS, the granularity may be larger only because the programs may be larger but the principle is nearly the same – progress is measured through significant accomplishments and the *exit criteria* used to verify their completion. The maturity of the program is measured by these significant accomplishments, NOT the simple passage of time.

The other principles described in the book can be found in good project management guides as well. (There are many guides beyond PMI's PMBOK) Here are some references that provide this background. Maybe of these guides are targeted at large and complex projects, but many of Mike's processes are directly applicable to a broad range of projects sizes and complexities. This is the power of the book. It is not just a book about planning small agile software development projects. Other project domains will find many useful ideas as well.

My Biased Approach to Book Reviews

My approach to a book review is to read the chapters in sequential order, making notes as I go. I then use these notes to comment on the chapters and write a summary recommendation at the end. Before starting the reading the chapters, I look through the bibliography to find papers or books that I have not come across before. Mike has done a good job of providing sources and other materials to augment the book. There are some references I had not seen before, so the bibliography alone added value. Mike has both sides covered – good original ideas and the references to back them up – especially on the estimating process side. Next, I look at the art work to see if the pictures are *notional* [1] or *substantive* in nature. For the most part they are notional, but this is probably the best that could be done for the audience.

I've assumed the substance of the book is valid, and I'm commenting on how I would use this content in my work.

[1] The idea of a notional description is powerful but it has limits if not bounded. These unbounded descriptions are very common in the agile literature. The exuberance of using the notional description as the executable description is common. The problem of course is the three infamous words *this doesn't scale* come into play. Especially once a non-trivial planning and estimating project comes along.

Chapter 1 –The Purpose of Planning

Figure 1.1 is the standard diagram for technical performance measures, [2] increasing maturity of IMP/IMS, or reducing performance variance for every product and process we use on our program. Changing the horizontal axis to the appropriate names allows the diagram to be the basis of every planning process. The only issue is that the variance at the end point is never 0, it always has some uncertainty. Here in aerospace we call such a diagram "notional," meaning it is a good representation of what the solution will look like, but is not actually correct in practice. The other thing *notional* with the figure (and therefore wrong) is the variances are never symmetrical in practice. When I see symmetrical variance it tells me the picture does not represent any real process.

The PMI approach to the estimating problem is naïve to start with and simply wrong in the end. Probabilistic schedule analysis is a core competency of any good systems engineering process. The development of 3–point estimates for a triangle distribution, plus or minus percentage from a mode (most likely) and a variety of other approaches is complex, time consuming and at times tedious.

The section titled *Why Do It?* is a start on the reasons for planning. [3] Also important is that planning is part of the risk reduction process for any project – especially software development projects. It is this risk reduction that is the primary reason for planning. Without a *plan*, "answering the mail" (as we say in the proposal business) on risk is not possible. Risk management becomes "making lists and checking them twice" and no actionable outcomes can take place because no tradeoffs can occur, because no impact on cost and schedule can be identified.

So now that that is little rant is over, let's continue

Page 5 – there is the suggestion that the dates for the plan are part of the plan. In fact the planning tool is what produces the dates, not the planners. If the planners produce the dates – say a planned set of features on a specified date – then the plan is called *time boxed*, with constraints on the end dates (Must Finish On or Finish No Later Than). This is the kiss of death for any planning process. It is forbidden on our program. The planning process needs to identify the work, the sequence of that work, the margins needed to mitigate risk and let the planning tool say when the work will be done. If the end date is fixed – which is the case many times – the planning tool will then tell us *when we should*

[2] TPMs are a way of defining what done looks like in units of measure of technical performance. The figure has this starting for the project schedule's potential overrun. What is missing of course is "how" these estimates came to be and how they will be reduced as time passes. This is why TPMs are used on large technical projects to drive the planning process – they define how the program should move forward with increasing maturity and not allow progress to be measured by the passage of time.
This by the way is the kiss of death for any real project plan and it is so common in the software development world that all the examples of project scheduling processes that agile rails against are really bad examples to start with. If the agile community could actually see how large complex programs are managed in very similar ways to agile, then all this wrangling over what not to do would die done and we could start talking about real executable value outside the agile only software development domain.

