

Software Process Improvement: What gets measured gets done

Petter Øgland

Department of Informatics, University of Oslo, Norway

petterog@ifi.uio.no

Abstract. A software quality management system (QMS) has been designed in a democratic fashion by programmers and has been approved by management. In order to motivate the programmers in following their own standards, measurement feedback is used for creating social pressure (peer pressure and management pressure). In nine out of ten cases this approach has been successful. Could the failure of the tenth case be an indication of the statement “what gets measured gets done” not being as obviously true as it is sometimes assumed to be in management literature? Trying to investigate this by reflecting on seven years of conversations with programmers and managers, various explanations are presented. Few seem to have any validity. The most likely explanation seems to be that “what gets measured gets done” is a statement about group behaviour. When it comes to individuals or very small groups, special predicaments may make behaviour less predictable. The theoretical insights for QMS design, provided by this case study, is that the “what gets measured gets done” design approach might prove more efficient if one manages to conceptualize lumps of individuals as self-managing teams or groups.

Introduction

The use of measurements in feedback loops plays a fundamental role in software process improvement (Sommerville, 2001). Not only are measurements useful for getting objective knowledge on process characteristics, but the measurement process may have motivational implications by itself as illustrated by the “what gets measured gets done” principle often mentioned in management literature (e.g. Waterman & Peters, 1982).

The purpose of this introduction is to describe the background for a case study where the “what gets measured gets done” principle was used for designing a quality management system (QMS) for COBOL software. The design of the system is treated as part of the background for the study, and thus presented in this introduction, where the motivation for research comes from unexpected results from how the QMS is performing.

Background for case study

Within the IT department of a Scandinavian public sector finance institution the lack of standardized software development practices has caused critical situations when trying to rotate software among programmers or whenever new programmers enter or old programmers leave. A near-fatal incident in 1997, where one of the main programmers responsible for vital parts of a nationally critical software system tragically died and left the remaining programmers with almost incomprehensible code to handle, caused the organization to rethink its strategies of software development. The IT strategy document of 1998 emphasized the need for developing and following internal programming standards that were compliant with international software standards. There were also updates in the IT security policies in 2000. Interviews with programmers and managers 2000-2008, carried out by the author of this paper (who was working as a quality manager from 2000-2005), indicated that all shareholders seemed to agree on the seriousness of the situation and all seemed to agree on the need for developing and implementing common standards (Øgland, 2006).

Designing a quality management system

In trying to solve the practical problem of making programmers align with common standards and make sure that all software, new and old, is gradually made compliant with the standards, a small quality management system (QMS) was designed by focusing on the following ideas:

- *Participatory Design* (the programmers themselves designing both the standard and the management system for making sure the standard is being followed)
- *Double-loop learning* (create a system that not only makes software comply with standards but also helps reflect upon whether the standards should be improved or revised)
- “*What gets measured gets done*” (performance is evaluated by measurements and used for objective feedback)

Consistent with ideas of Participatory Design, a system for guiding and monitoring the process of standardizing the software was developed by the

programmers themselves. The system consisted of an internal standard for the development of COBOL software, developed according to the local procedure used for developing standards, a software program for testing business software against a given set of tests (a subset of all the tests implied by the standard), and the management of the system consisting of one of the programmers running the test program and giving the results to the quality coordinator (external to the group) while the quality coordinator did statistical analyses and returned the results to the programmers.

The QMS was designed as a “double-loop learning” system (Argyris & Schön, 1978, pp. 1-6) as illustrated in figure 1. An initial programming standard was designed by the programmers and then used for testing the software. As deviations from standard (“defects”) were discovered, “single-loop learning” consisted of eliminating the defects and making software more uniform and hopefully more accessible for all programmers. The outer loop in figure 1 is a design feature to stimulate debate on whether the initial standard is too flexible, too restrictive or whether it could be “improved” in some way or another (“double-loop learning”).

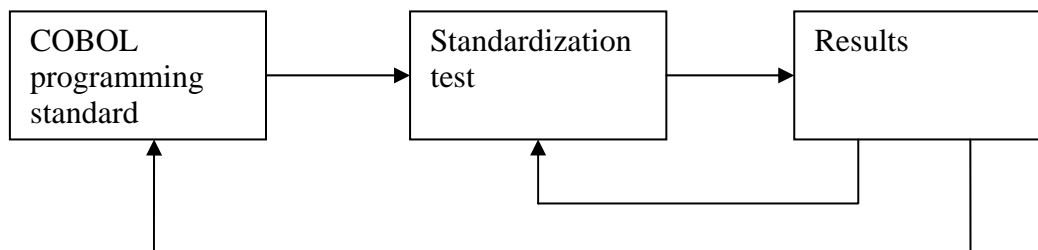


Figure 1. Software quality management system designed as a “double-loop learning” system

Immediately after the system was implemented in 2000, single-loop learning set in as measured in terms of the defect level systematically decreasing in most of the software packages being monitored. A few years later, in 2003, there was an instance of double-loop learning as the programmers decided that the initial version of the standard was too restrictive resulting in the standard and the test program being redefined (Øgland, 2006).

Another important part of the design for the QMS was stimulating improvement by benchmarking annual defect levels and annual improvement results among the various projects and informing all shareholders of the results, as illustrated in figure 2 with management pressure and peer pressure indicated through dotted lines.

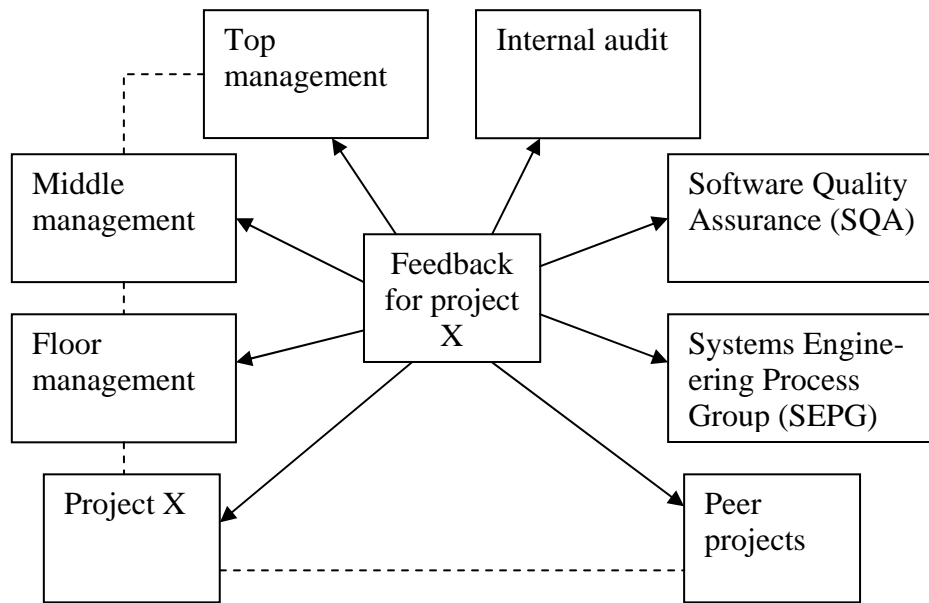


Figure 2. Design for buiding social tension by giving distributed feedback

The idea behind this aspect of the QMS design was to try to create tension needed to prevent the dynamics of the system reaching equilibrium of no further improvement rather than a homeostasis of continuous improvement, as suggested by total quality management theory based on reframing Kurt Lewin’s force field theory and action research within the framework of complex adaptive systems (Goldstein, 1996; Øgland, 2006; 2007b).

