

## Stretching Agile to fit CMMI Level 3

- *the story of creating MSF for CMMI® Process Improvement at Microsoft Corporation*

**David J. Anderson,  
Program Manager, Microsoft Corporation**

**June 2005**

### **Abstract**

Agile practitioners pride themselves on highly productive, responsive, low ceremony, lightweight, tacit knowledge processes with little waste, adaptive planning and frequent iterative delivery of value. It is often assumed that CMMI compliant processes need to be heavyweight, bureaucratic, slow moving, high ceremony and plan driven. Agile developers often skeptically perceive formal process improvement initiatives as management generated inefficiency that gets in the way of productivity. At Microsoft, we've adopted the teachings of W. Edwards Deming and stretched our MSF for Agile Software Development method to fit the requirements for CMMI Maturity Level 3. The resultant MSF for CMMI Process Improvement is a highly iterative, adaptive planning method, light on documentation, and heavily automated through tooling. It enables management and organization of software engineering through use of agile metrics such as velocity and cumulative flow but with an added dimension of an understanding of variation – adapted from Deming's teachings. This is the story of how mixing Deming with Agile produced a lightweight CMMI solution for .Net developers everywhere.

## Introduction

Back in August 2004, we (at Microsoft) started to look at developing a methodology for the new Team System product which would be compatible with the Software Engineering Institute's (SEI) Capability Maturity Model Integration (CMMI) [SEI 2002]. At that time, we did not envisage that the final solution would be seen as an agile method. We assumed that the CMMI world and the agile world were like oil and water. CMMI and its predecessor the Software CMM [SEI 1993] were largely associated with the government contracting, aerospace and defense businesses – a world of large, long duration programs on often life critical systems with government requirements for auditability and traceability.

Many software developers are suspicious of process generally. Process often gets in their way and slows the pace of software development to a frustrating level. The CMMI is associated in their minds with such process initiatives. We arrived at the start of the **MSF for CMMI® Process Improvement** method project with similar prejudices.

To provide some background, Microsoft Consulting Services has offered training in project management under the brand Microsoft Solutions Framework (MSF) since 1992. MSF has a long history and a body of knowledge built up over more than a decade. MSF v3.0 incorporated learning from within Microsoft and external sources. It incorporated a lot of established late 20<sup>th</sup> century management science and embraced the concepts of team accountability, shared responsibility, empowerment, delegation, systems thinking and many other aspects of general management practice. MSF doesn't look much like traditional software engineering project management guidance. In many respects it is very agile. The MSF v4.0 team had to respect the history of MSF whilst trying to create a method which would meet the CMMI requirements.

Our interpretation of the essence of agile software development was that it is enabled by trust – the act of trusting developers to do the right thing and building trust with customers through frequent delivery and attention to feedback. This provides the slack and empowerment needed to let good things happen. On the other hand, the government aerospace and defense industry seemed to be a place which lacked trust and relied on verification through audit - a place where nothing happened without committee review and approval - approval being required frequently throughout the life cycle. Everything must be traced bi-directionally in a fine-grained manner to facilitate auditing. In addition, the empowered self-organizing nature of agile software development seemed opposed to the *plan the plan, plan the work, work the plan*, command and control approach which seemed to be called for in the CMMI. It seemed that the CMMI method needed to be a *big planning up front* method and that there was no opportunity to re-use any of the work done for the **MSF for Agile Software Development** method already in development.

Some previous attempts to do agile within a CMMI framework, [e.g. Alleman 2003], essentially offered point solutions, such as the ability to use some aspects of extreme programming within an otherwise command and control, big planning up front, non-

adaptive environment. This seemed to underscore the difference in philosophy between the two worlds. What we sought to achieve was a full life cycle method that was truly agile but met the requirements for the CMMI (at least to Maturity Level 3). The SEI has [Paulk 2001, 2002] sought to offer up such guidance most notably this year [Konrad 2005]. However, this guidance has tended to be focused on point solutions such as XP, or high level, philosophical and difficult to implement in a general full life cycle sense. The existing literature has summarized that CMMI and agile were compatible but failed to provide specific details or examples.

We achieved our goal for an agile CMMI method by stretching the MSF for Agile Software Development method to fit. We have coverage of 20 of the 25 process areas with the exception of, Supplier Agreement Management, Integrated Supplier Management, Organizational Process Focus, Organizational Environment for Integration and Organizational Training. Those process areas in scope at Maturity Levels 2 and 3 on the CMMI model are covered in full and those at Maturity Levels 4 and 5 have about 50% coverage.

