Technical Report, University of Maryland, Computer Science Dep., College Park, MD 20742. August, 1995.

AINSI An Inductive Method for Software Process Improvement: Concrete Steps and Guidelines*

Lionel Briand, Khaled El Emam and Walcélio L. Melo**

Paper Published in the
Proc. of the ESI-ISCN'95: Measurement and Training Based Process Improvement, Sep. 11-12 1995, Vienna, Austria.

Abstract

Top-down approaches to process improvement based on generic "best practice" models (e.g., CMM, TRILLIUM, BOOTSTRAP, SPICE) have become popular. Despite the idiosyncrasies of each of these approaches, they share some common characteristics: all of them are based on numerous assumptions about what are best practices, and about the business goals of organizations and the problems they face. Other organizations, like the Software Engineering Laboratory of the NASA Goddard Space Flight Center, *HP and CRIM in Canada, have adopted the Quality Improvement Paradigm (QIP). The QIP stipulates a more bottom-up and inductive approach to process improvement. The focus of this paradigm is to first understand what processes exist in the organization and to determine what causes the most significant problems. Based on this, opportunities for improvement are devised, and empirical studies are conducted to evaluate potential solutions. In this paper, we present a method, named AINSI[1] (An INductive Software process Improvment method), which defines general but concrete steps and guidelines for putting in place the QIP. This method is the result of the collective experiences of the authors and integrates many lessons learned from process improvement efforts in different environments. It also integrates many complementary techniques such as qualitative analysis, methods for data collection (e.g., the Goal/Question/Metric paradigm), and quantitative evaluation.*

1. Introduction

Top-down process improvement approaches (e.g., the Software Engineering Institute's Capability Maturity Model for software (Paulk *et al.* 1993), TRILLIUM (Coallier 1995), BOOTSTRAP (Haase *et al.*, 1994), SPICE (Drouin 1995)) provide a high-level model of what ought to be the process of a software development organization. Such models are based on the consensus of a designated working group about how software should be developed or maintained. They are very useful in the sense that they provide general guidelines to people who do not know where to start improving, and in which order. Also, these models are useful in contract award situations where alternative bidders may be comparatively evaluated.

A general assumption of these models is that the more an organization's processes match the stipulations of the model, the greater its effectiveness on some criteria (e.g., product quality, productivity, predictability etc.). In general, there is a dearth of evidence supporting this

---

* Briand and El Emam are with the Centre de Recherche Informatique de Montréal (CRIM), 1801 McGill College Av., Suite 800, Montréal (Québec), H3A 2N4, Canada; Melo is with the University of Maryland, Institute for Advanced Computer Studies, College Park, MD, 20742. E-mails: { lbriand | kelemam}@crim.ca, melo@cs.umd.edu.

** Authors are listed in alphabetical order.

[1] *Ainsi* is a French word which means "in this way".

assumption. For instance, in the context of CMM-based assessments, Hersh (1993) states *"despite our own firm belief in process improvement and our intuitive expectation that substantial returns will result from moving up the SEI scale - we still can't prove it"* and Fenton (1993) notes that evaluating the validity of the SEI's process maturity scheme is a key contemporary research issue. There is now some *initial* evidence supporting a positive relationship between the CMM's maturity levels and some subjective criteria of including quality and productivity (Goldenson and Herbsleb 1995). In addition there is some *initial* evidence based on two BOOTSTRAP case studies showing that when weaknesses identified by an assessment are addressed, improvements in quality and productivity result (Haase *et al.*, 1994). Similar studies are being conducted as part of the SPICE project (El Emam and Goldenson 1995). However, despite such commendable efforts, the overall evidence remains weak and is contradicted by other works (El Emam and Madhavji 1995c).

Furthermore, in practice, it is not so obvious that these models will address the causes of problems in all organizations. For example, an analysis of findings from assessments based on the CMM (for which the appropriate data was available) identified that more than half the organizations had problems in areas not covered by the CMM (Kitson and Masters 1993).

Alternatively, one could adopt a more inductive approach to process improvement where the focus of the first step would be to understand what exists in an organization and determine what causes significant problems. Then, solutions could be devised and evaluated in pilot studies or even controlled experiments (Basili, 1992). Only after a solution is found to be effective and efficient, then it should be integrated into the existing process or the process may be modified. Such an approach is inspired by the Quality Improvement Paradigm (QIP) recommended by several researchers and practitioners (Basili, 1992). The QIP is a specialization of the scientific method for software engineering research.

The question now is to determine how such an approach can realistically be implemented in different software organizations. One well known and typical example is the Software Engineering Laboratory (SEL) at the NASA Goddard Space Flight Center, where the understand, assess, change paradigm has been in place for the last 20 years. Experience has shown that the cost of such a continuous software process improvement mechanism can be kept under a reasonable percentage of the development cost. This paper is in large part based on the SEL experience and a few other process improvement experiences in which the authors were involved (Briand *et al.*, 1994; 1995; El Emam and Madhavji 1995a; 1995b).

This paper is structured according to the phases of a process improvement activity as we see it. Although these phases are presented sequentially**,** in most cases they are overlapping. We try not to focus exclusively on technical issues but to provide a complete overview supported by key references.

