Software Process: A Roadmap

Alfonso Fuggetta
Politecnico di Milano
Dipartimento di Elettronica e Informazione
P.za Leonardo da Vinci, 32
20133 Milano (Italy)
Tel. +39-0223993623
Alfonso.Fuggetta@polimi.it

ABSTRACT
Software process research deals with the methods and technologies used to assess, support, and improve software development activities. The field has grown up during the 80s to address the increasing complexity and criticality of software development activities. This paper aims to briefly present the history and achievements of software process research, some critical evaluation of the results produced so far, and possible directions for future work.

1 INTRODUCTION

During the past 20 years, software has conquered an essential and critical role in our society. We increasingly depend on the features and services offered through computerized systems. Any modern product or service embeds and/or exploits some piece of software. As an example, companies sell (or plan to sell in the near future) systems to automate building operations and to embed Internet-features into home appliances.

Unfortunately, software applications are complex products that are difficult to develop and test. Very often, software exhibits unexpected and undesired behaviors that may even cause severe problems and damages. Every issue of the *ACM Software Engineering Notes*, a newsletter published by the ACM SIGSOFT interest group, contains a section that presents a comprehensive (and also frightening) report of the problems and accidents caused by software systems faults. For these reasons, researchers and practitioners have been paying increasing attention to understanding and improving the quality of the software being developed. This is accomplished through a number of approaches and techniques. One of the main directions pursued by researchers and practitioners is centered on the study and improvement of the process through which software is developed. The underlying assumption is that there is a direct correlation between the quality of the process and the quality of the developed software. The research area that deals with these issues is referred to using the term *software process*.

As an autonomous discipline, the software process area was started in the 80s, through a series of workshops and events (in particular, the *International Software Process Workshop*). Along the years, new events and journals on the subject have been started, such as the *European Workshop on Software Process Technology* and the *Software Process – Improvement and Practice* journal. Important institutions have been created in the USA and in Europe to study software processes: the *Software Engineering Institute* (SEI, Pittsburgh, USA) and the *European Software Institute* (ESI, Bilbao, Spain). Even standardization organizations have started important efforts centered on software processes. For example, ISO has created two important standards such as the ISO 12207 (software lifecycle activities) and 15504 (software process capability determination).

This paper aims to critically present and discuss the main results that the software process research area has achieved in the past decades. This is accomplished by evaluating both technological and methodological aspects. Indeed, there are other publications that offer a comprehensive overview of the results achieved so far in software process research (see for example [1], [2], and [3]). For this reason, the focus of this paper is on offering *a critical evaluation of the attitude and modes of the research work conducted so far*. Accordingly, the paper is structured in three sections.

- Section 2 presents a quick overview of the history and achievements of the software process research areas.
- Section 3 presents a critical evaluation of the work accomplished so far.
- Section 4 summarizes some possible directions for future work.
- Finally, Section 5 draws some conclusions.

2 A BRIEF OVERVIEW OF SOFTWARE PROCESS RESEARCH HISTORY AND ACHIEVEMENTS

The notion of process

The first important contribution of the software process research area has been the increasing awareness that *developing software is a* complex *process*. Researchers and practitioners have realized that developing software is not just a matter of creating effective programming languages and tools. Software development is a collective, complex, and creative effort. As such, the quality of a software product heavily depends on the people, organization, and procedures used to create and deliver it.

This vision has its roots in the work accomplished in the 60s and 70s. In those two decades, researchers and practitioners focused their activity on three main goals:

- Development of structured programming languages (e.g., Algol, Pascal, and C).
- Development of design methods and principles (e.g., information hiding, top-down refinement, functional decomposition).
- Definition of software lifecycles (e.g., waterfall, incremental development, prototype-based).

The third topic mentioned above (lifecycles) is directly related with the notion of software process. A software lifecycle defines the different stages in the lifetime of a software product. Typically, they are requirements analysis and specification, design, development, verification and validation, deployment, operation, maintenance, and retirement. Moreover, a software lifecycle defines the principles and guidelines according to which these different stages have to be carried out. For instance, the waterfall model suggests that a specific phase should be started only when the deliverables of the previous one have been completed. Conversely, the spiral model considers software development as the systematic iteration of a number of activities driven by risk analysis. In general, a software lifecycle defines the *skeleton* and *philosophy* according to which the software process has to be carried out. However, it does not prescribe a precise course of actions, an organization, tools and operating procedures, development policies and constraints. Thus a lifecycle is certainly an important starting point to define how software should be developed. Still, adopting a specific lifecycle is not enough to practically guide and control a software project.

