

# Agile Software Process Improvement

Joshua Klein ©

Senior Consultant

Process Improvement in Software Development

[j.h.klein@ieee.org](mailto:j.h.klein@ieee.org)

## Abstract

*Contrary to common belief, small groups and small projects can and should benefit from Software Process Improvement (SPI). Agile and evolutionary approaches make SPI scalable to all projects, regardless of scope. This paper presents a pragmatic method of process improvement that has emerged from the author's experience over the last five years in three software-development groups—each of a different size and in a different company. The methodology described is applicable to all aspects of SPI—technological, organizational and behavioral—thanks to its flexible and creative approach.*

## 1. Introduction

Today, the terms Software Process Improvement (SPI) and Software Engineering (SE) have an academic, rigid and complex ring to them. What's more, software developers have the idea that these methodologies are simply too time-consuming and expensive to implement for small, "pragmatic" enterprises. For SPI and SE to gain support in these settings, process improvement efforts must rapidly and continuously lead to tangible results in terms of software production.

The Agile [1] Software Process Improvement (ASPI) methodology brings about improvement in small cycles. The cycle starts with a minimal investment that yields measurable results, giving those involved confidence in the methodology, which, in turn, fuels the motivation for another improvement cycle. Every improvement achieved demonstrates the methodology's immediate usefulness to the project and inspires ongoing software process enhancement. This is an adaptive, pragmatic, fast-track approach to achieving process improvement in small steps. Gilb has shown the approach to be successful in project management.[2]

## 2. Discomfort and failure as primary trigger

Process Improvement implies change and change means resistance. To willingly accept change, people need to be convinced that it offers more benefits than risks. Alternatively, they may realize that the fear of change is preferable to undesirable results obtained when using existing work methods. The broad promise of "higher quality" is, unfortunately, not appealing enough; though many people say they are driven by quality, most are driven by schedule.[3]

Generally, after a few bad experiences, the threat of failure becomes stronger than the threat posed by change. Innumerable startups that initially underestimated software testing experienced painful failures, which consequently lead to a change in their testing policy. A disk crash causing costly data loss is a sure trigger for the adoption of configuration management. Unfortunately, we are more prone to learn, and improve, after failure than after success.

In addition to failure, the primary motivator for process improvement, discontent with the status quo can motivate the desire to change.[4] Let's not mislead ourselves, however: this is "permission to try" rather than a whole-hearted commitment to process improvement. ASPI builds from this humble starting point. From cycle to cycle, the team receives additional doses of positive feedback, leading to a progressively greater commitment to process improvement, until improvement becomes part of the company culture.

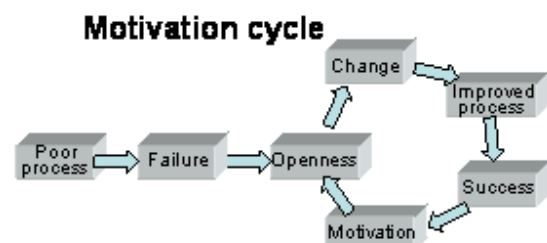


Figure 1: The motivation cycle

### 3. Agile assessment

#### 3.1 Improvement leader

In this model, "assessment" is integrated with the improvement process. The improvement leader (formerly "lead assessor") directs and coaches all improvement-related tasks.[5] This person performs the internal assessment, presents findings to the staff, develops the improvement plan and manages it. The improvement leader can be an external SQA consultant, the QA manager or a company SQA engineer, but he is not "merely" an assessor; he has the responsibility to show results, like any other project manager.

#### 3.2 Open assessment

Open assessment is the cornerstone of a motivation-driven improvement process. This is an *internal* assessment. The assessor does not build constraining checklists, which have a limiting effect on eliciting participation from interviewees.

Kerr stated that an imposed development process is likely to miss the mark and result in failure.[6] In fact, if developers play a role in designing the process, it has an excellent chance for promoting improvement: not only does it target the correct issues, but it also raises the developers' motivation for implementing it.

Dymond emphasized the decisive contribution engineers can bring to assessment findings.[7] Veteran engineers can offer valuable information about endemic problems in company processes as well as areas of strength. Newcomers are able to question established practices in light of their experience at a previous workplace. Administrative assistants can also recount process failures and achievements.