What made this possible? What was the bridge between the CMMI world and the philosophy of agile software development? The answer lies in the teachings of W. Edwards Deming. Deming's Theory of Profound Knowledge based on an understanding of natural variation in processes and its analysis with statistical process control was the key to unlock a full lifecycle agile CMMI approach.

## Deming and Agile

W. Edwards Deming's underlying philosophy of management is based on the concept of feedback. The Deming Cycle of Plan-Do-Check-Act (PDCA). Deming uses his feedback cycle to drive change based on objective statistical analysis. In addition to hard objective techniques like statistical control, Deming also introduced softer subjective guidance based on systems thinking with his 14 points for management [Deming 1982]. Of the 14 points, those of most interest to the agile community are:

- 3. Cease dependence on quality control to achieve quality, instead focus on quality assurance throughout the lifecycle.**
- 4. Build trust and loyalty with suppliers**
  
- 6. Training on the job**
- 7. Leadership**
- 8. Drive out fear**
- 9. Break down barriers between departments**
  
- 12. Remove barriers to pride of workmanship, focus management on quality rather than production numbers**

Deming's thinking is very aligned with the agile manifesto and the practices of many agile methods. Number 3 could be compared to the practice of test first development for example. Agile is all about building trust between developers and customers (no. 4).

Practices such as pair programming encourage training on-the-job (no. 6). Support for concepts such as sustainable pace, and transparency of reporting within self-organizing teams drive out fear (no.8). Generalist roles break and collaborative team-working break down barriers between departments (no.9). Short iterative delivery cycles where everyone on the team gets to see finished work and demonstrate it to the customer every few weeks remove barriers to pride of workmanship and focus everyone on quality (no.12). That quality according to Deming, is defined by the customer and not the specification (i.e. change must be embraced if that is what the customer wanted). This is very agile.

Deming taught the notion that *quality is conformance to process* rather than conformance to specification. Rather than measure conformance to specification using quality control departments, it is better to build quality assurance into the process and measure conformance to process. The implication is that if you are doing things the right way, then the right thing will be produced and the customer will be happy. It is 2<sup>nd</sup> order management – management of the process.

## Theory of Profound Knowledge

The Theory of Profound Knowledge tries to provide an objective mechanism for measuring whether a process is under control – whether or not a team is doing the right thing. The theory is very simple. It seeks to understand the natural variation in the tasks that workers are asked to perform and measure that variation. Using mathematics devised by Walter Shewhart [Shewhart 1939] upper and lower control limits are set. If the process is varying within the control limits then the process is said to be *under control*. Under control essentially means that the process is running as designed and is not unduly influenced by external factors out of the immediate management control. When the process is varying out with the control limits then it is said to be *out of control*. In other words, some external factor has affected the performance of the process and placed the quality of the output in jeopardy.

## The Deming Epiphany

The CMMI is a model for process improvement which is intended to take an organization to a level where continuous improvement in productivity and quality is possible [Chrissis 2003]. This basic philosophy is based on the work of W. Edwards Deming [Deming 1982]. However, the 5 level model which was introduced to the original Software CMM to enable the assessment of maturity in government contractors for United States Air Force contracts was based on the manufacturing model of Philip Crosby [1979]. Crosby's name is most associated with the definition of *quality as conformance to specification* [Crosby 1979]. Hence, to measure quality one must have a complete specification at the beginning and at the end the variance from the specification is measured and quality is assessed. For software projects, *success* is measured using this quality metric, interpreted as delivering a project *on-time, on-budget, with the agreed functionality*. The Crosby influence on the Software CMM is so prominent that it is not widely appreciated that the underlying philosophy and goal of the CMMI is to achieve continuous improvement as described and taught by Deming.

The CMMI maturity model is mapped to Deming thinking. Maturity Levels 2 through 4 are all about creating the organizational capability to eliminate special cause variation and hence avoiding management mistakes #1 and #2 (see section after Figure #3), whilst Maturity Level 5 provides for continuous improvement through the gradual reduction of common cause variation. **If the CMMI was truly rooted in Deming's philosophy, it seemed to us that it must be possible to create a truly agile CMMI method.**

It was this realization, in November 2004, that caused us to abandon the work done so far on MSF for CMMI Process Improvement. We decided to start again building a process template based on our MSF for Agile Software Development method.

