–"...your article contained a lot of important "lessons learned" and great points that organizations implementing CMMI should hear!" Mike Konrad, SEI

Energizing CMMI

C. C. Shelley

April 2010

Oxford Software Engineering
9 Spinners Court, 53 West End, Witney, Oxfordshire, England, OX28 1NH
shelley@osel.netkonect.co.uk

ABSTRACT

CMMI is an influential and widely used model for process improvement, yet it often fails to deliver the expected benefits. The origins of the model are examined and the reasons for this disappointing performance explored. Approaches to software process improvement that mitigate many of the risks inherent in model based 'Big SPI' are described and a set of good SPI 'rules' that capture the essence of these approaches are proposed.

1 Introduction

A discussion by software process engineers concerned at the state of software process improvement and the use of CMMI prompted the drafting of ten 'rules' of good SPI (listed at the end of this paper). These were developed into a webinar and later a BSC SPIN SG seminar in the summer of 2009. This paper is a development of that seminar:

CMMI an influential software process model, recognized across the industry. It has been used world wide by many diverse software organizations as a framework for software development and management processes, and as a template for process improvement.

Along with this recognition CMMI is also gaining a reputation for poor return on investment. Despite careful planning, methodical training, and ongoing appraisals and process development the anticipated outcomes can be slow to manifest and unexpected dysfunctional behaviours may appear. Something is not right. Too many organizations are taking too long, or failing to reach their potential with CMMI.

There is little wrong with CMMI itself. It has captured and organized the experience and knowledge of the best software organizations, but the way this knowledge is interpreted and applied, how a software organization actually uses it needs considerable care. Without a good understanding of the management of change in complex and subtle software development environments, and the software process models it is easy to build excessive and unnecessary risk into improvement work.

This paper reviews current software process improvement (SPI) practice and the issues that organizations undertaking SPI face. To understand these issues the origins and development of CMMI are reviewed and the reasons many of the difficulties encountered listed. A number of approaches and techniques that mitigate SPI risks and dysfunction are described and a set of rules encapsulating good SPI are presented.

2 Current state of SPI

Business continues to seek better software development capability, whether it is :

- faster delivery,
- increased responsiveness,
- reduced costs,
- better quality,
- more manageable quality,
- etc...

It also wants to demonstrate and market this.

CMMI offers both a means to increase capability and a means for recognizing this capability. It is supported by numerous suppliers of products, services.

However there are concerns. Use of CMMI has a high, but rarely reported, failure rate[1]. There are often high costs and low or negative returns on investment, which compare poorly with other, less comprehensive treatments. And there are systemic problems with the delivery of CMMI services. CMMI, and with it SPI, are being challenged by other approaches to improving software development practice.

Some of this may be due to the model being over extended, having the scope expanded and the content generalized. It may be due to the way improvement work has been distorted by the popularity of CMMI in diverse software development organizations, that are over concerned with model conformance and not sufficiently concerned with the delivery of business benefits.

To understand how this has happened it is necessary to look at the origins and history of CMMI.

3 Origins and History

CMMI was preceded by 'A method for assessing the software engineering[2] capability of contractors' (CMU/SEI-87-TR-23) and later the CMM (Capability Maturity Model). These were developed for the U.S. Department of Defense as risk management tools for the selection of dependable software suppliers. They were developed with a scoring system designed to minimize risk to the department. This scoring system is unusual in that it places most software engineering organizations at the low end of a spectrum of capability with few (none in the early days) at the high end. (A more normal system that may not have distorted SPI activity so severely, would have placed the majority at a mid point with unusually poor performers and unusually good performers at extreme ends of the spectrum.) Never-the-less this scoring system worked to identify higher performing organizations, as judged by the DoD, as candidate to bid for contracts. This scoring system persists in CMMI.

With most software organizations grouped at the lower end of a capability spectrum potential suppliers responded by working to meet the high performance criteria of CMM. Thus the early use of CMM by software organizations was *not* SPI but a desire to 'conform to standard' as a qualification to bid for defence contract work. These suppliers, as sophisticated software and systems engineering corporations, would have been familiar with both the intent of CMM and with working to engineering process and product standards.

