Lightweight Software Process Assessment and Improvement

Tom Feliz
Tom.Feliz@tektronix.com

Abstract

When most Software Engineers think of software process assessment and improvement, images of high paid consultants, byzantine process frameworks, and expensive audits come to mind. Established process frameworks, such as the Capability Maturity Model Integration (CMMI), are focused primarily on external certification and less on internal process improvement. Software process assessment and improvement need not be a heavyweight and burdensome endeavor. The best software organizations make process improvement an integral part of their organizational development and culture. The approach outlined in this paper describes a lightweight software process assessment and improvement approach that is practical, customizable, and can be implemented with reasonable effort. This approach is particularly relevant for smaller software organizations, which bear the costs of software process improvement (SPI) disproportionately.

For most software organizations, the value and benefit of assessing organizational maturity is in identifying tangible process improvement initiatives based upon well-established Software Engineering best practices. Considerable research has been conducted over the past 40 years into the practices that yield the biggest return on investment in regards to defect removal efficiency and developer productivity. Some of the most comprehensive research has been conducted by Capers Jones of Software Productivity Research (SPR). SPR has compiled a large database measuring the relative effectiveness of most common software practices. This data forms the basis of the software process assessment and improvement approach proposed in this paper.

Biography

Tom Feliz is a Senior Software Design Engineer and Software Quality Lead at Tektronix. He has been engineering software since he bought his first computer, a Commodore VIC-20, in 1983. Prior to joining Tektronix, Tom founded multiple technology startups and has worked in a variety of Software Engineering environments. His professional interests include Embedded Software Engineering, Hardware-Software Integration, and Software Quality.

Tom Feliz has a Master of Science in Computer Science and Engineering (MSCSE) from OHSU OGI School of Science and Engineering. He was awarded the IEEE Certified Software Development Professional (CSDP) credential in 2007, one of only a handful in Oregon.

1. Introduction

Software process improvement (SPI) has enabled thousands of companies worldwide to produce higher quality software in a more predictable fashion while meeting cost estimates and schedule commitments [1]. SPI, based on well-known reference models such as CMMI [2] or ISO 15504 [3], provides industry best practice benchmarks that software organizations can use to assess their software process performance both for internal process improvement efforts and for evaluation by outside vendors. Historically, these efforts have been targeted towards large software organizations with hundreds of developers and thousands of employees [4, 5]. This has left many smaller software organizations questioning the role of SPI in their software development efforts. Small organizations, with less than 50 employees, account for approximately 90% of the software and data-processing companies in the U.S. [6] and small data-processing companies with five employees or less constitute 65% of the U.S. industry [7].

Even though many smaller software organizations see the need for SPI, their needs have been disproportionately ignored in SPI research literature [4, 7]. Large software organizations tend to have formal processes based on established process frameworks, as well as dedicated software quality staff to oversee the management of software processes company-wide. Smaller software organizations, in contrast, generally lack internal software process expertise and software processes are often ad hoc. For SPI to be successful in the smaller 90% of software organizations, a more lightweight approach is needed.

This paper begins with a basic introduction to SPI, software process assessment, and a discussion of where existing process frameworks fall short. A simplified and practical process assessment and improvement approach is then presented with real-world example assessment artifacts. The hope is that by demonstrating a practical, simplified, and customizable cookbook approach to SPI and process assessment, the author might begin to dispel the common myth that SPI is out of reach for the great majority of software organizations (i.e. the other 90%).

2. Software Process Improvement

2.1 Introduction to SPI

Even the most chaotic of organizations use some form of process, even if it's just pure code-and-fix software development [8]. Software processes range from total chaos, with each individual working in isolation in his or her own style performing any tasks in whatever order is deemed appropriate, to regimented processes, where each finely-grained action is recorded, measured, managed, and documented. For most organizations, the ideal process is somewhere is between these two extremes. Projects don't necessarily have to be planned to still be using processes [9], which may even be quite effective in some contexts. The so-called "peopleware" process approach is based on just this theory. The goal is to employ the most gifted and talented people and place them in a well-supported environment conducive to intellectual / creative work and a highly collaborative process will arise yielding high quality outputs [9,10]. Clearly, there is some validity to such an approach. But when repeatable long-term organizational performance is required with minimal risk, then a little process can go a long way.

There is a common belief, especially among smaller software organizations, that software assessments and SPI only burdens creativity and stifles innovation with unnecessary overhead. In this view, as in the "peopleware" approach, high quality people know intuitively how to produce high-quality software. Certainly, high quality people will produce relatively higher quality output. But if that's all that is required, then the best software organizations with arguably some of the best software professionals, wouldn't need process and wouldn't experience the common problems associated with software quality and productivity. Certainly, smaller projects require smaller processes, but even the smallest projects involving more than a single person could benefit from rudimentary processes to facilitate communication and ensure the synchronization of work [1].

A well-defined and tailored process will actually enhance creativity and free employees to concentrate on the details of the projects rather than the process [5, 9]. By freeing the most talented software professionals from the predicaments created by others (by avoiding the problems in the first place), process not only enhances creativity, but maximizes the productivity of the individuals and increases predictability of the work products [1]. Well defined and proportionate software processes can and do enable a person to do their job better, and not just prevent them from doing harm [11].

Further, research suggests that employee morale improves along with a well-managed SPI program. Boeing's Space Transportation Systems organization made the journey from CMMI level 1 to level 5 and at the same time employee satisfaction increased significantly along the way. The increase in satisfaction led to increased motivation, innovative solutions, greatly improved attitudes, and an increased sense of trust between management and employees [12]. Brodman, et al, while assessing the return-on-investment among 35 companies of various sizes that implemented the CMM[1] in their organizations, made the observation that the most frequently cited benefit was the morale and confidence of the software development staff improved "significantly" [13]. Additionally, there is anecdotal evidence that SPI increases employee ownership of the quality system and employees appreciate the strengthened support infrastructure [11].

Fundamentally, SPI provides a framework so continuous controlled improvement can be established to better support an organization's business objectives [1, 8]. Organizations seek to pursue SPI for a number of reasons [1, 5, 8].

- A desire to reduce wasted time and/or effort
- A desire to stabilize current processes to ensure repeatability
- To ensure that failures happen only once
- To remedy dissatisfaction with the status quo
- To rein in cost estimates and meet schedule commitments
- To enhance predictability

Crisis can be a great motivator for organizational change, but SPI can also be implemented by forward-thinking organizations in anticipation of growth or to take advantage of current software best-practices represented by the chosen reference process model.