Observations

While some software projects started the process of standardizing in 1998, as a consequence of the near-fatal incident, the common COBOL software standard was accepted in 2000. There are ten software development projects being run in parallel, each project responsible for standardizing the COBOL software for one particular software system.

The solid curve on the left side diagram in figure 3 shows the results of repeated estimates of “year of zero defects”, using linear regression estimates on plots representing the average defect levels for all ten projects. The oscillations in the curve indicate that the estimates are not to be trusted too much. The rapid fall in the curve between 2002 and 2003 is the result of “double-loop learning” (the standard being made less strict, thus producing less “defects” and consequently advancing the expected year of zero defects although the improvement rate is unchanged). The current guess, based on estimates for the last four years, is that all relevant software will be compliant with the given standard around 2011, although the latest point estimate suggests 2014 as year of zero defects.

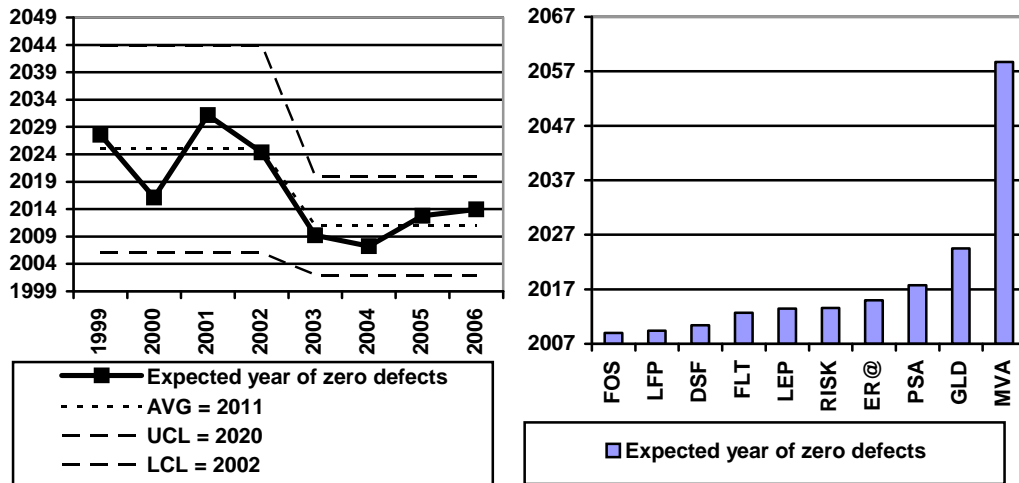


Figure 3. Results of linear regression for predicting the year of zero defects

The diagram on the right hand side of figure 3 shows the latest predictions for year of zero defects for each of the ten projects. While many projects are expected to reach the level of zero defects within a few years, for the tenth project the regression line predicts the year of zero defects lies more than fifty years ahead in time (more than nine standard deviations behind the average zero-defects-year for the other nine projects).

Evaluating the design in figure 1 and 2 based on the results shown in figure 3, the standardization process is not working as well as one might hope for. Why is the tenth project behaving differently from the others?

Research problem and structure of paper

Considering the ideas informing design of the QMS, such as user participation, flexibility for changing the governing variable, clear feedback in the shape of measurements and metrics and trying to design the management system as a game rather than something imposed upon the workers by management, in one of the ten cases the design is not working as well as the designer (writer of the paper) would have hoped. The research question is stated as:

- Do the observations challenge the assumption of “what gets measured gets done”?

The research question is motivated by the hope that a better understanding of why the tenth project is failing might contribute ideas on how to improve the design of quality management systems, making systems for solving similar problems more efficient and robust.

In the next section, performance measurement systems theory and flow theory will be used to frame the problem within a context that could perhaps provide some ideas for better design of quality management systems. The approach used for gaining insights on the behavior of the tenth project, consists of a series of interviews that have been carried out in combination with looking at output from the QMS in terms of numerical tables and diagrams. A general description of methods and material is presented in the third section while the case study results follow in the fourth section. In the final section the aim is to try to make sense out of the interviews and statistics through a discussion that leads to a conclusion and new design ideas.

Designing software quality management systems

According to Sommerville (2001, p. 536-7), software quality management can be structured into three principal activities:

1. *Quality assurance.* The establishment of a framework of organizational procedures and standards which lead to high-quality software.
2. *Quality planning.* The selection of appropriate procedures and standards from this framework and the adaptation of these for a specific software project.
3. *Quality control.* The definition and enactment of processes which ensure that the project quality procedures and standards are followed by the software development team.

Furthermore, there are two types of standards that may be established as part of the quality assurance process (Sommerville, 2001, p. 539):

1. *Product standards.* These are standards that apply to the software product being developed. They include document standards such as the structure of the requirements document which should be produced, documentation standards such as a standard comment header for an object class definition and coding standards which define how a programming language should be.
2. *Process standards.* These are standards that define the processes which should be followed during software development. They may include definitions of specifications, design and validation processes and a description of the documents which must be generated in the course of these processes.

According to this classification, the standardization method described in the introduction can be described as a product standard although the method in itself may be seen as a process standard.

Concerning some of the problems described in the introduction, Sommerville (2001, p. 540-1) describe these as well-know and gives advice on how to avoid such problems.

Software engineers sometimes consider standards to be bureaucratic and irrelevant to the technical activity of software development. This is particularly likely when the standards require tedious form-filling and work recording. Although they usually agree about the general need for standards, engineers often find good reasons why standards are not necessarily appropriate to their particular project.

To avoid these problems, quality managers who set the standards need to be adequately resourced and should take the following steps:

1. Involve software engineers in the development of product standards. They should understand the motivation behind the development of the standard and be committed to these standards. The standards document should not simply state a standard to be followed by should include a rationale of why particular standardization decisions have been made.
2. Review and modify standards regularly to reflect changing technologies. Once standards are developed they tend to be enshrined in a company standards handbook and there is often a reluctance to change them. A standards handbook is essential but it should evolve with changing circumstances and technology.
3. Provide software tools to support standards wherever possible. Clerical standards are the cause of many complaints because of the tedious work involved in implementing them. If tool support is available, there is not a great deal of additional effort involved in development to the standards.

