An empirical study on the utility of formal routines to transfer knowledge and experience

Reidar Conradi
Norwegian Univ. of Science and Technology (NTNU)
NO-7491 Trondheim, Norway
Phone: +47 7359 3444, Fax: +47 7359 4466
conradi@idi.ntnu.no

Tore Dybå
SINTEF Telecom and Informatics
NO-7465 Trondheim, Norway
Phone: +47 7359 2947, Fax +47 7359 2977
tore.dyba@informatics.sintef.no

Abstract

Most quality and software process improvement frameworks emphasize written (i.e. formal) documentation to convey recommended work practices. However, there is considerable skepticism among developers to learn from and adhere to prescribed process models. The latter are often perceived as overly "structured" or implying too much "control". Further, what is relevant knowledge has often been decided by "others" – often the quality manager.

The study was carried out in the context of a national software process improvement program in Norway for small- and medium-sized companies to assess the attitude to formalized knowledge and experience sources. The results show that developers are rather skeptical at using written routines, while quality and technical managers are taking this for granted. This is an explosive combination.

The conclusion is that formal routines must be supplemented by collaborative, social processes to promote effective dissemination and organizational learning. Trying to force a (well-intended) quality system down the developers' throats is both futile and demoralizing. The wider implications for quality and improvement work is that we must strike a balance between the "disciplined" and the "creative" way of working.

Keywords: Software process improvement, knowledge transfer, knowledge management, formal routines, developer attitudes.

1 INTRODUCTION

To regulate and improve the work in software development, many organizations and projects have documented their "best work practices" as *formal routines*. These may be prescribed process models, guidelines, rules, check-lists etc. On the other hand, we cannot expect adherence to formal routines ("process conformance") unless the routines are understood, respected and demonstrated to be useful in daily practice. We can mention two extremes: young programmers are famous for their improvisation and disregard for written laws and regulations (the "creative" work mode). On the other hand, we have the military commando system where rules and commands should be obeyed to the letter (the "disciplined" work mode).

The described study was carried out in the context of a Norwegian software process improvement program, SPIQ, involving a dozen small and medium-sized software-intensive companies. Many of these companies have installed their own quality systems

and/or experience bases, but not all these were fully exploited by the developers. Thus, it was natural to investigate how different user groups, such as developers and managers, perceived formal routines as a medium to express and disseminate knowledge and experience.

The structure of the rest of the paper is as follows: Section 2 gives a summary of related work. Section 3 explains the issues and hypotheses raised and the research method to explore these. Section 4 reports the results, Section 5 discusses these, and Section 6 contains a conclusion.

2 RELATED WORK

This section summarizes related work on a more general level. The discussion in Section 5 contains some more specific references to related work.

The emphasis on formal routines reflects a common assumption in software process improvement (SPI) and quality assurance (QA), that *"the quality of a software product is largely governed by the quality of the process used to develop and maintain it"* [Pau95], p. 8. This often means that relevant work practices (processes) must be systematically documented as formal routines, often as standard process models. These routines must then be communicated to the developers, customized and adopted by them and later revised. Many companies have established *quality system* to encode the routines. Such a quality system may be coupled to a *software experience base (SEB)*, containing experimental data and aggregated models (i.e. "knowledge") based on such data. With the advent of Internet and Web technologies, many quality systems and experience bases are now using such media. We should, however, emphasize that technological remedies are only a means to reach a goal. In all quality and improvement work, the ultimate success criterion is *satisfied customers* in the spirit of Total Quality Management (TQM) [Dem86].