The key characteristic of SPI is that improvement is both incremental and continuous. Processes are improved over time and experience is fed back into the improvement meta-process in a cyclical manner. Feedback occurs through the use of project retrospectives and regular process assessments. As an organization matures, its processes also mature as industry best practices are integrated into an organization's *modus operandi*. In this way, a mature software organization is dynamic, evolving, and improvement is continual.

2.2 SPI for Smaller Software Organizations

Smaller software organizations may have particular concerns that existing process models won't provide the flexibility required to address the great diversity of projects encountered. Though many organizations feel their particular project and process challenges are unique, software organizations all confront similar problems [1]. On one level, all software projects are different. But in many fundamental ways, all software projects are the same [5]. SPI must be tailored to match each organization's unique requirements, but at the same time leverage the field-tested best-practices [14].

Smaller software organizations (those with 50 employees or less) face unique challenges when pursuing SPI initiatives. Software Engineering for this 90% isn't simply a "degenerate" form of what is practiced in larger organizations, but has its own set of problems and solutions [6]. The following issues have been identified as particular impediments to smaller software organizations in their pursuit of SPI [15,16,17,18]:

[1] CMM was the precursor to CMMI.

- Questionable relevance of reference process models
- Cost of SPI implementation
- The trade-off of SPI versus other needs
- Incompatible corporate culture
- Customers not supportive
- Reference model organizational structure conflicts with current structure
- Resource limitations for pursuing SPI
- Heavy reliance on software “heroes”
- Long-term investment in SPI is not understood or appreciated
- Employees are often not aware of basic principles of quality management
- Perceive SPI to be for larger organizations only
- Not convinced by SPI data from larger organizations
- Lack of in-house expertise in SPI
- Not aware of potential business benefits
- Forced to adopt customer process standards
- Employees “opt-out” of larger organizations to avoid “process”

Among the larger themes from this list of issues is the sensitivity of smaller software organizations to cost and resource issues. Return-on-investment (ROI) is of particular importance, if not paramount importance. ROI must be justified at each step of the way. Smaller software organizations absorb the fixed costs of SPI in a disproportionate manner compared to larger organizations. Thus on a percentage basis, SPI costs smaller organizations more [15].

The following are basic principles to consider when implementing a SPI program [1, 8].

- Major changes must start at the top
- Everyone must be involved
- Knowledge of current processes is essential
- Process change is “continuous”
- Software process change requires reinforcement
- SPI requires investment (long term)

Organizations that attempt to improve process from the “bottom-up” end up with “islands of excellence”, but lack the follow-through to propagate the best-practices throughout the organization [5]. This phenomenon is particularly prevalent in smaller software organizations that tend to address process issues at the project level. SPI encourages the broadening of the myopic tendency to make process decisions based on expediency rather than on a planned basis. As mentioned earlier, SPI programs in small software organizations must always keep in mind the potential ROI and expend limited resources that will provide the most benefit with the least risk [8].

To ensure SPI efforts do not impinge on organizational innovation and creativity, the focus of SPI efforts should be on project-specific issues, rather than process centered ones. Standards and procedures should concentrate on essential details, but leave plenty of flexibility to accommodate the project variations found in typical small software organizations. One approach is to specify two levels of standards and procedures. The base would consist of mandatory elements whereas the top layer would be a set of best-practices that are recommended, but not required [11]. It’s also important to recognize that there’s no single right way to do SPI. The goal of a SPI effort must meet the requirements of the organization pursuing the effort [19].

Software organizations should also keep in mind the key factors of a successful software process improvement program [15, 8, 20].

- Company-wide SPI emphasis / Goals are widely understood
- Customer support of SPI efforts
- Management business alignment with SPI / Management support

- Sub-contracting relationship with a larger company with a SPI focus
- Membership in a third-party process improvement organization
- Process related training
- Developer / technical staff involvement
- Maintenance of momentum
- Corporate cultural awareness of SPI
- Separation of process and product concerns (for a product company)
- Presence and empowerment of champions
- Frequent process assessments
- Visibility into the SPI process
- Compensated SPI responsibilities
- SPI staff well-respected in the organization

Obviously, not every organization will be able to conform to these success factors. Instead, the list is meant to provide practical guidance on the factors that have been observed among small software organizations with successful SPI programs. Besides the support of top-level management, one other factor deserves special emphasis because of its repeated emphasis in SPI literature. Training in software process and SPI is almost universally highlighted and its ROI in regards to SPI is always emphasized as highly favorable [1, 5, 15, 11, 8, 19].

3. Software Process Assessment

3.1 Introduction to Software Process Assessment

Software process assessments are the mechanism by which SPI is measured, evaluated, and opportunities for improvement are discovered. Assessments are not an *ad hoc* exercise and imply a process standard to be evaluated against [1]. Watts S. Humphrey in his seminal work, *Managing the Software Process*, describes three main objectives for software process assessments - to learn the organization, identify problems, and enroll opinion leaders in SPI [1]. Problems identified during the assessment process serve as the raw material for developing process improvement initiatives. Assessments can also serve an educational function by encouraging process-level discussion, continuous improvement orientation, and facilitating a forum for creative group thinking in regards to process [21]. In a way, assessments provide a ten-thousand foot view of how a software organization is performing in meeting its business objectives.

Watts S. Humphrey identifies five guiding principles for software process assessments [1]:

1. Need for a process model
2. Absolute confidentiality
3. Senior Management Involvement
4. Attitude of respect towards all participants
5. Action orientation

A process model provides the framework or skeleton to organize the assessment and process improvement efforts. Without a process model framework, assessment and improvement efforts can devolve into a battle of personal opinion and conjecture. A process model provides an “ideal” based on known best practices. The purpose of an assessment is to evaluate an organization against these proven best practices and identify initiatives with the greatest ROI potential.

Confidentiality ensures that assessment interview questions and questionnaires are answered openly and honestly without fear of retribution or blame. The assessment process should always start with the assurance of confidentiality and the assessor should carefully guard personally identifiable information

from disclosure (e.g. names and projects). Further, freeform answers to assessment questions should be sanitized and only reported in aggregate form.

Since the assessment process requires resources and SPI is only effective with long-term follow-through, management approval and support is important. This is especially true if one of the formal process frameworks such as CMMI is employed, which often require extensive training, outside consultants, and time from project members. One of the most effective ways to ensure management buy-in of the assessment and SPI efforts is to involve management from the beginning of the project. Managers are particularly concerned about efficiency and even skeptical management can be won-over if the focus of SPI is on increased productivity, improved quality, and improved customer satisfaction.

Some of the best and most creative ideas come from non-management software development staff. It's always surprising how aware individual software team members are about process problems and how forthcoming they can be if there is a spirit of mutual trust between the team and the assessor. It's therefore critical the entire assessment process be based upon mutual respect. Process assessment concerns more than just management and all team members will have valuable input.