Sommerville does not include advice on what to do when these design principles have been used and some of the software engineers still ignore the standards, as was the case mentioned in the introduction.

Offen and Jeffrey (1997) include issues such as “a quality environment has already been established” and “there is senior management/sponsor commitment” in their list of success factor framework in related research on establishing software measurement programs (using the M3P model), but they make no suggestions on how to succeed with the model if these two factors are not present. On the contrary, they report problems with installing a measurement program in a government organization where success factors were not present.

In a comparative study of two organizations IS and ES implementing software metrics programs, Hall and Fenton (1997) draw some conclusions on practitioner attitude:

One of the most important factors in metrics program success is practitioner attitude: If you fail to generate positive feelings toward the program, you seriously undermine your likelihood of success.

A significant influence on attitude towards metrics was job seniority. In both organizations, managers were much more positive than develops about metrics use,

introduction, and management. Furthermore, the more senior managers were, the more enthusiastic they were about using metrics. At IS, for example, 87 percent of senior managers were positive about metrics compared to 72 percent of middle managers and 45 percent of junior analysts and programmers.

However, we also found that developers were more positive about metrics than conventional wisdom has led us to believe (Tom DeMarco suspected this was the case.) It is generally thought, especially by managers, that developers are unenthusiastic about quality mechanisms like metrics and cannot see their value. Our results actually show that 80 percent of ES developers and 50 percent of IS developers were positive about metrics use.

It has been said that when developers are asked their opinion about software quality mechanisms, they say such things are useful, but when they're asked to participate they find many reasons why their work must be exempt. This could explain the positive attitudes towards metrics that we found in this study. If this is the case – and developers are only positive about metrics in the abstract – then organizations need to work toward realizing this positive potential. In any case, the relationship between positive perceptions and negative action warrants more research.

A difference between the measurement system design described in the introductory section and the designs of the systems described by Hall and Fenton is the fact that the quality standards, measurements and metrics were defined by the software engineers themselves without management interference beyond the fact that the standards were sanctioned by management and the feedback was not only given to the engineers but also given to managers and other process shareholders.

Although some of the research on the design of software measurement systems mentioned above uses ethnographic studies as a research approach, the researchers do not explicitly suggest applying ethnographic studies as an integrated part of the ongoing QMS implementation processes. Sommerville (2001, p. 562) recommends, however, ethnographic studies as a technique for process analysis in order to understand the nature of software development as a human activity on a more general basis, not discussing software measurements in particular but development in general.

What gets measured gets done

There are numerous references to the principle of “what gets measured gets done” in management and quality control literature. The principle is attributed to Mason Haire in the following context (quoted in Lynch & Cross, 1995, p. 159):

What gets measured gets done. If you are looking for quick ways to change how an organization behaves, change the measurement system.

There are numerous references to this principle in management and quality control literature (e.g. Peters & Waterman, 1982). Based on literature research

carried out so far, it has been difficult to find research questioning or contradicting Haire's statement, although some researchers mention the effect of unexpected results of using measurements, e.g. Behn (2003):

“What gets measured gets done” is, perhaps, the most famous aphorism of performance measurement. If you measure it, people will do it. Unfortunately, what people measure often is not precisely what they want done. And people – responding to the explicit or implicit incentives of the measurement – will do what people are measuring, not what these people actually want done. [...] Thus, although performance measures shape behavior, they may shape behavior in both desirable and undesirable ways.

Although it has been difficult finding research that challenges the verbatim “what gets measured gets done”, there has been written extensively on similar phenomena, such as “the Hawthorne effect”. The Hawthorne effect refers to the work efficiency studies carried out by the Hawthorne Works outside Chicago 1924-32 and is sometimes described as “a short-term improvement caused by observing worker performance” (Landsberger, 1968). There seems to be some controversy as to whether the data from the experiments actually support the anecdote about workers increasing their productivity because of the presence of the observers or whether the “Hawthorne effect” is management mythology unsupported by empirical evidence (Wikipedia, 2008).

Other effects of similar nature to “what gets measured gets done” and the “Hawthorne effect” include the “Pygmalion effect” and the “placebo effect”.

Participatory Design

Among the web sites for the Computer Professionals for Social Responsibility organization there is a web page suggesting that there can be no single definition of Participatory Design (PD) as PD practitioners are so diverse in their perspectives, backgrounds and areas of concern (CPSR, 2008). Nevertheless, the following list of issues is used for giving some guidelines on the sort of thinking that is associated with PD:

- Respect the users of technology, regardless of their status in the workplace, technical know-how, or access to their organization's purse strings. View every participant in a PD project as an expert in what they do, as a stakeholder whose voice needs to be heard.
- Recognize that workers are a prime source of innovation, that design ideas arise in collaboration with participants from diverse backgrounds, and that technology is but one option in addressing emergent problems.
- View a "system" as more than a collection of software encased in hardware boxes. In PD, we see systems as networks of people, practices, and technology embedded in particular organizational contexts.

- Understand the organization and the relevant work on its own terms, in its own settings. This is why PD practitioners prefer to spend time with users in their workplaces rather than "test" them in laboratories.
- Address problems that exist and arise in the workplace, articulated by or in collaboration with the affected parties, rather than attributed from the outside.
- Find concrete ways to improve the working lives of co-participants by, for example, reducing the tedium associated with work tasks; co-designing new opportunities for exercising creativity; increasing worker control over work content, measurement and reporting; and helping workers communicate and organize across hierarchical lines within the organization and with peers elsewhere.
- Be conscious of one's own role in PD processes; try to be a "reflective practitioner."

For the purpose of designing quality management systems it is not obvious that a PD approach may be the best approach. Some quality control experts seem to be of the persuasion that it is the people who do the work who are best capable of describing their processes, but there are others who believe that those doing the work are inside the system and thus not able to see the system properly in the same way as an external specialist would do (e.g. Deming, 1992, p. 54).

Materials and Methods

Insights related to why a certain group of software engineers appears to be behaving inconsistently with the principle "what gets measured gets done" is needed in order to improve the design of the QMS outlined in the introductory section. If one were to give a procedural description P of the management system from the introduction, i.e. a flowchart or a detailed list of steps of what is actually being done in order to monitor and provide feedback to the software engineers standardizing COBOL software, the aim of the research is to come up with a revised procedure P' that would generate better results than P .