2. Set up a process improvement team

The first and most critical activity at start-up is getting the commitment of management to software process improvement. One of the major reasons why process improvement programs fail is that management did not provide the adequate resources to make it happen. It is the case that many organizations whose main business is not software development may not consider software to be important. Therefore they are reluctant to invest in software process improvement. Efforts within such an organizational context are commonly an uphill battle with management to make resources available for improvement

It is widely recognized that a process improvement team separate from the developers must be set up at the start of an improvement effort (Fowler and Rifkin 1990; Software Engineering Laboratory 1995). The main reason for this is that the priority of the developers is to produce systems, and therefore process improvement activities will not receive as much attention as they deserve or need. A process improvement team has process improvement as its priority.

The SEI calls this process improvement team a Software Engineering Process Group. This group would be staffed, ideally full-time, by experienced technical and credible personnel from the organization, and may also be staffed by researchers and external consultants. Similarly, the SEL calls such a group the "analysts" who have a mix of backgrounds in empirical methods, data analysis, and software development and maintenance. Some are experienced researchers and scientists, others are senior managers or software engineers. It is important that the individuals in this group have strong technical knowledge and are respected in the organization. One sign of a lack of management commitment to process improvement is when they assign those people that they "don't know what else to do with" to a process improvement team.

Whatever the label, the process improvement team is responsible for initiating and carrying out many of the process improvement activities, such as setting up training, designing evaluation studies, defining the goals of measurement, studying developed projects to feedback to the organization any lessons learned, following the execution of actual activities in order to find deviations and opportunities for improvement, etc.

The mandate of the process improvement team must be well defined and must be tied to important business goals. In some cases, the process improvement team is formed with the mandate to help the organization become ISO 9001 certified within, say, one year. This may considered to be tied to business goals because it will help the organization get contracts or because it is demanded by a major customer (especially in Europe). Similarly, some organizations consider achieving Level 3 on the CMM to be important for business because it would help them get contracts (especially for the defense community in the US). Progress is then measured by the proportion of Level 3 practices that have been implemented or the extent of compliance to ISO 9001 clauses. Success is measured by having all necessary practices implemented and ISO 9001 clauses adequately satisfied. This, using the terminology in (Schaffer and Thomson 1992), is termed an activity-driven approach to improvement.

Strict adherence to the activity-driven approach has a number of disadvantages. The organization does not need to actively measure important variables like product quality, productivity, and meeting budget and schedule targets, because these are not needed to evaluate progress and success. If they do, this data is not used to evaluate progress and success. Therefore, the organization may continue to put practices in place without knowing the effects on product quality, productivity, and predictability. Consequently, the organization would not know if all the activities that have been implemented really provided any quality, productivity, or predictability benefits.

Putting practices in place on the faith that they are somehow "good" is not prudent. As noted in (Basili, 1992), many of our intuitive ideas about software development may turn out to be incorrect once put to empirical test. In general, many practices that are in current use in software engineering have not been adequately empirically tested, and so we do not know if they really work, and under what conditions they do work. If the organization expects practice X to improve, say product quality, then the process improvement team's mandate would include improving product quality. The process improvement team may subsequently decide that practice X is a worthwhile candidate, and evaluate it internally within the organization to determine if it really improves product quality and by how much. It is therefore important not to confuse or be ignorant of the difference between cause and effect. Process improvement efforts ought to be driven by the desired effect, i.e., results driven.

The process improvement team should start by creating an awareness in the organization about the existence of the team, its mandate, objectives, and approach. The purpose of this is to enlist the support and cooperation of the organization's members. This is necessary in preparation for the subsequent activities of the improvement effort.

3. Model the existing process

Process modeling may serve many purposes and is believed to have many benefits (Finkelstein *et al.* 1994). In the context of process improvement, descriptive process models are useful for understanding the way things currently work in the organization and for communicating this understanding. Below we discuss four important issues in process modeling: (a) how to collect information for process models, (b) what information to collect, (c) the formalisms to use for modeling, and (d) the appropriate level of detail of process models.

There are a number of methods for eliciting process modeling information. One method based on experiences at Contel is given in (McGowan and Bohner 1993). This has four main phases: observe the organization and interview staff, determine context, purpose, and viewpoint of model, construct process model, and verify the process model. Madhavji *et al.* (1994) have generalized from their experiences and presented a method for eliciting information suitable for the construction of process models. Some of this method's main steps that are most relevant here are: understanding the organizational environment, defining the objectives of the modeling exercise, planning the elicitation strategy, developing the process model, and validating the process model. Similarly, Briand *et al.* (1994; 1995) have described their acquisition process for collecting information suitable for the construction of models. The steps in the acquisition process include: determining the hierarchical structure in the organization, determining the roles covered by the positions in the hierarchy, and identifying the various dependencies within the reporting hierarchy. These three methods should serve as reasonable starting points for a process modeling effort. Furthermore, a discussion of issues to consider in a real world process modeling exercise is given in (Henry and Blasewitz 1992). In particular, the authors of that article suggest tasking the developer groups to build models describing their own processes.