[3] No where else in the project–based world (other than agile) does the question *why plan* come up. Only in agile is the need for planning questioned. Although XP has a planning session activity. Agile is so polluted with bad examples of how planning and other processes have failed, that it takes special effort to bring out the good aspects of these processes.

have started this project. Fixing the start date, the fixed end date and the predefined work to be performed is not planning. It is a picture of failure.

Page 9 – the statement “this book is about agile planning, not agile plans” misses an important point about the planning process in mature organizations. The conjecture that plans are somehow fixed and separated from the planning process is simply not true, it was never true. In our aerospace environment we use Microsoft Project Server to mange the database of planning elements. There is a separation between the process of planning and the results of planning. Both the plans and the planning process need to be agile. Mike discusses the planning process. But the plan can be agile as well. In most aerospace projects an agile plan is represented by a risk tolerant schedule, on and off ramps for decision making, margin and probablistic analysis of the attributes of the plan – criticality, cruciality, the probabilistic critical path, which means the impact of “near critical” paths on the completion of the project.

It is more than having a plan that can be easily changed. It means having a plan that has integrity, is robust to these changes, and whose architecture is “tolerant” of these changes – that allows the project to be *managed in the presence of change*.

Chapter 1 is a nice introduction to agile planning and plans, with some focused actions on Page 10.

Chapter 2 – Why Planning Fails

This chapter makes the conjecture that project failure is a failure of planning. Mike has five causes of planning failure. This conjecture needs a bit of clarification before it can be useful in a general sense.

1. Planning by activity versus features – this is correct. In the IMP/IMS process the plan describes the increasing maturity of the product not the passage of time for the functional effort. The “features” should be defined in the statement of work or a functional specification and then the plan should show how these features are being delivered in an increasing maturity manner. That way incremental and iterative process can be shown explicitly in the plan. This approach is standardized in IMP/IMS based procurement processes.
2. Multitasking causes further delay – multitasking is one of the hot buttons for agile projects. They claim there is a cost for task switching, but do not describe the underlying details of this approach. The problem is the curve in Figure 2.2 appears to be notional as best and from a 1993 paper (going on 13 years now).
3. Features are not developed by priority – in the aerospace planning process the sequencing of the Significant Accomplishments of the IMP/IMS is critical to building a robust plan. This “flow” describes the architecture of the program – that is how the program will develop the product in its increasing maturity process. This must be carefully thought out since any “streamlining” processes must actually add value to the program, reduce the programmatic risk and increase the likelihood that the completion date will be held.
4. We ignore uncertainty – “It is moronic to predict without first establishing an error rate for a prediction and keeping track of one’s past record of accuracy,” –

Nassim, Nicholas Taleb in *Fooled by Randomness*. This says it all. If you are planning in the absence of a probabilistic estimate you're not planning, you're guessing.

5. Estimates become commitments – understanding the probability distribution for each estimate is critical. It is also difficult without an understanding of the statistical processes that drive the schedule. This includes things like: merge bias for parallel tasks, long tailed distributions, ill–formed distributions. The total length of a plan is not the sum of the individual durations, it is the convolution of the most likely values (mode) of the distributions for each task along the multiple paths of the network. This area of study belongs in statistical programmatic risk analysis and is part of any mature planning activity.

Chapter 3–An Agile Approach

All of the agile manifesto words are here. They describe what agile values, how they work together, the idea of short iterations, who is the product customer etc.

