Improving Software Process Improvement

Reidar Conradi, *Norwegian University of Science and Technology*

Alfonso Fuggetta, *Politecnico di Milano, Italy*

The past years have seen an increasing interest in methods, models, and techniques (so-called frameworks) to support the assessment and improvement of software development processes. Here, software process improvement encompasses process assessment, process refinement (traditional SPI), and process innovation (introducing major process changes). National and international bodies have proposed and supported frameworks such as the Quality Improvement Paradigm, ISO 9000, the Capability Maturity Model,[1,2] Spice (Software Process Improvement and Capability Determination, also known as ISO/IEC 15504), and Bootstrap to promote mature software development practices. The CMM has been supplemented with the Ideal improvement model, the Personal Software Process, and Team CMM. Most of these frameworks have become well known among practitioners and researchers. In the US, more than 1,000 CMM-based assessments have been carried out. From 1994 to 2000, the Commission of the European Union funded more than 450 so-called process improvement experiments (PIEs) through its European Systems and Software Initiative (ESSI).

Two dichotomies characterize software process improvement efforts and approaches: disciplined vs. creative work and procurer risks vs. user satisfaction. Based on these perspectives, the authors introduce six theses to illuminate the problems of pursuing SPI.

Despite the amount of resources spent on these improvement efforts, several critical issues are still open, as the many ongoing initiatives, projects, and standardization efforts testify. In particular, evaluating whether an assessment and consequent improvement plan have had a causal, significant, and positive impact on company success is difficult. Certainly, some studies show that process assessment techniques benefit software development organizations.[3] In reality, these positive effects are not so evident. Some researchers claim that most improvement initiatives have had limited success.[4] Further, companies such as Microsoft have developed successful commercial software without adhering to external process standards or pursuing ambitious improvement initiatives.

The problems go deeper than just needing to downsize and tailor SPI frameworks. As the Spice and CMM experience has shown, rather than just repair and adjust the process (model), we should also question some underlying assumptions and refocus SPI to be more company- and user-oriented. Indeed, one of the basic SPI-QA (quality assurance) assumptions is that "the quality of a software product is largely governed by the qual-

0740-7459/02/$17.00 © 2002 IEEE

Authorized licensed use limited to: Chalmers University of Technology. Downloaded on February 8, 2010 at 08:12 from IEEE Xplore. Restrictions apply.

ity of the process used to develop and maintain it."[2] The CMM also states that "if a process is not under statistical control, sustained progress is not possible until it is."[1] However, software work, like other design work, is not like mechanized or disciplined manufacture. It has a strong creative component involving human and social interaction that cannot be totally preplanned in a standardized and detailed process model.

SPI should ultimately lead to a learning- and user-oriented organization, as emphasized by the "mother" of all SPI-QA frameworks, Total Quality Management (TQM).[5] A good business process is simply one that leads to satisfied users—the only ones who can define product quality. Such a process is not necessarily one with high CMM or ISO 9000 ratings, though software procurers who think they represent potential users often emphasize these ratings. Just take a look at today's cellular phone providers to observe the disparity between business success and formal CMM levels. (For example, Motorola is partly at CMM Level 5, Ericsson partly at Level 3, and Nokia at Level 1.)

The SPI community knows most of the arguments on all sides of this issue. In spite of this, substantial parts of SPI frameworks and efforts have been biased toward discipline rather than creativity in their internal focus and toward procurers rather than users in their external focus. These two pairs of opposing biases, or dichotomies, cover the range of attitudes; the challenge is to reconcile them.

The value of SPI debated

In a recent report on the Spice trials,[6] we found some disturbing observations:

> *Approximately three-fifths do not believe that the assessment has had a major impact on the organization. This may be due to there not being sufficient time since the assessment for SPI to have taken root or due to the organizations not being able to act on the recommendations and findings from the assessment.…This is further reinforced by the finding that only 28 percent disagree with the statement that nothing much has changed since the assessment. It is interesting to note that 79 percent believe that SPI was overcome by events and crises, indicating potentially that the organizations were not in a state that is ready for long-term SPI initiatives (i.e. there were other more important things to attend to).*