The outputs of a software process assessment are recommendations for tangible and actionable SPI initiatives. The guiding principle should be action-orientation instead of conformance to a particular process model or framework. The action plan must also support the strategic business goals of the organization.

3.2 Traditional Software Process Assessment

Traditional software process assessments consist of three phases, preparation, assessment, and recommendations [1]. The preparation phase is where assessment goals are determined, the process model framework is chosen, management buy-in is achieved, opinion-leaders are engaged, outside consultants are hired, and assessment materials are prepared. The assessment phase is the execution period when the assessment team performs the actual process assessment. The assessment phase represents the period of maximum involvement from the organization. Finally, the recommendation phase consists of analyzing the assessment results, prioritizing process areas in need of improvement, and reporting on the results.

Software process assessments can be performed by one of several groups depending on the level of formality required and the purpose of the assessment. For example, assessments performed for external compliance or certifications are usually performed by outside consultants. However, assessments performed for internal process improvement needs can be performed by a champion within the organization. Larger companies might have a process or quality group that performs process assessments. Typically, process assessments are performed by a cross-disciplinary group of key individuals such as management, team leads, SQA, and so forth. In the author's opinion, the more organic and indigenous the assessment team, the higher the probability of success.

Assessment questionnaires are created based on the key process areas (KPAs) enumerated in the reference process model. A scoring system is used to rate the completeness of the organization's deployment of each KPA. Each appraisal score is then aggregated into a summary score to generate an overall process maturity appraisal score (e.g. CMMI maturity level). Project artifacts are collected to provide verifiable objective evidence (VOE) to support the organization's appraisal. The process questionnaire results combined with the assessment scores are then reported back the organization along with process improvement recommendations.

Process assessments are typically performed on a periodic basis every 6-24 months. Process improvement takes time and assessments require resources to perform. The effort must be balanced against the potential benefit. As an example, the average time for an organization to move from CMMI level 2 to level 3 is 20 months for 2011 [22]. Thus, performing a process assessment every 3 months for a CMMI level 2 organization may not be an efficient use of resources.

3.3 CMMI-DEV and SCAMPI Appraisals

The Standard CMMI Appraisal Method for Process Improvement (SCAMPI) is the recommended software process appraisal method of the Software Engineering Institute (SEI). The purpose of a SCAMPI appraisal is to evaluate an organization's strengths and weaknesses against the CMMI process framework. According to the SEI, SCAMPI appraisals are conducted for the following reasons [23].

1. To compare an organization's process performance to CMMI best practices.
2. To provide a means of external certification to outside customers and suppliers.
3. To fulfill contractual obligations to customers.

As with all software process assessment frameworks, CMMI-DEV is based on Software Engineering best practices as defined by the SEI in association with industry and government. KPAs for CMMI-DEV are listed in Table 1 [23].

Process Area	***Category***	***Maturity Level***
Causal Analysis and Resolution (CAR)	Support	5
Configuration Management (CM)	Support	2
Decision Analysis and Resolution (DAR)	Support	3
Integrated Project Management (IPM)	Project Management	3
Measurement and Analysis (MA)	Support	2
Organizational Process Definition (OPD)	Process Management	3
Organizational Process Focus (OPF)	Process Management	3
Organizational Performance Management (OPM)	Process Management	5
Organizational Process Performance (OPP)	Process Management	4
Organizational Training (OT)	Process Management	3
Product Integration (PI)	Engineering	3
Project Monitoring and Control (PMC)	Project Management	2
Project Planning (PP)	Project Management	2
Process and Product Quality Assurance (PPQA)	Support	2
Quantitative Project Management (QPM)	Project Management	4
Requirements Development (RD)	Engineering	3
Requirements Management (REQM)	Project Management	2
Risk Management (RSKM)	Project Management	3
Supplier Agreement Management (SAM)	Project Management	2
Technical Solution (TS)	Engineering	3
Validation (VAL)	Engineering	3
Verification (VER)	Engineering	3

Table 1 - CMMI-DEV Key Process Areas

Each CMMI-DEV KPA is broken down further into sub-processes, which are evaluated separately during an assessment. CMMI uses a 5-level maturity scale (4-level for continuous representation) [23].

1 – Initial
2 – Managed
3 – Defined
4 – Quantitatively Managed
5 – Optimizing

Between 2006 and 2011, only about 7% of all organizations reporting to the SEI had been assessed at level 4 or 5 [22]. This suggests most software organizations are not motivated to pursue maturity levels

beyond 3 (managed). This supports concerns expressed by Karl Wiegers of Process Impact concerning the expected benefit of CMMI levels 4 and 5 versus the effort to achieve them [24].

One thing to note about the CMMI-DEV is the emphasis on process and project management KPAs. Many of the CMMI KPAs can rightly be considered meta-processes. CMMI specifies the “what”, but not the “how” of the processes and associated practices. As a result, there is considerable flexibility in the implementation of the process areas.

Due to the prestige of the SEI and ubiquity of the CMMI process framework throughout industry and government, CMMI process maturity ratings are influential when evaluating customers and suppliers. This “process report card” aspect to the CMMI allows business and government to evaluate the process maturity (and indirectly quality) of its vendors. Likewise, a vendor can use a high CMMI appraisal level to promote its process maturity. Of the1333 SCAMPI Class A appraisals reported to the SEI in 2011, 633 were reported by software organizations in China and India versus only 270 reported from all U.S. software organizations [22]. The large number of offshore software development organizations in both China and India likely account for most of the disparity.

3.4 Where the CMMI and SCAMPI Appraisals Fall Short

The CMM and later the CMMI were developed by the SEI, which is largely funded by grants from the U.S. Department of Defense (DoD). Given this lineage, CMMI has traditionally been used by the DoD to evaluate suppliers. For many software organizations doing business with the DoD, CMMI appraisals and a minimal maturity level are required for contract compliance. For these organizations, large and small, there are strategic business reasons to adopt a formal CMMI approach. Large software organizations (i.e. hundreds of developers) may also find the rigor and formality of a full CMMI approach compelling. For large organizations, the cost and resources required to fully implement a CMMI SPI program are easier to absorb.

What about the other 90% of software organizations with less than 50 employees? For many smaller software organizations, the resources required to pursue a CMMI-based SPI program are often prohibitive. A SCAMPI appraisal team consists of at least four people and one of the four must be a certified lead appraiser. Lead appraisers must take at least 3 CMMI training courses, pass a number of tests, and have previous experience with SCAMPI appraisals. Other members of the appraisal team must take at least one CMMI training class. The expense and effort is considerable, which is why most software organizations hire external consultants to perform SCAMPI appraisals. Lead Appraisers typically charge between $1,500 and $2,500 per day [25]. It’s easy to see how costs for a SCAMPI appraisal can approach $50,000-100,000. For many smaller software organizations, the potential ROI simply does not justify the cost.