## CMMI Organization

The CMMI model can be viewed as either a staged representation or as a continuous representation. The model can be implemented in a *staged* fashion. This is the idea that an organization achieves maturity level 2 across all relevant practices and then maturity level 3 and so on. It can also be implemented in a *continuous* fashion where capability in individual process area is measured. An organization can achieve a high capability level in a particular process area whilst holding only a capability level 2 in other process areas. It is the staged method of assigning maturity levels that is associated with the American government contracting market and is hence best known in North America.

The CMMI model actually consists of 25 process areas. Within each process area, a number of specific goals are enumerated. Each specific goal in turn has an enumerated set of specific practices. The practices are at an expected level where they describe what should be done and what artifacts might be produced as a result. For example, Project Planning (PP) 2.1 asks us to *"identify task dependencies"* and expects a *"Critical Path Method (CPM) or Program Evaluation and Review Technique (PERT)"* [Chrissis 2003] chart as a result.

Many of these practices as documented would be unacceptable to an agile software developer. For example PP 1.1 sub-practice 2 asks us to *"Identify the work packages in sufficient detail as to specify estimates of project, tasks, responsibilities and schedule"* and goes on with *"The amount of detail in the WBS at this more detailed level helps in developing realistic schedules, thereby minimizing the need for management reserve."* It is all too easy to interpret this as a requirement for *big planning up front*. Project Monitoring and Control (PMC) 1.1 asks us to *compare "actual completion of activities and milestones against the schedule documented in the project plan, identifying significant deviations from the schedule estimates in the project plan."* This sounds like heavy planning, command and control and is antithetical to the agile concept of self-organization and postponed work allocation. Technical Solution (TS) 2.2 asks us to *"establish a technical data package"* and suggests artifacts like *"product architecture description, allocated requirements, product component descriptions, product characteristics, interface requirements, conditions of use"* and so forth. Again it is easy to interpret this as a requirement for heavyweight documentation. It also prescribes design before coding which is abhorrent to the extreme test-driven development philosophy where design is emergent through refactoring. In fairness to the CMMI, such

specific guidance is defined as “informative” and is provided only to help readers understand what is expected.

In addition, the CMMI has a reputation for heavyweight documentation. Some literature [Ahern 2005] advises that as many as 400 document types and 1000 artifacts are required to facilitate an appraisal.

## Achieving Agility with CMMI

A key to achieving more agility with the CMMI is to realize that the practices are primarily advisory or indicative only. To meet a CMMI appraisal, an organization must demonstrate that the goals of a process area are being achieved. This is done by identifying evidence of practices. However, the practices need not be the ones described in the CMMI specification. The organization is free to propose alternative practices and appropriate evidence. The appraisal team must then agree to whether this is appropriate to demonstrate the goal. This provides a process designer with a great deal of freedom.

As it turned out, MSF for CMMI Process Improvement did not require any alternative practices with the exception of the practice and evidence for specific practice REQM 1.4 which calls for full end-to-end traceability of artifacts. This could be considered onerous for many agile projects. The reason alternative practices weren't required is more to do with interpretation of intent. When one practice asks for a plan to be made and another for a commitment to a schedule, budget and specification, there is often an assumption that these commitments must be precise - that there is an underlying philosophy of conformance to plan and specification. When another practice asks for a review of planned versus actual and creation of a report detailing deviation from specification, there is an assumption that this is to audit quality as conformance to specification – any deviation being a failure in quality. Though this is a common understanding in many organizations practicing CMMI, it need not be true. **It is acceptable to document the approach as conformance to process and the measurement mechanisms to be variation aware and defined within common cause variation limits.** This is the key which unlocks the possibility of a truly agile CMMI implementation across the entire lifecycle.

Hence, plans and specifications need not define precise numbers and dates but may define a range of acceptable numbers and dates. So long as this is the communicated organizational understanding and is acceptable to the customer then it is acceptable within the CMMI specification and appraisal scheme. For example, a plan based on historical velocity data of 1.7 scenarios per developer per week with a low end variability of 0.8 and a high end of 2.5, might conservatively offer to develop 80 scenarios in 10 weeks with 10 developers and define a stretch goal of 250 scenarios. A slightly more aggressive plan might use 8 weeks with a 2 week buffer for variation and offer a minimum of  $8 \times 17 = 136$  with the same stretch goal of 250.