CMM use spread both as a requirement for defence contracts, with potential suppliers evaluated against CMM requirements using software capability evaluations (SCEs), and, later, as a framework for process improvement, with organizations using CMM based appraisals for internal process improvement (CBA IPIs) as a diagnostic tool to identify areas for improvement. With this use as a framework for SPI came a belief that improvement meant a move from low scores against the model to higher scores; moving from low maturity (level one), to higher maturity (levels 2 and up), i.e. higher maturity is better.[3]

The increasing influence and popularity of CMM encouraged the spread of CMM and SPI services across industry sectors, spilling out from defence contractors and software engineering, to the wider IT community in the commercial and financial sectors, and world wide, from the U.S. to Europe and then globally.

The success of CMM also triggered the development of other models, both by the SEI, that developed other CMMs, and other organizations' 'me too' models: Bootstrap, SPICE, Trillium and in house variants of CMM.

The SEI decided to consolidate its family of CMMs into a single integrated CMM, the CMMI. This model has an expanded scope, including systems development as well as the integration of teams and supplier management. It also adopted a continuous representation borrowed from SPICE. This continuous representation is rarely used. The failure of this continuous representation is interesting and significant. While it is valuable as an approach to SPI there is no formal way of recognizing or communicating improvement, unlike the staged representation with is achievement of maturity levels.

The number of products and services related to CMMI, and the number of service providers has increased to service a world wide customer base.

At the time of writing, on the positive side:

– CMMI has global recognition and use as a model of software practice and improvement,

[1] Where 'failure' can mean a number of things, typically: failing to reach a level of maturity, taking too long or costing too much to achieve a recognized level of maturity, not achieving the benefits expected at a given level of maturity, or, having achieved a level of maturity or performance, failing to sustain it.

[2] Note: 'engineering' – not development

[3] CMM (and later CMMI) and SPI are now seen as almost synonymous, but SPI was already in place in the software industry, (mostly large US corporations) before the advent of CMM. CMM provided a useful, systematic approach to understanding and improving software engineering practice and popularized SPI to the extent that they are now difficult to disentangle.

– it provides an excellent framework for understanding software development and management,

– introducing CMMI can act as an incentive and motivator for SPI , at least initially,

– CMMI conformance is required by many software customers, introducing good practice to places it would not otherwise have done.

But:

– CMMI conformance is required by many software customers, introducing it to places where it is not understood or wanted,

– CMMI services are aggressively marketed and sold beyond their areas of applicability, and with unrealistic statements of potential benefits,

– unquestioning belief that model conformance automatically means better performance is misplaced and causes loss of trust in the model when this is found not to be so,

– there is increasing concern with the poor ROI and high failure rate,

– genuine SPI is being distorted or marginalized as process improvement becomes a 'tick box' exercise to ensure conformance.

These problems are not intrinsic to CMMI. The CMMI model retains much of value from CMM. (However it would have been preferable to refine and elaborate understanding developed with the CMM, rather than generalize and extend it.) It has a rich, if poorly reported, history, and many lessons have been learned. Many other software and quality models and standards - TQM, ISO 9000-3, RUP.. - have suffered similar problems, and the agile community is beginning to encounter the same with the over selling of their models.

4 SPI Issues

SPI is perceived as *risky* (and, if mismanaged, it is.) It can be very expensive, time consuming, and ultimately unattractive to technical staff and managers, and deliver very little. Why is this?