Some of the basic questions that need to be answered in a process model are: What are the tasks in the process? What are the dependencies between these tasks? When do the tasks start and end? Who are the actors that perform these tasks? What are the interdependencies between the actors? A conceptual framework defining information that could be collected has been presented in (Armitage and Kellner 1994). This framework has three entity classes (activities, agents, and artifacts) and two aspects (relationships within and among entity classes, and behavior of the entity classes and the relationships). Another article presents a set of process modeling attributes which has perspectives (such as process steps, roles, resources, and constraints) and properties for each of the perspectives (Madhavji *et al.* 1994). Based on contemporary knowledge, these articles provide a comprehensive view of process information that can be collected.

Sources of information that we have found useful for answering these questions and for collecting relevant information are: interviews with members of the organization, inspection of process documentation if any exists, inspection of project management documentation (such as project plans), and direct observation of development staff (for example "shadowing" and attendance of meetings). One issue that ought to be considered, especially in a large and diverse organization and when the modeling effort has a wide scope, is differing terminologies used within the organization. A common terminology (that is acceptable to all developers in the organization being modeled) should be found. Suggestions for achieving this include: debating the issue until an acceptable compromise has been found, giving the components of the model (e.g., the documents produced or the activities) identifying numbers, and using terminology from public standards (e.g., military, IEEE or ISO standards).

Many formalisms for process modeling have been developed. Few of these formalisms have actually been used in practice. Some that have been used and that have been found to be useful are Statemate (Kellner and Hansen 1989), structured English, ETVX (Radice et al. 1985), the Actor-Dependency modeling approach (Briand et al. 1995; Yu and Mylopolous 1994), the commonly used Data Flow Diagrams (Frailey 1991) and the SADT notation (McGowan and Bohner 1993). The latter four support a single perspective, while the former integrates multiple perspectives (these are state-transition, data flow, and structure).

The models developed at this point will be useful for the analyses to be conducted later. It should be remembered, however, that the developers must be able to understand the formalism that is used. Therefore, a formalism that is most similar to those that are commonly used in software development, for example data flow diagrams or control flow diagrams, is likely to be the most easily understandable and rapidly accepted, even if it is considered less “powerful” than some other formalism. Another issue that needs to be considered is the ease of change of the process models. As the process improvement effort progresses, it will be necessary to change the process models. The formalism that is chosen ought to be easy to change. This may be facilitated by the use of an automated tool that supports the chosen formalism, e.g. TEMPO (Belkhatir and Melo 1994).

It is difficult to decide how deep we must go into the software process definition and modeling. In general, we would discourage the development of very detailed models, mainly because we have not found this excessive to be useful and their construction consumes considerable resources. It must be remembered that the objective is not to automate software processes, it is only to understand and communicate.

The process models that are developed will be used in subsequent steps of the improvement effort. However, the models also help the developers understand their process and how it fits into the overall organization. To be useful for developers, the models must make sense to them. The best sign of this is when the developers make copies of (parts of the) models and hang them in their offices; then they are finding them useful.

4. Conduct qualitative analysis

The goal at this stage is to identify the causes and inner mechanisms that lead to costly or risky problems related to the quality of the delivered products or the efficiency of the development process. The analysis described in Section 2 is providing the context in which the analyst can inquire about the problems. A well defined process model will help differentiate realistic working hypotheses about problems from non-relevant issues. There are three main alternative techniques for performing qualitative analysis in the current context: structured interviews as performed in knowledge acquisition, causal analysis of problems, and well-designed questionnaires. In fact, these techniques are usually complementary since they allow cross-checking of information.

When performing causal analysis, the analyst usually starts from a problem report coming from the testing phases or operation of a particular, but representative, software system. The analysis that will lead him/her to determine what happened will be based on multiple sources of information: the process documentation, the product documentation, the project (release or new development) documentation, (structured) interviews (Vogel, 1988) with the people involved in the process and, if possible, the originator of the problem report and the owner(s) of the product part where the problem was identified.

Of course, it is difficult to provide a precise procedure to perform such an analysis and the experience and talent of the analyst will be an important criterion for success. However, several useful guidelines exist in the literature (Collofello and Gosalia 1993; Chillarege et al. 1992; Nakajo and Kume 1991) and, for example, the authors of this paper were involved in defining various methodologies focusing on maintenance (Briand et al. 1994; 1995) and the requirements engineering process (El Emam and Madhavji 1995a; 1995b). Eventually, in a given organization, typical causal chains leading to problems can be devised by senior project managers and validated by causal analyses such as the ones discussed in this paper. This would help the originator of an error or a problem report to provide an explanation for what happened in a consistent terminology. This would, as a consequence, facilitate any future analysis to a great extent.