Page 27 – Hal Macomber is referenced in the book, that project plans can be viewed as *rapidly and reliably generating a flow of useful new capabilities and new knowledge*. The capabilities are generated by the products being produced, but the knowledge can come from the plan as well. The IMP/IMS approach says…

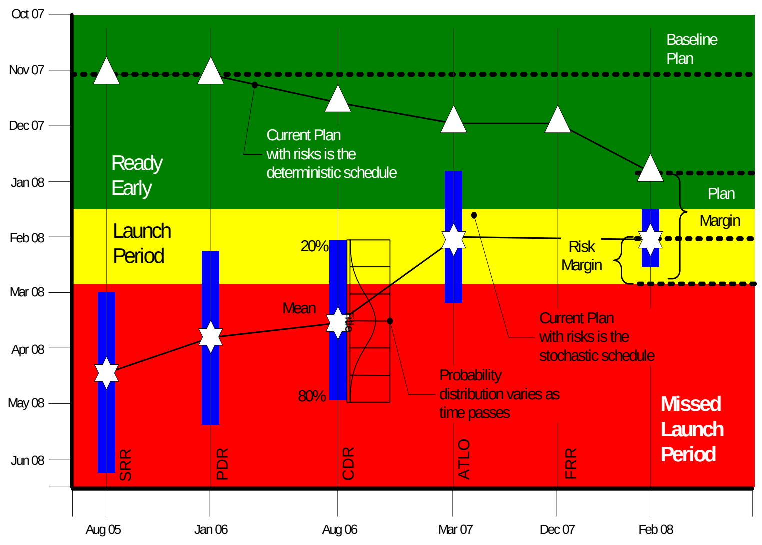
> *This plan shows how the program "can" produce new capabilities and new knowledge…with risk mitigation, process flow and probabilistic estimates embedded in the plan. The performance monitoring of the plan provides feedback for estimating future performance (Earned Value is a useful way to estimate future performance).*

This capability and knowledge must come from the products produced by the execution of the tasks of the plan, not the plan itself. Mike's view gives too much credence to the plan – the plan simply represents the agreed on intent of the planners, engineers, and managers. It itself is nothing more than a "plan" for the production of capabilities and knowledge.

The trick is to install a work process that gathers from the resulting product information that can be fed into the plan to produce knowledge. This loop is referred to as the Business Rhythm of the project.

1. Cost and schedule performance – earned value is the typical means of capturing this information. XP velocity can be used but it provides only half of the equation, a useful half, but only half.
2. Value performance – the Budgeted Cost for Work Performance (BCWP) describes the "value" delivered.
3. Quality – technical performance measures (TPMs) are a means of measuring the increasing quality of the product. This is not an ISO type quality, but a quality of the product when it is compared against the plan. For example in a spacecraft development I need to know the mass of the vehicle. This is important for many reasons, none the least if which is *how much fuel to I need to get to where I'm going?* I decide that during the early stages of the project I need to know the mass

to ±20%. (This is symmetric I know and a real TPM would be asymmetric). At the first Critical Design Review (where all the pieces come together) then the mass should be known to ±10%. This is a Technical Performance Measure. Similar measures can be made for cost and schedule. Here's a example of a notional TPM



The increasing maturity of the estimator – in this case the launch date – is shown as a function of the milestones in the master schedule. Risk and risk margin – both deterministic and probablistic – are shown in the two primary paths through the plan.

4. Maturity assessment – the measurement of the increasing maturity is the critical success factor for all projects. Agile projects do this better. But this measurement must be done if the project has a chance of success. This connects back to Mike's guidance to iteration, increments, producing working products, measuring progress etc. This is the standard (now mandatory) way large programs are managed and measured.

Page 28 – the concept of multiple levels of planning is mandated in DID 81650. Agile PM is now discovering this.

Chapter 4 Estimating Size with Story Points

Estimating the duration, complexity or cost of development activities is fraught with problems. Mike has some of the base references on how to construct these estimates but this is a very complex problem in the planning domain. It starts with how to remove the estimating bias from the units of measure describing the duration – in either days or story points. That can be done in a straight forward manner. What is the killer problem with Story Points or Ideal Days and the use of Velocity is the assumption that the units of measure stay constant on the same "metric" (the underlying scale) across the project. This is almost never the case. As the project proceeds the units of measure change their "metric."