– CMMI (and other models) are often poorly understood , misinterpreted, and misused leading to poor SPI practice,

– the models may now be too large, complex or subtle,

– they are necessarily incomplete, but this may not be recognized,

– they are often treated as a design to be 'implemented', rather than as *part* of a set of software process requirements,

– it is presumed that increasing CMMI maturity (conformance) automatically brings better performance[4],

– CMMI/SPI work is often treated as a (big, expensive) project with synthetic, unrealistic and incomplete project objectives – e.g. ML3 in two years and year on year 'productivity' improvements,

– poor or incorrect mapping of the model to the organization, or, more alarming, the organization to the model,

– assessments, originally a very effective diagnostic and investigative tool, have now become expensive, high stakes audits or conformance focussed 'gap' analyses,

– performance improvement is obstructed as poor processes are over controlled or 'frozen' to sustain presumed conformance and capability,

– CMMI and SPI specialists are frequently placed in ambiguous positions. Presumed to have more software process knowledge than they do, with too much influence and having to deal with conflicts of interest they too often, and with the best of intentions, do more harm than good,

– CMMI and SPI specialists, together with their clients' management, are far too focussed on models and model conformance, and are remote from the effects (good or bad) of their SPI work.

5 What to do about this?

Many of the problems encountered in model based SPI work are due to a failure of the SPI community to share an understanding of the character of the models and the nature of SPI. It is not a simple matter of encouraging imitation of the models or requiring staff to mimic best practice.

SPI is essentially exploratory, this needs to be understood and embraced. Recognize the nature of SPI and plan and act accordingly. If not then it becomes extraordinarily risky.

[4]Capable organization will be found to to conform to CMMI requirements, but this does not mean that organizations conforming to CMMI requirements will be capable – *from their business perspective.*

However, it is essential to be realistic and acknowledge the expectations of most investors in SPI and CMMI. Many believe, and will continue to believe that model conformance and high maturity automatically delivers, often unspecified, improvements to development and management performance, and they will have made major investments in those beliefs. 'Big SPI', organized as high profile, closely managed projects will continue, with CMMI, or other models, imperfectly understood or interpreted and trusted to deliver unclear and unrealistic benefits.

Genuine SPI is different. It has different values and motivations originating from before CMM and maturity levels. It places business objectives first, explicitly, and model conformance second – always[5]. It may use the models, but not necessarily conform to them. They are tools to be used.

Real SPI work has a different structure and character to 'Big SPI' projects with their schedules, milestones and conformance objectives. The work is exploratory. Changes are many and small, but with compounding, and occasional major benefits, and with readily acknowledged, carefully studied failures. Improvements or changes are developed and delivered collaboratively, as a continual stream. Changes are made fast, the results are evaluated, based on evidence and data, and shared. Learning is critical. Real SPI is part of 'business as usual'. And always SPI helps technical staff and management to do a genuinely better job. *This is the only and proper purpose of SPI.*

This real SPI often has to work within the context of Big SPI, conformance oriented projects. It is Big SPI that gets senior management's attention, gets the funding and resourcing, and invests in motivating technical staff and managers. This is difficult but can be done - and when it is reduces the risks inherent in Big SPI, conformance projects.

While Big SPI and real SPI are not mutually exclusive, co-existence can be difficult: there will be tensions when Big SPI project schedules and requirements to demonstrate conformance distort the work to find better ways of working. Exploratory methods, learning and failures become difficult to tolerate, and pressure is exerted to roll out standard, model conformant processes.

The next section describes a number of approaches and techniques that exemplify real SPI and that should be integrated into SPI initiatives, reducing the risks inherent in Big SPI projects.

[5]This may sound odd. Big SPI invariably claims to place business objective first. But this claim presumes that conformance – attractive in itself for commercial or contractual reasons - will deliver better (but unspecified) ways of working. Real SPI requires these better ways (more predictable, more responsiveness....etc.) to be identified explicitly, and their achievement to be validated, preferably measurably.

These approaches help direct software improvement away from unrealistic trust in models to deliver unspecified improvements and unthinking conformance, towards identifying and solving software development problems that get in the way of business objectives, towards using the models, and, in the process, improving the probability of achieving Big SPI objectives.

They are not intended as a complete set, or an end to end methodology, but do illustrate the nature of real SPI where:

- engaged software professionals...
- ...and managers,
- do useful work that delivers real, early, ongoing and compounding benefits,
- while developing a genuinely better software development capability,
- and earlier achievement of a robust and demonstrable capability.