The notion of software process builds on the notion of lifecycle and provides a broad and comprehensive concept to frame and organize the different factors and issues related to software development activities. *A software process can be defined as the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product*. Thus, a software process exploits a number of contributions and concepts:

1. *Software development technology: technological support used in the process*. Certainly, to accomplish software development activities we need tools, infrastructures, and environments. We need the proper technology that makes it possible and economically feasible to create the complex software products our society needs.

2. *Software development methods and techniques: guidelines on how to use technology and accomplish software development activities*. The methodological support is essential to exploit technology effectively.

3. *Organizational behavior: the science of organizations and people*. In general, software development is carried out by teams of people that have to be coordinated and managed within an effective organizational structure.

4. *Marketing and economy*. Software development is not a self-contained endeavor. As any other product, software must address real customers' needs in specific market settings. Thus different stages of software development (e.g., requirements specification and development/deployment) must be shaped in such a way to properly take into account the context where software is supposed to be sold and used.

Viewing software development as a process has significantly helped identify the different dimensions of software development and the problems that need to be addressed in order to establish effective practices. Indeed, addressing the problems and issues of software development is not just a matter of introducing some effective tool and environment. It is not sufficient to select a reasonable lifecycle strategy either. Rather, we must pay attention to the *complex interrelation of a number of organizational, cultural, technological, and economic factors*.

Process modeling and support

The emphasis placed on the notion of software process has motivated a number of research initiatives. A first area of investigation is related to the techniques and methods to model software processes and to support their execution (or enactment). Because software processes are complex entities, researchers have created a number of languages and modeling formalisms (often called Process Modeling Languages or PMLs) that make it possible to represent in a precise and comprehensive way a number of software process features and facets:

- Activities that have to be accomplished to achieve the process objectives (e.g., develop and test a module).

- Roles of the people in the process (e.g., software analyst and project manager).
- Structure and nature of the artifacts to be created and maintained (e.g., requirements specification documents, code modules, and test cases).
- Tools to be used (e.g., CASE tools and compilers).

There are many different types of PMLs. For a detailed discussion of the existing approaches, the reader is invited to refer to a number of surveys published in the past years ([1], [2], [4]). In general, existing PMLs are based on a number of linguistic paradigms that are extended in order to increase their expressive power. For instance, several approaches exploit Petri nets (SPADE, FUNSOFT nets), while others are centered on logical languages (Sentinel/Latin). Lee Osterweil has adopted a somewhat different approach with the notion of process programming. This approach is based on the idea that processes can be described using the same kind of languages that are exploited to create conventional software. This view has been initially pursued with the development of a language based on Ada (called APPL/A) and recently of a new language (called JIL) that incorporates constructs and concepts typical of different programming languages.

PMLs can be used for different purposes:

- Process understanding. A PML can be used to represent in a precise way how a process is structured and organized [5]. This can be instrumental to eliminate inconsistencies in the process specification (i.e., the company quality manual).
- Process design. Proactively, a PML can be used to design a new process, by describing its structure and organization.
- Training and education. A precise description of the process can be useful to teach company procedures and operations to newly hired personnel.
- Process simulation and optimization. A process description can be simulated to evaluate possible problems, bottlenecks, and opportunities for improvement.
- Process support. A precise description of the process can be interpreted and used to provide different levels of support to the people operating in the process [6].

An environment that supports the creation and exploitation of software process models is often called Process-centered Software Engineering Environment (PSEE).

Process improvement

As any other human-centered endeavor, software processes can exhibit unexpected or undesired performance and behaviors. The experiences of the past years have emphasized a wide range of situations where this phenomenon can be observed. Let's consider some typical examples:

- Delivered products do not exhibit the desired quality profile in terms of reliability, functionality, or performance.
- A specific sequencing of process operations introduces unnecessary delay and overhead that can be eliminated or at least reduced by allowing a redistribution of responsibilities and work assignments.
- It is difficult to keep track of the changes and variations of the software products generated by different members of the developed team.

The above situations are meant to be just examples and do not represent the entire range of problems faced by software engineers. In general, researchers and practitioners have realized that processes cannot be defined and "frozen" once for all. Processes need to continuously undergo changes and refinements to increase their ability to deal with the requirements and expectations of the market and of the company stakeholders. Hence, *process need to be continuously assessed and improved*.