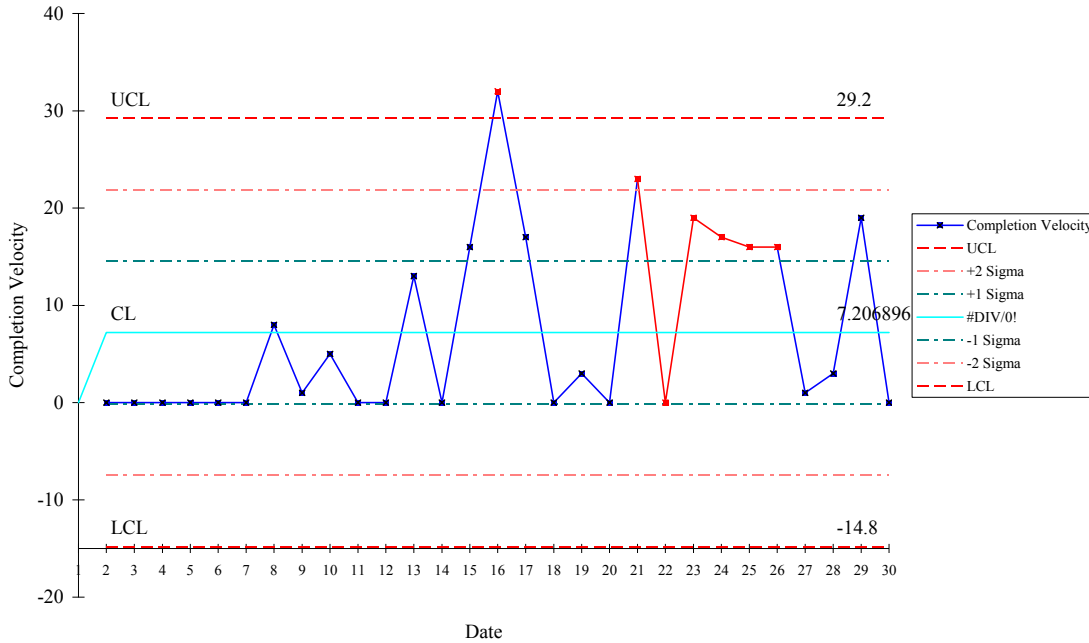
## Velocity and Queue Management

At the highest level of maturity, the CMMI calls for continuous improvement enacted under conditions of objective control. It is recommended that this be achieved using

statistical control in the fashion promoted by Deming in his Theory of Profound Knowledge. It was desirable for us to deliver a method which met the CMMI Maturity Level 3 but to lay the ground for a future version which could take our customers the whole way to Maturity Level 5. The SEI firmly believes that the economic benefits of the CMMI are delivered at the higher maturity levels. At Microsoft, we believed that our customers will ultimately want to make the transition to Maturity Level 5 and we wanted to facilitate that with a smooth incremental path using our existing process template. It was therefore necessary to consider Maturity Level 5 in our planning whilst focusing on delivering a Maturity Level 3 solution.

If we are to follow a Deming approach to achieving CMMI level 5 then there must be a mechanism to allow monitoring of statistical control. Compatibility with agile methods can be achieved by measuring velocity of production of customer valued items and using this metric for planning. In MSF value items are defined as usage scenarios [Carroll 2000]. Scenarios are not to be confused with use cases or paths through use cases. Their heritage is quite different and born out of the usability engineering field. Scenarios are concrete descriptions of usage which do not try to second guess underlying systems implementation. In MSF, scenarios, as customer requirements, are analyzed and broken into architecture, development, test and user experience tasks. Figure 1 shows the velocity of task completion (or the production rate of the process). This is very different from the more traditional project management approach of estimating time on task for each task and then tracking planned versus actual performance. The latter merely measures variation in estimation ability or lack of conformance to individual local commitment, whilst the former measures variation in the whole system of value delivery to the customer.