The interview starts with a brief description of ASPI as an opportunity for better performance based on the information gathered from the interviews. The objective of the interview is to anonymously gather interviewees' opinions on both weaknesses and strengths. The assessor asks the interviewee about his work, about successes and failures, about smooth processes versus problematic ones. For every practice, the assessor asks three basic questions:

1. What goes wrong now?
2. Optimally, how should the process be performed?
3. How should the process be changed?

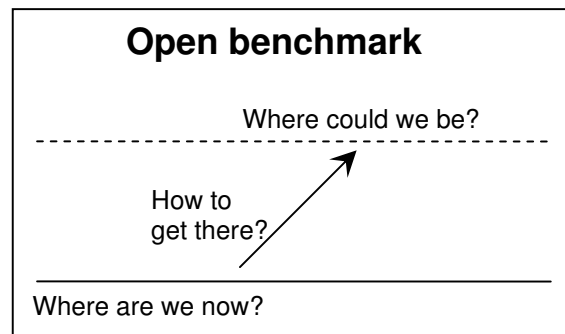


Figure 2: Open benchmark

The assessor encourages the interviewee to speak freely, the only restriction being not to refer to persons but to processes. The rule is to avoid names of people; roles or titles may be referred to if absolutely necessary.

#### 3.3. Objectivity and normalization

"Open assessment" findings are a mix of subjective feelings, personal frustrations and actual process clues. Different people can see a given practice from quite different points of view. Determining objective findings is simple: the more a finding recurs in interviews, the greater the chance it is fact.

The assessor must find a link between reports. To this end, additional clarification questions are presented. The assessor then collects the clues that point to one problematic practice. This method is typical of scientific research, where a new theory explains a group of apparently unrelated findings.

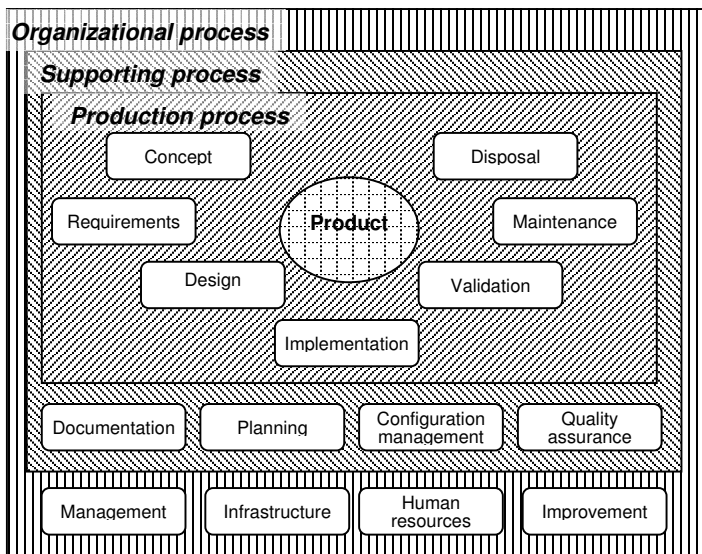
### 4. Plan

#### 4.1 Quick and tangible results

Changes in software development practice must make a real contribution to production before they are accepted in "pragmatic" companies. Management is more likely to focus on practices that directly affect software production—for example, coding (implementation). Initial improvement efforts in this area, therefore, have good chances of management's support.

In the free interpretation of ISO 12207 shown in the illustration below, the software manufacturing process is divided into three distinct layers. Each layer sustains the layer above, up to the product itself. Managers typically choose to start with activity areas at the "production process" level, such as *implementation*, *requirements* or *design*.

# SW manufacturing process



**Figure 3: Software manufacturing processes**

## 4.2. Prioritization

When prioritizing processes for improvement, the greatest weight should be given to those areas in which there is broad acceptance of assessment findings, even if these are not objectively the most critical. The assessor, therefore, submits his report to a mid-management forum, the level that prioritizes the KPAs. This ensures that at least initial ASPI activities benefit from broad managerial support.

It is wise to postpone the most challenging tasks to subsequent phases in the improvement process—when confidence in the improvement process has increased and patience is greater.

## 4.3 Improvement cycles

After the first assessment, KPA identification and prioritization, each KPA becomes its own independent improvement cycle.[8] This is a very short improvement process that aims to demonstrate added value in a limited area, within short time, and requiring only a small investment. The importance of this was shown by Boehm.[9]

Mini-improvement-cycle	
1. KPA assessment	Infrastructure
2. Methods & Procedures	
3. Pilot	
4. Training	
5. Deployment	
6. Assimilation	