These observations have motivated a range of projects devoted to the creation of *quality models* and *improvement methods* for *software process improvement*. A quality model (such as the SEI Capability Maturity Model – CMM – and the ISO 9001 standard [7]) defines the requirements of an ideal company, i.e., a reference model to be used in order to assess the state of a company and the degree of improvement achieved or to be achieved. An improvement method (i.e., SPICE and IDEAL) suggests the steps to be accomplished in order to improve the quality of a software process. Basically, improvement methods indicate how to carry out the "process of improving a process".

An important part of process improvement is *process assessment*, i.e., the determination of the degree of maturity of a process with respect to a quality model. Indeed, some of the most important contributions in process improvement have been originally started with the goal of creating assessment models and methods (e.g., CMM).

Metrics and empirical studies

The techniques and methods discussed in the previous two sections (process modeling and support, and process improvement) need to be based on reliable and effective practices. PMLs and support environments may certainly be useful, but we need to know how to structure and organize the process to be described and supported using PMLs/PSEEs. Similarly, to improve we need to identify the techniques and tools that are really instrumental to enhancing the performance of a specific process. Basically, we need answers to a number of questions such as the following ones:

- What are the indicators that can tell us something about the quality of a process?

- What techniques are more effective to improve a specific process?
- What is the cost and expected impact of a tool on the performance of software processes?

In general, researchers and practitioners have realized that there is an increasing need for a systematic evaluation of the quality of a process, of its constituents (tools, procedures, …), and of the resulting products. This evaluation is essential to support the implementation of improvement strategies and of any other decision-making activity related to software development. For this reason, in the past decade there has been a significant development of techniques and methods related to *software metrics and empirical studies*. In this context, there are three main kinds of contributions:

- *Definition of (new) metrics*. We need indicators that are able to quantify in a coherent and simple way the properties of the entities involved in software development [8]. For instance, how can we evaluate the size and complexity of a Java program? Or also, what is the productivity of a Java programmer?
- *Empirical methods*. Defining (new) metrics is not enough. We also need experimental approaches to guide the evaluation of a specific process [9]. In order to derive meaningful insights, we must be confident that the approach followed in studying the process is appropriate and sound.
- *Empirical results*. Metrics and empirical methods are the means that we use to study a phenomenon. Once defined, we apply these means to understand and assess specific problems and settings, in order to learn something on the nature of software development processes. These lessons learned (i.e., *empirical results* such as "technique X is not effective in context Y") increase our ability to successfully manage software development projects. The quality of empirical results has to be proved with respect to two different kinds of validity criteria [10]. We need to be sure that the study has been designed correctly (*internal validity*). Moreover, we need to understand if and under what circumstances the results of the study can be applied in different settings (*external validity*).

Processes, eventually!

The consolidated experiences of researchers and practitioners have been instrumental to define and consolidate successful processes. It is worthwhile to mention here two well-known examples: the Personal Software Process [11] and the Unified Software Development Process [12].

- The Personal Software Process (or PSP) is a collection of practices and techniques that are meant to guide the work of a software engineer. PSP has been defined by Watts Humphrey on the basis of his experiences and observations of real software development organizations.
- The Unified Software Development Process has been recently created by Jacobson, Booch, and Rambough. The Unified process is a set of guidelines and process steps that should be followed to apply UML in the different stages of software development.

Summing up

This section has provided a very quick and high-level overview of the activities that are often qualified as software process research. Clearly, the presentation is not intended to be exhaustive and technically complete. Rather, the goal was just to frame the different contributions and activities in order to give an overall picture of the work that has been carried out in the previous decade. Notice also that some of the topics that have been qualified as software process research, such as software metrics and empirical studies, can be considered autonomous research areas. Indeed, other papers in this volume discuss these topics. Still, I think it is important to mention here specific research activities on metrics and empirical studies whose subject of study is the software process.

As a general comment, it is possible to observe that there are a number of important achievements that have increased the quality and effectiveness of software development processes. Nowadays, we are able to conceive, create, and deploy software systems whose complexity is orders of magnitude larger than 15 years ago. Still, despite the large amount of results produced so far, software process research is undergoing a crisis that is visible through a number of symptoms:

- Most technologies developed by the software process community have not been transferred into industrial use.
- The number of papers on the software process modeling and technology presented at conferences and published in journals is decreasing.
- There is an increasing feeling that the community is stuck and unable to produce innovative and effective contributions.