These comments are serious and raise important points that go well beyond accidental management or priority problems. Furthermore, as James D. Herbsleb and Dennis R. Goldenson document,[3] quite a few people working with the CMM describe SPI as having no or very little effect (26 percent) or even being counterproductive (4 percent). Forty-nine percent said they were disillusioned by the lack of results. Other researchers have written about the problems of achieving SPI in small organizations.[7]

On the other hand, there is a growing body of success stories, where CMM-based SPI gets credit for significant productivity gains and reliability improvements.[8] There are also observations around Cocomo II[9] indicating that each increased CMM level (aside from other effects) means a productivity gain of 4 to 11 percent. However, in most of this work, causally relating the claimed gains to a chosen improvement method's application is difficult. That is, would any systematic SPI-QA (quality assurance) initiative have yielded similar results cost-efficiently? Is there a reference group? Likewise, could the application of just a few novel core techniques—estimation, inspections, or reuse, for example—have led to a similar gain or major share of the gain?

We have performed several European studies in small- and medium-sized enterprises (SMEs), both in Scandinavia[10] and Italy.[11] These studies conclude that short-term priorities, combined with business and market turbulence, may severely prevent, hamper, and even abort well-planned, low-key SPI efforts. Further, many of the cited SPI frameworks are hardly suited or applicable in their present forms for SMEs. Consequently, the Norwegian Software Process Improvement for Better Quality project, known as SPIQ, has defined a downscaled and pragmatic subset of existing SPI frameworks aimed at busy and often small IT companies.[12]

Internal SPI focus: Discipline versus creativity

The first perspective, discipline, refers to introduction of and adherence to more structured work processes, usually as nor-

A good business process is simply one that leads to satisfied users—the only ones who can define product quality.

Authorized licensed use limited to: Chalmers University of Technology. Downloaded on February 8, 2010 at 08:12 from IEEE Xplore. Restrictions apply.

Software development has three main roles in a value chain: software developer, procurer, and user. The role of the customer, who is paying for the actual software, may be either of the latter.

mative process models and plans. This is in line with most QA efforts and usually involves setting up procedures, rules, and guidelines and possibly certifying or standardizing many of these processes. There are more than 250 proposed software standards, many of them recommending standard process models—but how many of these are in practical use?

The other perspective, creativity, emphasizes that software development relies on a collaborative design process known as participatory development. This applies to both the project team and the future user, where even requirements are subject to continuous clarification and (re)negotiation. Although there is no excuse for not trying to make an initial set of clear, fixed requirements, the reality of software development is not well-defined and stable.

Many of the ideas and techniques on quality improvement (TQM and others) come from manufacturing, which involves stable products, processes, and organizations. Information technology, on the other hand, is characterized by rapid product innovation, not process refinement. Under such circumstances, Stan Rifkin claims, we have offered the "wrong medicine" to small software companies, which have consequently failed their SPI programs.[4] An English software consultancy company, Oxford Software Engineering, is advocating what they call rapid process improvement. In Internet time, one year is like a "dog year" (seven years) in other disciplines, and time-to-market seems sacred. The strength of many small software companies lies in fast turnaround and their ability to convert next week's technologies into radically new products and services. According to IBM Software manager Daniel Sabbah in a conversation at ICSE 2001, even his large organization, with 4,500 software developers working on middleware, spends 30 percent of its resources on technology-driven process changes.

We must therefore adopt a set of quality and improvement technologies that can operate under constant change.[13] For instance, we should be careful in overly reducing process variance (as implied by CMM Level 4), because this constrains future technological opportunities. That is, we run the risk of refining an outdated process. On the other hand, we should not let software work degenerate into an unstructured happening, discarding discipline and formalization altogether. We need to balance the "odd couple" of discipline and creativity.[14] Appropriate perspectives should be applied to different parts of the software process—for example, discipline and rigor to inspections and configuration management, and creativity and collaboration to requirements engineering and high-level software design.

External SPI focus: Procurer risks versus user satisfaction

Software development has three main roles in a value chain: software developer, procurer, and user. The role of the customer, who is paying for the actual software, may be either of the latter.