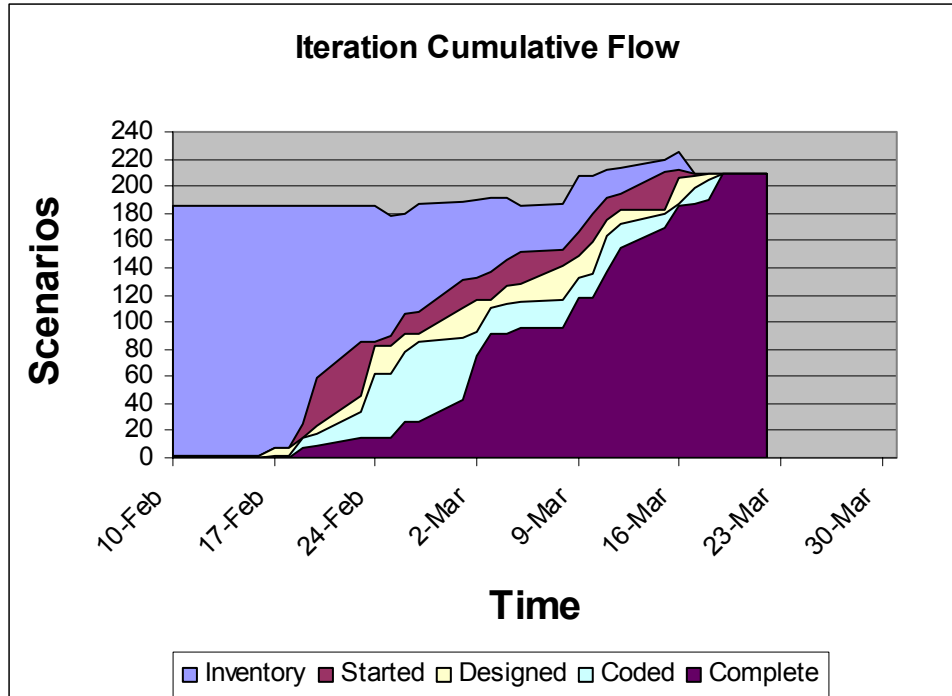
### Completion Velocity Chart



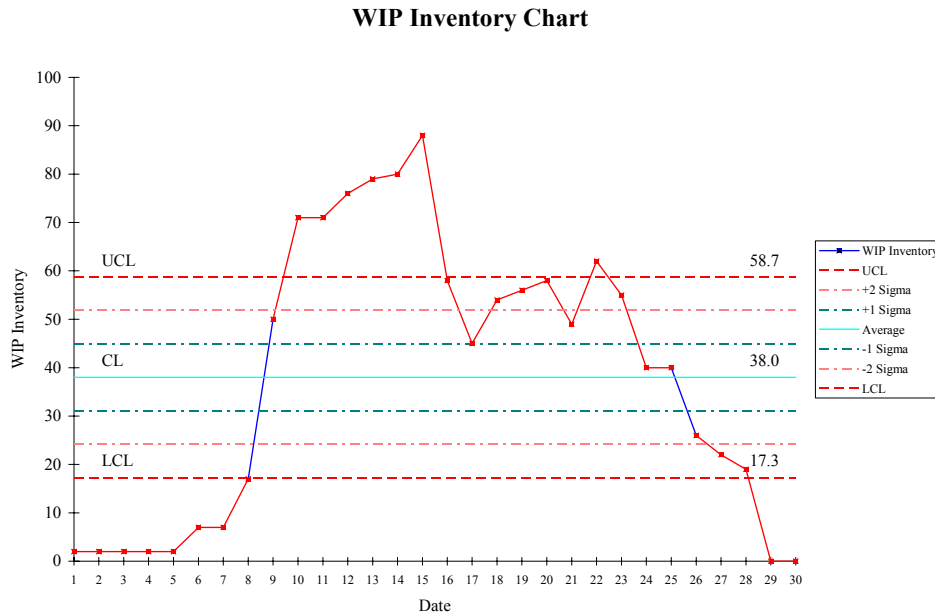
**[Figure 1. Completion velocity (or production rate) in a XMR control chart]**

The velocity measurement is not sufficient on its own to identify **special cause** variation in the process. Shewhart called special causes, “assignable causes”. In other words, there ought to be a specific assignable event that can be identified as the root cause. A record of project issues is a good source for identifying special cause variation. The issue log is storing the history of engineering requests for help. By implication requests for help indicate potential or actual special cause variation. It could be the case, that a special cause will be unidentified initially and invisible. Management can choose to investigate or simply shrug it off as unexplained. Clearly more mature organizations are better at this root cause investigation and exhibit processes under greater control.

We can track blocking work-in-process (WIP) by monitoring work queuing for processing. David Anderson [2003] showed how to do this using cumulative flow diagrams [Figure 2] based on work by Reinertsen [1997]. Monitoring the WIP level using statistical control [Figure 3] enables several things. WIP levels predict lead time, and hence, future affect on the project schedule. WIP levels can also indicate blocking work items and where in the life cycle the blockage is occurring.