From this perspective, the research process can be seen as an integrated part of an engineering process that consists of finding the optimal procedural design P^* for this particular quality control problem. The current research does not aim for an optimal design, but conceptually it could be possible to think of iterations of design research that would produce a series of designs that would ultimately converge into an optimal design where improvement is no longer possible; P, P', P'', \dots, P^* .

In traditional engineering literature science and engineering have sometimes been described as inverse processes. Trying to explain the difference between science and engineering to engineering students, Krick (1969, p. 36-7) explains as follows:

Full appreciation of the role of engineering is difficult if you do not understand the basic distinction between science and engineering. They differ with respect to the basic processes characteristic of each (research versus design), predominant day-to-day concerns, and primary end product (knowledge versus physical contrivances).

Science is a body of knowledge, specifically, man's accumulated understanding of nature. Scientists direct their efforts primarily to improving this understanding. They search for useful explanations, classifications, and means of predicting natural phenomena. In his search for new knowledge the scientist engages in a process called research and in so doing he devotes much of his time to the following activities:

- Hypothesizing explanations of natural phenomena.
- Obtaining data with which to test these theories.
- Conceiving, planning, instrumenting, and executing experiments.
- Analyzing observations and drawing conclusions.
- Attempting to describe natural phenomena in the language of mathematics.
- Attempting to generalize from what has been learned.
- Making known his findings through articles and papers.

The scientist's prime objective is knowledge as an end in itself.

In contrast, the usual end product of the engineer's efforts is a physical device, structure, or process. Let there be no mistake – the gyroscope, the weather satellite, the radio telescope, the electrocardiograph, the nuclear power station, the electronic computer, and the artificial kidney are the fruits of engineering. The engineer develops these contrivances through a create process referred to as design (in contrast to the scientist's central activity – research). Some of the engineer's prime concerns as he executes this process are the economic feasibility, the safeness, the public acceptance, and the manufacturability of his creations. In contrast, the scientist's prime concerns as he performs his function include the validity of his theories, the reproducibility of his experiments, and the adequacy of his methods of observing natural phenomena.

In the case of quality engineering, i.e. the engineering standards and procedures to be included in a quality management system, Krick's references to natural science could be replaced with social science. Juran (1964) suggests that the relationship between management and social research can be seen as identical to the relationship between engineering and natural science.

The design cycle in figure 4 illustrates engineering design and design research as reverse processes.

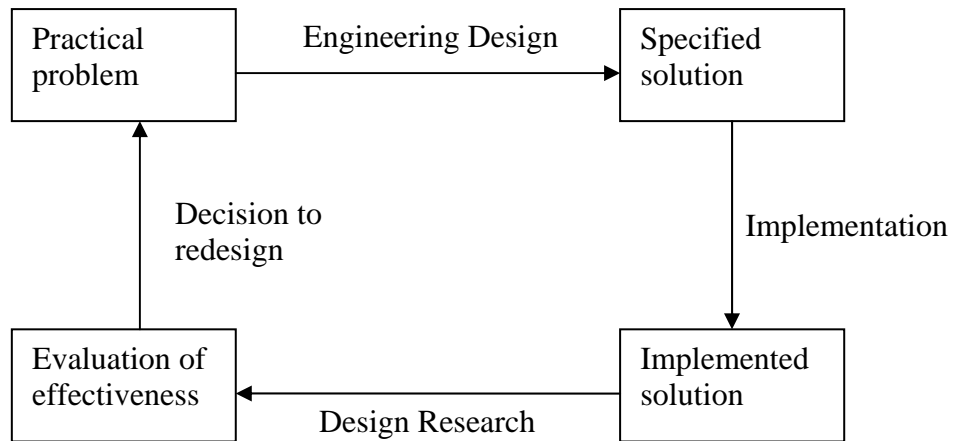


Figure 4. Design Research Cycle (adapted from Krick, 1969, p. 160)

The major difference between the diagram in figure 4 and the original diagram in Krick’s book is that the process named “monitoring the system” has been replaced by “design research”; the tagging of the bottom arrow. What this means in the context of this paper is that not only has the procedure *P* been monitored in terms of plotting results to graphs and interviewing the people involved, but the procedure *P* has been identified as a special case of a class of procedures that consist of measuring performance, plotting and sharing the results as joint feedback to the group. Typical examples of such procedures would be the regular evaluations (“sprints”) that are carried out as a part of agile software development practices such as Scrum (Schwaber, 2004). The goal is then try to question the statement “what gets measured gets done” within this general context of measurement and feedback driven software development, rather than just adding practical knowledge to the practical quality engineering problem in question.

The research design has been inspired by an attempt to interpret aspects of “science of the artificial” (Simon, 1996) in the context of the “research wheel” as described by Rudestam and Newton (1992, pp. 5-8). The process of integrating engineering and research into the same process should be consistent with the research activities defined by March and Smith (1995) as is also the idea of a procedure *P*’ being the output of this kind of research. It is also the belief of the author that the research method explained is in compliance with the seven guidelines for design science as suggested by Hevner et al. (2004).

In previous papers based on earlier stages of the same case study, the action research approach has been used (Øgland, 2006; 2007a; 2007b). However, as pointed out by Cole, Puroo, Rossi & Sein (2005), action research and design research have much in common, even to the extent that it may even be possible to see them as special cases of a generalized “action design science”. The reason why design research has been chosen rather than action research in this third iteration of the same “action design research” study is the author’s impression

that action research seems more useful for researching topics in psychology or sociology while trying to create organizational change while design research aims at adding to the knowledge of designing artifacts (e.g. quality management systems) while, similarly, trying to create organizational change. In other words, while the aim of this paper is to add insights on interventions in a software engineering culture can create organizational change, the intervention has to be of a kind that can be described as a procedure to be added as part of the quality management system of the organization.

Data collection and analysis

Certain parts of the data collection and analysis consist of the data collection and analysis done as a part of the running of the quality management system, as illustrated in figure 3. As the researcher is also the designer of the system, all system documentation, input and output for the system have been ready at hand. Furthermore, it has been possible to collect data on the cost of running the system (in terms of man-hours spent by the engineer/researcher for running and improving the system) and other measurement related to the management of the system.

When it comes to understanding the weaknesses of the design in terms of understanding why the quality control principle of “what gets measured gets done” seem to have been violated to a certain degree for at least one of the projects that interact with the system, it is necessary to conduct interviews with the people involved. Not only the software engineers apparently not behaving as expected are necessary to interview, but also their managers who are the ones who are responsible for standards and procedures being followed. In order to get an even fuller picture of the environment, various other system shareholders have been interviewed, such as software engineers who have been behaving particularly consistent with the expectations of the systems, managers on different levels and people responsible for software engineering process improvement and software quality assurance.

