Software Process Improvement and Rapid Process Improvement

Software process improvement (SPI) is one of the many approaches used to improve the quality and predictability of software development. It has its origins in the larger U.S. organizations, especially defense contractors, of the 1980s. These organizations had, and have, ongoing needs to improve ways of working to meet contractual obligations, reduce costs and improve profitability.

As part of this drive for improved ways of working the Software Engineering Institute and Mitre Corporation were contracted by the U.S. Department of Defense to produce a model for evaluation software development organizations. This model, the Capability Maturity Model, is a synthesis of many ideas and techniques from software engineering and manufacturing. It became very successful and influential, and dominates the SPI field. Many of the beliefs and assumptions of the model's creators, implicit in the model, are carried forward into many organizations undertaking process improvement, whether using the CMM or not, such is its influence.

Some of these assumptions are:

-       software development activity is best described as processes – it is acknowledged that these include people and technology too;
-       improvement is incremental or evolutionary, in contrast to 'Business Process Re-engineering' which was popular at the time the CMM was developed;
-       improvement takes time;
-       'institutionalization' (a CMM term) makes processes robust.

There are good reasons for these assumptions, and misunderstanding or ignoring them can lead to failure to improve software development processes. However these assumptions are not necessarily valid or helpful in all situations. They reflect the origins of the CMM and the, primarily engineering and military cultures of the large organizations that inspired the CMM and were evaluated using the CMM.

There are also beliefs and assumptions about the nature of software engineers (and the term *engineers* is important) and their behaviour that may not be appropriate in the wider

industrial and commercial contexts were software development is now conducted, in the early twenty first century, twenty years after the CMM was first developed. Many organizations undertaking process improvement now do not have an engineering culture; software *developers* produce software. Many organizations are small, or new, or are reorganized frequently and have no process to improve, unlike the large corporations with large markets or major defense contracts. And development tools have developed to the extent that they can now determine processes and ways of working. In addition software processes are now being offered commercially as a commodity.

In the course of process improvement work across diverse industry sectors, and in a wide variety of organizational cultures a number of themes have emerged.

1. An ongoing and stable commitment to understanding and improving software development practices and software quality is required from senior managers. This need not be a detailed understanding of particular development practices; in fact this can be a hindrance if senior managers attempt to get too closely involved. But a clear appreciation of the value of software development processes and practices, analogous to an appreciation of manufacturing processes is required.

2. Long term process improvement plans are of little use. They are made obsolete by events.

3. 'Traditional' SPI takes too long to deliver benefits to software developers, managers or the organization. Traditional SPI planning often schedules improvements in terms of many months or years before useful results or objectives are achieved. Small or rapidly changing organizations will, quite reasonably, loose patience at these timescales, or may be reorganized or restructured before they are achieved. (It was remarked in 'The Capability Maturity Model' that "... the CMM is [...] an application of the process management concepts of TQM to software...". TQM is pervasive, tactical, results driven process improvement; something often forgotten in process improvement planning which treats process improvement in a similar manner to traditional, project oriented software development.)

4. SPI is intrinsically exploratory and risky. It can be inefficient, expensive and difficult to manage.

5. The benefits expected of SPI activity are rarely identified explicitly.

6. SPI work tends to focus on model requirements – conformance to standard for certification or marketing purposes – rather than improving the effectiveness or efficiency of working practices. There is a wide spread assumption that 'implementing' model requirements automatically delivers benefits. This is not so. Unless software processes are consciously designed to meet business needs and constraints there will be few benefits from conforming to model requirements or standards alone. The models and standards are reasonable *indicators* of improved ways of working, but conformance alone is no guarantee of improvement - but will increase overheads and costs[1].

7. Many benefits arise from small and simple improvements, made quickly.

8. 'What's in it for me?' is fundamental and must be addressed. While no one can gainsay the need for process improvement everyone will be looking for some advantage from their involvement. If none is evident support will be limited. This is not unreasonable, and can be used to validate the process improvement work: if the benefits are all related to nominal organizational benefits, but not evident in better day to day practice as seen by managers and developers the value of the organizational benefits should be questioned. (See 5 above.) Furthermore if there is no reasonable response to the 'what's in it for me?' reaction then progress in making change can be expected to be slow.