[Figure 2. Cumulative flow of scenarios in an iteration]



[Figure 3. WIP scenarios shown in a control chart]

Deming suggested that there were only 2 mistakes a manager could make. He called these **Mistake #1** and **Mistake #2**. Mistake #1 is interfering when everything is normal – when the process exhibits statistical control. He called this *tampering*. When a process is under control, it should be left alone. Micromanagers have a tendency to tamper because

they react to fluctuations without considering the bigger picture. Hence, micromanagement often leads to mistake #1. When a process is under control the workers should be left to self-organize without management intervention. This concept sounds very agile. Mistake #2, is a failure to intervene when a process is out of control. A process gets out of control when something external affects it. In common language, we call these *issues*. When an issue is blocking the flow of value in an agile software development process, everyone understands that someone (usually a project manager) must be assigned to eliminate it. Methods like Scrum [Schwaber 2001] make this very explicit. Avoiding mistake #2 is all about aggressive issue management and good risk management. A risk after all is simply an issue which hasn't happened yet!

The issue log should reflect the trends in WIP. The issue log should contain details of assignable causes for special cause variations.

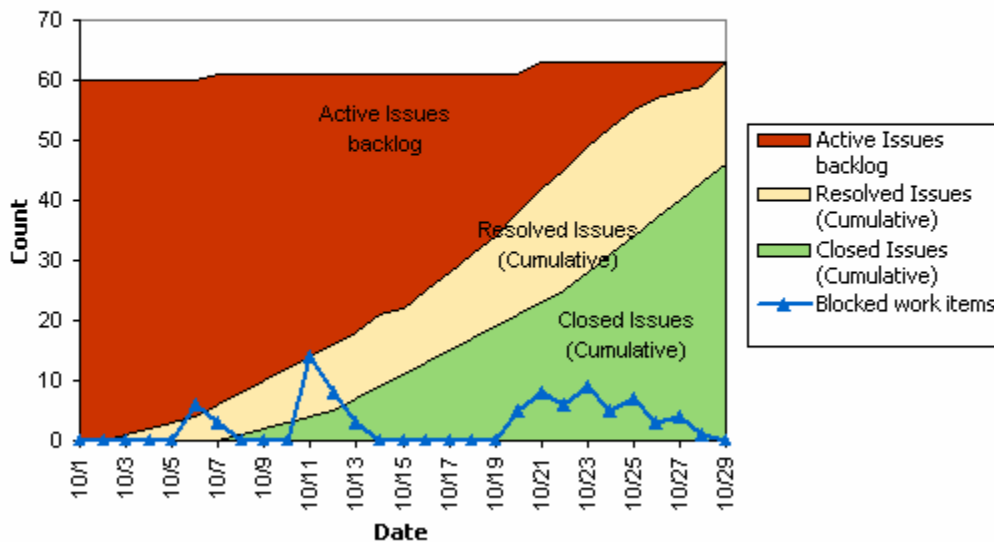
The issue log and blocked WIP can be used to assess the capability of the organization to eliminate special cause variation. Figure 4 shows the cumulative flow of issues in the issue log. Project managers will work to resolve issues and close them out. On occasion issues will not close fast enough resulting in blocked work-in-progress in the main project (or iteration) backlog. This is shown in the overlaid line graph. A lack of blocking work would indicate a lack of special cause variation, whilst the WIP inventory and Completion Velocity charts show the extent of the common cause variation.

#### Issues and Blocked Work Items

Report generated: 11/04/2004 11:25 AM by jackerman

Description...

[View report documentation](#)



[Figure 4. Issue log cumulative flow with blocking scenario inventory]

An understanding of common cause variation can be used for project planning and estimation. An understanding of the risk associated with special cause variation can be

used to pad the project plan with buffers (effectively, *insurance*) against schedule slippage, or delivery of reduced functionality, due to velocity slippage.

## Iterative and Adaptive Planning

By defining a set of agile metrics but constructing them within a framework of statistical control, we have laid the ground for an agile method compatible with CMMI Maturity Level 5. What is needed now is an iterative and adaptive planning mechanism that would be recognizable as agile.