Some definitions: Within the context of this study, we define *formalization* as written rules, procedures, and instructions. *Explicit* knowledge is formalized knowledge, e.g. as process models or guidelines. *Tacit* knowledge is the operational skills that practitioners possess, including practical judgement capabilities. Tacit knowledge cannot always be made explicit. *Learning* is lasting modification in behavior, based on experience and understanding. It requires *both* formal training and informal information exchange. For transfer of experience, a useful model is shown in **Figure 1** below [Non95]:

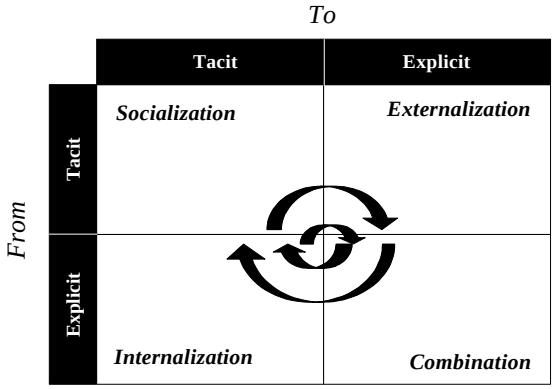

Figure 1: A model for knowledge conversion between tacit and explicit knowledge.

This figure expresses that practitioners first *internalize* new knowledge (i.e. individual learning). The new knowledge is then *socialized* into revised work processes and changed behavior (group learning). The new work processes and the changed behavior are then observed and abstracted, i.e. *externalized*. This new knowledge is then *combined* to refine and extend the existing knowledge (organizational learning). This process continues in new cycles etc.

To enable *learning* is the crucial issue, both at the individual, group, and organizational level. The latter means creating and sustaining a *learning organization* that constantly improves its work, by letting employees share experience with each other. Around the underlying experience bases, there may be special (sub-)organizations to manage and disseminate the stored experience and knowledge, as exemplified by the *Experience Factory* [Bas84]. We also refer to the workshop series of *Learning Software Organizations* [Bom99]. The *knowledge engineering* community has also worked on experience bases, often with emphasis on effective knowledge representations, deduction techniques, case-based reasoning etc., and towards a wide range of applications.

Social anthropologists and *psychologists* have studied how organizations “learn”, and how their employees make use of information sources in their daily work. Much R&D effort has been spent on the “externalizing" flow, looking for valid experience that can be analyzed, generalized, synthesized, packaged and disseminated in the form of improved models or concepts. For instance, to make, calibrate and improve an estimation model based on the performance of previous software projects. Explicit knowledge may nevertheless be misunderstood due to lack of context and nuances, e.g. how to understand the context of post-mortems?

However, the hard part is the “internalizing" flow. That is, how to make an impact on current practice, even if updated knowledge may be convincingly available? See for instance the ethnographic study on the use of quality systems in [Zub88]. Typical inhibitors are “not-invented-here", mistrust (“been-burned-before"), lack of extra time/resources (“not-getting started”), or plain unwillingness to try something new or different (like adhering to formal procedures in a quality system). A study of maintenance technicians for copy machines indicated that such experts were most likely to ask their colleagues for advice rather than to look it up in or even to follow the “book”, or to informally exchange experience and news around the waterpost, termed *story-telling* in [Bro91].

Furthermore, the existence of *software quality manuals*, either on paper in thick binders (sometimes 1-2 m in the shelves) or in web documents on an Intranet, is no guarantee for their use. In fact, since manuals may dictate people on how to perform their job, traditional quality departments in many software organizations are not looked upon with high esteem by developers. For instance, there are over 250 proposed *software standards* [Pfl94], often “standard” process models, but how many are in practical use?

So, if we are to succeed with formal routines and explicit knowledge in a quality system or a SEB to achieve *learning*, we must *not* carry the traditional “QA hat” of *control*. This does *not* mean that all formal knowledge in the form of books, reports etc. (like this article) has to be discarded. The lesson is just that formal routines must be formulated and introduced with proper participation from the people involved, in order to have the intended effect on practice.

Lastly, many of the ideas and techniques on quality improvement (TQM and similar) come from manufacturing, with rather stable products, processes and organizations.

Information technology, on the other hand, is characterized by rapid *product innovation*, not gradual process refinement [Rif99]. One "IT" year is like a "dog" year (7 years) in other disciplines, and time-to-market seems sacred (i.e. schedule pressure). The strength of many software SMEs (Small and Medium-sized Enterprises) lies in their very ability to turn around fast and to convert next week's technologies into radically new products and services. Barrett [Bar98] has used the term *improvisation*, a jazz metaphor, to characterize performers that execute evolving activities while employing a large competence base. With reference to our software development context, we must carefully adopt a set of quality and improvement technologies that can function in a very dynamic environment – so how to *manage constant change* [Dyb00b]? Since SPI assumes that there is "something" stable that can be "improved", we must pick our learning focus accordingly. For instance, the Norwegian Computer Society (www.dnd.no) is now offering a course in "chaos and complexity theory" as an alternative to manage highly evolving projects.