This might be a pessimistic view. Still, as in any other area of software engineering [13], we need to rethink the way we are carrying out the research activity. This is instrumental to identify new directions and approaches to research. Consequently, the next section will present some considerations and observations on the work done so far and propose some criteria to guide future research activities.

3 CRITICAL ISSUES IN SOFTWARE PROCESS RESEARCH

The critical issues and problems in software process research can be summarized by four position statements (see also [14]).

Software processes are processes too

I took the liberty to rephrase one of the most successful and well-known mottos of the past decade. Created by Lee Osterweil for his invited talk at ICSE 87, the expression "Software processes are software too" has driven the work of many researchers and practitioners (including myself). The variation of Osterweil's motto proposed here is meant to be provocative and to stimulate a reflection on the attitude and approach of most software process research. We have often considered software processes as a "special" and "unique" form of processes. Consequently, we have basically assumed that it was inappropriate and even impossible to reuse the approaches and results produced by other communities (e.g., workflow and CSCW). Indeed, this attitude has caused a major problem. The software process community has redone some of the work accomplished by other communities, without taking advantage of the existing experiences. This insufficient willingness to analyze the results and contributions of other areas has slowed down the rate of innovation. Moreover, we have not taken the opportunity to learn from other researchers' mistakes. We should heavily invest in finding and evaluating commonalities and similarities [15], rather than identify differences that often appear to be quite artificial.

The purpose and nature of PMLs/PSEEs must be rethought

As discussed in Section 2, one of the key topics of software process research has been the development of PMLs and related PSEEs. As a matter of fact, after more than 10 years of research on the topic, few (if any) of the proposed approaches have been transferred into industrial practice.

If we consider process modeling, we realize that practitioners do not use the PMLs we have defined. Indeed, practitioners' most important need is to describe processes with the purpose of understanding and communicating them. Consequently, PMLs must be easy to use, intuitive, and "tolerant", i.e., their formality should not become a burden for the modeler. Conversely, existing PMLs are complex, extremely sophisticated, strongly oriented towards detailed modeling of processes. This is justified by the desire to be precise and to provide enough and coherent information to enable "process enactment", i.e., the execution and, often, the "automation" of the process. Moreover, this attitude is often exacerbated by the desire of most software process researchers to model "too much of a process", i.e., all the details concerning software development (e.g., steps and procedures of a design method). Unfortunately, this creates significant barriers to entry and, consequently, limits the possibility for PMLs to be adopted in practice.

The problems with existing PMLs are reflected into PSEEs. Very often, PSEEs are complex and intrusive. In order to pursue even simple operations such as editing and compiling a program, the initial effort needed to setup a PSEE is often very high. Moreover, the attitude towards modeling all the details of a process tends to make PSEEs rigid and inflexible. If we look at the market, we may observe that successful environments are characterized by a somewhat different philosophy. For instance, several researchers believe that Configuration Management (CM) environments (e.g., Continuus and CCC) are "the" real process-centered environments. Even if the effort needed to setup a CM environment is significant, the activities (i.e., process fragments) automated by this class of products are very complex and, at the same time, extremely boring and repetitive. For instance, managing the checkout of a software release can be automated, relieving software engineers from a lot of highly repetitive work, reducing the chances of mistakes, and shortening delivery time significantly. CM environments have become so important to software engineers that no large-scale development initiative can be launched without setting up an appropriate CM environment. *The motivation for this success is that CM environments automate in a very effective way only those process fragments that are reasonable to automate*. This should be considered an important lesson for software process researchers. Can we claim the same for existing PSEEs? Aren't we trying to model and automate something that intrinsically can't be modeled and automated? Isn't the failure of PSEEs due to an inappropriate and unrealistic definition of the goals?

Empirical studies are a means, not an end

In the past years, research in empirical software engineering has increased at a very fast pace. New events and journals have been created and the number of submissions on the subject to conferences is dramatically increasing. The motivation for this growing interest in empirical studies is the legitimate desire to increase our understanding of the principles and nature of software development. In other scientific domains, empirical scientists have made a substantial contribution to the development of our knowledge. Therefore, it is reasonable and appropriate to apply the methods and approaches of empirical sciences to software engineering.