6 Some Approaches

1. Identify the business's technical objectives: Identify the real, technical objectives that will be delivered by the process improvements. Model conformance is sometimes a given; needed as a qualification to bid for work, or for marketing purposes, but it is essential that the changes made to achieve this conformance be beneficial to technical and management practice too.

What needs to be improved, and by how much? Process improvement depends, critically, on an understanding of what 'improvement' really, and specifically, means. The desire or need to demonstrate conformance to a process model may be of some, but only some help, as a motivator, but can positively hinder efforts to provide technical staff and managers with more effective working practices unless there is a realistic idea of what 'better' is.

It can take time to convince senior management of the need to identify technical improvement objectives. Disturbing the notion that 'conformance = better' can be difficult. Where an organization's process improvement efforts are focussed solely on model conformance it may make the technical objectives more compelling and achievable if a vision of what it will be like when conformance is achieved can be developed. Ask why ML3 (say) is required. What it will be like. Consider developing a detailed and shared vision of before (now) and after (conformance) is achieved.

Try to find out who really cares about the business objectives. Ask managers to prioritize 'faster, cheaper, better'. Just occasionally you will received a very clear answer, not just equivocation, but perhaps a clear need for better predictability, responsiveness, reduced rework, improved system maintainability, or other

attributes that are important to the success of the organization. When these are recognized and shared they can be aimed for and achieved by an genuinely improved (and model conformant) software capability.

Look at the reports from post implementation reviews or retrospectives. What are they telling you? Patterns or trends – persistent problems arising time after time are signalling business tensions and drivers (And if you don't have good retrospectives in place then get them started now.)

Another approach to get a clear and shared view of business objectives is bootstrapping, using part of CMMI itself. CMMI expects management to articulate their organization's 'policies' on various aspects of software development. This CMMI requirement gets management to think about what they want from their software development, and why – this is one of the best uses of CMMI.

Policies should be drafted, shared for comment and revision, published and maintained. They need to be exceptionally clear and unambiguous; to be specific and compelling statements of *what* is wanted.

2. Identify problems getting in the way of the business objectives: A good first step to identifying problems to be fixed by process improvement is to return to retrospectives. (You do have them in place now, don't you?). Recurring problems are symptoms of business drivers being impeded by technical issues. These these are good candidates for fixing.

An investigation or appraisal is another means for identifying problems.

Find owners for these problems. This will not be difficult in a small organization or department, but in larger organizations where people will tend to be more cautious, and unwilling to rock the boat finding someone who wants the problems solved can be a problem in itself. Find out who suffers from the problem – these are the natural owners. (And if no one is interested in the problems it may be worth considering why SPI is being undertaken at all.)

When problems have been identified group these with the objective they are associated with. This provides a scope for the remedial work and traceability when someone asks 'why are we doing this?'. Discard any problems that do not map onto the business's objectives.

3. *Identify and agree fixes to the problems* – and fix them: Retrospectives again. During retrospectives solutions (frequently startlingly good solutions) to problems will be volunteered.

And explore CMMI, or your favoured model, for candidate solutions. This is another one of the more useful ways to use these models – mine them for their ideas and wisdom.

Envisage what it will be like when the problem is fixed, characterize 'before' and 'after'. And it is highly desirable to quantify this difference. This act of quantifying is a further opportunity to discover more about the problem (and an approach for this is described within CMMI).

Take care to partition the work to identify and implement fixes into small manageable solutions, each of which is capable of either a) delivering a small benefit, b) or failing, or c) delivering an unexpectedly large benefit. Plan many small, fast exploratory steps – try and see, try and see. Failures are not expensive or critical and can be valuable – providing insights and suggesting other solutions[6].

4. Begin working with those that need the fix, or are keen to find better ways of working. The success of process improvement work is critically dependent on the attitudes of those involved. When working to identify solutions and make fixes involve those that directly benefit from the fix, or are keen to find better ways of working. This enthusiastic and resilient minority of innovators and a early adopters are the natural owners of the improvements, and will work to make the fixes successful, and in making process improvement a success. Success breeds success and the more conservative staff can then begin to get involved with increased confidence that they are not making a mistake.