The solution is to provide a loose project plan which approximates the scope of a whole project and lays out a plan for a series of iterations that define approximately what will be developed in each one. The secret is to avoid big planning up front. We achieve this by defining a small series of critical to quality (CTQ) end-to-end scenarios and storyboarding each one. As stated earlier, quality is defined by the customer. It's important to gain an early, if somewhat approximate mutual understanding of this. The customer must agree that these CTQ scenarios represent the broad intent of the product or service envisaged as the deliverable from a project. The CTQs are then assigned against iterations in the project plan. Hence, the project plan is very loose and approximate but detailed enough to communicate to the customer the essence of what they will get at each stage in the project. It's important that the customer communicate what represents a suitable iteration cadence from their perspective. The natural business cycle is likely to dictate how often the customer can absorb new functionality and realize value from it. Hence, the customer will determine the iteration length. A commitment to a project plan is therefore possible from all stakeholders. This commitment is a critical part of the CMMI requirements.

Detailed planning is then left to the start of an iteration where a set of specific detailed scenarios are fleshed out around the previously identified CTQ scenarios. Acceptance tests are written for each scenario and together the requirements and tests form the scope of the iteration plan. Scenarios are ranked using MoSCoW rules [DSDM 2005]. The metric for velocity and variation is used to define a minimum delivery level for the iteration, based on the low end of the capability range and a stretch goal based on the high end is defined. "Must have" requirements are assigned against the minimum level and "should have" or "could have" requirements against the stretch goal. The plan is then agreed by all stakeholders and commitments are made.

During the iteration, the engineers are asked to continue to focus on conformance to process, using the velocity and cumulative flow reports along with the issue log and blocked WIP report. Attention will be paid to variation in velocity. A precise estimate of the scope that will actually be delivered can be made on a daily basis. It should continue to fall within the bounds of the minimum level and the stretch goal, or there should be an assignable cause to explain why not. Management attention can then be focused on such problems and recovery actions initiated. By doing so, requirements for root cause analysis and corrective action in CMMI are covered.

Again, even at the iteration level, planning remains loose. The plan for an iteration contains only a backlog sorted into “must have” requirements which make up the minimum commitment level and others which represent a buffer for the natural variation in the software development method being used. The team is empowered to self-organize within an iteration. The plan is calculated using averages from historical data. When no history is available the plan should be based on a best guess and monitored frequently using the reports shown in figures 1 through 4. The plan can be adjusted accordingly. This only ever affects the first iteration of a new project with a new team.

## **Stretching MSF for Agile Software Development**

What it lacked was some formality in specific guidance for some process areas. In many agile development methods, much of the infrastructure to develop software reliably is either implied or left to resolution through a general purpose issue management method. For example, if a lack of a configuration management environment is a blocking issue then the resolution would be to put one in place. The CMMI, based on decades of software engineering experience, already includes many of these aspects. Creating appropriate environments, commissioning appropriate tools and initiating projects with sufficient vision, communicating that vision with launch events, and planning for the realization of value through deployment, are all part of the CMMI prescription.

There is nothing particularly at odds with the agile philosophy in these things. The main difference is that they are spelled out explicitly.

It was therefore necessary to enhance MSF for Agile Software Development with additional activities to cover these aspects of the CMMI. This was non-trivial. The footprint of the guidance material for MSF for CMMI Process Improvement is 150% larger than that of MSF for Agile Software Development. So some large amount of stretching was necessary. For example, MSF for Agile Software Development has 25 work product artifacts whilst the CMMI method has 59. There are 9 metric charts with the agile method whilst the CMMI method has 12. However, a typical CMMI process implementation has over 400 artifacts [Ahern 2005]. A combination of integrated tooling and an overall agile approach to the process design has reduced the overhead – the heaviness – in the process by around 85%.

The result is a process template which is larger than a typical agile process with slightly more formality with respect to approvals, sign offs and audits, but one that has an agile, adaptive core and is lightweight in comparison to a traditional CMMI approach. We believe that MSF for CMMI Process Improvement represents an agile, highly productive methodology which includes a formal quality assurance mechanism and encourages a culture of continuous improvement.

## **Conclusions**

CMMI process implementations are often associated with conformance to plan, low trust environments, with command and control structures. These require a big design up front approach with auditing of conformance and by implication punishment for non-conformance. This generates fear and fear encourages greater focus on more detailed

planning coupled with a blind willingness to follow such a plan. As a result, the customer's needs become disconnected from the work and customer driven change becomes unacceptable because change must be reflected in the plan in order that the result conforms to the plan. The effect is a great deal of non-value adding rework.