Performing causal analysis for a fault consists of determining: the fault type or category, the phase where the fault was found, the verification process that uncovered the fault, the phase where the fault was created, the cause(s) of the fault in terms of organizational or process flaw (when relevant), its severity, and the solutions that might prevent the fault from occurring in the future. In a given organization, typical fault occurrence patterns should be identified by associating types of faults to likely process and human causes. Based on our experience, an empirical investigation of a few representative projects ought to be sufficient to identify such patterns. In ideal cases, causal analysis should be performed shortly after errors have been detected and changes have been implemented and tested. However, very often, in order to speed up the analysis, the analyst has to study completed and carefully selected developments or releases. Also, not only faults should be investigated but delays or budget overruns could provide interesting insights.

Examples of fault classification schemes are provided in the references above. Such schemes need to be developed based on the specific experience of each organization so that they make sense for the organization's software engineers and technical managers. However, at the beginning, external experience reports can be a very useful source of ideas. The references provided above should be suitable in that context. Because causal analysis can be expensive and time intensive (it requires talented and experienced people), it should focus on costly changes or high-severity faults that could have led to significant problems in operations.

For example, Nakajo and Kume (1991) describe the following procedure. They link program faults with system failures. Then, they attempt to determine what type of human error is involved (e.g., communication error within the development team). Last, the human error is explained in terms of work system flaws (i.e., inherent characteristic of methods, workers and environments affecting human error occurrence and deviation.) Patterns of fault occurrences were identified by looking at the statistical associations between program fault types, human error categories and work system flaw categories. For example, interface faults (mismatching between software components) was found to be closely associated with communication problems between various component's development teams. It is important to note that taxonomies of work system flaws are inherently specific to the organization under study and, as a consequence, reusing causal analysis techniques is not always straightforward.

A general procedure to perform causal analysis would follow the following outline (Briand et al 1994; Collofello and Gosalia 1993):

1. Select representative system development(s) or release(s).
2. Obtain all problem reports.
3. Classify program faults according to pre-defined severity and type taxonomies.
4. Determine the causes of high-severity faults in terms of human errors and then process flaws.
5. Develop recommendations for process changes and validate them with the participants of the causal analysis (one should attempt to reach a maximum consensus on issues).
6. Refine causal analysis guidelines and taxonomies to help improve future analyses.

Performing such procedures assumes the existence of a well-defined process and organization model. Without a clear context for reflection, these steps are difficult, if not impossible, to perform, since consistency checks between the description of the process and organization, the documentation available, and the stories gathered during interviews are necessary. Such verification may lead to refine the perceptions the analyst has of a problem report but may also lead to the refinement of the defined process and organization models.

5. Define and document an action plan

Once the process in place and its main weaknesses and strength are well understood, an action plan must be defined. In (Fowler and Rifkin 1990), the action plan is defined as: *"A formal, written response to the (process) assessment, and the 'map' for improvement."* The action plan is extremely important for several reasons. First, it is required in order to get a suitable budget for the next phases (evaluation of solutions, changes to the process). Second, it is crucial to convey the right information to management and developers about the importance and difficulty of what is going to be achieved. This is particularly important since two common sources of failures for improvement programs are, among others, the lack of adequate funding/resources and senior management commitment.

In practice, it is very common to see action plans which do not specify all the activities involved (e.g., non-technical activities such as training), the responsibilities and prerogatives of all participants, the risks taken, and corresponding contingency plans. Process improvement in software is always exploratory and risky. An action plan should be carefully thought out and not overlooked. Once promises have been made to senior management and once participants have expectations, the software process improvement team has to make sure not to disappoint them, or the consequences might be disastrous for the future of process improvement in the organization. The participants should be made aware of all uncertainties and new challenges to come.

Another important point is that interdependencies between action plan tasks should be carefully studied. The impact of underestimating the efforts of certain tasks can create significant damages to the pertinence of the remainder of the action plan. It is very frequent, for example, to see action plans based on a very superficial understanding of the processes in place, or inspections introduced without adequate data collection to evaluate their impact or refine them.

Actions plans contain, at least, three levels of decomposition. Each step of improvement describes an increment from a lower to a higher state of practice, e.g., substitute informal peer reviews by inspections. Within each step, sequential and overlapping phases of improvement actions are described, e.g., for inspections, phases such as preparation of participants and material, implementation, and evaluation on pilot projects are commonplace. Within each phase, a set of (usually unordered) activities should be described to better convey the size of the phase and better estimate its required effort, e.g., for the preparation phase, design the inspection checklists and error log forms. Each document to be written, tool to be developed, training course to be given, etc. should be listed in the description of the activities.

A possible high-level outline of what an action plan could look like is as follows:

1. Corporate improvement objectives and motivations.
2. The various groups and participants of the process improvement action plan, their responsibilities and prerogatives (high level at this stage).
2. Improvement steps, their connections to corporate objectives, their entry and exit criteria.
3. The phases leading incrementally to the exit criterion of each improvement step, the risks and uncertainty associated with each phase and the corresponding contingency plans, the managers in charge of monitoring and organizing the execution of phases.
4. The set of activities involved in each phase of each step (normal procedures or contingency plans), the participants in each activity and their responsibilities (outputs, quality, schedule).
5. The effort and cost for each step and phase, with uncertainty intervals.
6. The expected benefits, based on external experiences and whatever information exists about the organization's productivity and quality.