Even if the CMMI is only utilized for the process model without performing SCAMPI appraisals, CMMI may not be the best choice for reasons beyond cost. Of the 22 KPAs in the CMMI-DEV profile, 12 are focused on project and process management activities. Only 5 of the 22 KPAs are categorized as engineering process areas. Yet, most software organizations pursue SPI to reduce waste, reduce occurrence of failure, improve estimates, enhance predictability, and produce higher quality outputs. The CMMI process areas are weighted toward practices that don’t necessarily support these goals. Ultimately, the CMMI can help develop an elaborate process infrastructure, but says surprisingly little about how to develop better software.

The CMMI provides little prescriptive guidance on how to implement the required processes. For example, CMMI-DEV specifies that peer reviews and code reviews are to be performed, documented, and metrics collected. There’s only generic guidance of the style or rigor of the reviews, whether this includes such best practices as static code analysis, how often reviews should be performed, or which work products are to be reviewed. The net effect is that organizations can develop elaborate software processes based on the CMMI, but still not have a handle on how effective the processes are in regards to overall software quality (e.g. defect removal efficiency) or productivity.

What's needed is a lightweight and adaptable SPI and assessment approach that leverages established industry software best practice research and provides specific and tangible process guidance. Considerable research has been done on the relative effectiveness and productivity impact of most common Software Engineering best practices [26]. Such an approach would not be usable as an external assessment or certification vehicle like the CMMI. But, most software organizations (i.e. the other 90%), are primarily interested in SPI for internal improvement and only require a process framework to provide a rough guide for long-term continuous SPI. The standardized and externally-facing aspects of the CMMI are of little value in this scenario.

4. Lightweight Process Assessments

4.1 Software Engineering Best Practices

Software assessments must be performed in relation to known best practices to be effective. Without an ideal to compare against, a software organization won't know where it falls short. Ideally, these best practices are based on sound Software Engineering research. The most common software process frameworks are very complex and can be difficult to implement. Yet, most software organizations don't have internal expertise in SPI. Unless an organization is willing to expend the resources to hire outside consultants, another approach is required.

Capers Jones and his company, Software Productivity Research, have been conducting Software Engineering research for over 25 years. Capers Jones' recently published book, *Software Engineering Best Practices*, contains compiled data from 675 companies in 24 countries and 13,500 projects [26]. The data is analyzed with three dimensions, code size, type of software, and development activity. Each practice/process is ranked based on defect removal efficiency (DRE)[2] and productivity. 200 practices are scored and ranked. Scores range from 10.00 to -10.00, with negative scores indicating practices that do more harm than good. The following table shows the first 50 best practices overall.

Rank	Methodology, Practice, Result	Score
1.	Reusability (> 85% zero-defect materials)	9.65
2.	Defect potentials < 3.00 per function point	9.35
3.	Defect removal efficiency > 95%	9.32
4.	Personal Software Process (PSP)	9.25
5.	Team Software Process (TSP)	9.18
6.	Automated static analysis	9.17
7.	Inspections (code)	9.15
8.	Measurement of defect removal efficiency	9.08
9.	Hybrid (CMM + TSP/PSP + others)	9.06
10.	Reusable feature certification	9.00
11.	Reusable feature change controls	9.00
12.	Reusable feature recall method	9.00
13.	Reusable feature warranties	9.00
14.	Reusable source code (zero defect)	9.00
15.	Early estimates of defect potentials	8.83
16.	Object-oriented (OO) development	8.83
17.	Automated security testing	8.58
18.	Measurement of bad-fix injections	8.50
19.	Reusable test cases (zero defect)	8.50
20.	Formal security analysis	8.43
21.	Agile development	8.41
22.	Inspections (requirements)	8.40

[2] DRE = Defects Discovered Before Release / Total Discovered Defects Before and After Release.

23.	Time boxing	8.38
24.	Activity-based productivity measures	8.33
25.	Reusable designs (scalable)	8.33
26.	Formal risk management	8.27
27.	Automated defect tracking tools	8.17
28.	Measurement of defect origins	8.17
29.	Benchmarks against industry data	8.15
30.	Function point analysis (high speed)	8.15
31.	Formal progress reports (weekly)	8.06
32.	Formal measurement programs	8.00
33.	Reusable architecture (scalable)	8.00
34.	Inspections (design)	7.94
35.	Lean Six Sigma	7.94
36.	Six Sigma for software	7.94
37.	Automated cost-estimating tools	7.92
38.	Automated maintenance workbenches	7.90
39.	Formal cost-tracking reports	7.89
40.	Formal test plans	7.81
41.	Automated unit testing	7.75
42.	Automated sizing tools (function points)	7.73
43.	Scrum session (daily)	7.70
44.	Automated configuration control	7.69
45.	Reusable requirements (scalable)	7.67
46.	Automated project management tools	7.63
47.	Formal requirements analysis	7.63
48.	Data mining for business rule extraction	7.60
49.	Function point analysis (pattern matches)	7.58
50.	High-level languages (current)	7.53

Table 2 - Top 50 Practices from Software Engineering Best Practices

The reader should immediately notice how specific and tangible many of the practices are. Unlike the CMMI, which is very abstract and management focused, SPR's best practices are focused on practices and processes that improve early defect detection, such as Team Software Process, automated static analysis, and code inspections. The CMMI only refers to these practices indirectly, if at all. For software organizations looking to improve software quality as efficiently as possible, seeking to improve DRE is a good first step.

The best practices presented here and in Capers Jones' book are meant to be a starting point and not all practices will be relevant in every context. Different organizations will have different strengths and weaknesses. The history and experience of each organization will also vary. There may be practices not on the list that an organization would like to address due to familiarity or an existing partial implementation. Since the goal is to improve software processes internally, there is considerable flexibility and opportunity for customization. To maximize ROI, it's essential to focus on process improvement generally and not get distracted by the allure of conformity to a process model or an arbitrary set of practices.

4.2 The Assessment Process

A software process assessment should be managed as a project with the requisite planning. The high-level assessment tasks are enumerated in Figure 1. As with all projects, the assessment process begins with a charter to ensure buy-in from key stakeholders. The charter need not be particularly formal, but due to the time and effort required to perform an assessment, at least tacit management approval is needed. The charter also establishes the general goals and nature of the assessment. Issues such as the

assessment scope, level of formality, the composition of the assessment team, rough timeframe, and the disposition of the assessment outputs should be defined early in the process.