However, as the engineering design of the COBOL standardization quality management system started as a quality engineering project and has only more recently been transformed into a action/design research project, interviews, conversations and observations done up until quite recently have been done either as a part of managing and improving the system and partly at random intervals in an unplanned manner.

Furthermore, while many of the principles from ethnographic studies (e.g. Silverman, 2004) have been followed as an integrated part of the application of international guidelines on how to conduct quality audits (ISO, 2002), neither tape recorders nor notebooks have been used. At certain periods the researcher was also not allowed to interview software engineers (unless accompanied by a

manager), something that often resulted in interviews not being conducted, although the QMS has been running continuously for seven years.

As the system has interfaced with ten project groups, normal management procedure for the running of the QMS should have consisted of carrying out seventy (or more) interviews with software engineers plus additional interviews with managers and other shareholders. However, due to restrictions mentioned and sometimes practical difficulties in making meetings with the software engineers, the research so far has consisted of about 40-50 interviews and/or conversations.

The data analysis has consisted of the researcher trying to remember what has been said and discussed and then trying to make sense out of this (Weick, 1995, pp. 1-16) for the purpose of understanding how to improve the efficiency of the quality management system.

Population characteristics

The interpretative part of the study has consisted of interviewing members from ten projects over a period of seven years. The projects were managed by three group leaders, handling a total of about 40 programmers.

Seven of the projects were related to software to be maintained in consistency with annual waterfall-like life cycles (Sommerville, 2001, p. 45-6), while the remaining three projects were related to on-line computer systems that followed principles more in the style of evolutionary development (Sommerville, 2001, p. 46-8).

The distribution between male and female programmers was about 50/50. The age distribution was from about mid thirties to mid sixties, with most of the people being in the age slot between 40 and 50. Each of the ten projects are manned with 2 to 6-7 people.

The tenth project consists of two people, one person responsible for the MVA system (female, age approx. 40) and one person responsible for the ER system (male, age approx. 50). Although the two systems do not interact functionally, source code is stored on the same disk on the mainframe which is the domain that defines the tenth project.

Results

In the first part of figure 5, the progress is compared between the average defect level of the first nine projects and defect level of the tenth project. The second part of the figure compare the progress rate for each of the ten projects as defined by the regression line made up of plotted values from 2004 and onwards. Due to the revision of the software standard in 2002/2003, the gap produced by the revision is a cause of changes in the evaluation system and not due to “dramatic

improvements” (as mentioned in the introductory section), thus making period from 2004 and onwards more suitable for predicting future progress than looking at the complete time series.

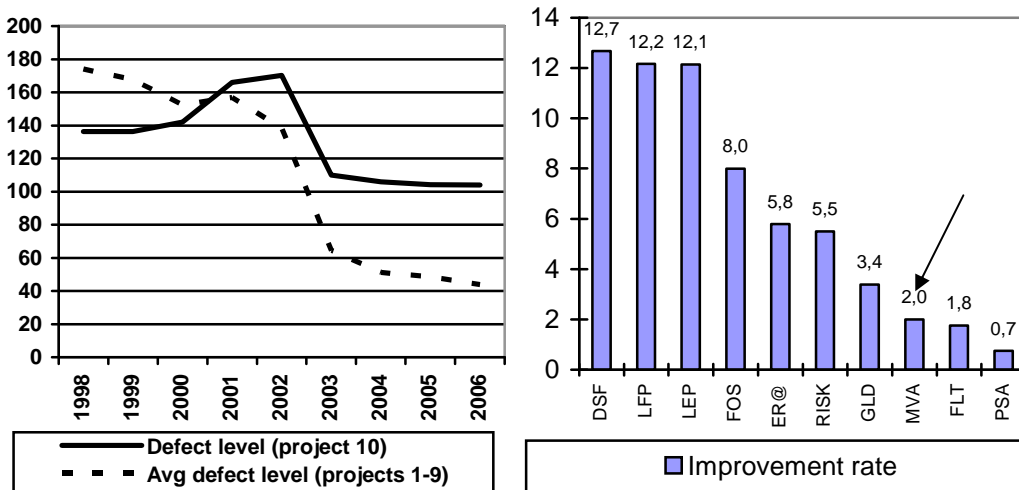


Figure 5. Defect index for the tenth project compared with the average defect index for the remaining nine projects

By visually inspecting the right-hand diagram in figure 5, it is possible that the improvement rate for the solid line was slightly more rapid in the earlier period 1998-2002 than it was in the later period 2003-2006 (although not obvious). The “year of zero defects” results in figure 3 were based on making the best linear fit for the observations 2003-2006 and then calculate where that line would cross the x axis. Perhaps it would be more realistic to believe that the improvements would reach asymptotically towards zero (never reaching the “year of zero defects”), but for the case of trying to understand why the tenth project behaves radically different from the other projects, the linear regression model is mathematically simpler to deal with and provides a good enough approximation for investigating this particular problem.

The individual improvement rates on the right-hand side of figure 5 shows that then tenth project (MVA) is not the project progressing at the slowest rate. In fact, there are two projects making even less annual progress (FLT and PSA). However, when we compare the columns in this figure 5 with the columns in this figure 3, the reason why the projects FLT and PSA score better is because the software they represent have already been standardized to a high level and it is no longer necessary to improve at a very rapid rate in order to reach the “year of zero defects” within a timeline that is comparable with most of the other projects.

Discussions with representatives of the tenth project

During the time of the first conversations, in 2000, the project coordinator for the tenth project was sharing office with the project coordinator for one of the more consistently successful projects (FOS). When discussing first year results with the FOS project coordinator it was possible also to give some hints to the other project coordinator. As can be seen in figure 5, there was no progress in the tenth project the first few years, and as the quality manager (writer of this paper) was of the persuasion that positive results should be give positive and explicit feedback while negative results should be handled more diplomatically, the discussion with the coordinator of the tenth project mostly consisted in saying that the defect level of the software was not bad compared to defect levels in other projects (actually the defect level was better than average as can be seen in figure 5).

The coordinator for the tenth project responded to this by asking whether it was necessary to comply with the standard. From the viewpoint of the quality manager this question was surprising as the point of quality management is to make sure that all standards and procedures are being followed unless there should be some well-argued reason for not doing so. The answer the project coordinator was given was that it could perhaps be a good idea to try to do some small improvements, as most of the other projects were already focused on standardizing and were already producing good results.

Annual conversations during the next few years were of a similar kind. The statistics showed little improvement, but conversations were kept on a friendly level, emphasizing that the project was still scoring okay with respect to benchmark results and also discussing the rationale behind the standardization process, namely to make sure that it should be possible for programmer A to take care of software developed by programmer B in an easy manner as they were both following the same programming standard.