**Figure 4: Mini-improvement-cycle**

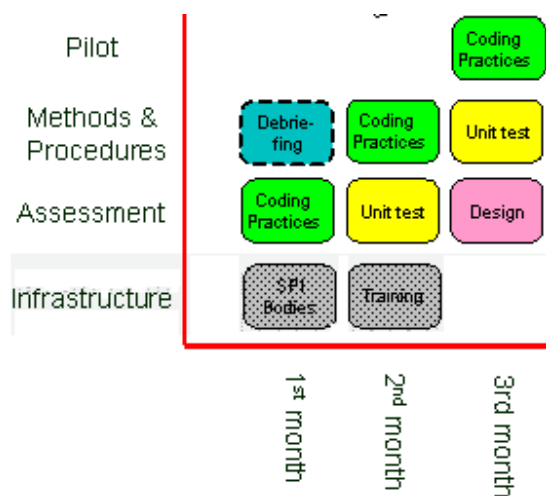
Within the mini-improvement-cycle, most of the SPI activities must follow a logical progression, where one activity ends before the next one begins. (That is, the "waterfall model" is generally applicable for a given KPA.) For example, method research and the pilot cannot start before the completion of the KPA assessment. Similarly, training must take place before deployment. On the other hand, pilots and method research can take place concurrently and, in fact, have a beneficial effect on each other.

A new KPA mini-cycle starts as soon as resources become available. This planning approach has several advantages: it not only saves time, but maintains SPI momentum. In addition, it distributes improvement costs among a number of resources, keeping the current improvement investment light.

## 4.4 Case study

LADj is part of ICG, which develops communications products at Intel. Two years ago, when the group had an SE team developing templates for software-design documents, it decided that process improvement was needed in additional areas. The LADj staff was aware that I was running a small process-improvement project for another group in the same building. Because of the group's informal exposure to process improvement, its members were not resistant to the concept.

The first issue selected following first assessment and the ensuing brainstorming was "coding rules." This is part of *implementation*, which is closest to production itself (see figure 3). It therefore sounds "productive" even in a culture unfamiliar with process-improvement methodologies. Not surprisingly, the second choice was "unit (developer) tests"—part of *validation*. The third issue selected was *design*.



**Figure 5: Roadmap (partial view)**

The illustration above shows how a new mini-improvement-cycle was started each month. We strove to have three improvement tracks running at the same time. The “unit test” track started when the “coding practices” track completed assessment. In practice, our progress was very flexible, in accordance with the resources available. Peak activity in other projects caused delay. We never claimed priority over software-development activities, so ongoing projects were only minimally affected.

The Software Engineering Process Group (SEPG) created a sub-committee to define the coding rules. The members of the sub-committee were all prominent software engineers, professionally respected within LADj, so that the developers had confidence that the coding rules established had relevance to their work.

The process of assimilation of the coding rules established continues to this day, long after the SEPG formally completed this phase, demonstrating a true commitment to SPI. One of the engineers, for example, recently wrote patterns for the rules and posted the page on the SPI intranet site.

Today, process improvement is institutionalized for the entire LAD group, worldwide: there is a dedicated team that directs the process for all of the group's branches, around the world.

## 5. Conclusion

ASPI answers the paradoxical need for a process of improvement that does not demand much investment. It is especially appropriate in the current economic climate, which reduces the willingness to take risks and dictates a higher degree of competitiveness. Like other “light CMM” methods, its added value is that it introduces

process improvement and software engineering principles into the work culture.

## 6. References

- [1] "Agile" was used in connection with software development by Alistair, Cockburn, et al in “Manifesto for Agile Software Development,” <http://agilemanifesto.org>, February 11-13, 2001.
- [2] Gilb, T. *Evo: The Evolutionary Project Managers Handbook*, [www.gilb.com/Download/EvoProjectMan.pdf](http://www.gilb.com/Download/EvoProjectMan.pdf), August 1997.
- [3] Fowler, M., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Pub Co; 28 June 1999.
- [4] M.C. Paul, “Software Process Proverbs,” *Software Engineering Institute*, Carnegie-Mellon University, Jan 1997.
- [5] J. Puffer, A. D. Little, “Action Planning,” *Software Process Newsletter*, IEEE Computer Society TCSE, Spring 1997.
- [6] S. Kerr, “The Best-Laid Incentive Plans,” *Harvard Business Review*, Harvard Business School Publishing, January 2003, pp 27-37.
- [7] K. Dymond, “Essence and Accidents in SEI-style Assessments,” International Software Consulting Network, <http://www.iscn.ie/news/iscn95/doc-19.html>, 18-Aug-97.
- [8] Caputo, K., *CMM Implementation Guide: Choreographing Software Process Improvement*, Appendix G, Software Engineering Process Group, Addison-Wesley, April 14, 1998.
- [9] B. Boehm, “CMMI and the Balance of Discipline and Agility,” CMMI Technology Conference 2002, University of Southern California—Center for Software Engineering, November 13, 2002.