This approach also lends itself to evolutionary and incremental improvements and piloting. However, do be aware that the success of early work may in part be due to the enthusiasm of the early users and things may not work quite so well when the wider development community get their hands on the new ways of working.

5. Don't manage the work as a project: There are two reasons for this alarming suggestion. Firstly, projects are prone to project management. They are planned, costed, budgeted., scheduled, resource constrained,

[6]The set of approaches above, if considered in sequence: identify objectives, then obstructing problems, and then find and make fixes, may appear obvious. However, experience shows that there is a strong tendency by those involved in this type of work to select and plan ambitious and expensive solutions when both the business objectives and the obstructing problems are presumed, but still unclear, especially when the solutions are attractive or when the objective appears clear and agreed – model conformance, say. In both cases SPI will run into difficulties. (The similarities to software development undertaken without a clear understanding and agreement of the requirements are clear.)

and are expected to converge on a successful outcome, and then finish. And second, they compete with other projects for resources, and need to be appear successful.

SPI is different. It is systemic, inclusive, collaborative, exploratory, ongoing, open ended, and rooted in reality.

The two approaches are not mutually exclusive, but the desirable attributes of SPI will be far more difficult to achieve if a classic project oriented approach is adopted. By adopting a project oriented approach the exploratory approach with its acceptance of failures, and learning transforms into waste and uncertainty and risk (to the project) threatening the schedule or budget. Risk mitigation will stifle the work. There is also a tendency for projects to drift towards a waterfall type approach. It isn't inevitable - iterative and incremental approaches can be project managed - but the drift to BDUF is always there.

SPI does need to be directed and controlled, but as part of 'business as usual', integrated and adding value to every day work, and with no overt need to be perceived as 'successful'.

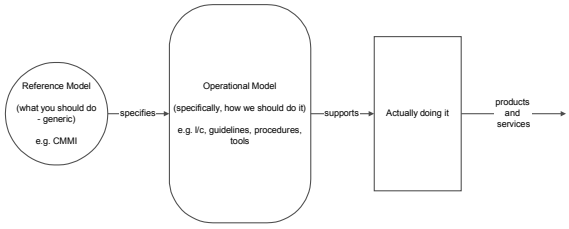
Consider other approaches to managing technical work: perhaps a programme with a simple, fine grained structure, setting up and dissolving PATs, or an index of process improvement change requests, or iterations or improvement cycles, that can be shaped and directed as business requirements change (which they will). Use means that avoid fragile project plans and manage the work without invoking the full majesty of project management.

6. Use a framework for problem solving and delivering improvements: The reason for this is similar to the reason for using a lifecycle for software development. A simple shared framework for SPI problem solving, understood and shared by those engaged in this work, provides a familiar way of working, in which to structure work and information. It provides a degree of familiarity and confidence to SPI workers that may otherwise feel uncertain about the work and wary of the reactions of others. A good improvement framework allows SPI workers to maintain integrity, and it removes blame from occasional failures. And, by providing this template structure for the work it eases planning and management. The well known PDCA (Plan Do Check/Study Act) model is designed for just this purpose. Our own TCM framework is based on this. But it is perfectly acceptable to use an existing change management system with the change control process to provide a structure for many, modest changes. (But remember it is not the framework or associated methods that deliver the solutions, it is the people using them.)

7. Use the models – but understand them, and know who's in charge: CMMI and other software process models contain many valuable ideas and insights. They provide a structure for software process knowledge, and (more problematically) a template for process improvement plans.