6. Set up a measurement program

At first, a measurement program is generally designed to (1) provide a quantitative baseline of comparison for future process changes, (2) better understand the issues that may or not have been identified in the qualitative analyses preceding measurement (e.g., the high-cost of certain activities may make them priority improvement targets), and (3) make the process of decision making less risky by providing accurate and on time quantitative management information about the software products and the processes used to produce these products. There is no possible improvement or even technology assessment if one does not know the current quality and productivity baselines of his/her organization in terms, for example, of error density and KLOC's per man-month. Such a common sense principle is very frequently forgotten under the pressure for quick improvement and immediate tangible results!

In addition, it has been shown in many occasions that a measurement program is more likely to succeed if its definition is driven by corporate and project goals, e.g., reduce development costs by reducing requirements' definition instability (Bassman *et al.* 1994; Basili and Rombach 1988). The data collected must closely reflect in their definition, format, and collection procedures, the local constraints, practices, and needs of the organization.

Typically, a measurement program collects a mix of product and process data about pre- and post-release defects, the effort breakdown per activity and phase, the complexity and size of the systems under development or maintenance, etc. In the details, however, the definition of what a phase or an activity means, the classification taxonomies of defects, and the complexity measures assessed as relevant, can significantly vary from one environment to another.

Once a measurement program has been defined, participants must be trained in the data collection procedures and their feedback must be considered in order to pre-validate and refine the data collection program. It is important to recognize that the commitment of participants is one of the most important success criteria. Also, a great deal of useful information cannot be collected under the form of quantitative data. In some circumstances, questionnaires may be needed. A substantial literature exists in the social sciences in order to help collecting qualitative data (e.g., qualitative scoring scale) .

In general, data collection can be used for many different purposes. Post-mortem analysis of projects is one, e.g., build a project cost estimation model. Understanding project dynamics and evolution across phases is another one, e.g., the distribution of cost across phases. Measurement can be used for the sake of characterization (e.g., what are the most frequent origins of errors), evaluation (e.g., are my inspection techniques effective?), prediction (e.g., cost estimation model), and, of course, improvement (e.g., what are the most frequent causes of high-severity errors).

The usual steps of implementation of a measurement program can be summarized as follows :

1. Define corporate and measurement goals.
2. Derive models (evaluation, prediction) and measures that appear suitable in the specific context of measurement.
3. Define data collection procedures to collect valid and accurate data.
4. Train participants to data collection procedures.
5. Test data collection and validation procedures on pilot projects, simplify and refine them as needed.
6. Expand the use of measurement to the whole organization.
7. Analyze collected data to assess the usefulness of the data and the accuracy of the models.
8. Refine models, measures, and data collection procedures, return to step 4.

Some usual sources of problems when implementing a measurement program are:

- A lack of thorough technical reflection about the defined measures and models, their implicit assumptions and implications, their properties, and how valid they are in the specific context of measurement.
- Data collection is not goal driven.
- The goals of measurement are not clearly specified and communicated to participants.
- Too many goals are addressed at once.
- Conflict between high-level (corporate) goals and lower-level (practitioners) goals. If only top-down (manager) goals are defined, developers may feel unmotivated in participating in the data collection. On the other hand, if only bottom-up (developer) goals are set, managers may not be interested in spending resources on a measurement program. Compromises should, thus, be found.
- Data collection procedures are not well-designed and disruptive.
- Developers are asked to collect too much process data.
- The people in charge of measurement do not have the right training, education and/or experience.
- There is no real management commitment, participants are not clearly convinced and/or feel threatened by measurement.

For more details, see (Bassman *et al.* 1994; Briand, Morasca and Basili, 1994; Hall and Fenton 1994; Grady 1992).

7 Perform a pilot project

Once problems are identified, options for improvement ought to be defined and evaluated. Example options would be the introduction of new practices, like code inspections, changing of programming language (e.g., Waligora *et al.* 1995), the introduction of new tools, like tools for enforcing coding standards, etc. In all cases, however, the identified problems should guide the identification and selection of possible options.

Ideally, a pilot project would be initiated to evaluate the new practice(s) and/or tool(s) (henceforth referred to by the generic term *technology*) that have been selected. Evaluation in the context of the host organization is important. This is because for many new, or even old, technologies in software engineering, there is little empirical evidence supporting their claimed benefits. A good discussion of this is given in (Fenton *et al.*, 1994) for formal methods, and a good example in (Melo *et al.*, 1995; Basili *et al.*, 1995) for Object-Oriented methods. For instance, most OO methods currently available have never been empirically validated. In addition, even when there are a handful of empirical studies evaluating a technology, and showing positive results, one would not be prudent to conclude that the benefits are achievable in his/her organization. Pilot projects would also have other benefits, such as providing feedback for *tailoring the technology* to the organization, and *generating buy-in* within the organization for the new technology.

The scope of the pilot project has to be defined. This is the number of projects and personnel that will take part in it. In general, a small part of the overall organization (in terms of projects or personnel) would be selected. In small organizations, it may sometimes be necessary to include all personnel in the pilot project.