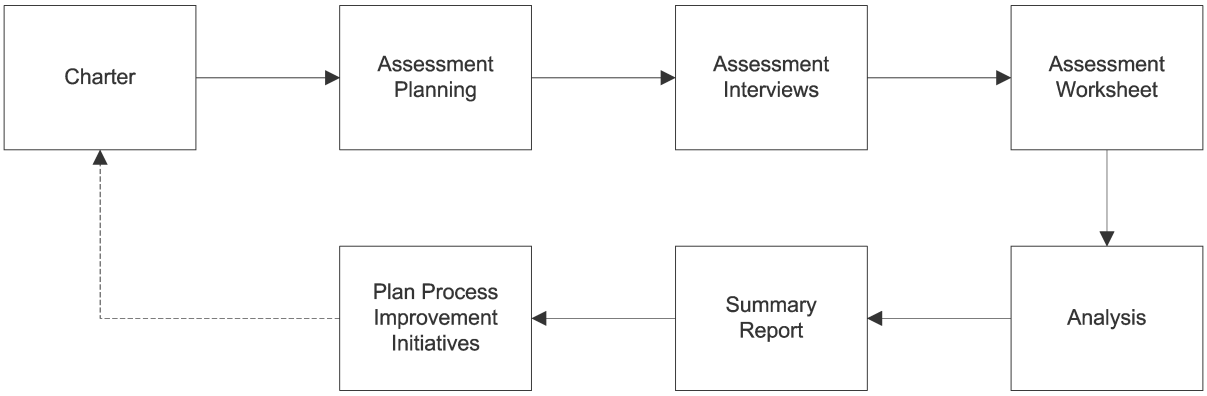

Figure 1 - Software Process Assessment Process Diagram

Assessment planning establishes the roadmap for the assessment process and sets expectations. The primary goal of the planning phase is scheduling assessment meetings and interviews. The assessment plan may be as informal as an email thread or as formal as a complete project plan. It's recommended that the assessment plan not be too prescriptive to allow the process to evolve dynamically. For instance, it is often necessary to engage additional resources based on new information discovered during the course of the assessment process.

There are two aspects to a software process assessment – qualitative and quantitative. The qualitative aspect captures the non-quantifiable input to the assessment and represents the "soft side" of the assessment. Qualitative data is generally gathered from assessment interviews performed one-on-one with key members of the organization. The number of interview subjects will vary from organization to organization. But, generally, interviews with a quarter to half of an organization provide adequate input to the process. Interviews are typically about an hour in duration, but definitely not longer than two hours. Interview questions are customized for an organization based on the broad process areas common to most process frameworks (e.g. Management Processes, Organizational Processes, Engineering Processes, etc.). The qualitative data is captured "raw" and archived for later analysis. As stated previously, the answers to the interview questions should be sanitized for personally identifiable information to ensure the anonymity the participants. The appendix to this paper includes an actual assessment survey from a software process assessment performed by the author in 2005.

The quantitative aspect of an assessment is a rough attempt to aggregate assessment results into a single number or set of numbers. Figure 2 is an example software process assessment worksheet for a single software organization.[3] Assessment results are summarized by process area, which enables analysis by related categories of best practices. The best practices in this worksheet were chosen based on Capers Jones' research published in *Software Engineering Best Practices* discussed in the previous section [26].

The implementation of each best practice is rated on a simple 3-level scale.

1. Currently doing nothing or very little
2. Partial implementation, but opportunity for improvement
3. Fully Implemented

This scale, which is a simplified version of the CMMI scale, makes assessment very easy and reduces ambiguity overall. Scores for individual best practices are then averaged together for each process area and the process areas are averaged together to produce an overall assessment rating. Individual

[3] Document templates are available from the author upon request.

organizations can modify this approach by using a more graduated scale and/or weighting certain process areas over others.

By nature, quantitative measures are inexact and abstract. However, in the aggregate, quantitative measures provide valuable trending information to guide SPI efforts over time. Quantitative measures are meant to be used as relative measures and not as absolute measures. An organization that is increasing in process maturity should see its assessment score rise over time. The author has also observed a strong correlation between the implementation of proven best practices and software quality.

Software Best Practices Assessment Worksheet

Rating Key: 1 - Currently Doing Nothing or Very Little; 2 - Partial Implementation, but Opportunity for Improvement; 3 - Fully Implemented

Best Practice	Sep-10	Dec-10	Mar-11	Jun-11	Sep-11	Dec-11
Software Process/Project Management	**2.33**	**2.50**	**2.50**			
Regular Progress Reports/Standup Meetings	3	3	3			
Customer Visits/AE Interviews/VOC Data/Trip Reports	2	3	3			
Robust Configuration Management of All Deliverables	3	3	3			
Software Process Explicitly Defined and Documented	2	2	2			
Clear Exit Criteria/Checklists for All Project Milestones	2	2	2			
Formal Estimation Discipline	2	2	2			
Software Requirements	**2.29**	**2.43**	**2.71**			
Requirements Inspections/Reviews	2	2	3			
Formal Requirements Tracking/Product Backlog	3	3	3			
Measurement of Requirements Changes	1	2	2			
Changes Managed via Change Control Board (CCB)	3	3	3			
Non-Functional Requirements Explicitly Defined	1	1	2			
Usability is a Core Part of the Requirements Process	3	3	3			
Use Case/User Story-Driven Requirements	3	3	3			
Software Design/Architecture	**1.50**	**2.00**	**2.17**			
Design Inspections/Reviews	2	2	2			
Formal Architecture Documentation	1	2	2			
Reusable Architecture	2	3	3			
Architecture Follows Solid Design Principles	2	2	2			
Design Documents Updated Throughout the Lifecycle	1	1	2			
Design Documentation is Traceable to Requirements	1	2	2			
Software Construction	**1.78**	**2.33**	**2.33**			
Software Reuse	2	3	3			
Source Code Control (e.g. ClearCase)	3	3	3			
Static Code Analysis	1	2	2			
Automated Defect Tracking (e.g. ClearQuest)	3	3	3			
Peer Code Inspections	1	2	2			
Agile/Scrum Development/Incremental Delivery	3	3	3			
Continuous Integration	1	2	2			
Formal Coding Standards	1	2	2			
Preference for COTS Components versus Rolling Your Own	1	1	1			
Software Testing	**2.00**	**2.29**	**2.29**			
Reusable Test Cases	2	2	2			
Formal Test Plan (reviewed)	2	2	2			
Automated Testing (Unit, Regression, and/or Integration)	2	2	2			
Expert User Testing/Lighthouse Customers/Beta Testing	2	2	3			
Integration Testing on Actual Target Hardware Early	2	3	2			
Test Case Traceability to Requirements / User Stories	2	3	3			
Unit Testing/Test Driven Development	2	2	2			
Software Quality Assurance (SQA)	**1.75**	**2.50**	**2.75**			
Formal SQA Plan (reviewed and approved)	1	1	2			
Separate/Independent SQA Function	3	3	3			
Formal Risk Management	2	3	3			
Defect Removal Efficiency is Measured	1	3	3			
Overall Assessment Rating	**1.94**	**2.34**	**2.46**			

Figure 2 - Example Software Assessment Worksheet

4.3 Assessment Analysis

Once the software process assessment is complete, the raw results must be analyzed to identify actionable process improvement initiatives. There are a number of ways to evaluate assessment results and generally it's good to start with the goals and priorities laid out in the assessment charter. The qualitative information gathered from the assessment interviews is very important for setting the context and providing background for the assessment analysis. Common and recurring themes will often arise and should be carefully considered. The best improvement ideas often come from the assessment interviews and the assessment team should seek to keep an open mind when analyzing the results. The assessment interviews also provide the bulk of the narrative for the software process assessment report.