However, it is fair to say that especially TQM *is* aware of the cultural and social dimensions of quality work. TQM has a strong emphasis on creating a learning organization, and having all employees participate and involve themselves in order to satisfy their customers. So, how can software organizations best systematize, organize and exploit previous experience in order to improve their work?

3 CONTEXT, QUESTIONS, AND METHOD

3.1 The SPIQ project

The **SPIQ** project [Con96] was run for three years in 1997-99, after a half-year pre-study in 1996. SPIQ stands for **SPI for better Quality**. The project, which was funded in part by the Research Council of Norway, involved three research institutions and 12 IT companies, mostly SMEs. More than 20 SPI pilot projects have been run in these companies. The main result of the SPIQ project was a pragmatic *method handbook* [Dyb00a]. Typical work in the cooperating companies included pilot projects to test out a certain improvement technology, like novel inspection techniques, incremental development, or use of measurement and software experience bases. A follow-up project called PROFIT is now carried out in 2000-2002. For further results from comparative studies of SPI success, emphasizing organizational and cultural factors see e.g. [Con00].

3.2 How the study was performed

Overall organization: The actual study was carried out between NTNU/SINTEF and five companies participating in the SPIQ project. Data collection was carried out by two NTNU students in the last year of their MSc study, as part of a "pre-thesis" project [Car99]. The two students were advised by the two authors of this paper, the former being their teacher and also a SPIQ researcher, the latter being a researcher and PhD student attached to the SPIQ project.

Preparation: First, the student read and learnt about the project and relevant literature. Then we tried an initial formulation of some important issues in the form of research questions, and discussed these. At the same time, we contacted potential companies and checked their willingness to participate and their availability. Then a more detailed

interview guide (see 3.3) was designed, in dialogue between the students and their advisors. The companies had been briefed about the questions, and when and how the interviews were going to be run.

Research questions – important issues to address are:
Q1: What is the knowledge of the routines being used?
Q2: How are these routines being used?
Q3: How are they updated?
Q4: How effective are they as a medium for transfer of knowledge and experience?
And, furthermore, are there important differences between developers and managers, and how much cooperation is involved in making and updating the routines?

Subjects: Initially, we tried to have a dozen companies involved, but the time frame of the students' availability (three months in spring of 1999) only allowed five companies. One of these was in Trondheim, the rest in Oslo. Three of the companies were ISO-9001 certified. Two of the companies were IT/telecom companies, the rest were software houses. A convenience sample (i.e. available volunteers) of 23 persons were interviewed based on their experience with SPI, whereof 13 developers and 10 managers. The latter group included one quality manager and one software manager (e.g. division or project manager) from each of the five companies.

Data collection: After finishing the interview guide, this was sent by email to the respondents. A few days later, the two students visited the companies, and spent a full day at each company. At each place they spent up to one hour with each respondent in semi-structured interviews. In each interview, the four questions were treated one after another. One of the students was asking the questions, and both students made notes (interview records) during the interview. The other student served as a scribe, and wrote down a structured summary immediately after the interview. The first student then checked this against his own notes.

Data analysis: The ensuing categorization and data analysis was done by the two students, in cooperation with the authors, and reported in the students' pre-diploma thesis.

3.3 The interview guide

As mentioned, we formulated **four main research questions** and 14 sub-questions:

Q1. Knowledge of the routines.

1.1 Describe the (possible) contents in routines being used for software development.

1.2 How were these routines introduced in the company?

1.3 What was the purpose of these routines?

1.4 What is you personal impression of the routines?

Q2. Use of routines.

2.1 What status does a given routine have among developers?

2.2 To what degree are the routines actually being used?

2.3 Who are the most active/passive users?

2.4 What is the availability of the routines?

2.5 How is follow-up and control of usage done?

Q3. Updating of routines.