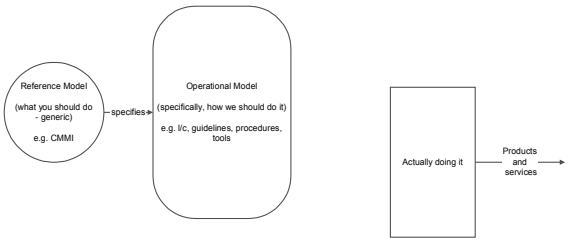
These models are reference models and necessarily generic. To have any value they require mapping onto the organization's operational model, which should be a reasonable representation of actual practice, and certainly not the other way round. An organization's operational model is more important than the reference model, providing a process baseline and shared basis for understanding and supporting actual practice. With a good mapping the software organization's development problems and process improvement opportunities can be traced back to the reference model, which can then be mined for ideas and potential solutions. To do this requires a good understanding of the reference model *and* the organization's operational model, *and* actual practice *and* their relationships to each other.

It can seem easier to short cut this, especially if the operational model is ill defined, incomplete, inconsistent, or non-existent. It seems less messy, more logical, and quicker to use the reference model as a specification for an operational model which is then developed, baselined and 'rolled out'. Technical staff and managers are then trained and required to use the new conformant operational model:

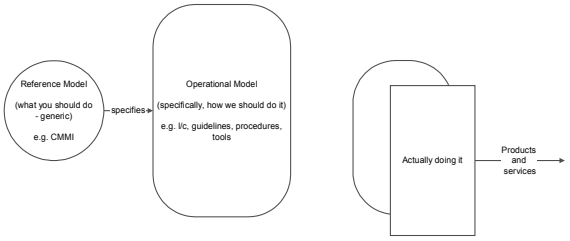


This top down approach rarely works and indicates a misunderstanding of what a capable software organization really is. It sounds reasonable, but takes little account of the subtlety and complexity of actual development practice or the 'soft' socio-technical infrastructure. While the reference models recognize this they are too generic and incomplete to provide an adequate specification for operational models, which need to reflect business needs and capabilities, and relate to actual practice and behaviour.

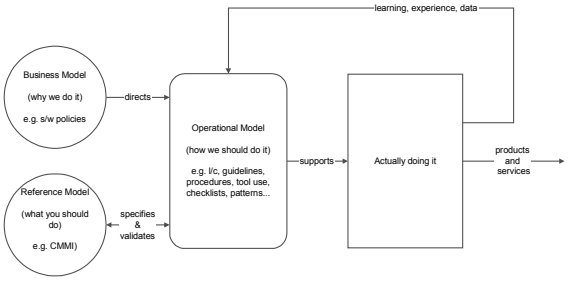
An operational model, developed from a reference model alone (often by well meaning software model or SPI experts) will be found to be incomplete and unrealistic, with little in common with actual practice and providing little support to technical staff who will recognize this and reject it if they are to work effectively:



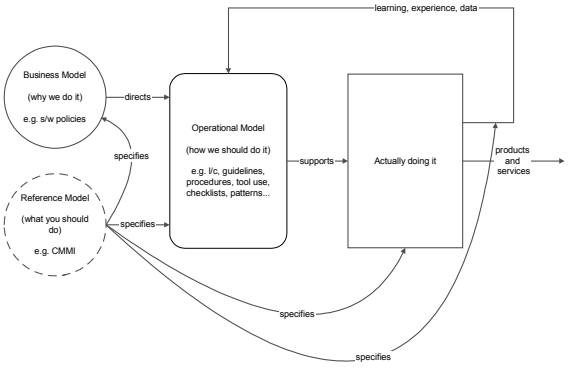
...and who may well have their own ad hoc operational model hidden away from those who would seek to improve it...



However if CMMI is understood and *used*, not just copied, to inform the way in which the operational model is developed, and if it is used to guide the development of the supporting infrastructure something altogether different emerges:



The operational model is shaped by data, learning and experience from actual practice, and by the business model (perhaps as software development policies) as well as by a reference model. Ways of working are supported by a useful functioning system, supporting and sustaining development practices directed to meet business needs and capabilities, *guided* by the reference model - as intended by CMMI:



Directing attention from the reference model, to business needs and real experience, and putting in the infrastructure to make the operational model 'live' has the effect of making the achievement of reference model conformance far more likely.