A software procurer is in charge of acquiring software from a software developer (provider or contractor) according to stated and often fixed requirements. It might be a mass-produced article but is usually a tailor-made system that's part of a new information system. Providers are frequently picked after a round of bidding, evaluation, and contract negotiation, which covers product quality (functionality, reliability, and so on), price, delivery scheduling, and possibly future maintenance. Naturally, the aim is to reduce procurer risks—how can the user be confident that the provider can deliver the promised software on time and budget? Here, certification, a well-proven technique in other engineering and manufacturing disciplines, comes into play.

The Software Engineering Institute's work on the CMM has identified five certification levels of software maturity. The SEI asserts that these levels and their key process areas capture relevant company processes that, in combination, will lead to desired product quality as well as budget and schedule compliance from a procurer's viewpoint. The CMM and similar frameworks are thus meant to evaluate (assess) engineering and methodological aspects of software processes, such as project management, requirement engineering, configuration management, and so on. However, they do not consider the success factors of the contracting company (software developer), such as profitability, competitiveness, market strategy, and user satisfaction. Indeed, these frameworks hardly touch on the contractor's success; their focus is work disci-

Authorized licensed use limited to: Chalmers University of Technology. Downloaded on February 8, 2010 at 08:12 from IEEE Xplore. Restrictions apply.

pline and risk reduction.

A user (or end user) is one who interacts with the delivered software, directly or indirectly. The user, being a person or an organization, implicitly defines product quality—whether the functionality, price, and delivery time of a software piece (and what comes with it) satisfy the user's evolving needs.

Poor product quality can lead to disastrous errors (for example, unsafe, unreliable, or insecure systems) and "dead-upon-delivery" incidents of misconceived information systems. However, time-to-market is becoming more important, both for software companies and other organizations that rely on software. Internet time has even reached the large telecom companies, and Ericsson has stated its priorities: "faster, better, cheaper"—in that order. An end user is not interested in the procurement or development of a piece of software but in how the software helps create a high-quality product. The delivery of such products should therefore be the software provider's and developers' dominating concern, as TQM clearly states. Given the risks of developing information systems with unclear or evolving requirements, many organizations have turned to more incremental methods with more interaction between users and developers. Methods such as the Dynamic Systems Development Method and the Rational Unified Process are rapidly taking hold. Organizations consider the static view of formal product procurement harmful and are rapidly abandoning it. Instead, they are forming strategic partnerships and signing development contracts without always agreeing on a final set of requirements. As another example, the Norwegian Computer Society (www.dnd.no) offered a course in spring 2001 on chaos and complexity theory to help its members prepare for modern IT system development.

The essential problems of SPI initiatives: Six theses

We believe that the essential problems undermining most companies' SPI initiatives come from using these two dichotomies to interpret SPI. Frameworks such as CMM, which is inherently discipline- and procurer-oriented, have greatly influenced such initiatives. CMM assessments and corresponding SPI initiatives have not consistently correlated with company success, as the examples of Nokia, Motorola, and Ericsson demonstrate. The results of recent empirical studies are mixed; they show that "improving" (or structuring) the process does not automatically improve the company's competitiveness. Nevertheless, a software procurer might force a developer to initiate and carry out SPI or a quality certification (ISO-9000, for example).

The two identified foci and their perspectives are not intrinsically in conflict. They all identify legitimate and useful visions for assessing and improving a company. We should decide which SPI approach or framework is best for which context. For instance, the CMM might work well for stable companies. The real problem is managing goals, expectations, and viewpoints. A software procurer's goal is to select the best contractor objectively and rationally. A software company's goal is to survive and grow in a competitive market. An end user's goal is to acquire the software product that can solve the right problem, at the right time, at an acceptable price. We cannot expect the same SPI approach and consequent effort to accommodate all these viewpoints.

These observations suggest a strong need to revise the approach and assumptions underlying existing SPI frameworks and related company initiatives. Over the past five years, we have been involved in several SPI initiatives. Based on this experience, we present six theses on how to enhance and better apply SPI frameworks.

These observations suggest a strong need to revise the approach and assumptions underlying existing SPI frameworks and related company initiatives.