One method of analyzing and prioritizing best practices based on effectiveness and effort is an impact matrix, shown in Figure 3. The vertical axis is an estimate of the effort required and the horizontal axis is the defect removal efficiency based on Capers Jones research. The practices in the lower-right quadrant are considered low-hanging fruit due to the relatively low effort required for implementation and the high defect removal efficiency potential. Whereas the defect removal efficiency of a practice is objectively measured based on research, the implementation effort will often vary between organizations depending on history, process maturity, and existing capabilities. In general, practices such as software reuse, peer review, static code analysis, formal risk management, and coding standards provide a "big bang for the buck" regarding overall software quality improvement.

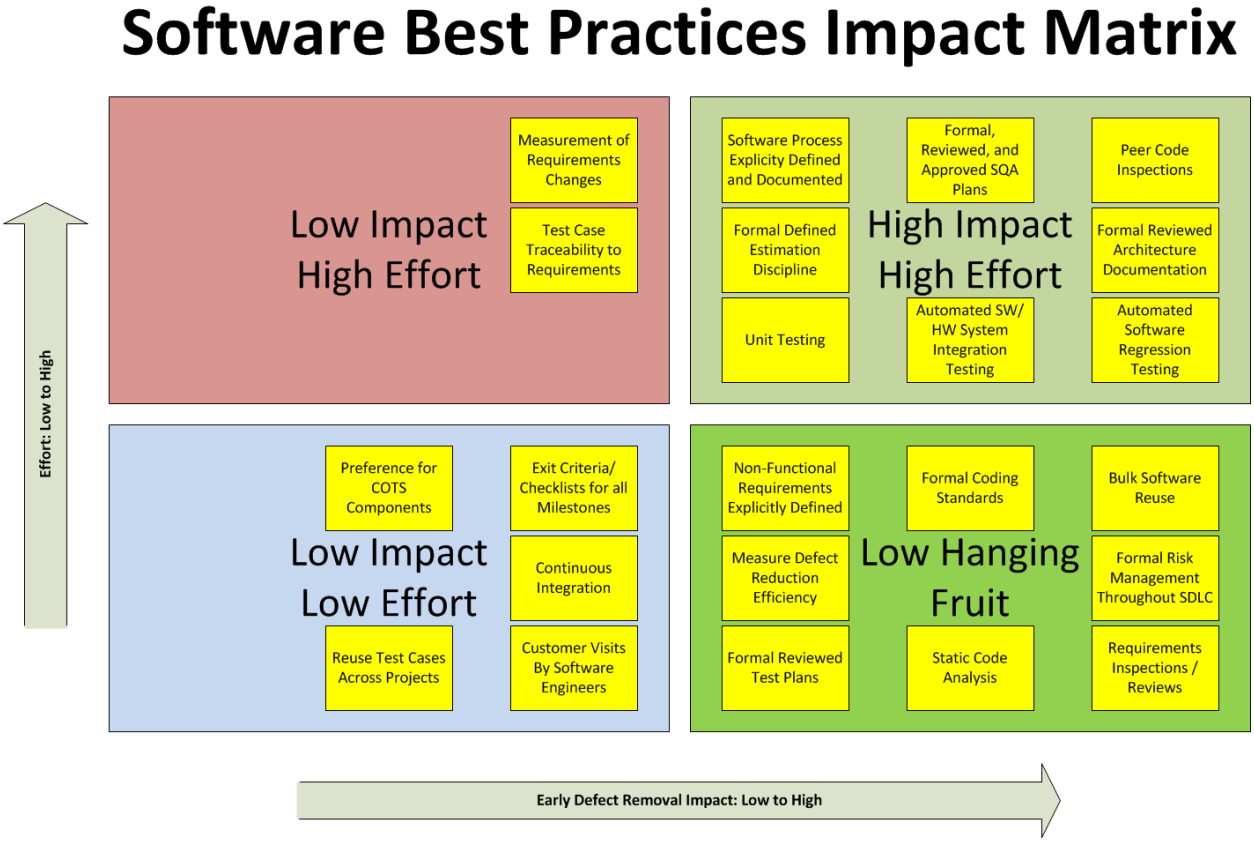

Figure 3 - Impact Matrix Demonstrating One Method for Analyzing Possible SPI initiatives

The typical deliverables of a software process assessment are as follows.

1. Description of the current state of software quality - It's important to be sensitive to an organization's historical context and prior process improvement efforts.

2. An analysis of organizational strengths and weaknesses – It's best to lead with the strengths, but an organization will usually accept a carefully considered presentation of its weaknesses.
3. Hard data and metrics – In addition to the software process assessment worksheet, other metrics can be useful to paint a full picture (e.g. defect removal rates, project schedule delays, customer satisfaction survey results, etc.).
4. A methodology overview – How did the assessment team reach the conclusions and recommendations being proposed?
5. Description of the future state of software quality – It's essential to explain the path from the present state to the future state is an incremental one.
6. A list of software process improvement initiatives – The number of initiatives must be realistically considered. In general, the list should not be longer than can be achieved in 12 months or less.

There are a number of ways to present the software process assessment results, but typically an assessment report will be issued followed up by a presentation of the results. Some organizations will also integrate training for specific practices with the presentation of the assessment results. Each organization will ultimately have to decide the most effective way to communicate and initiate the proposed changes. At a minimum, it's essential to plan and schedule the proposed initiatives.

Figure 4 shows an example software process improvement backlog. Each initiative is broken down into subtasks with task ownership identified, a tentative target date, and current status. Figure 5 shows the same tasks presented on a visual Kanban board for display in a public area such as a shared hallway.