The results of the empirical studies conducted so far, however, have produced mixed feelings. There are two major problems that several researchers have raised on the subject:

- *Significance*. The results of most empirical studies appear to be not very significant. Even if it is certainly true that most empirical studies have the purpose of providing a formal and credible foundation for practitioners and researchers' beliefs, often the added value of these experiments is limited. For instance, a paper that has been recently accepted for publication in a major journal spends about 50 pages of data and statistics to state that there is some evidence that the adoption of requirement engineering techniques is often positively correlated to improved software process

performance. What is the reader supposed to learn from this study? Shouldn't researchers' energy be directed at studying more promising questions?

- *External validity*. Many empirical studies carried out so far tend to be characterized by a very limited external validity. Namely, it is difficult to generalize the conclusions of the study outside the context where the study was carried out. There is an increasing sense of dismay in reading papers whose results are difficult to reuse. Certainly, an important added value of an experiment is its design and structure, since it can be reused in different contexts. But this does not remove the sensation that the reuse of these empirical results is limited and problematic.

In general, as any other scientific domain, we should keep in mind that *empirical studies are a means, not an end*. Thus, we should pay more attention to their significance and contribution, and not just to the quality of the experiment design or, worse, to the amount of statistical curve fitting. Moreover, we should not automatically disqualify as "non" scientific those efforts that are not based on statistical evidence and controlled experiments. In a landmark paper [16], Lee states that "… the natural science model does not involve, as objectives, the utilization of any of the following … : laboratory control, statistical controls, mathematical propositions, and replicable observations. Instead, each of these happens to be a means to an objective in scientific research rather the objective itself. *MIS case studies are capable of achieving the same scientific objectives through different means*". [1] Some of the most important contributions in computer science were not based on empirical studies (as we define them today) and statistical evidence. Did Parnas statistically verify that the adoption of information hiding is positively correlated to the quality of the developed software (and vice versa)? Certainly, Parnas made his assertion on the basis of a deep and mature experience. But the relevance of his intuition was illustrated by qualitative observations. As a provocation, I claim that by today's evaluation criteria, his work would probably not be considered scientifically valid. Nowadays, would we accept Parnas's paper "On the criteria to be used to decompose systems into modules" for publication in IEEE TSE or ACM TOSEM or ICSE?

[1] The executive overview of the paper reads as follows: "The classical research requirements cannot be met in a case study. Confounding variables typically make it exceedingly difficult to sort out causal relationships. The imposition of classical experimental controls and rigor, aimed at overcoming these problems, may require such an artificial environment that the validity of the results is called into question."

Software process improvement is process improvement too

I have once again "stolen" Osterweil's paper title to assert that software process improvement should take much more into account what other disciplines and researchers have discovered about process quality and process improvement. As in the case of software process technology, we often consider software processes as special and different from any other engineering and design process. Therefore, we derive that software process deserves specific improvement methods. Unfortunately, often these new and specific methods ignore or overlook the contributions of organizational scientists [17], [18]. Thus the risk is to reinvent the wheel and ignore important issues that may play a critical role in any improvement initiative. For instance, most of the indications suggested by the CMM focus on engineering aspects only. Unfortunately, the successful implementation of these indications often requires a deep reconsideration of the organization carrying out the development activity. This kind of implications is inadequately addressed by most software process improvement methods [19]. Certainly, software development is characterized by specific issues and problems. Still, we cannot forget that software development is carried out by teams of people involved in a highly creative activity. *It is, indeed, a human-centered process as many others engineering and design processes in our society*.

4 LOOKING FOR RESEARCH DIRECTIONS

Constructively, the observations and comments presented in this paper can be used to propose the following directions for future research:

- PMLs must be tolerant and allow for incomplete, informal, and partial specification. The goal should be to ease the adoption of PMLs. Practitioners should be able to incrementally build their process models, being informal and incomplete during the early stages of the modeling activity when it is impossible or inconvenient to be precise and exhaustive. If needed, the model should be incrementally enriched and made formal to address specific issues such as enactment and simulation.
- PSEEs must be non-intrusive, i.e., they should smoothly integrate and complement a "traditional" development environment. Moreover, it must be possible to deploy them incrementally so that the transition to the new technology is facilitated and risks are reduced.
- PSEEs must tolerate and manage inconsistencies and deviations. This requirement reflects the nature of a creative activity such as software development, where consistency is the exception and not the rule [20].
- PSEEs must provide the software engineer with a clear state of the software development process (from many different viewpoints).