This does not damage the general concept of velocity, story points or ideal days but leaving this out leads many to conclude that story point and ideal days are good unbiased point estimators of the project duration – they are not. This is a highly biased statistical estimating point of view, but it is one that has burned me several times on large programs where Monte Carlo simulation is used as a standard approach to programmatic and technical risk assessment. This book should probably not address this issue other than in a footnote. The end point of the topic is Ed Conrow's book *Effective Risk Management: Some Keys to Success*. Ed is a consultant on our program and his book is invaluable to anyone interested in all the gory details of probabilistic assessment of risk.

So what's the real problem? Velocity is an un–calibrated estimator whose "metric" (the scale by which the unit is measured) must remain unchanged over the life of the project. This has not been shown to be the case in practice. In fact, the biases associated with maturing project estimators' change as time passes. This is a core problem in probabilistic risk estimating in large programs like Joint Strike Fighter and SBIRS (Space Based Infrared System). There are many degrees of freedom for the underlying stochastic process of cost, schedule, risk, task interaction, and task productivity.

The example Mike provides on **Page 39** is flawed, since the dimensions of the walls of the room remain unchanged during the painting project. For a software "wall" its original size is unknown and the final size is unknowable at the start of the project. It is good that estimation of effort can be separated from the estimation of duration, but the velocity metric is *driving in the rear view mirror* – it can be done but it sure is sporty.

This topic is at the core of Mike's thesis and XP, so it is unlikely to go away. Nor is there a simple solution that involves the approaches of agile. But for estimating of software development effort and duration, better approaches are around and in use in large projects. No matter Mike provides a clear and concise description of the approach – regardless of the underlying problems with the biased statistical estimator.

Chapter 5 – Estimating in Ideal Days

The ideal days approach has the same statistical problems but is a useful process all the same.

Chapter 6 – Techniques for Estimating

The assumption that there is a sweet spot for estimating is true in principle, but Mike does not describe how to find this sweet spot. This is a non–linear optimization problem that requires the independent variables be calibrated and the dependent variable correlations be defined. This problem falls into a class of problems call "Isoperformance optimizations." A recent paper "Isoperformance: Analysis and Design of Complex Systems" gives some background.

Page 51–52 – this example here falls into that class of "stories" that make me cringe. Why would Kristy be so naïve as to not consider the overhead and past problems in estimating the future effort? I'm sure it happens. This is a common narrative approach in agile books for some reason. *Tell a story of someone doing dumb things on purpose and then give an example of how to correct the outcome using an agile method*. I would think it would be better to provide a check list of all the considerations of estimating and have the reader use this check list when making an estimate. This is a biased view I know. And I know there are people who do *dumb things on purpose*. It's just that I personally don't like to read about them in what is essentially a fine book on agile estimating.

Page 52 – the estimating scale section references two papers, but there are loads of other approaches and this is a controversial issue to begin with. Here in aerospace we use a geometric scale. Mike's scale is subjective, and like many others, is arbitrary and has no statistical basis. We have these arguments all the time. Mike's scale is probably useful, but like the biased estimators above doesn't get too far once the statistics start to be questioned.

Page 54 – the sources of estimation are correctly defined. Starting with an expert opinion. The real problem in software estimating is there is rarely a historical database. I think there is a missed opportunity here to introduce COCOMO–style estimates not as a tool for agile, but as an anchor on how large complex defense system projects are estimated in the presence of similar uncertainty.

Chapters 7 – 11

This information, while informative was not that interesting to me, since it is focused on software development. Not that this isn't the real purpose of the book, but I'm looking at this from the general purpose planning point of view.