Software Process Improvement Backlog				
Initiative / Task	**Owner**	**Target Date**	**Status**	**Notes**
Resharper		**End of Q1 2011**	**In progress**	
Acquire Resharper licenses.	Steve	12/2/2010	Complete	
Install Resharper on each workstation.	Team	12/9/2010	Complete	
Provide additional Resharper training for the software team (if needed).	Steve	Q1 2011	Complete	not needed
Create a Resharper code style profile for the DAPL C# coding standard.	Bob	Q1 2011	Not started	
Peer Reviews / Code Inspections		**End of Q1 2011**	**In progress**	
Decide on a Peer Review tool.	Bob	10/27/2010	Complete	
Acquire PeerReview Complete	Bob and Jill	12/16/2010	Complete	
Establish a permanent server hosting environment for PeerReview Complete.	Bob	1/27/2011	Complete	
Install and Configure PeerReview Complete	Bob	2/17/2011	Not started	
Unit Testing		**End of Q1 2011**	**In progress**	
Retrofit existing Apollo unit tests into Microsoft Test framework.	Steve	11/18/2010	Complete	
Create a DAPL unit test standards document.	Bob	1/27/2011	In progress	
Setup initial unit test project structure in the Apollo .Net solution.	Steve or Bob	1/27/2011	In progress	
Revise sprint completion criteria checklist to specify unit testing requirements.	Bob, Steve, or Joe	Q1 2011	Not started	
Research unit test alternatives for embedded C/C++ code.	Bob	11/25/2010	Complete	FlexeLint is the best option.
Coding Standards		**End of Q1 2011**	**In progress**	
Investigate and adopt a C# / .Net coding standard.	Bob and Steve	2/10/2011	In progress	Current proposal is to use the *.Net Framework Design Guidelines* book and FxCop standards.
Document new coding standard.	Bob	2/24/2011	Not started	If we use the *.Net Framework Design Guidelines* coding standards, we'll just purchase multiple copies of the book.
Provide additional review and training of the coding standard as needed.	Bob and/or Steve	Q1 2011	Not started	This may include training on defensive programming and common gotchas.
C# Static Code Analysis (FxCop)		**End of Q2 2011**	**Not started**	
Determine the policies and guidelines for applying static code analysis to the existing code base (e.g. which projects to analyze, how to manage findings).	Bob, Steve, and Joe	4/13/2011	Not started	
Perform initial static code analysis, tune exceptions, and generate report.	Bob	5/18/2011	Not started	
Integrate static code analysis into build process.	Bob, Steve, Pam	Q2 2011	Not started	
Manage static code analysis findings on an ongoing basis.	Bob	Ongoing	Not started	
Continuous Integration		**End of Q2 2011**	**Not started**	
Acquire a server to host the continuous integration environment.	Bob	Q1 2011	In progress	We will be using CruiseControl for continuous integration.
Determine if ClearCase will work for continuous integration or if we need to use Subversion.	Bob	Q1 2011	Not started	
Install and configure continuous integration environment.	Bob	Q2 2011	Not started	
Provide continuous integration training for the software team.	Steve	Q2 2011	Not started	
Integrate continuous integration environment into day-to-day development.	Team	Q2 2011	Not started	

Figure 4 - Example Software Process Improvement Backlog

Figure 5 - Example Software Process Improvement Kanban Board

5. Conclusion and Lessons Learned

The SPI process is not a one-time endeavor. Change must be incremental and continuous to have a lasting impact. The basic continuous improvement framework is based on the Plan-Do-Check-Act (PDCA) model, otherwise known as the Deming Cycle or Schewhart Cycle. The PDCA model is the fundamental basis for all continuous improvement. It's expected that most organizations will customize PDCA to meet their unique needs.

In a software organization committed to continuous improvement, software process assessments should be performed every year or two and new process improvement initiatives should be regularly proposed and implemented, even if it's just one or two initiatives at a time. SPI is ultimately an investment that is amortized over time. As such, SPI introduces overhead and consumes organizational resources. Each organization will have to determine the optimum rate of change.

To a certain extent, all software organizations are complacent and driving change is essentially hard. One should expect resistance to SPI, but this should not be discouraging. If SPI were easy, then every software organization would always deliver high quality software on time. There are a number of ways to mitigate change resistance.

- Aggressively pursue management support.
- Enlist the organizational thought leaders in the SPI process.
- Utilize pilot initiatives limited to one project team or a subset of the organization.
- Track quality metrics to illustrate both improvement opportunities and SPI successes.
- Always propose initiatives with the highest ROI potential first.

The assessment process itself should also be continuously improved. Learn from your assessment experiences. Make use of post-assessment retrospectives to collect feedback on what went wrong, what went right, and ideas for improving the assessment process in the future. Between assessments, look for ways to improve the SPI process overall (e.g. keep a SPI notebook). Promote software quality and process maturity continuously.

In the end, SPI is as much of an art as it is a science and art must be experienced. The following is a list of hard-learned lessons based on the author's experience. Your mileage may vary.

1. If there's one thing the reader should take away is that SPI does not need to be a heavyweight undertaking. By focusing on continuously improving high ROI processes a few practices at a time, rapid and permanent change is very attainable.
2. Be aware of the Hawthorne Effect, which says that simply observing an organization often changes the behavior of the organization. Utilize this phenomenon by regularly shining a light on quality problems and process areas that need improvement.
3. SPI works best if change develops from within a team or an organization and is not imposed from the outside. Stand-alone quality groups rarely are able to effect the long-term change required to fundamentally improve software quality.
4. Very often, the best SPI feedback and ideas come from front-line employees who perform the actual productive work (e.g. developers, project managers, SQA staff, etc.).
5. It's almost always best to implement the best practices with the highest potential ROI and then heavily leverage early successes to motivate further improvement and change.
6. Long-lasting and transformative SPI is essentially cultural change. Changing the culture of an organization requires concerted effort and takes considerable time. Persevere, but be patient.
7. Crisis can be a great opportunity to implement change into an organization. If your software organization is resistant to change, be ready to seize crisis opportunities.
8. Software productivity and quality go hand in hand. However, quality always leads productivity. When a software organization fails to deliver on time, quality is almost always at fault.
9. It's essential to be realistic about what can be expected regarding the level and pace of change. Every organization has its own velocity at which change can be absorbed. Know this velocity.
10. Management support for SPI is very important. But, SPI can also be pursued "under the radar," especially if done organically at the team level.
11. Most successful SPI efforts are driven by a process champion who is an integral and respected member of a team or an organization.
12. Carefully chosen metrics can be very insightful and motivational. Nothing convinces management like solid numbers. Leverage metrics to drive SPI, if possible.
13. Ideally, SPI initiatives should be incentivized by management.
14. Ironically, resistance to change is often greatest from managers and team leads. It's up to the process improvement champion to convince management that SPI improves an organization's performance in ways relevant to them such as improving productivity, quality, and the organization's ability to innovate.