When the 2001 results arrived, however, it was necessary to make inquiries into why the results now not only showed lack of improvement but were in fact getting worse. This year it was not possible to conduct an interview with the project coordinator, but discussing the issue with a project member, he said that they had been discussing the issue of trying to take the standardization challenge more seriously. He found the issue of the defect level getting worse both surprising and amusing, but after some reflection he suggested that the reason might have something to do with development of new programs often consisted of making copies of old programs as a starting point, and if those old programs contained lots of defects, these defects would be copied over to the new programs and thus increase the total level of defects.

The next year the defect level increased even more, but after having pointed this out to a higher level manager, the quality manager was told that he would not be allowed to do more interviews with the tenth project.

The role of the software development model

When confronted with lack of progress in project ten as compared with the nine other projects, the head of the software maintenance department suggested that this could perhaps be explained due to the software development model. The tenth project consisted of making “evolutionary” updates (Sommerville, 2001, pp. 46-8) on a “real-time” system (monthly production runs) while many of the other projects dealt with annual production systems. In the case of annual production systems, the maintenance process followed a strict procedure of updates in specification documents, design documents, programming, testing, production and evaluation in compliance with various annual waterfall models (Sommerville, 2001, pp. 45-6). Working within disciplined waterfall framework, it should not be a great surprise that the standardization of software for systems following such a rigid development model would be more quickly standardized than systems without such a rigid model, using a more evolutionary approach.

His conclusion was that we should not interview the people in the tenth project as they were probably doing their best, and being confronted with poor trend data would only make them annoyed and less productive.

However, the tenth project was not the only project following an “evolutionary development” software process model. In figure 6 below the progress results from the tenth project is compared with progress results from the RISK project and the DSF project.

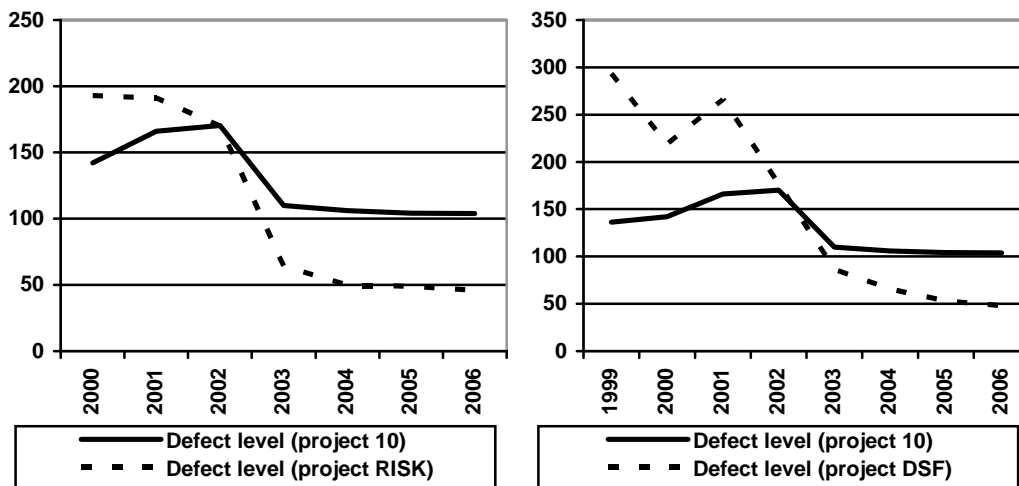


Figure 6. Comparison of progress on the tenth project and two other projects that also maintain software within the framework of an “evolutionary development” software process modell.

The diagram on the left-hand side of figure 6 seems somewhat consistent with the hypothesis that it might be more difficult to create improvement for “evolutionary development” systems where standardization is not part of a well-defined and time-bound waterfall phase for “implementation and unit testing”. The diagram on the right-hand side of figure 6, however, shows good results from

a project trying to standardize software within the framework of evolutionary development.

In both comparisons, the progress during the last few years (since 2003) is more rapid for the reference projects than in the case of the tenth project, as can be seen from the diagram in figure 6 where the prognosis for when the tenth project will be completed is far beyond the time horizon for the reference projects.

The Pygmalion effect

In the year 2000, the quality manager (writer of this paper) was able to discuss some issues of software quality assurance with a visiting SQA person from a large CMMI-level 5 company in India. This was the time when the COBOL standard had been developed and the first iteration of the quality control was about to commence. The discussion evolved around the idea of internal benchmarking and distribution of results as a way for creating needed and sufficient tension for creating improvements, as illustrated in figure 2 and explained in the introduction of this paper. Personally, he would not try something like that, he said, as he expected benchmarking to create little motivation for improvement for those who got top results and would probably work like negative motivation for those who got bottom results.

Reflecting upon this discussion, perhaps the reason for the poor performance in the tenth project could have something to do with the tenth project repeatedly receiving poor benchmark results, each time contributing to a manifestation of a poor self-image that the programmers were poor performers and there consequently being no reason to improve as the results only verified their own pessimistic beliefs.

Producing a scatter diagram with defect level along the x axis and improvement rates along the y axis, the pattern suggested by the Indian SQA consultant actually seem to be confirmed (right-hand side of diagram in figure 7). Using a parabolic regression curve to model the quality improvements (from the past to the present year) as a function of the quality level from the past year, the tenth project fits on the far left (low quality, little improvement) while projects on the far right are those of high quality and little improvement. Those who score low and those who score high on the benchmark results show trends of little annual improvement. Those who are ranked in the middle range of the benchmark results show trends of great improvement.

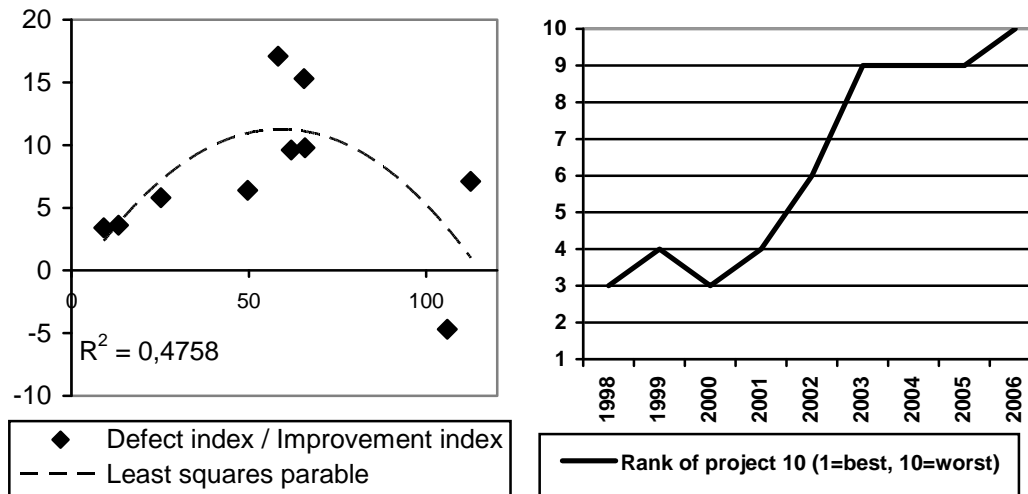


Figure 7. (a) Parabolical relationship between defect indexes and improvement indexes, and (b) history of defect benchmark results (rank) for the tenth project when compared with the others.