- The scope of software improvement methods and models should be widened in order to consider all the different factors affecting software development activities. We should reuse the experiences gained in other business domain and in organizational behavior research.

5 CONCLUSIONS

Software development is a critical activity of our society, as we increasingly depend on software in most modern products and services. Therefore, software process research has an important role to play in the future of the software engineering research and practice. To face this challenge effectively, however, we as software process researchers should frankly and openly evaluate the errors and mistakes of the past, in order to avoid them in the future and to increase the effectiveness of the solutions we are going to propose. In this paper, I have presented four propositions that summarize some of the concerns raised in the community in the past years. In general, software process researchers and practitioners should reuse the experiences and achievements of other areas and disciplines. Moreover, we should rethink the approach we have adopted in studying and supporting software processes. These observations might appear as quite obvious and even trivial. Still, I do believe that they are the underlying motivations for the partial lack of results we observe in the discipline.

ACKNOWLEDGEMENTS

The author wishes to thank Anthony Finkelstein, Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli, David Rosenblum, and Alex Wolf for their comments and suggestions.

REFERENCES

[1] V. Ambriola, R. Conradi, and A. Fuggetta, "Assessing process-centered software engineering environments," *ACM Transactions on Software Engineering and Methodology*, vol. 6, 1997.

[2] G. Cugola and C. Ghezzi, "Software processes: a retrospective and a path to the future," *Software process - Improvement and practice*, vol. 4, pp. 101-123, 1998.

[3] A. Fuggetta and A. Wolf, "Trends in Software Processes," in *Trends in Software*, B. Khrisnamurthy, Ed.: John Wiley, 1995.

[4] P. Garg and M. Jazayeri, "Process-centered Software Engineering Environments," : IEEE Computer Society Press, 1996.

[5] S. Bandinelli, A. Fuggetta, L. Lavazza, M. Loi, and G. P. Picco, "Modeling and improving an industrial software process," *IEEE Transactions on Software Engineering*, 1995.

[6] S. Bandinelli, E. Di Nitto, and A. Fuggetta, "Supporting cooperation in the SPADE-1 Environment," *IEEE Transactions on Software Engineering*, vol. 22, 1996.

[7] M. O. Tingey, *Comparing ISO 9000, Malcolm Baldrige, and the SEI CMM for Software*: Prentice Hall, 1997.

[8] N. Fenton, "Software measurement: a necessary scientific basis," *IEEE Transactions on Software Engineering*, vol. 20, pp. 199-206, 1994.

[9] C. M. Judd, E. R. Smith, and L. H. Kidder, *Research methods in social relations*, Sixth ed. Fort Worth, TX (USA): Holt, Rinehart and Winston, Inc., 1991.

[10] L. Votta, A. Porter, and D. Perry, "Experimental Software Engineering: a report on the state of the art," presented at 17th International Conference on Software Engineering (ICSE 17), Seattle (WA), 1995.

[11] W. S. Humphrey, *A discipline for Software Engineering*: Addison-Wesley Publishing Company, 1995.

[12] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Reading, Massachusetts 01867: Addison Wesley Longman, Inc., 1999.

[13] A. Fuggetta, "Rethinking the modes of software engineering research," *The Journal of Systems and Software*, vol. 47, pp. 133-138, 1999.

[14] R. Conradi, A. Fuggetta, and M. L. Jaccheri, "Six theses on software process research," presented at 6th European Workshop on Software Process Technology (EWSPT '98), Weybridge (UK), 1998.

[15] G. A. Bolcer and R. N. Taylor, "Advanced workflow management technologies," *Software process - Improvement and practice*, vol. 4, pp. 125-171, 1998.

[16] A. S. Lee, "A scientific methodology for MIS case studies," *MIS Quartely*, vol. 13, pp. 33-50, 1989.

[17] F. Cattaneo, A. Fuggetta, and L. Lavazza, "An experience in process assessment," presented at ICSE 17 - 17th International Conference on Software Engineering, Seattle (USA), 1995.

[18] P. Carlson, "Information technology and organizational change," presented at Seventeenth annual International Conference on Computer documentation, New Orleans (LA), 1999.

[19] F. Cattaneo, A. Fuggetta, and D. Sciuto, "Pursuing coherence in software process assessment and improvement," CEFRIEL, Milano, Technical Report September 1998.

[20] B. Balzer, “Tolerating inconsistencies,” presented at International Conference on Software Engineering (ICSE 13), Austin (TX), 1991.