3.1 What procedures are used to update the routines?

3.2 Who participates in the update activities?

Q4. Routines as a medium for transfer of knowledge and experience.

4.1 Do you regard written routines as an efficient medium for transfer of knowledge and experience?

4.2 What alternatives to written routines do you think are useful or in use in the company?

4.3 What barriers against transfer of experiences do you think are most important?

The interview guide contained advice on how to deal with structured questions, usually with three answer categories such as "yes-maybe-no" or "little-some-much". We allowed more unstructured commentaries in the form of prose answers to solicit more direct and commentary opinions.

4 RESULTS

In this section, we present the results of our study regarding the utility of formal routines as a medium for transfer of knowledge and experience. The focus is on participation and potential differences in opinion between developers and managers in this matter.

On Q1: Knowledge of the routines

All respondents had a fairly good knowledge of the routines that were in place in their respective companies. In fact, two thirds of the respondents showed good knowledge about the content of the routines. **Table 1** illustrates this, and shows how well developers and managers were able to describe the specific contents of the routines in their company.

Table 1: Knowledge of company routines.

	Software developers (n = 13)		**Managers** (n = 10)	
	Frequency	Percent	Frequency	Percent
Little	-	-	-	-
Some	6	46	2	20
Much	7	54	8	80

However, when it came to knowledge about how the routines were introduced, 50% of the developers did not know anything about this process. On the other hand, only one manager did not know about the introduction process. All in all, it turned out that about 30% of the developers and 70% of the managers had actively *participated* in the introduction of routines (**Table 2**).

Table 2: Degree of involvement during introduction of routines.

	Degree of involvement			
	Low		High	
	Freq.	Percent	Freq.	Percent
Developers	9	69	4	31
Managers	3	30	7	70

Furthermore, it seemed to be a common understanding regarding the *objective* of having formal routines. Most respondents said that such routines were useful with respect to quality assurance. Other respondents said that they would enable a more unified way of working. However they emphasized that:

"*Routines should not be formalistic, but rather useful and necessary*".

Respondents in the three ISO-9001 certified companies claimed that their routines were first and foremost established to "get the certificate on the wall", and that the quality of their software processes had gained little or nothing from the ISO certification. One of the respondents expressed his views on this by the following example:

"*You might be ISO certified to produce lifebelts in concrete, as long as you put the exact same amount of concrete in each lifebelt.*"

Although some of the respondents were critical to the routines, stating that:

"*10% of the routines are useful, while the remaining 90% is nonsense*"

Most respondents, nevertheless, had a good impression of the routines, typically stating that:

"*Routines are a prerequisite for internal collaboration.*"

"*Routines are a reassurance and of great help.*"

On Q2: Use of routines

Software developers and managers agreed on the degree to which the routines were used. In general, they answered that about 50% of the routines were in use, and that the more experienced developers used the routines to a lesser extent than the more inexperienced developers do. Furthermore, it was a common agreement that:

"*There is no point in having routines that are not considered useful*".

However, the status of the routines among the software developers was highly divergent, as seen from the following statements:

"*The routines are generally good and useful, but some developers are frustrated regarding their use.*"

"*The system is bureaucratic – it was better before, when we had more freedom to decide for ourselves what should best be done.*"

"*The routines are easy to use.*"

"*Routines are uninteresting and revision meetings are boring.*"

On Q3: Updating of routines

None of the companies had scheduled revisions as part of the process for updating their routines. Most answers to this issue were rather vague. Some respondents explained that such revisions were informally triggered, while other respondents did not know how to propose and implement changes to existing routines. However, respondents from all of the companies, both managers and software developers, said that all employees in their respective companies could participate in the revision activities if they wanted to.

On Q4: Routines as a medium for transfer of knowledge and experience

The answers to this issue varied a lot, and indicated highly different attitudes regarding the effectiveness of formal routines for knowledge and experience transfer. Particularly, it seemed to be a clear difference in judgment between software developers and managers. While seven of the ten managers regarded written routines as an efficient medium for knowledge transfer, none of the developers did! Furthermore, half of the developers considered such routines to be inefficient for knowledge transfer, while only one of the managers shared this view.