Thesis 1

> ***SPI frameworks should support improvement strategies that focus on goal orientation and product innovation.***

Most SPI frameworks implicitly require a high stability level. The CMM, for instance, assumes that applying several significant (and often expensive and time-consuming) changes causes improvement. These changes will progressively move the company from a low maturity level to a higher one. However, most companies cannot base their improvement strategies on such a long-term effort and do not have a well-established baseline for comparison. The environment is fast-paced: time-to-market is crucial, and companies are continuously created, merged, and destroyed. Most of them would never consider this kind of SPI as a priority. For these companies, the real issue is product innovation, not process stabilization and refine-

Authorized licensed use limited to: Chalmers University of Technology. Downloaded on February 8, 2010 at 08:12 from IEEE Xplore. Restrictions apply.

Developers are the ones most likely to see what is possibly wrong. Many frameworks emphasize that we should first seek top-management commitment.

ment. Processes (if any) must "follow the stream" and let the company effectively pursue product innovation. If a CEO's problem is to deliver a product within six months, he will not seriously consider an SPI initiative that won't be finished for at least one year. That is, we need specific goal orientation—not general symptom orientation.

In a study of Norwegian and Swedish PIEs supported by the EU's ESSI program, all the failed PIEs (10 of 29) were due to personal or organizational instabilities[10] In the SPIQ project,[12] a collaborative Norwegian SPI project, only 2 out of 15 companies completely avoided such instabilities from 1997 to 1999, and 6 of 15 companies aborted their improvement efforts during either planning, startup, or implementation.

Generally, SPI frameworks should identify an action list that offers sufficient local leverage and a time line consistent with the company's market dynamics. This requires heavily extending our assessment methods and considering a range of currently excluded dimensions (for example, market and economic variables). If the company context is suited for applying the CMM, that is fine, but one size does not fit all. The remedy is to

- Make the SPI initiative consistent with business goals and strategy to ensure that it will have the desired effects on company performance.
- Start with improvement activities that deal with the most pressing needs.
- Define realistic and visible short- and long-term goals.

Thesis 2

Developers are motivated for change; if possible, start bottom-up with concrete initiatives.

Developers are the ones most likely to see what is possibly wrong. Many frameworks emphasize that we should first seek top-management commitment. Our experience shows that mid-level management resistance can also be a problem, because these managers are the main players in improvement initiatives. The CMM, which has a top-down orientation, also acknowledges the problem with middle managers.

As part of the SPIQ project, we interviewed software managers and developers in five SMEs about their attitudes toward improvement and quality work and about how improvement activities should be prioritized (for example, using a scorecard). We found that developers indeed are motivated for change, and often there was broad consensus when identifying the most critical or possibly most cost-effective topics to address. By starting with pressing needs such as estimation, inspections, or incremental development, improvement is coupled to concrete goals and can be effectively monitored, analyzed, and evaluated. In the SPIQ project, we have worked on more than 20 concrete improvement initiatives (ranging from six to 12 months down to two to three months) and cooperated closely with developers and quality and project managers. Top-level managers regularly receive reports and results from this work. This secures ongoing support, paves the way for more long-term improvement initiatives (two to four years), and encourages experience reuse and organizational learning. Showing concrete results to those whose processes "need" improvement promotes situated learning. Improvement and learning cannot be forced from the outside. They must become an ingrained part of the work process. This is also consistent with how knowledge workers perform their jobs. Such workers easily learn to work around and even sabotage projects involving ineffective processes and fruitless directives.

So, our remedies for this thesis are to

- Use a simple and focused scorecard to start with, not a large assessment.
- Rely on and enlarge developer participation through situated learning.
- Establish feedback lines inside and between improvement initiatives and make them visible to top-level management.

Thesis 3

Automated software process support has been overemphasized.

Over the last 12 years, there has been a massive research and development effort to support the software process using computerized tools (so-called software process technology). SPT involves modeling lan-

Authorized licensed use limited to: Chalmers University of Technology. Downloaded on February 8, 2010 at 08:12 from IEEE Xplore. Restrictions apply.

guages, editors, interpreters, and possibly experience bases—sometimes combined into a process-sensitive software engineering environment (PSEE). However, because software processes are largely dynamic and cooperative activities, the impact of such technologies has been meager.

The problems and issues that SPT and workflow–groupware management address are the same. Indeed, we can largely reuse the experiences and results from workflow and groupware research for software processes. Further, configuration management (CM) tools are the real PSEEs. CM is the underlying process controlling a large portion of any software development activity. The fact that CM processes are amenable to automation explains why CM tools have been so successful on the market.