However, the diagram on the right-hand side shows that the tenth project has not always been the worst project. In fact, it started out as one of the best and has gradually fallen down on the ranking table. There is no indication of the motivation for standardization having changed during the seven years the tenth project has been observed, although it has been ranked among the best, the mid-level and the worst. Project performance for the tenth project has been consistently below the performance level of the other projects all the time.

Interview with IT top management

At the time when the tenth project had reached a climax of not only failing to improve but actually getting less and less compliant with the standards over several years, the quality manager (writer of this paper) brought the issue up with the IT general director. The IT general director asked whether the quality managers had any ideas why it was like this, why this particular project was performing totally opposite of the other projects. As the quality manager answered that he did not know, the IT general director took up the issue with his board of managers on the next weekly meeting. Although it is unclear what was said during this meeting and what was communicated down the line of three management levels above the software engineers in the tenth project, after this incident the tenth project has been reporting slow but steady improvements, as can be seen in the time series diagrams in figures 6 and 7.

Due to business reorganization in 2007, the former IT general director was replaced with one of the people who had previously been on his board of senior managers. As a part of the ongoing research on how to improve the QMS design, in February 2008 the new director was asked whether he had any suggestions on

what could be the reason why the tenth project was consistently performing at a significantly worse than the other projects. The new IT general director, who previously had worked mostly with external consultant software engineers (for large software development projects) rather than in-house software maintenance engineers, interpreted the situation as typical for difficulty of working with in-house people in a public sector organization.

In private organizations it is possible to remove people not willing to follow the rules of the organization, he said, but in a large public sector organization the employers are protected in ways that makes it close to impossible to handle difficult employees.

Self-assessment: Quality costs

Total quality costs are sometimes defined as prevention costs plus appraisal costs plus failure costs (ASQ, 2008). While the cost of failure, such as the cost of having people trying to make changes into source code they have difficulty understanding, may be difficult to predict, the prevention costs are kept at a minimum by having the software engineers themselves decide which programs to standardize and when to do this (as long as all programs are standardized in the long run). The appraisal costs are the costs of running and improving the quality management system. The running of the system, consisting of collecting numerical data, carrying out the statistical analysis, producing written feedback and conducting interviews, plus writing and annual evaluation report of about 100 pages (25,000 words), amounts to about 70 man-hours (about one man-day for each of the ten projects each year). Improving the system by the use of action research or design research, including the writing of academic papers at a quality that will get them accepted in decent academic outlets, can be costly although exact man-hours haven't been calculated yet.

There have been some minor changes in the system from year to year, trying to improve efficiency and make the process more "scientific" (i.e. compliant with action research and/or design research), but the basic principles of measuring software against a common standard, benchmarking and distributing the results has remained the same all along since the first year the system was run.

Discussion

The study was not designed to compare and contrast a PD approach with a non-PD approach for developing standards and quality management systems. However, the study indicates that it is possible to get quite far by a quantitative engineering approach, producing 90% "automatic improvement" in this particular case. For the final 10%, nevertheless, the mechanical approach does not work (as of yet) and it is necessary to carry out some kind of qualitative studies in order to

find out what actually happens in the social world and see whether this might give some insights that could be used for improving upon the design described in the introductory section.

As was mentioned in the theory section, ethnographic studies should be treated not only as an academic research approach but as an integrated part of software engineering when trying to understand the nature of software engineering as a human activity (Sommerville, 2001, p. 562). One of the reasons for the prolonged weak results for the tenth project may have something to do with a lack of initiative in terms of carrying out interpretative research to find out in more detail what the software engineers in question were thinking and feeling rather than simply rely on the behaviorist hypothesis of “what gets measured gets done”.

On the other hand, as mentioned in the results part, the quality manager / researcher was at times not allowed to interview these people as some of the managers was of the opinion that it would be better to leave these people alone doing their best than to provoke them by asking them about the quality management system.

This protective attitude among some of the managers is different from the case studies reported by Hall and Fenton (1997). In other studies it seems to be more common that managers are focused on standards, measurements, control and improvement. Following the logic implied by the new IT director general, the reason why there seems to be less focus on control and improvement in public sector organizations may be because the survival of the organization does not depend on customers to the same degree as private organizations, and the fact that strong unions and protective culture makes it difficult to apply the philosophy of “managing by measuring” anyway.

Participatory Design

One success factor often mentioned for implementing software measurement systems is the importance of having management commitment (e.g. Offen and Jeffrey, 1997). When there is little management commitment, as in this case, it may be useful to design a quality management system that exploit those few elements of management commitment there might be or to create an illusion of management commitment and use that illusion in order to try to create sufficient social pressure to make sure that people comply with the system (Øgland, 2007b).

Where PD has sometimes been motivated as a part of political research where the aim has been to create a greater level of democracy in the work place or a better culture for communication, in the COBOL programmer case the aim has been that the programmers themselves should take responsibility for the quality of their work by defining the standards, the procedures and measurement systems themselves rather than having such elements pushed upon them from above.

What gets measured gets done?

In nine of the ten projects, the results were consistent with “what gets measured gets done”. This observation is consistent with certain leadership theories, such as the principle that whatever management focuses on tends to get done (Schein, 1985). In this case study, however, the managers did not show much interest in measurements. Rather than getting enthusiastic about the improvement trends, they got worried about non-optimal trends and sometimes tried to argue against measuring or prevent the researcher for doing qualitative studies that might help to understand why the trends were not as good as expected.

In this respect the software measurement design seems to have more in common with agile software development methods such as Scrum where the Scrum coordinator (Scrum Master) is not a traditional project manager in terms of being responsible for progress, but the Scrum Master is a person who, among other things, is responsible for collecting and presenting statistics while it is the team itself that decides what to do (Schwaber, 2004, p. 6-7).

If one should read a “what gets measured gets done” principle into the Scrum framework, an essential part of the Scrum philosophy seems to be the focus on self-managed development teams. Many of the COBOL projects could be defined as self-managed teams in a similar way, as there is no real management pressure and the benchmark statistics are used for turning the process of standardization more into a game-like process. As reported in a previous paper focusing on the programmers who were the leaders in the standardization game, not all the programmers within the group were equally enthusiastic about being measured and benchmarked, but the fact that the group consisted of several people a culture of quality improvement emerged and made the project excel (Øgland, 2006).