Typically, managers said that written routines were important as means for replacing the knowledge of the people that had left the company. Software developers, on the other hand, did not make such a clear connection between experience, knowledge transfer and formal routines. One software developer said that different groups within the company never read each other's reports, while another developer maintained that it would take too much time to learn about the experience of the other groups. Several of the developers explained their views by stating that the documentation was not good enough, it was hard to find, boring to use and that it takes too much time.

When asked about useful alternatives to written routines, the respondents answered that they regarded some kind of "Experience base" or "Newsgroup" as the highest ranked alternative. Other high-ranked alternatives were "Socialization", "Discussion groups", "Experience reports", "Group meetings", and "On-the-job training". **Table 3** shows these alternatives in rank order (1 is best) for software developers and managers respectively.

Table 3: Alternative media for knowledge transfer.

	Rank	
Medium	Developers	Managers
Experience base/newsgroups	1	1
Socialization	2	3
Discussion groups	3	2
Experience reports	4	3
On-the-job-training	5	6
Work w/ ext. consultants	6	-
Group meetings	7	5

We also asked the respondents about what they regarded as the most important barriers

against transfer of knowledge and experience. Nearly all of them said that such transfer, first and foremost, is a personal matter depending on how much each individual whishes to teach their lessons-learned to others. Furthermore, the willingness to share depends on available time, personality, self-interest, and company culture.

As shown in **Table 4**, seven (six developers and one manager) of the 23 respondents answered a definite "No" to the question regarding the efficiency of written routines as a medium for transfer of knowledge and experience. Likewise, seven respondents (managers only) answered an equally clear "Yes" to the same question. The last nine respondents answered somewhere in between.

Table 4: Do you regard written routines as an efficient medium for transfer of knowledge and experience?

	Software developers (n = 13)		**Managers** (n = 10)	
	Frequency	Percent	Frequency	Percent
Yes	-	-	7	70
Both	7	54	2	20
No	6	46	1	10

Due to the rather small sample size in this study, and the low expected frequency in several of the cells in **Table 4**, we compared the respondents' assessments of the routines and their job function using Fisher's exact probability test. With this test, the exact probability (or significance level) that the obtained result is purely a product of chance is calculated. The test statistic of 13.02 was highly significant (p=0.002, two-tailed). Thus, we rejected the hypothesis of independence and concluded that there is a difference in the distribution of assessment of the usefulness of formal routines as an efficient medium for transfer of knowledge and experience between software developers and managers.

Table 5: Degree of involvement vs. assessment of formal routines as an efficient medium for transfer of knowledge and experience.

	Degree of involvement	
Efficient medium?	Low	High
Yes	-	7
Both	5	4
No	7	-

Since software developers had been involved in the process of introducing the routines to a much lesser extent than the managers, we compared the respondent's assessment of the routines with the level of involvement using Fisher's exact test (**Table 5**). The test statistic of 14.71 was highly significant (p<0.0005, two-tailed). Thus, we concluded that there is a difference in the assessment of the usefulness of formal routines as an efficient medium for transfer of knowledge and experience with respect to the degree of involvement in the introduction process.

5 DISCUSSION

In this section, we restrict the discussion to possible explanations of why none of the software developers in our study regarded formal routines as an efficient medium for transfer of knowledge and experience. Below we present three main reasons for the observed diversity regarding the assessment of the utility of routines.

1. Occupational culture: Software development is radically different from manufacturing. The former is not a mechanical process with strong causal and linear models, where we just need to establish the "right" formal routines. Rather, developers view software development largely as a creative and social activity. The reality for most software organizations is a non-deterministic, multi-directional flux, that involves constant negotiation and renegotiation among and between the social groups shaping the software [Dyb00b]. This does not mean that we should discard discipline and formalization altogether. What is needed, is to *balance* the "odd couple" of discipline and creativity in such work [Gla95].

2. Developer participation around formal routines: Employee participation is crucial, and the developers are a software organization's most important "capital". Adler and Borys [Adl96] have introduced two different types of formalization: an *enabling* type where procedures capture lessons-learned or best practice, and a *coercive* type where procedures are presented without a motivating rationale and thus tend to be disobeyed. In our investigation developers are clearly not against formal routines, especially those that captured prior project experience. Rather, they wanted enabling routines resembling those in a "experience base" or "newsgroup".