8. Measure Progress: The reasons for measuring progress are straightforward – to provide visibility and control of SPI work, and to enable a true evaluation of what works and what doesn't. The visibility and evaluation are particularly important. The rationale of SPI is improvement. It is fundamental to be able to say whether a change made to ways of working really has been an improvement, or not, and the only way to do this – apart from expressing an opinion – is to measure the effect.

Visibility is important too because SPI is a matter of changing behaviour and expectations. If work performed and results achieved can be shared then the value of the work is amplified. Good data is one of the best ways to communicate what is happening.

The need for control may be more contentious. Managers will desire control, but the extent to which this type of work can be controlled – rather than directed and encouraged will vary widely, depending on the nature of the organization and the calibre of the staff.

The most useful measurement technique is, predictably, GQM, but with a change. GQM originated in an abnormally capable software environment and it reflects that. It was used to develop top down systemic measurement systems. In most organizations this approach tends to be risky[7] and triggers dysfunctional measurement. But when it is applied to locally and tactically then it is very useful indeed. As a guide, the application of measurement for SPI is most effective when similar to that recommended for TQM and TQC type work[8].

Some top level, overall measures of process improvement will be needed too. Use GQM at a high level to develop simple and easy to interpret measures: for example changes in defect numbers, improvements in delivery predictability, reductions in rework, numbers of fixes made, numbers of failures too, and lessons learned. These and similar data, can be produced to meet a clear, traceable information need of senior managers.

Inevitably, if the work is being undertaken within the context of a process improvement project, progress towards achieving conformance will be wanted. Such measures are straightforward. Use evaluations, formal

[7]See 'eXtreme Measurement: or Why Software Measurement Fails and what to do about it', http://www.osel.co.uk/papers/eXtremeMeasurementpaper.pdf

[8]Humphrey noted, in the original CMM book that '...CMM is therefore an application of the process management concepts of Total Quality Management.'

or not, to determine the number of practices satisfied, and the rate of progress – but only as a secondary measure, subordinate to other measures of business benefit delivered.

SPI work is measured by progress made – quality and value of changes made, not by degree of conformance, or against a schedule terminating with an appraisal that is expected to demonstrate conformance.

9. Keep going: If the SPI work is delivering real benefits to the organization, if it is delivering value for money - why stop?

7 The ten rules of SPI[9]

These 'rules' prompted by concern about current SPI practice formed the basis for a webinar, a BCS SPIN seminar in 2009 and the development of this paper. They were not expected to be complete or correct, and few will agree with all of them, but they have succeeded in provoking debate and discussion about the character of SPI and its objectives:

1. Concentrate on fixing real problems getting in the way of business goals. If you aren't have a d****d good reason.

2. Require rapid feedback (results) on the effect of your changes: solve lots of small problems fast...

3. ...and evaluate (measure and analyse) them, and then act on them.

4. Software process improvements are owned by those that do the work.

5. Use a model to provide a conceptual framework and scope if it helps (actually, experience shows that two are better). Know how to use it, and who's in charge. Don't let model conformance become the primary objective.

6. Don't manage SPI as a project.

7. Measure progress by results, not schedule.

8. SPI is exploratory; some, many even, improvements will not work as you expect. Failures should be regarded as learning opportunities. They are more than compensated for by those improvements that work well.

9. Tactics determine strategy. That is, strategies are valueless until you know what you can actually change in practice.

10. SPI must pay for itself. Demonstrate this or stop.

[9]Now extended and elaborated. See www.osel.co.uk/ttrospi.pdf

8 Closing Remarks

Software Process Improvement has become distorted by the misunderstanding and misuse of software process models such as CMMI. This leads to disappointing results and a poor return on investment.

The models themselves are not to blame. They contain much of value but are sophisticated tools that need to be understood and used carefully and well.

A return to good SPI practice, requiring rapid feedback and analysis of results, with a primary focus on improved performance, and, perhaps, a secondary focus on model conformance will result in greater SPI success, with, paradoxically, reduced timescales and costs to achieve model conformance.

The author would welcome comments and suggestions on 'good SPI' and the ten rules.