A decision has to be made regarding the evaluation strategy. Kitchenham et al. (1994) have identified three strategies that may be followed: (a) formal experiments, (b) case studies and (c) surveys. In the same the authors express a strong preference for case studies in conducting evaluations in industrial settings. However, another evaluation strategy may be followed, namely quasi-experiments (Cook and Campbell 1979).

In quasi-experimental designs, it is recognized that there are many variables that are difficult to control (as one would attempt to do in a formal experiment). In particular, subjects are not assigned to treatment and control groups randomly, resulting in nonequivalent groups. To illustrate this point, we take a simple example. Assume we were to evaluate a technology by letting a group of people use it (the treatment group), and comparing their performance to another group that does not use the technology (the control group). In a formal experiment one would assign subjects to the two groups randomly (e.g., using random number tables).

For a pilot project in an industrial setting, this approach may not work because of the setting and because there are other objectives to the pilot study than just evaluation (e.g., create buy-in). For instance, there may be specific individuals that you want to be in the treatment group. Also, there may be a diffusion of treatment from the treatment to the control groups if they are in close physical proximity and/or if they interact extensively on a regular basis; in such a case we may select the two groups to minimize interaction. If there is diffusion of treatment, then the control group will no longer serve the no-cause-baseline function, and the two groups will be more similar than planned. Another effect that has to be considered is that of withholding treatment. Given that in some cases subjects in the pilot study may have volunteered (by themselves or by their managers), it may not be prudent to withhold the treatment (i.e., the technology use). In the case of random assignment, the subjects will not know until after volunteering whether they will be in the treatment or control groups. This may create resentment, which is not a good thing if you want to create buy-in within the organization.

For the above reasons, quasi-experiments are suggested as an alternative for formal experiments. It should be noted however that if the unit of analysis is larger than the individual programmer or small team, the expense of any such experimental design may be too large to justify.

A simpler strategy is to use case studies. With case studies one compares the outcome of the pilot with an existing baseline. It would not be useful to conduct a case study where no baseline exists. A good set of examples of case studies are the experiences of the SEL with ADA and an Object-Oriented analysis/design method (Waligora *et al.* 1995).

The selection of projects and/or personnel for pilot studies must balance three concerns (Leonard-Barton and Kraus 1985; Veryard 1987): (a) the risk of failure, (b) achieving the benefits that are expected (i.e., the value of the technology), and (c) their representativeness and credibility to the rest of the organization.

If, for example, a low risk project is selected in order for the improvement effort to survive politically, it may not provide benefits or be representative. If the personnel selected are the most capable in the organization, then it would not be clear from the outcomes whether the benefits were due to the people or the technology. The same applies if the worst capable personnel were selected. The motivation of subjects or project managers taking part in the pilot should also be examined. If such personnel are not too eager, then the pilot project may fail. Conversely, if such personnel have too much enthusiasm, this may be indicative of another agenda that may place the project at risk. Some general recommendations on selection of personnel are (Kitchenham et al. 1994):

- do not allow technology "champions" to run evaluation exercises
- use normal procedures to staff pilot projects; do not select people who are particularly enthusiastic or particularly cynical about the technology

The pilot project should be sufficiently small and simple (otherwise it may take too long and may overtax the as yet novice personnel). It must also be large and complex enough to be credible to the remainder of the organization and to provide realistic feedback for tailoring the technology.

It is critical that the participants in the pilot project receive adequate training in the new technology. Training is NOT an overview or briefing session. A training course will properly prepare the participants to use the technology to solve realistic problems. For the training course(s) all the necessary material for conducting the pilot must be prepared. For example, for training on inspections, all the data gathering forms and inspections procedures must be available. Furthermore, the pilot project should commence shortly after the training so that participants do not lose interest and knowledge with the passing of time.

8. Change the process and the organization

Subsequent to the pilot project and the review of its results, the technology would be transferred to the remainder of the organization. Lessons learned from the pilot should be used to tailor the technology and its training materials to the organization. Also, the data collection procedures may have to be modified to take account of lessons learned in the pilot project.

To transfer the technology to the remainder of the organization, a number of issues require particular attention. First, the reward structure must be changed to reflect the goals of the technology and/or to be congruent with the technology. For example, if the current organizational culture rewards productivity, then the introduction of structured analysis and design methods, which slow down the early phases of a project, are unlikely to be adopted by the organization, or if adopted are unlikely to be used effectively. Second, there must be a number of consultants expert in the technology available to solve problems and, if necessary, to provide coaching to members of the organization.

Given that the introduction of the technology will involve changes to the process and possibly to the organizational structure, the process and organizational models that were developed earlier have to be modified to reflect the changes. This ensures that the models are current.

Depending on the resources available and the organization's capacity to change and to support change, the implementation of the technology may have to be incremental. This means that only a part of the organization changes at a time rather than the whole organization.

It is also necessary to continue the collection of data on an on-going basis. This will allow for continuous monitoring and evaluation of the technology that was inserted in the organization.