3. Working and learning: Our results support the theories of *situated learning* based on "story-telling" [Bro91]. That is, we should emphasize informal, as opposed to formal learning. The latter alone are inadequate. It is not surprising, therefore, that "socialization" and "discussion groups" were among the highest ranked alternatives to formal routines.

Some implications:
First, *studies of the effects of formalization*, whether they are enabling or coercive, should focus on the features of the actual routines as well as their implementation. In addition, we should pay more attention to designing the features and the associated goals.
Second, we must recognize and confront the implications of the deeply *embedded and tacit assumptions* of the different occupational cultures. And, furthermore, learn how to establish better cross-cultural dialogues in order to enable organizational learning and SPI.
Third, a practical implication is that managers should recognize the needs of *balancing discipline and creativity*, in order to supplement formal routines with collaborative, social processes. Only by a deep and honest appreciation of this, can managers expect effective dissemination of knowledge and experience within their organization.

Based on the findings of this study, we conclude that both software managers and developers must maintain an open dialogue regarding the utility of formal routines. Such a dialogue will open the way for empirically based learning and SPI, and thus attain the rewards of an enabling type of formalization.

Limitations and recommendations for future research:
This study focused on the utility of formal routines to transfer knowledge and experience. Although it can provide valuable insights for introduction of formal routines in the software industry, our initial study is not without limitations.

First, the small sample and lack of randomness in the choice of respondents may be a threat to *external validity*. In general, most work on SPI suffers from non-representative participation, since companies that voluntarily engage in systematic improvement activities must be assumed to be better-than-average.

Second, a major threat to *internal validity* is that we have not assessed the reliability of our measures. Variables such as degree of involvement and efficiency of routines are measured on a subjective ordinal scale. An important issue for future studies is therefore to ensure reliability and validity of all measures used. We may also ask if the respondents were truthful in their answers. For instance, they may have sensed we were "looking for trouble", and thus "giving us what we wanted". However, their answers to the four main questions and their added qualitative comments show a consistent picture of skepticism and lack of participation on formal routines. We therefore choose to generally believe their answers.

However, despite the mentioned limitations and lack of cross-checks, we feel that this study makes an important contribution to the understanding of formal routines and their role in organizational learning and SPI. Further studies should be extended, operationalized and used for cross-sectional and longitudinal studies of a much larger number of companies. Furthermore, such studies should include a multiple respondent approach to cover all major occupational cultures. They should also perform supplementary, ethnographic studies on how developers *really work* and how their work relate to formal routines – see [Per94] on observational studies of developers at ATT.

6 CONCLUSION

Results from the study show that software developers do not perceive formal routines alone as an efficient way to transfer knowledge and experience. The study confirms our suspicions about large differences in perception of the utility of formal routines on these matters. That is, developers are skeptical to adopt formal routines from traditional quality systems. They also want such routines to be introduced and updated in a more cooperative manner. These results coincide with many other investigations on similar themes, see Section 2 and also [Par86] on how to fake a rational design process. So in spite of a small sample, we think that the results are *representative* for a large class of software companies.

The basic remedy to all this is to create a more cooperative work atmosphere to promote situated learning, with strong developer participation in designing and promoting future quality systems. The developers also seem open to start exploiting new electronic media as a means for collaboration and linking to newer SEB technologies – see also our previous studies [Con00] on this. Lastly, we were not able to make precise *hypotheses* on our four issues beforehand, so the study has a character of a *preliminary* investigation. Later studies may be undertaken with more precise hypotheses and on a larger sample.

Acknowledgements: Thanks to colleagues in the SPIQ project and not at least to the students Jon E. Carlsen and Marius Fornæss that did the fieldwork [Car99].

REFERENCES

[Adl96] Paul S. Adler and Bryan Borys, "Two Types of Bureaucracy: Enabling and Coercive", *Administrative Science Quarterly*, Vol. 41 (1996), pp. 61-89.

