

Energizing CMMI: The Case for Real Software Improvement

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. This paper explores the reasons for this disappointing performance and the thinking behind “The Ten Rules of (Real) SPI”

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

Unfortunately, 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.

Current state of SPI

There is little wrong with CMMI itself. It has captured and organized the experience and knowledge of the best software organizations. The observation of experienced process improvement specialists is that the problem lies mostly in the way this knowledge is interpreted and applied. Without a good understanding of the management of change in complex and subtle software development environments, it is easy to build unnecessary risk and burdensome costs into improvement work.

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

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

Business also needs to be able to demonstrate and market this improved capability, and this is frequently the driver for adopting CMMI as both a means to increase capability and a means for recognizing this capability. However use of CMMI has a high, but rarely reported, failure rate. 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.

Why CMMI fails to deliver

'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.

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.

CMMI was preceded by 'A method for assessing the software engineering capability of contractors' (CMU/SEI-87-TR-23) and later the CMM (Capability Maturity Model). Note the use of the term, "software engineering" not "software development". 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. CMM (and later CMMI) provided a useful, systematic approach to understanding and improving software engineering practice and popularized SPI to the extent that they are now seen as almost synonymous. However, SPI was already in place in the software industry before the advent of CMM.

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 its 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,
- 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.

Software Process Improvement Issues

SPI is perceived as *risky* (and, if mismanaged, it is.) It can be very expensive, time consuming, and deliver very little, so it becomes ultimately unattractive to technical staff and managers. 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. However, while capable organization will be found to conform to CMMI requirements, this does not mean that organizations conforming to CMMI requirements will be capable – *from their business perspective*.
- 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.

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

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.

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.

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.

Ten Rules of Good SPI

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 (Published in May 2009). These are 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. They are in no particular order:

1. Improvements are owned by those affected by them, i.e. those that use or perform the affected activities.
2. Concentrate on fixing real problems getting in the way of business goals - if you aren't have a d****d good reason.
3. Require rapid feedback (results) on the effect of changes (solve lots of small problems fast)...
4. ...and evaluate (measure and analyse) them, and then act on them.
5. Use a model to provide a conceptual framework and scope if you want (actually experience shows that two are better), and know how to use it, and who's in charge - don't let model compliance become the primary objective.
6. Don't manage SPI as a project.
7. Measure progress by results, not schedule.
8. Tactics determine strategy – that is, strategies are valueless until you know what you can actually change in practice.
9. SPI is exploratory; some, many even, improvements will fail. But these failures are offset by what you learn and those improvements that do work well (and a few that work spectacularly well).
10. SPI must pay for itself. Demonstrate this or stop.

For the latest thinking on these guidelines see www.osel.co.uk/ttrospi.pdf.