As mentioned, we have so far not substantially demonstrated that SPT is useful. SPT environments and PSEEs are not used in real software processes, as the significant lack of commercial vendors offering this kind of product testifies. SPI should apply results from organizational and behavioral sciences that emphasize the creativity and cooperation in software development.

These observations apply not only to software activities, but to a large set of creative design processes. For example, ongoing empirical work in computer-supported cooperative work is exploring the use of computer-assisted tools to support unstructured processes. In general, we must reconsider the efforts and research in process technology and relate them to other disciplines that have intensively studied cooperative and creative processes.

So, to remedy thesis 3, we propose that

- Only stable processes—for inspection, testing, or configuration management, for example—are well suited for SPT support, especially with computer-assisted enactment.
- An SPT tool must adapt to the specific needs of the application; building an advanced tool for the wrong application is technological overkill.

Thesis 4

Cost–benefit analysis requires novel amortization models.

The proof in the pudding is whether SPI results in a positive return-on-investment (ROI)—that is, whether it contributes to increased competitiveness and value creation. As we mentioned in Thesis 1, many companies want a six- to 12-month time line for improvement efforts. A recent Norwegian survey indicates that software companies' average project length is three to six months, with increments of three to six weeks inside projects.[15] On the other hand, it takes four to five years to move from CMM Level 1 to Level 3. Admittedly, SEI never claimed to have designed the CMM to increase productivity, although there are many CMM-related and other SPI papers on ROI. Regrettably, they are often based on uncertain data.

Because many SPI frameworks (CMM, Spice, ISO 9000) advocate clustering many different techniques as part of a certification procedure, it becomes difficult to show a causal relationship between future benefit and any actual applied techniques. Moreover, methods for incremental or component-based development assume that many techniques are combined in new and complex ways that may not fit the grouping in the SPI frameworks. A general problem with all software improvement initiatives (reuse programs, metrics programs, or SPI) is that they spend resources now, while—as with any other investment—the gain lies in the future. We therefore need a standardized market-value model, such as the one employed in Cocomo II,[9] to account for software improvement initiatives. On the other hand, many companies have no uniform, internal interest rate even for physical investments, such as new buildings, office furniture, or computer equipment. In spite of this, they often require SPI efforts to show a positive ROI in one year or less. Maybe the mistrust in yet another software technology is rooted in the upper-management sphere?

Remedies for this problem are to

- Develop incremental and novel cost–benefit models (such as Cocomo II).
- Strive for a common company-internal cost–benefit model.
- Execute one SPI effort (experiment) at a time, possibly against baseline projects.
- Tell management that reproducible and convincing results might take some time.

SPT environments and PSEEs are not used in real software processes, as the significant lack of commercial vendors offering this kind of product testifies.

Authorized licensed use limited to: Chalmers University of Technology. Downloaded on February 8, 2010 at 08:12 from IEEE Xplore. Restrictions apply.

Often, good quality is associated with zero defects—high reliability or guaranteed safety, for example.

Thesis 5

SPI assumes cultural changes, so we need expertise from the social sciences.

The engineering profession, including software engineering, combines technical and social aspects. The same must apply for SPI to properly initiate, carry out, study, and learn from improvement efforts. Thus, we should draw on the social sciences (to conduct proper case studies, for example), social anthropology (to study how people really work), organizational sciences (for organizational learning), and so on. This applies to both software managers and researchers.

Software developers are knowledge workers. They tend to respond negatively to top-level dictates on how to do work or change processes. Organizations must integrate new tools such as PSEEs or experience bases into normal work practices, otherwise such tools are doomed to fail. A focus on continuous improvement and quality requires participative attention by all employees and cannot be delegated to internal visionaries, quality managers, or process champions. Managing SPI and quality is not something that an organization can do, for example, for one year and then put aside for the next three. Management books taught by management "gurus" worldwide admittedly emphasize most of this. However, the heavy time pressure in innovative fields like IT makes this a demanding task.