[Bar98] F. J. Barrett, "Creativity and Improvisation in Jazz and Organization: Implications for Organizational Learning", *Organization Science*, 1998, Vol. 9, No. 5, pp. 605-622.

[Bas84] Victor R. Basili, Gianluigi Caldiera, and Hans-Dieter Rombach, "The Experience Factory", In John J. Marciniak, editor, *Encyclopedia of Software Engineering – 2 Volume Set*, John Wiley and Sons, 1994, pp. 469-476.

[Bom99] Frank Bomarius, editor, *Proc. 1st Workshop on Learning Software Organizations* (associated to SEKE'99), Kaiserslautern, 16 June 1999, 126 p. Publisher: Fraunhofer IESE, Kaiserslautern.

[Bro91] John S. Brown and Paul Duguid, "Organizational Learning and Communities of Practice: Toward a Unified View of Working, Learning, and Innovation", *Organization Science*, Vol. 2, No. 1 (Feb. 1991), pp. 40-57.

[Car99] Jon E. Carlsen and Marius Fornæss, "Undersøkelse om Prosessforbedring" (in Norwegian), IDI, NTNU, Trondheim, 30 April 1999, 72 p., EPOS TR 357 (pre-diploma project thesis).

[Con96] Reidar Conradi, "SPIQ: A Revised Agenda for Software Process Support", In Carlo Montangero, editor, *Proc. 4th European Workshop on Software Process Technology (EWSPT'96)*, pp. 36-41, Nancy, France, 9-11 Oct. 1996. Springer Verlag LNCS 1149.

[Con00] Reidar Conradi, Mikael Lindvall, and Carolyn Seaman, "Success Factors for Software Experience Bases: What We Need to Learn from Other Disciplines", In Janice Singer et al. (Eds.), *Proc. ICSE'2000 Workshop on Beg, Borrow or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research*', Limerick, 5 June 2000, pp. 113-119.

[Dem86] W. Edwards Deming, *Out of the crisis*, MIT Center for Advanced Engineering Study, MIT Press, Cambridge, MA, 1986.

[Dyb00a] Tore Dybå, editor, *SPIQ metodebok for prosessforbedring i programvareutvikling – v3.0* (in Norwegian), SINTEF/NTNU/UiO, Trondheim and Oslo, Norway, Jan. 2000, ca. 200 p.

[Dyb00b] Tore Dybå, "Improvisation in Small Software Organizations: Implications for Software Process Improvement", *IEEE Software*, Vol. 17, No 5, Sept.-Oct. 2000, pp. 82-87.

[Gla95] Robert L. Glass, *Software Creativity*, Prentice Hall, Englewood Cliffs, N.J., 1995.

[Non95] Ikujiro Nonaka and Hirotaka Takeuchi, *The Knowledge-Creating Company*, Oxford University Press, 1995.

[Par86] David L. Parnas and Paul C. Clements, "A Rational Design Process – How and Why to Fake it", *IEEE Trans. on Software Engineering*, Vol. 12, No. 2, pp. 251-257, February 1986.

[Pau95] Marc C. Paulk, Charles V. Weber, Bill Curtis, and Mary B. Chrissis, *The Capability Maturity Model for Software: Guidelines for Improving the Software Process*, SEI Series in Software Engineering, Addison-Wesley, 1995, 640 p.

[Per94] Dewayne E. Perry, Nancy Staudenmayer, and Lawrence G. Votta, "People, Organizations, and Process Improvement", *IEEE Software*, Vol. 11, No. 4, July 1994, pp. 36-45.

[Pfl94] Shari Lawrence Pfleeger, Norman Fenton, and Stella Page, "Evaluating Software Engineering Standards", *IEEE Computer*, Sept. 1994, pp. 71-79.

[Rif99] Stan Rifkin, "Discipline of Market Leaders and Other Accelerators to Measurement", *Proc. 24th Annual NASA-SEL Software Engineering Workshop* (on CD-ROM), NASA Goddard Space Flight Center, MD 20771, USA, 1-2 Dec. 1999, 6 p.

[Zub88] Soshana Zuboff, *In the Age of the Smart Machine*, Basic Books, New York, 1988.