9. Conclusions

We have presented in this paper a bottom-up approach for the practical improvement of software processes and products (AINSI: An INductive Software process Improvement method). It may be seen as an instantiation of the more general Quality Improvement Paradigm (Basili, 1992). AINSI differs from top-down approaches which are intended to provide an ideal framework for process improvement, e.g., SEI CMM, because it relies on a more detailed interpretation of available and interdependent sources of information on the software organization. Therefore, we attempt to identify issues with very little preconceived ideas about how the process should look like. In order to facilitate its application in different software development and maintenance organizations, we have provided a set of concrete steps and guidelines. Actually, we have tested and constantly refined it based on the experience we have been acquiring by conducting qualitative and quantitative studies on public and private organizations. In the remainder of this section, we present some lessons learned based on these studies.

First, we think one should not confine oneself to a single learning strategy (e.g., only controlled experiments). On the contrary, it is important to remember that there are many different ways to learn. We should use our common sense to decide which strategy is most appropriate given the decision we want to make and the resources we have available. For example, controlled experiments allow a better level of control of the factors under study; and they can be more easily replicated in different environments. However, this is because it may be difficult or impossible to apply such an approach that social scientists have developed techniques such as quasi-experimentation (Cook and Campbell, 1979). Since it takes time for a technology to mature in an organization, case studies are in general also necessary in order to tailor it to the organization based on the feedback from developers and managers [Kitchenham *et al.*, 1995]. For example, at the SEL, only after the execution of some case studies did it become evident that Cleanroom produced benefits in quality and productivity (Basili and Green, 1994) only if a few modifications to the original method were implemented .

Second, qualitative studies should be seen as complementary to quantitative studies. In general, developers in an organization have many useful insights into existing problems and ways to address them; they are a valuable source of information in process improvement. Knowledge acquisition methods like structured interviews (Vogel, 1988) would help interpret quantitative data and provide ideas for solving problems.

In closing, one problem we have witnessed is the attempt by some organizations to try to solve their problems too rapidly. This is manifested in improvement programs with dozens of working groups and technologies being introduced concurrently. In practice, many organizations do not have the capacity to change very rapidly without considerable disruption to ongoing projects. Developers cannot continue to produce software at the previous rate, and change all their practices at the same time. There is no magic.

References

J. Armitage and M. Kellner (1994). “A conceptual schema for process definitions and models”. In *Proc. of the 3rd Int'l Conf. on the S/W Process*, pp.153-165.

V. Basili (1992). “The experimental paradigm in software engineering”. In D. Rombach, V. Basili and R. Selby (editors), *Experimental Software Engineering Issues: Critical Assessment and Future Directions*. LNCS, Vol 706. Spring-Verlag.

V. Basili and S. Green (1994). “Software process evolution at the SEL”. In *IEEE Software*, July 1994.

V. Basili and H. D. Rombach (1988). “The TAME project: towards improvement-oriented software environments”. In *IEEE Transactions on Software Engineering*, 14(6):758-773, June.

V. Basili; L. Briand; W. Melo (1995). “An Validation of Object-Oriented Design Metrics”. University of Maryland, Dep. of Computer Science, College Park, MD 20742. CS-TR-3443.

M. J. Bassman; F. McGarry; R. Pajerski (1994). “Software Measurement Guidebook.” Software Engineering Series, SEL-94-002.

N. Belkhatir and W. L. Melo (1994).” Evolving softwre processes by tailoring the behavior of software objects”. In *Proc. of the IEEE Int’l Conf. on S/W Maintenance*, Victoria, Canada.

L. Briand; V. Basili; Y.M. Kim; D. R. Squier (1994). "A change analysis process to characterize software maintenance projects". Proc. of the *Int'l Conf. on S/W Maintenance*. Victoria, Canada.

L. Briand; W. L Melo; C. Seaman; V. Basili (1995). "Characterizing and assessing a large-scale software maintenance organization". In *Proc. of the 17th Int'l Conf. on S/W Eng.*, Seattle, WA.

L. Briand, S. Morasca, V. Basili (1994). *"Property-Based Software Engineering Measurement*", CS-TR-3368, University of Maryland, College Park, MD, 20742, November 1994.

R. Chillarege, I. Bhandhari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M.Y. Wong. “Orthogonal defect classification - a concept for in-process measurement”. *IEEE Transactions on Software Engineering*, 18(11):943-956, November.

F. Coallier (1995). “TRILLIUM: A model for the assessment of telecom product development and support capability”. In *Software Process Newsletter*, No. 2, pp. 3-8, Winter.

J. Collofello and B. Gosalia (1993). “An application of causal analysis to the software production process”. In *Software Practice and Experience*, 23(10):1095-1105, October.

T. Cook and D. Campbell (1979). *Quasi-experimentation: Design and Analysis Issues for Field Settings*, Rand McNally, Chicago.

J-N Drouin (1995). “The SPICE project: An overview”. In *Software Process Newsletter*, No. 2, pp. 8-9, Winter.

K. El Emam and N. H. Madhavji (1995a). “A field study of requirements engineeirng practices in information systems development”. In *Proc. of the 2nd IEEE Int’l Syposium on Requirements Engineering*, pages 68-80.