9. Some software engineering techniques are fundamental. Professional software *engineering* practices, well understood in other engineering disciplines too, are essential if working practices are to be improved. These practices may need to be recast or

---

[1] Consider two organizations working in different industry sectors, one CMM L2 and the other CMM L3. If they were able to swap software processes the CMM L2 organization would become CMM L3, improved nominally, but its processes would be unsuited to its needs. Processes *must* be designed to meet business needs – which may be *guided*, by processes models. NB This is not a criticism of the CMM specifically, it is a limitation that applies to all models and standards that tends to be sidelined by the need to demonstrate conformance.

reformulated to make them acceptable in particular organizational cultures but are never-the-less essentially the same as engineering practice.

10. 'Faster, Cheaper, Better' is a slogan, and unhelpful. For most organizations managing quality needs to come first; it is fundamental.[2] If software quality is not managed, neither can schedule or cost. As software quality comes under control improvements in schedules and costs can be expected. For organizations with good software development practices schedule, cost and quality are carefully balanced and require well-considered engineering trade offs. If measurable improvements of all three are necessary this will require investment in Research and Development (in contrast to TQM approaches where improvement is intrinsic to working practices, but cannot be expected to deliver major, predictable improvement[3].)

11. 'Think Strategically but act Locally'. Unfortunately this has become a business cliché; never-the-less it has some truth to it. In larger organizations 'global' process improvement initiatives are occasionally launched. These do have value in providing organizational incentives and can help in sharing information. However they can damage practical process improvement work as process experts spend their time liasing, setting up committees, and attempt to develop a common, enterprise wide ways of working; neglecting the day to day process improvement work. These strategic approaches can distract from useful process improvements, or deter or undermine it, and frequently fail to deliver anything useful. If a strategic initiative is underway then exploit it for what it can offer but treat it with caution. In particular do not wait for it to deliver practical solutions or useful ways of working. Useful improvements are always developed and owned locally, albeit exploiting assets from elsewhere. By acting locally, with small, frequent feedback loops, new ways of working can be tried, and results assessed, with further improvements being built on the successes.

12. Tactics determine strategy. The set of process improvement tactics and tools that will work in any given environment is quite limited. A track record of process improvement, with a growing understanding of which tools and tactics deliver the

---

[2] This does not mean high or highest quality, simply the ability to manage working practices to ensure the appropriate level of quality.
[3] Unless allied to ToC, which actively searches for and manages process 'constraints'.

best results will determine the feasibility of long term process improvement objectives. An effective way of working is to begin making tactical fixes, establish patterns of process improvement activity, finding out what works and what doesn't. This builds confidence. Then when process improvement capability is understood decide what can and cannot be achieved. In this way risks are reduced, a track record of improvement is established – together with their benefits.

Rapid Process Improvement (RPI) is a response to these observations. It is an attempt to make Weinberg's 'Law of Twins - Inverted'[4] a practical approach to process improvement supported by methods and tools.

RPI is driven by results – either data or outcomes - that are delivered by many small improvement cycles each with its data or outcomes. They need not all be good - but there *are* results, and frequently. Those involved can learn from, and are motivated by the successes, and can learn from the affordable, and therefore admissible, mistakes (forgive and remember). Because the improvement activity is small scale it can be inexpensive, easy to plan and the successes will deliver a good return on investment and necessarily short break even times.

RPI also minimizes the use of generic solutions and training. People use *their* processes not generic ones, so process improvement work focuses on specifics and making fixes, eliminating the inessential.

In particular:

1. RPI work focuses on solving real problems. If the work is not solving problems question why the work is being undertaken. The assumption is that software development is difficult and should be made easier. This is done by fixing problems. Each fix is a benefit. By focusing on solving problems to deliver benefits rather than making difficult to substantiate leaps of faith that, for example, 'Lifecycle model <X> will improve development times', or 'achievement of CMM L3 will eliminate our quality problems' are kept under control. The intent is fixes, not pious hopes.

---

[4] "Some of the time, in some places, significant change happens – especially when people aren't working hard at it."

2. Desired results are stated explicitly and when possible measurably. The expenditure of effort and difficulties encountered by changing ways of working are mitigated by a clear, shared view of what will be achieved. By stating the desired result of the work it is also possible to know when it has been achieved. Process improvement work will remain focused and not become self serving.