Chapter 12 – Splitting User Stories

The quote from Mary Poppendieck makes the huge assumption that the features of the system can be partitioned (made orthogonal) from the modules. I say huge assumption because this is one of those quotes made by agile thought leaders that are so generalized as to be worthless in practice, and therefore dangerous to the implementation team. I say this from direct experience of having cleaned up several "agile projects" that built their management process on the platitudes of book quotes with absolutely no consideration for the impacts of following these platitudes to their final conclusion. Mary is fond of statements like this – and I understand her approach as a "sales person" for agile. But care is needed in practice.

The splitting of stories approach in this chapter is classical systems engineering. "On what boundaries should we divided the system requirements?" In other words how do we flow down the requirements into subsystems and sub–subsystems? This is a system architecture problem.

When the features represented by the stories are re–cast into "capabilities" of the system there are many resources about partitioning capabilities across systems–of–systems in the Systems Engineering literature. If there is anything I'm learning from this book it is that Systems Engineering already has a lot to say about what Mike is writing about. This is good, since the agile planning world is now starting to become mainstream; there is an underlying collection of principles already in place for them to draw on.

Page 124 – splitting stories on operational boundaries is the same as Systems Engineers partitioning the flow of work along the architectural decomposition of the product. The plan then reflects how this partitioning flows as the product development matures the capabilities of the product. Mike's got this absolutely right – straight out of the NASA System Engineering handbook.

Page 125 – it is interesting there is a section of *removing cross cutting concerns*. In the spacecraft planning business *cross cutting concerns* is an assigned discipline of Systems

Engineering. It is their job to manage all the cross cutting activities in one place. I'm not convinced they can be removed from a sufficiently complex project.

Mike's solution is to split out the cross cutting concerns into a separate story. In the complex systems world this would be assigned to the *cross cutting* systems engineering (usually called Big SEIT – systems engineering integration and test). The I&T part is done separately for all the obvious reasons. The interactions between all the parts needs to be verified independently for the individual parts. This is especially important on ERP style projects, where unit and integration testing assures that the static parts work, but since there are 1,000's of interaction combinations of features a Systems Integration and Test activity is needed.

Stories and Tasks

At this point in the book it has dawned on me how to classify *stories*. They are Accomplishment Criteria in the IMP/IMS vernacular. AC's are *exit criteria* for the increasing maturity of the program. *Significant Accomplishments* (SA) are *entry criteria* for the *Program Events* (PE). PE's are the assessment point for the capabilities of the program. A typical PE for a manned spaceflight program would be ***Risk Reduction Flight 2 Complete*** or more typically ***Critical Design Review Complete***. These are points in the program were a specific set of capabilities is available for assessment. The customer then acknowledges that these capabilities are acceptable and authorization to proceed to the next PE is given at the acceptance review. This approach works very well with enterprise software systems as well – since small incremental releases aren't very useful for the enterprise class user base. They need a integrated cohesive set of *capabilities* that can be put work in production – not half finished (but working) features.

The CRUD paradigm is very common in document management and configuration control of manufactured good (Product Data Management systems use this as well). Mike's paradigm of partitioning stories on CRUD boundaries is really great. Again this is a good Systems Engineering activity.

Chapter 13 – Release Planning Essentials

The premise that planning is somehow fixed and unchangeable has permeated the conversation around agile. This is not the truth on real programs. *You improvise, adapt and overcome* all the scheduling difficulties.

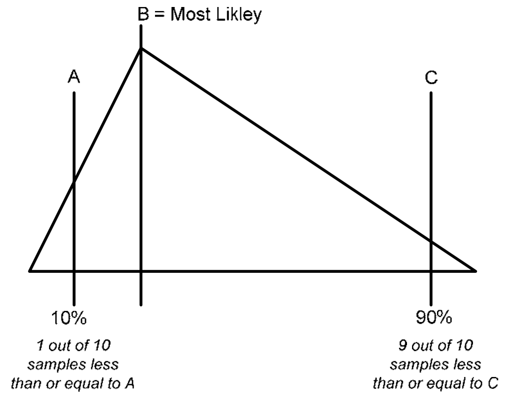
Page 135 – conditions of satisfaction are called *Accomplishment Criteria* – exit criteria for the collection of tasks.