K. El Emam and N. H. Madhavji. (1995b). “Measuring the success of requirements engineering processes”. In *Proc. of the 2nd IEEE Int’l Syposium on Requirements Engineering*, pages 204-211.

K. El Emam and N. H. Madhavji (1995c). “The reliability of measuring organizational maturity”. In *Software Process Improvement and Practice Journal*, 1(1).

K. El Emam and D. R. Goldenson (1995). “SPICE: An empiricist’s perspective”. In *Proceedings of the Second IEEE International Software Engineering Standards Syposium*, August.

N. Fenton (1993). “Objectives and context of measurement/experimentation”. In D. Rombach, V. Basili and R. Selby (editors), *Experimental Software Engineering Issues: Critical Assessment and Future Directions*. LNCS, Vol 706. Spring-Verlag.

N. Fenton; S.L. Pfleeger, R.L Glass (1994). “Science and substance: A challenge to software engineers”. IEEE Software, July, pp. 86-95.

A. Finkelstein; J. Kramer; B. Nuseibeh, eds. (1994). *Software Process Modeling and Technology*. Research Studies Press (distributed by Wiley & Sons).

P. Fowler and S. Rifkin (1990). “Software Engineering process Group Guide”. Technical Report CMU/SEI-90-TR-24, Software Engineering Institute.

D. Frailey (1991). “Defining a corporate-wide software process”. In *Proc. of the 1st Int'l Conf. on the Software Process*, pp. 113-121.

D. Goldenson and J. Herbsleb (1995). “What happens after the appraisal? A survey of process improvement efforts”. Paper presented at the *1995 SEPG Conf.*.

R. B. Grady (1992). *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, 1992.

V. Haase, R. Messnarz, G. Koch, H. Kugler, and P. Decrinis (1994). “Bootstrap: Fine tuning processs assessment”. In *IEEE Software*, pp. 25-35, July.

T. Hall and N. Fenton (1994). “Implementing software metrics – the critical success factors”. In *Software Quality Journal*, 3(4):195–208.

J. Henry and B. Blasewitz (1992). “Process definition: theory and reality”. In *IEEE Software*, pp.105, November.

A. Hersh (1993). “Where's the return on process improvement?”. In *IEEE Software*, page 12, July.

M. Kellner and G. Hansen (1989). “Software process modeling: a case study”. In *Proc. of the 22nd Annual Hawaii Int'l Conf. on System Sciences*, vol. II, pp. 175-188.

B. Kitchenham, S. Linkman, and D. Law (1994). “Critical review of quantitative assessment”. In *Software Engineering Journal*, 9(2):43-53, March.

B. Kitchenham; L. Pickard; S.L. Pfleeger (1995). “Case studies for method and tool evaluation”. IEEE Software, 12(4):52–62.

D. Kitson and S. Masters (1993). “An analysis of SEI software process assessment results: 1987-1991”. In *Proc. of the 15th Int'l Conf. on S/W Engineering*, pp. 68-77.

D. Leonard-Barton and W. Kraus (1985). “Implementing new technology”. In *Harvard Business Review*, pp. 102-110, November-December.

N. Madhavji, D. Hoeltje, W.K. Hong, and T. Bruckhaus (1994). “Elicit: a method for eliciting process models”. In *Proc. of the 3rd Int'l Conf. on the S/W Process*, pp.111-122.

W. Melo; L. Briand; V. Basili (1995). “Measuring the Impact of Software Reuse on Productivity and Quality in Object-Oriented Systems”. University of Maryland, Dep. of Computer Science, College Park, MD, 20742. CS-TR-3395. [To appear in the Communications of the ACM]

C. McGowan and S. Bohner (1993). “Model based process assessments”. In *Proc. of the Int'l Conf. on S/W Engineering*, pp. 202-211.

T. Nakajo and H. Kume (1991). “A case history analysis of software error cause-effect relationship”. In *IEEE Transactions on Software Engineering*, 17(8), August.

M. Paulk, B. Curtis, M-B Chrissis, and C. Weber (1993). “Capability Maturity Model, version 1.1”. In *IEEE Software*, pp. 18-27, July.

R. Radice, N. Roth, A. O’Hara, Jr., W. Ciarfella (1985). “A programming process architecture”. In *IBM Systems Journal*, 24(2):79-90.

R. Schaffer and H. Thomson (1992). “Successful change programs begin with results”. In *Harvard Business Review*, pp. 80-89, January-February.

Software Engineering Laboratory (1995). “*Software Process Improvement Guidebook*”. Software Engineering Laboratory Series, April, SEL-95-002.

R. Veryard (1987). “Implementing a methodology”. In *Information and Software Technology*, 29(9): 46-474, November.

C. Vogel (1988). “Génie cognitif”, Masson, Paris.

S. Waligora, J. Bailey, and M. Stark (1995). “*Impact of ADA and Object-Oriented Design in the Flight Dynamics Division at Goddard Space Flight Center*”. Software Engineering Laboratories Series, March, SEL-95-001.

E. Yu and J. Mylopolous (1994). “Understanding ‘why’ in software process modeling, analysis, and design”. In *Proc. of the 16th Int'l Conf. on Software Engineering*, Italy.