This brings up the fundamental method problem of how to influence and study social organizations. The SPI advocate or researcher will often serve as an "influence agent" in a moving-target organization and work closely with the persons under study. These SPI initiatives must

- Use multidisciplinary teams—for example, combined with action research.
- Establish participative engagement on all process changes and associated activities.
- Emphasize the cultural, learning, and long-term dimensions of SPI work.
- Start humbly with small steps.

Thesis 6

SPI is about learning—not control, as in QA.

There are many textbooks on learning organizations and knowledge management. Long-term quality work is not about punishing those who commit errors, but about systematizing and learning from past experiences and users' needs and reactions collectively and without ego. The slogan of the NASA Software Engineering Laboratory in the 1980s and '90s was that measurement and learning from experience were "part of our way of doing business." However, "smoke-screening" or "scape-goating" will typically prevent systematic learning from failed projects in organizations, both internally and externally. According to recent reports from other parts of NASA, in autumn 1999, an engineer could be fired for reporting a problem, whereas in the pioneering 1960s it was a "problem" not to report a problem. The medical profession has similar reporting inhibitors.

Often, good quality is associated with zero defects—high reliability or guaranteed safety, for example. Many software-intensive companies, therefore, have a separate QA department or quality manager who maintains the quality system (formal routines) for the company. However, developers might not embrace these formal routines (see Thesis 5), which tend to collect dust on the shelves. Web versions of such standards might not fare much better. In a Norwegian survey of 23 software developers and managers, developers saw formal routines as ineffective in transferring knowledge; the routines were often prepared and introduced without their participation.[16] AT&T studies on how workers actually perform processes also report a lack of process conformance.[17] All this corroborates the observation that employees will not necessarily turn formalized and externalized experience into better work practices. The same observation has been made regarding promoting reuse through common reuse repositories.[18]

The remedies for this problem are to

- Create an SPI team separate from the QA department (as in the CMM's software engineering process groups), or, better yet, combine SPI and QA in one team.
- Perform empirical studies on how people actually work.
- Make a reward system for reporting problems or suggesting improvements.

Authorized licensed use limited to: Chalmers University of Technology. Downloaded on February 8, 2010 at 08:12 from IEEE Xplore. Restrictions apply.

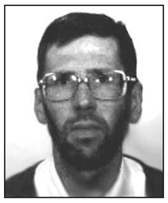
Improving the way we develop software or software processes is a key challenge for society. Software is the core of any modern service or product. Therefore, ensuring its quality is vital. As a consequence, SPI is one of the most critical and important efforts that any software company pursues. Nevertheless, many SPI initiatives have not proven successful—that is, significantly and directly related to relevant improvements in company performance and product quality. Software development companies' lack of commitment and investment might cause this. Our opinion is that existing SPI frameworks are unable to address the critical challenges of a software development company, causing this lack of interest, commitment, and goal-oriented success.

The conclusions we propose might appear obvious to some readers. Still, it is surprising that despite the increasing awareness that SPI cannot only be based on CMM-like assessments, the number of initiatives and projects aimed at innovating SPI methods is limited. Similarly, there are few documented attempts to apply new or different kinds of assessment methods. Indeed, given the reported problems with existing SPI frameworks and company SPI initiatives, it is time to redirect the course of action to make SPI methods and technologies more effective and widely used. This means being more flexible in choosing and applying SPI frameworks, learning from organizational scientists, and, most importantly, listening to a company's customers and aligning with a company's technology and business context. The discussion offered here is an initial attempt to provide direction on how to address these challenging issues.