Page 138 – if this figure were laid out as an IMP/IMS the topology would be identical.

Chapter 14 – Iteration Planning

The idea of a *commitment based* planning process removes the time box scheduled approach found in most bad–planning paradigms.

Page 164 – the distribution shown in Figure 14.5 almost NEVER occurs in real life. The concept of a symmetric distribution for the duration is


B = Most Likley
A
C
10%
90%
1 out of 10 samples less than or equal to A
9 out of 10 samples less than or equal to C

task is way too *notional* for any useful discussion. Asymmetric distributions with long tails to the right – *right skewed* – is the starting point for the model of the completion times. Figure 14.6 has the same problem. For the casual reader this is probably not a problem, but it shows how little the discussion has moved from simple concepts to actual projects.

The terminology of *on average it will take 12 hours to complete a one–point user story* is the wrong approach for several reasons:

1. The *average* duration cannot be stated without the number of samples being stated as well.
2. The *average* is not what we're after – we're after the *Most Likely* duration. The average is a point estimate; the most likely is a statistical estimate. Point estimates cannot be summed across the totality of stories.
3. The real question is *for 1 out of 10 times I perform this task what is the duration or less. For 9 out of 10 times what is the duration of less?* The figure above uses a triangle distribution to describe the task duration estimates. The 10/90 points on the triangle distribution or one way to confidently describe a statistical model for task completion. In the Triangle distribution the Most Likely and the Mean are the same value. This is very useful when "adding" statistics – statistical distributions can not be "added" they must be convolved in order to arrive at the total duration of a string of tasks.
4. The concepts in this chapter are valid, the math is not.

<u>**Chapter 15 – Selecting an Iteration Length**</u>

The length of the iteration is a critically important question. Mike describes to problem well as well as the solution. In large complex projects a *rolling wave* is equivalent to the iteration. There are endless discussions about the length of the rolling wave, same for the agile iteration.

<u>**Chapter 16 – Estimating Velocity**</u>

The approach described here starts with *time boxed scheduling* – since the iteration are fixed. Velocity is an adequate measure of progress in XP environment but it is a poor estimator of cost AND schedule. Velocity is like *driving in the rear view mirror* – it can be done, but it's a real rough drive. As well velocity is a *relative* measure of performance not an absolute measure of performance. This is similar to the pair comparison used for estimating software size. PPA generates relative estimates of size, but unless there is a basis of estimate, these relative estimates are not very meaningful outside the group using them to size their efforts *in a relative manner*. Not that velocity and PPA are not useful, they are. But they are not replacements for more sophisticated methods on larger more complex projects.

<u>**Chapter 17 – Buffering Plans for Uncertainty**</u>

Here again the statistical approach is *notional* and actually not correct at times.

Figure 17.1 shows the probability of completion times. This is technically not correct. A curve like this shows the number of times an identical task completes *on or before* a specific time. Like the triangle distribution diagram above. It is not the probability of a single task completing in a given time. Although it is interpreted most often in this manner it is not technically correct. That said it is a useful conversation piece at this point in the book. A footnote would have provided a bridge between notional and factual.

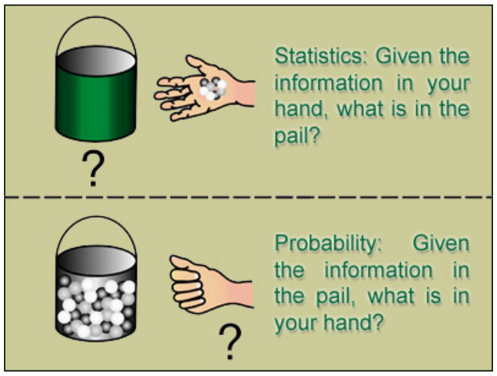
This is the time to sort out the difference between statistics and probability.