This paper shows that it is wrong to associate these undesirable software engineering behaviors with the CMMI. It doesn't have to be that way.

By embracing the teachings of W. Edwards Deming and understanding their relationship to agile principles and practices, it is possible to develop a truly agile full life cycle process which meets the requirements for all 5 Maturity Levels in the CMMI model. Specifically by using agile metrics such as velocity, cumulative flow and trends in open issues, we have designed planning and monitoring methods which embrace variation and allow for postponed, late commitment and adaptive iterative planning.

By doing so the process embraces the agile manifesto [Cunningham 2001] ideas such as valuing **customer collaboration over contract negotiation** and **responding to change over following a plan**. In addition, by embracing variation and accepting it as part of the process, the people aspect of **individuals and interactions over processes and tools** is embraced. Through use of iterative cycles agreed at a cadence acceptable to the customer, the first half, **working software over comprehensive documentation** is embraced.

## References

- [Ahern 2005] Ahren, Dennis M., Jim Armstrong, Aaron Clause, Jack R. Ferguson, Will Hayes, Kenneth E. Nidiffer, *CMMI SCAMPI Distilled – appraisals for process improvement*, SEI Series, Addison Wesley, Boston, MA 2005
- [Alleman 2003] Alleman, Glen B. and Michael Henderson, *Making Agile Development Work in a Government Contracting Environment – Measuring Velocity with Earned Value*, Agile Development Conference, Salt Lake City, Utah, June 2003
- [Anderson 2003] Anderson, David J., *Agile Management for Software Engineering – applying the Theory of Constraints for Business Results*, Prentice Hall PTR, Saddle River, New Jersey, 2003
- [Carroll 2000] Carroll, John M., *Making Use – scenario-based design of human-computer interactions*, MIT Press, Cambridge, MA 2000
- [Chrissis 2003] Chrissis, Mary Beth, Mike Konrad and Sandy Shrum, *CMMI – Guidelines for Process Integration and Product Improvement*, SEI Series in Software Engineering, Addison Wesley, Boston, MA 2003
- [Crosby 1979] Crosby, Philip B., *Quality is Free*, Signet (Re-issue edition) 1980
- [Crosby 1995] Crosby, Philip B., *Quality is Still Free – Making Quality Certain in Uncertain Times*, McGraw Hill Harvard Business School Publications, 1995
- [Cunningham 2001] Cunningham, Ward et al, *The Manifesto for Agile Development*, <http://www.agilemanifesto.org/>
- [Deming 1982] Deming, W. Edwards, *Out of Crisis*, The MIT Press, Cambridge, MA 2000
- [DSDM 2005] DSDM Consortium, *MoSCoW Rules*, Dynamic Systems Development Method, Version 3.0, <http://na.dsdm.org/en/about/moscow.asp>

- [Konrad 2005] Konrad, Mike, and James W. Over, *Agile CMMI: No Oxymoron*, Software Development Magazine, March 2005
- [Paulk 2001] Paulk, Mark C., *Extreme Programming from a CMM Perspective*, IEE Software November 2001, <ftp://ftp.sei.cmu.edu/pub/documents/articles/pdf/xp-from-a-cmm-perspective.pdf>
- [Paulk 2002] Paulk, Mark C., *Agile Methodologies and Process Discipline*, Crosstalk, October 2002, <http://www.stsc.hill.af.mil/crosstalk/2002/10/paulk.html>
- [Reinertsen 1997] Reinertsen, Donald G., *Managing the Design Factory – A Product Developer’s Toolkit*, The Free Press, New York, NY, 1997
- [Schwaber 2001] Schwaber, Ken and Mike Beedle, *Agile Software Development with Scrum*, Prentice Hall PTR, Saddle River, NJ 2001
- [SEI 1993] Paulk, Mark C.; Curtis, Bill; Chrissis, Mary Beth Chrissis, and Weber, Charles, *Capability Maturity Model for Software Version 1.1*, Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403, February 1993.
- [SEI 2002] Software Engineering Institute, *CMMI® for System Engineering, Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Staged Representation (CMMI-SE/SW/IPPD/SS, V1.1, Staged)*, Carnegie Mellon University, Pittsburgh, PA 2002
- [Shewhart 1939] Shewhart, Walter A., *Statistical Method from the Viewpoint of Quality Control*, Dover Publications, Mineola, NY 1986