References

1. W.S. Humphrey, *Managing the Software Process,* SEI Series in Software Eng., Addison-Wesley, Reading, Mass., 1989.
2. M.C. Paulk et al., *The Capability Maturity Model for Software: Guidelines for Improving the Software Process,* SEI Series in Software Eng., Addison-Wesley, Reading, Mass., 1995.
3. J.D. Herbsleb and D.R. Goldenson, "A Systematic Survey of CMM Experience and Results," *Proc. 18th Int'l Conf. Software Eng.* (ICSE 96), IEEE CS Press, Los Alamitos, Calif., 1996, pp. 323–330. See also tech. report SEI-94-TR-13, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, Pa., 1994.
4. S. Rifkin, "Is Process Improvement Irrelevant to Produce New Era Software?" *Proc. Software Quality: Quality Connection* (ECSQ 02), Springer-Verlag, Heidelberg, Germany, LNCS 2349, 2002, pp. 13–16.
5. W.E. Deming, *Out of the Crisis,* MIT Center for Advanced Eng. Study, MIT Press, Cambridge, Mass., 1986.
6. Spice, Phase 2 Trials Interim Report, version 1.00, 17 June 1998, p. 159; www.sqi.gu.edu.au/spice/trials/p2summ.html.
7. R.P. Ward, M.E. Fayad, and M. Laitinen, "Thinking Objectively: Software Improvement in the Small," *Comm. ACM,* vol. 44, no. 4, Apr. 2001, pp. 105–107.
8. B. Curtis, "The Global Pursuit of Process Maturity," *IEEE Software*, vol. 17, no. 4, July/Aug. 2000, pp. 76–78.
9. B.W. Boehm et al., *Software Cost Estimation with Cocomo II* (with CD-ROM), Prentice Hall, Upper Saddle River, N.J., 2000.
10. T. Stålhane and K.J. Wedde, "SPI—Why Isn't It More Used?" *Proc. EuroSPI '99,* Pori School of Technology and Economics, Pori, Finland, Serie A25, pp. 1.34–1.39.
11. F. Cattaneo, A. Fuggetta, and D. Sciuto, "Pursuing Coherence in Software Process Assessment and Improvement," *Software Process: Improvement and Practice,* vol. 6, no. 1, Mar. 2001, pp. 3–22.
12. T. Dybå et al., *SPIQ Metodebok for Prosessforbedring [SPIQ Method Book for Software Process Improvement]*, V3, ISSN 0802-6394 Univ. of Oslo, SINTEF, and Norwegian Univ. of Science and Technology, 2000 (in Norwegian); www.geomatikk.no/spiq.
13. T. Dybå, "Improvisation in Small Software Organizations: Implications for Software Process Improvement," *IEEE Software,* vol. 17, no. 5, Sept./Oct. 2000, pp. 82–87.
14. R.L. Glass, *Software Creativity,* Prentice Hall, Upper Saddle River, N.J., 1995.
15. S.F. Andersen and P.C. Nødtvedt, *A Survey of Incremental Development in Norwegian IT Industry,* prediploma thesis, Norwegian Univ. of Science and Technology, Trondheim, Norway, 2001.
16. R. Conradi and T. Dybå, "An Empirical Study on the Utility of Formal Routines to Transfer Knowledge and Experience," *Proc. European Software Eng. Conf. 2001* (ESEC 2001), ACM Press, New York, pp. 268–276.
17. D.E. Perry, N. Staudenmayer, and L.G. Votta, "People, Organizations, and Process Improvement," *IEEE Software,* vol. 11, no. 4, July/Aug. 1994, pp. 36–45.
18. J.S. Poulin, "Populating Software Repositories: Incentives and Domain-Specific Software," *J. Systems and Software,* vol. 30, no. 3, Sept. 1995, pp. 187–199.

About the Authors



Reidar Conradi is a professor in the Department of Computer and Information Science at the Norwegian University of Science and Technology in Trondheim (NTNU, previously called NTH). He was a visiting scientist at Fraunhofer Center for Experimental Software Engineering in Maryland and at Politecnico di Milano in 1999–2000, where he did the work for this article. His interests are process modeling, software process improvement, software engineering databases, versioning, object orientation, component-based development, and programming languages. He received his MS and PhD in informatics from NTNU. Contact him at conradi@idi.ntnu.no.

Alfonso Fuggetta is a professor in the Department of Electronics and Informatics at Politecnico di Milano, Italy, and scientific vice-director of the affiliated CEFRIEL research institute (Education and Research Center for Information Technology). He is also a faculty associate of the Institute for Software Research at the University of California, Irvine. His interests are software process improvement, software architecture, mobile software, Web-based and event-based systems, and applications of such technology. He graduated from the Politecnico di Milano with a Laurea degree (MS) in electronic engineering. Contact him at Alfonso.Fuggetta@polimi.it.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.

Authorized licensed use limited to: Chalmers University of Technology. Downloaded on February 8, 2010 at 08:12 from IEEE Xplore. Restrictions apply.