1. *What is the probability of making a desired completion date?* Is a probability question. This question is important to the planning process as well as customers, so they can some planning on their own.

2. *Given the historical data, what is the "most likely" completion time for this task?* This is the question asked by the planning process.

What the planner needs to know is two things – what is the *most likely duration* of this work and *with all the work lined up – what is the confidence that the project will complete on or before a specified date?*

Page 192 – Mike mentions a 90% confidence interval for duration. This is highly unlikely on any real project. On large aerospace projects an 80% confidence interval is typically used – *there is an 80% chance of finishing on or before the target date.*


?
Statistics: Given the information in your hand, what is in the pail?
?
Probability: Given the information in the pail, what is in your hand?

The core problem with adding up the 50% and 90% buffered numbers in the example is Mike has switched the probability units of measure to scalar units of measure. This is not a problem in this context, since the 50% and 90% *buffers* have been converted to duration of fixed – NON Statistical – duration. But in fact those buffer durations represent a 50% chance and a 90% chance of completion so they have probability distribution associated with them and cannot be added as scalar numbers. This is a very common mistake in project management literature. There no harm in the *notional* aspects of the book, but on a real program – say with 8,000 task activities all connected in the complex network – this would a definite No–No from the programmatic risk analysis point of view. If a moderately sophisticated buyer – say NASA – saw this in a proposal, a RED would be given to the plan. RED is not good. Submitters with RED plans loose.

But again for the purposes of the book, the *notion* that buffers can be added and the duration of these buffers represent some confidence in completing the task is probably OK. [4]

Page 194 – Mike then falls into the *addition* trap that PERT and any other algebraic addition approach does. He takes the standard deviations of the buffer durations and adds them. This is wrong. It does produce a number, but statistically this number is optimistically biased. This error is well documented in the literature, is probably to obtuse to be addressed in a book review. But this error is so common it has a name – *ordinal summation errors*.

[4] This approach is a Theory of Constraints gospel, but it is wrong statistically. See Ed Conrow's Effective Risk Management, book on this complex and convoluted topic.

> You cannot algebraically add the *low*, *most likely* or *high* durations derived from a statistical distribution and get a valid duration. They must be combined statistically – convolved.

The PMI books, the standard off the shelf PM books, and most any paper on PERT or TOC shows how to do this – but it is wrong. Not terribly wrong, but wrong all the same. In a complex network it can be wrong as much as 20% (or so) depending on the topology of the network. See Conrow for the gory details [4] as well as others. [5] But again this is way too detailed for a book review or for any application Mike is suggesting.

Page 199 – schedule is not padding, this is true in any good schedule. The buffering is called *margin* in aerospace. Schedule margin is discovered by building a *deterministic* schedule with all the durations laid in. By the way a *good* schedule NEVER defines dates other than the start date (authorization to proceed). All dates are derived from the network of tasks and their durations. The ONLY constraints in a schedule are the MUST START ON for ATP and AS SOON AS POSSIBLE. Now in agile iteration based planning this will be difficult because of the *time boxed* approach to planning. From the deterministic schedule a Monte Carlo simulation is run to identify how much margin is needed to get to an 80% confidence level for the critical deliverables. This margin is then placed in front of these deliverables as an explicit task flagged as *margin*. TOC calls this a buffer, DID 81650 calls it margin.

<u>Chapter 18 – Planning the Multi–Team Project</u>

The chapter starts with one of the dumbest statements ever made about planning – *do the planning, but throw out the plans* – Mary Poppendieck. This is pure agile gobbledygook that has no meaning and no value and does not deserve to be in a book of this quality.

