

## The Ten Rules of SPI

May 2009: Recently we have been involved in several discussions where people (including us) have expressed discontent and concern about the current state of software process improvement (SPI). It has prompted the drafting of a list to capture the essence of 'good' SPI. Our list of rules looks like this at the moment:

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.

June 2009: After drafting these we found another set on the web by Yingxu Wang and Graham King. Similar but different ...

Rule 1: Software process improvement is complicated system engineering.

Rule 2: Software process improvement itself is a goal-driven and a continuous process

Rule 3: Software process improvement is an experiment process.

Rule 4: Software process improvement is risk-prone.

Rule 5: Software process improvement is a time varying system.

Rule 6: Software process improvement is a random system dominated by human factors

Rule 7: Software process improvement has preconditions.

Rule 8: Process improvement is based on process system reengineering.

Rule 9: Software process improvement achievement is cumulative.

They're good and cover much of the same ground, albeit with bigger words. And they pre-empt ours by ten years. But they only have nine rules and we've got ten, so we win : )

June 2009: Having started thinking about these rules I've now come up with some more – the 'other ten':

- I. Know what the problem is your working to resolve (don't start work on solutions to unclear problems)

- II. Know what the solution to a problem should look like before you start work –so you know when you've solved the problem. You should know what effect your improvement will have before you make it (and say how it will be measured). Try thinking about 'before and after' or 'as is' and 'to be'.
- III. Use PIRs (retrospectives) to find the problems, and potential solutions.
- IV. SPI is inclusive and collaborative – sharing and working with others and in teams. It isn't a solitary activity, although individuals can make major contributions.
- V. Recognize excellence and learn from it – *sixth* sigma (look to PIRs (retrospectives), technical reviews, project registers... ..?)
- VI. Management shows constancy of purpose (what about the other 13 points of Deming?) If they don't, or if they aren't interested don't let them know what you're doing, or don't do it.
- VII. The rewards of SPI are the solutions and fixes produced (not bonuses or other prizes).
- VIII. Learn to manage quality first – cost and schedule will follow.
- IX. Use a repeatable, teachable framework for SPI work to give SPI workers comfort and safety - like PDCA or TCM.
- X. Even if you know its going to take longer than you think it'll take it still takes even longer than that... ..but this isn't a problem; you're continually fixing problems, so you've got more time to fix more problems. (aka continuous process improvement)

October 2009: This next rule was prompted by some exceptional work seen in Austria. It seems odd to have this as a rule, it seems more like a tactic, but has a profound effect on the acceptability of common processes. This approach was always assumed by the author and described as a procedure in 1997 ( c –2, proc. def. on the rapid process improvement page), recorded by Ken Dymond in his CMM training and seen in action several times since. It is added to the first ten rules...

Rule 11. Document processes, practices and activities 'as is' and use this to establish a credible process baseline which can then be developed. (Do not - ever - document processes as you would wish them to be and attempt to introduce, or roll them out.)

Can these rules be grouped or categorized?

(categorize according to:

- cultural context/environment,
- tactics and tools,
- infrastructure,
- and strategies/business alignment?)

The last category suggests another rule which will seem initially to be at odds with the lemming like focus on business and customer needs, however, at the risk of calling down ridicule and contempt...

Rule 12. Focus software process improvement on fundamental software development capabilities initially (managing quality, configuration management, requirements engineering... (thank you CMM)), then when these are in place, but not before, align development capability with business and customer needs.

This is essentially a requirement to learn to walk before trying to run. This seems to conflict with rule 2. Not sure how to resolve this at present. Perhaps it is a matter of putting the tools in place before you can use them (to meet business needs).

Dec 2009: There is an 'SPI manifesto' being prepared by attendees at the last EuroSPI conference. It is still in draft and needs work to remove some ambiguities and oddities but has an interesting structure. It proposes three core 'values' under which are stated a number of principles. This could work well. The values themselves seem to work, but need refining and aligning. The principles still need the bugs removed and will no doubt provoke discussion and continue to be corrected, added to, and refined.

December 2009: The first principle identified in the draft SPI manifesto is a requirement to 'know the culture of the organization and focus SPI on their needs'. This is an excellent point. The need to understand the culture is essential.

Behaviours and expectations of managers and technical staff in, say, financial institutions and engineering organizations can be very different. This needs to be recognized, and understood, if improvement efforts are to have any chance of success. So another rule, with full acknowledgement to the EuroSPI manifesto authors:

Rule 13. Understand (and have some empathy with) the culture of the organization where SPI is being undertaken.

January 2010: *The Trust Trap*. Rule 5 (use models, but be careful) is important because of an odd effect of software process standards. Most standards can be trusted to save time and deliver results to a minimum 'standard' – that is their purpose. Simply complying to the standard delivers the desired outcome. If this was not so the effect could be serious. It would seem reasonable that software process models would do the same – that they can be trusted to deliver the desired outcome. But this does not seem to be so.

Software organizations with high levels of performance (high levels of software quality, dependable schedules, or other attractive attributes) will usually be found to align to a process model, and can achieve compliance with little difficulty. This leads, quite naturally, to the assumption that lower performing organizations would achieve similar levels of performance if they too complied with the process model or standard. But again and again software organizations having trusted models or standards to deliver the results they wanted, and been disappointed. Attempting to achieve CMM(I) (M)level 2 or 3 or, more recently, adopt agile methods usually disappoints. This is because while the models may *indicate* a high performance software process they cannot, *guarantee* it - and do not specify it either.

This suggests that the models and standards are incomplete; and not even 'necessary but not sufficient', but not even necessary. So what are they?

They are not good standards, because achieving a level of conformance or compliance does not specify or guarantee levels of performance. They are not dependable models either, being too generic. Clearly these models must be used very carefully if they are to have any value.

The value of these process models appears to be in providing a partial, indicative framework for areas of software development, management or technology that may, if carefully analysed and improved, deliver some benefits – if these are identified and specified. In effect acting as *part* of a set of software process requirements. And as with software requirements, decisions about how they are to be satisfied must be made, carefully. And as with software requirements, poor design decisions, even if they satisfy the requirements, or no decisions at all, will lead to disappointment.

If you want to benefit from these software process models you must ignore the hype and marketing and understand their limitations. They are generic and incomplete. The specifics that actually deliver the benefits must be developed on a case by case basis if you want to get the benefits. The models cannot be trusted, by themselves to specify how to improve performance – they simply point you to some of the areas where improvements could be made – but you will have to figure out how.