References

[1] W. Humphrey. Managing the Software Process, Addison-Wesley Reading Mass., 1989.

[2] Software Engineering Institute, Carnegie Mellon University, http://www.sei.cmu.edu/cmmi/.

[3] Available at http://www.iso.org/iso/search.htm?qt=15504&published=on&active_tab=standards.

[4] M.E. Fayad, M. Laitinen and R.P. Ward, "Software Engineering in the Small," Communications of ACM, vol. 43, no. 3, March 2000, pp. 115-118.

[5] M. Paulk, "Using the software CMM in small organizations," in Joint Proceedings of the Pacific Northwest Software Quality Conference and the Eighth International Conference on Software Quality, Portland. 1998, pp. 350–361.

[6] M. Laitinen, M. Fayad, R. Ward, "Guest Editors' Introduction: Software Engineering in the Small," IEEE Software, vol. 17, no. 5, September 2000, pp. 75-77.

[7] A.P. Cater-Steel, "Low-rigour, Rapid Software Process Assessments for Small Software Development Firms," in Proceedings of the Australian Software Engineering Conference, 2004.

[8] SPIRE Project Team, The SPIRE handbook; Better, Faster, Cheaper Software Development in Small Organizations, Centre for Software Engineering, Dublin, 1998.

[9] J. Bach, "Enough about process; What we need are heroes," IEEE Software. Vol. 12, no. 2, Feb. 1995, pp. 96-98.

[10] T. DeMarco, T. Lister, Peopleware: Productive Projects and Teams. New York: Dorset House Publishing Co.,1999.

[11] D.P. Kelly, B. Culleton, "Process Improvement for Small Organizations," IEEE Computer, vol. 32, no. 10, October 1999, pp. 41-47.

[12] G. Yamamura, "Head to Head: Process Improvement Satisfies Employees," IEEE Software, vol.16, no. 5, Sept./Oct. 1999, pp.83-85.

[13] J. Brodman and D. Johnson, "Return on Investment from Software Process Improvement as Measured by U.S. Industry". Crosstalk, Vol. 9, no.4, April 1996, pp.23-29.

[14] M.E. Fayad, M. Laitinen (1998) "Process Assessment Considered Wasteful," Communications of the ACM, November 1997, vol. 40, no 11, pp.125-128.

[15] J. Brodman and D.L. Johnson, "What Small Businesses and Small Organizations Say about the CMM" Proc. 20th Int'l Conf. Software Eng., IEEE CS Press, 1994, pp. 331-340.

[16] P. Grunbacher, "A Software Assessment Process for Small Software Enterprises," in Proceedings of the 23rd EUROMICRO Conference '97 New Frontiers of Information Technology, IEEE CS Press, 1997, pp. 123-128.

[17] G. Chroust, F. Stallinger, "Software Process Capability in Europe: A Survey," presented at EUROMICRO 99 – European Software Day, Milan, Italy, 1999.

[18] S.M. Garcia, "Thoughts on Applying CMMI in Small Settings," 2005, http://www.sei.cmu.edu/cmmi/adoption/pdf/garcia-thoughts.pdf.

[19] T. Pyzdek, "To Improve your Process: Keep it Simple," IEEE Software, vol. 9, no. 9, September, 1992, pp. 112-113.

[20] F. Guerrero and Y. Eterovic, "Adopting the SW-CMM in a Small IT Organization," IEEE Software, vol. 21, no. 4, July/August 2004, pp. 29-35.

[21] K. Wiegers, "Software Process Improvement: Ten Traps to Avoid," SD Times, May 1996.

[22] Software Engineering Institute, "CMMI for SCAMPI Class A Appraisal Results 2011 End-Year Update," March 2012, http://www.sei.cmu.edu/cmmi/why/profiles/upload/2012MarV3CMMI.pdf.

[23] Software Engineering Institute, "CMMI for Development, Version 1.3", November 2010, http://www.sei.cmu.edu/reports/10tr033.pdf.

[24] K. Wiegers, Creating a Software Engineering Culture, New York, NY: Dorset House Publishing, 1996.

[25] Entinex, Inc. (2011, October 9). *CMMIFAQ* [Online]. Available: http://www.cmmifaq.info/

[26] C. Jones, Software Engineering Best Practices, New York: McGraw-Hill, 2010.

Appendix - Sample Software Process Assessment Questionnaire

The following is a sample software process assessment questionnaire based on the ISO 15504 Software Process Improvement and Capability dEtermination (SPICE) process framework. The questionnaire was used by the author for a process assessment performed in 2005. Each interview took 1-2 hours to complete.

Management Processes

1) What is your opinion of the company's management team? Do you feel the management style in place is effective?

2) What would you change about the company's management style (if you could)?

3) What is your opinion of the company's project management? Do you feel that projects are well managed and executed? What changes / improvements would you like to see?

4) What is the company's basic project management approach? Is there a repeatable project management standard in place?

5) How is project progress reported and how often? How is a project monitored for acceptable progress?

6) Are there instances or generally areas where the existing project management approach may not be adequate?

7) What does "quality management" mean to you? Do you think the company has a quality focus? How is quality managed here?

8) How is risk managed in a typical project? What's the typical approach when a risk is realized and becomes a problem?

Organizational Processes

1) Do you feel the company relies on exceptional people (i.e. heroes)? Is there any concern that we may not be able to continue hiring heroes?

2) What role does training have at the company? Would more or less training be desirable?

1) How does the company typically illicit *customer* requirements? Is it always done in the form of and RFP / Proposal? What are the typical constraints placed on a project by a customer? Does the company use standard document templates for proposals?

2) Describe the process for initial customer requirements elicitation? Who's involved and how are stakeholders identified? What's the process for ongoing customer requirements changes?

4) Once coding actually commences, how is the actual code produced verified against the requirements and design?

8) What does the term "validation" mean to you? What validation processes does the company have in place (as far as you know)?

9) At what points in the software process do "joint reviews" occur? In your opinion, are they valuable?

10) In your opinion, do you think internal formal code and design reviews would be valuable?

11) How are problems typically resolved? Is there a formal process? How do contractual obligations play into the problem resolution process?

General Questions

1) What effect does the company's desire for follow-on work effect its software process discipline?

2) Are the company's projects growing in size and complexity?

3) Do you feel the company's existing software processes will scale as it grows?

4) How do you see the company's "culture" and how amenable do you think it is to a more formal software process approach?

5) Are there particular communication problems in the company? If so, when and where?

6) In your opinion / experience, what are the most pressing software process problem areas?

7) Finally, if you could change one thing about the company's approach to software development, project management, or culture, what would it be?