> *Planning is not "plan and forget" as Mary suggests, but it is an ongoing dynamic activity that peers in to future for indications as to where the solution may emerge and treats the plan as a complex situation adapting to an emerging solution.*

This chapter is actually the antithesis of Mary's statement – bad copy editing I guess. Mike lists 4 activities for planning

1. Establish a common basis of estimate – in large projects BOE's are derived from the plan.
2. Add detail to the user stories – the accomplishment criteria need tasks.
3. Perform the look ahead planning – this is going to be hard if you throw out the plan.
4. Incorporate buffer – the durations can be derived from the margin analysis.

[5] "Advanced Quantitative Schedule Risk Analysis," David T. Hulett, Hulett & Associates; "Inverse Statistical Estimation via Order Statistics: A Resolution of the Ill-Posed Inverse Problem of PERT Scheduling," William F. Pickard, *Inverse Problems*, 20, pp. 1565 – 1581, 2004; "PERT Completion Time Revisited," Fred E. Williams, School of Management, University of Michigan-Flint, July 2005; "Three Point Approximations for Continuous Random Variables," Donald Keefer and Samuel Bodily, *Management Science*, 29(5), pp. 595 – 609.

Chapter 19 – Monitoring the Release Plan

The project tracking processes described here include many of the processes found in large formal projects. Burn down charts, velocity measurements, parking lot views. These and as many as can be discovered are all useful. For an agile project management method to be useful it needs to serve the needs of all the participants. Mike makes it clear that the suggestions are just the beginning – each project needs to produce a set of *Big Visible Charts* that can be used for communicating the message – *we're making progress*.

Chapter 20 – Monitoring the Iteration Plan

Fact based management is a critical success factor for project management. This chapter shows how to start with facts and convey them in a meaningful manner.

Chapter 21 – Communicating about Plans

I like the quote that opens this chapter – *The more elaborate our means of communication, the less we communicate*. But let's not follow the path suggested by Mary. Building clear and concise communication processes is very difficult. It requires enough visibility into the future to describe what *done* looks like in some measurable terms so we can move forward in a measurable manner.

Keeping everyone informed starts with charts and graphs. One very useful format is an end–to–end flow of all the work to be done. For the current and the next iteration this would have detailed fidelity. For future iterations much less detail is available. This *picture* is updated every iteration to reflect the emerging understanding of what *done* looks like.

Figure 21.1 on Page 238 is a notional example of this. But instead of a Gantt chart (which shows progress as a function of time), use a PERT–like chart with Program Events (Releases), Significant Accomplishments (Collections of stories describing capabilities), and Accomplishment Criteria (Stories) and tasks for the current iteration.

Chapter 22 – Why Agile Planning Works

There are no guarantees in any planning process, let alone agile. There are various degrees of confidence, credibility, criticality, cruciality, risk, visibility, understanding, and delivered capability depending on the project management methodology. These are some of the attributes of a planning process.

It is a red herring to consider that agile planning processes provides these attributes in some way that is dramatically better than a mature planning process – emphasize a mature project management method. I say this from the experience of managing the development of plans for multi–billion dollar programs as well as smaller but equally as critical projects. The process of planning involves some basic activities. The dynamics of these activities is what makes them agile or not. How these activities respond in the presence of a changing environment. It is the *management in the presence of change* that makes one agile. This always seems to be confused with trying to *manage change* in the more formal process.

The management of change can not be ignored in any paradigm – agile or not. How we manage in the presence of change is what is important.

Page 254 – has a dozen guidelines for agile estimating and planning. This is the essence of the book and comes right before the case studies. One of the tests of a good book is whether you can boil the concepts down to one or two pages – you can and these 12 guidelines should be put on to a poster and hung on the wall.

Case Studies

The case studies are interesting and should be seen as *examples* of how other projects might work. The problem with a small number of case studies is they are small and not very representative of software projects in general. It is not to be expected any other way for a book like this. Here's an example of a broader set of projects in a case study. *High Risk Series: An Update*.

Conclusion

Buy this book, read it, read the material from the bibliography. Go searching on the web for other project management processes besides those produced by PMI. Look in government agency locations – especially NASA.