3. RPI driven by results. This is directly inspired by the 'plan do check act' improvement cycle, data and outcomes provide invaluable information for evaluating working practices and process improvement capability.

4. Speed: Work is carefully scaled and time-boxed. This is to make the work more manageable. (Process improvement work is at best difficult to manage with limited resources available and with operational work always taking priority.) And also to ensure that results and out comes are produced.

5. RPI is perhaps guided by a model – but not model driven. It is rare for process improvement to be undertaken without adopting a model, usually the CMM or CMMI. These valuable models provide a framework and context for process improvement but over time can come to dominate process improvement with the achievement of capability or maturity being a seductive, and often the only way of demonstrating 'improvement'. RPI attempts to reduce dependence on these models and the assumption of benefits they deliver and focuses on clearly articulated, locally determined benefits and improvements.

6. RPI work is highly visible. While it does have a technical aspect process improvement is primarily concerned with changing expectations and behaviour. This is best done when people can see what is being done; it also reduces NIH (Not Invented Here) syndrome.

7. The work is inclusive, integrating work into routine day to day work and business as usual. It encourages involvement of everyone in process improvement, and subsequent interest and commitment to good ways of working. (It also helps overcome the ongoing resourcing problem of process improvement work – there is *never* enough effort for process improvement.)

8. RPI is structured - for exactly the same reasons that software development work is structured. Frameworks for working, reduce uncertainty and risk, and make process improvement work more predictable and accountable. This is especially important for process improvement work where the work involves dealing with social aspects – getting interest, involvement and buy in; where results are uncertain, and many involved may have little experience in process improvement.

9. RPI supports local ownership and accountability. This maintains involvement and commitment to process improvement. It also helps ensure that results are useful and satisfy the 'what's in it for me?' questions.

10. RPI is opportunistic. The emphasis is on fixing problems. If a problem arises it should be looked at. Failing to consider problems will bring the rationale for process improvement into question. Never allow 'That's a CMM L3 issue, we're L1'.

These characteristics seem to be reasonable and self evident. When people are asked to comment on them the consensus view is that they seem, in general, to be a good way of working. Why are they not found more commonly in practice? There are a number of reasons that tend to drive process improvement work towards large 'project oriented' working with a focus on model requirements rather than measurable improvements in product quality, or development effectiveness:

1. Process engineers tend to be trained to understand process models and assessments and evaluations. Knowledge of models is often a prerequisite for process improvement work. The focus of process improvement becomes, by default, the achievement of excellence by conforming to the models' requirements – in the mid- to long-term - with little thought to improving performance in the near term.

2. Process engineers are often drawn from the software development community, which is excellent for ensuring first hand knowledge of working practices, but does mean that software development experiences and practices are applied to process improvement, which is less satisfactory: changing behaviour is not the

same as producing software. Process engineers can spend too much time designing and documenting new processes and not evaluating or improving existing practices.

3. Many process engineers are new to the role (most process engineers appear to spend between eighteen months to three years in the role) and have limited time to develop knowledge of process improvement tools and techniques.

4. It is easier to set or accept synthetic 'conformance to standard' objectives, rather than identify specific performance targets. Most senior managers will understand and be attracted to conformance targets rather than difficult to define, less easy understood or justify performance targets. Low level process and performance improvement is even more difficult to justify when busy managers and developers are focussed on operational matters, rather than 'overhead' activities.

5. Process improvement plans will tend to mimic project plans, with a tendency to overlong phases, distinct tangible deliverables (documents usually)- rather than changed practices, and a waterfall type approach that defers benefits until late in the project. This leads, unknowingly, towards risky and over-ambitious objectives that often only become evident late in the project (assuming the plan is not overtaken by events).

To minimize these problems sets of process improvement tools that support RPI type approaches, with continuous improvement, rapid feedback, low costs, low risks and widespread involvement, have been collected, developed and documented.

PS. With the rise of 'agile' software development methods it is striking how many characteristics are shared by these and RPI; and for the same reasons. Could RPI be considered as an agile improvement method – and this something that all SPI should exhibit?

CCS - 2009