In the case of the tenth project, the project consists of a group of one (or two). There are also other projects that consist only of one person, and still perform fairly well, e.g. the RISK project, but a single person project might be more vulnerable. The fact that the tenth project behaves in a different way than the other projects may not be as surprising after all. The implications in Mason Haire’s “what gets measured gets done” quote, as stated in the theory section is a statement about organizational change, not of individual change.

In the case of the tenth project, the more recent measurements show a slight improvement, something that appears to be a consequence of top management involvement. The fact that summary statistics were reported to top management over several years did not seem to have much of an impact on the tenth project, but when there was top management action, improvement were quick to follow.

Conclusions and further research

When doing interviews with various actors in order to get various interpretations on why the project is behaving differently, there are three main suggestions:

- 1) The software life-cycle is different
- 2) The programmers are badly motivated by repeatedly being ranked as worst in class, thus accepting this image and becoming worst in class
- 3) The attempt to socially construct peer pressure and management pressure is not successful. The programmers continue to act as if they were living on an isolated island.

It was shown that neither the first nor the second point is valid. In the case of the first point, there are other projects with a similar underlying life-cycle model that create better improvement results. In the case of the second point, the tenth project has not always been worst-in-class. In the beginning the project was performing better than average but as no action was taken, the project gradually fell down the ranking list until it became “worst-in-class”.

Although the current IT general director made a rather cynical remark about problems with management of behavior in public sector organizations, the comment seems to carry a certain level of truth. If people decide not to follow internal rules and regulations, it may be that such problems need different management methods than some of the most common methods used in private organizations.

The “what gets measured gets done” principle is extracted from a statement about organizational change. It may be “common knowledge” that the principle works for groups of people (Behn, 2003), but whether it works on individuals is not so clear. At least this study has illustrated the principle being set to use for a period of seven years with poor results in one case.

The theoretical contribution from this research is that quality management systems design based on “what gets measured gets done” should make sure that the topology of the larger socio-technical system does not fragment into isolated islands. It may not be possible for the designer of the quality management system to suggest that programmers are periodically rotated among various groups or that they add practices such as peer review of source code, as there may be organizational reasons why such changes may be difficult to achieve, but it should be within the responsibility of the QMS designer to question whether it may be possible to strengthen the social network of programmers so they may be able to communicate and exchange views on the QMS process as something they share.

If it should happen that the key person in the tenth project should either be moved to another project or this person should be asked to be member of another

standardization project in addition to the tenth project, an interesting hypothesis for further research would be to investigate whether the socialization within a new project would carry back inspiration for increasing the improvement rate for the tenth project.

References

- Argyris, C. and Schön, D. (1978). *Organizational Learning: A Theory of Action Perspective*, Addison-Wesley Publishing Company: Reading, Massachusetts.
- ASQ (2008). Cost of Quality (COQ), <http://www.asq.org/learn-about-quality/cost-of-quality/overview/overview.html> (accessed on April 8th, 2008)
- Behn, R.D. (2003). "Why Measure Performance? Different Purposes Require Different Measures", *Public Administration Review*, vol. 63, no. 5., 586-606.
- Cole, R., Purao, S., Rossi M. & Sein, M.K. (2005). "Being proactive: where action research meets design research", *Proceedings of the Twenty-Sixth International Conference on Information Systems*, 325-336.
- CPSR (2008). *What is Participatory Design?* <http://www.cpsr.org/issues/pd/introInfo>, (accessed on March 28th 2008)
- Deming, W.E. (1992). *The New Economics for Industry, Government, Education, Second Edition*, The MIT Press: Cambridge, Massachusetts.
- Goldstein, J. (1994). *The Unshackled Organization: Facing the Challenge of Unpredictability Through Spontaneous Reorganization*, Productivity Press: Portland, Oregon.
- Hall, T. and Fenton, N. (1997). "Implementing Effective Software Metrics Programs", *IEEE Software*, vol. 14, no. 2, 55-64.
- Hevner, A., March, S., Park, J., and Ram, S. (2004). "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, no. 1, 75-105.
- Hoyle, D. (2006). *ISO 9000 Quality Management Systems Handbook, Fifth Edition*, Butterworth-Heinemann: London.
- Imai, K. (1986). *Kaizen: The Key to Japan's Competitive Success*. McGraw-Hill: New York.
- ISO (2002). *Guidelines for quality and/or environmental management systems auditing (ISO 19011:2002)*, International Standards Organization: Geneva.
- Juran, J.M. (1964). *Managerial Breakthrough*, McGraw-Hill: New York.
- Krick, E.V (1969). *An Introduction to Engineering and Engineering Design, Second Edition*, John Wiley & Sons: New York.
- Landsberger, H. (1968). *Hawthorne Revisited*, Cornell University: New York.
- Lynch, R. and Cross. K. (1995) *Measure Up!: Yardsticks for Continuous Improvement*, Wiley: New York.
- March, S.T. & Smith, G.F. (1995). "Design and natural science research on information technology", *Decision Support Systems*, vol. 15, no. 4, 251-266.
- Offen, R.J. and Jeffrey, R. (1997). "Establishing Software Measurement Programs", *IEEE Software*, vol. 7, no. 1, 39-54.
- Peters, T. & Waterman, R.H. (1982). *In Search of Excellence: Lessons From America's Best-Run Companies*, Harpercollins: New York.
- Rudestam, K.E. & Newton, R.R. (1992). *Surviving your Dissertation: A Comprehensive Guide to Content and Process*, Sage Publishing: London.

- Schein, E. (1985). *Organizational culture and leadership: A dynamic view*, Jossey-Bass: San Francisco.
- Schlickman, J. (2003). *ISO 9000 Quality Management Systems Design*, ASQ Quality Press: New York.
- Schwaber, K. (2004). *Agile Project Management with Scrum*, Microsoft Press: Redmond, Washington.
- Seddon, J. (2006). *The Vanguard Standards: A systems thinker's guide to interpretation and use of ISO 9000:2000*, <http://www.lean-service.com/3-1.asp>
- Silverman, D. (2004). *Qualitative Research: Theory, Method and Practice, Second Edition*, Sage Publications: London.
- Simon, H.A. (1996). *The Sciences of the Artificial, Third Edition*, The MIT Press, Cambridge, Massachusetts.
- Sommerville, I. (2001). *Software Engineering, 6th Edition*, Addison-Wesley: Harlow, England.
- Vaishnavi, V., & Kuechler, W. (2006). Design Research in Information Systems. Retrieved April 12, 2007, from <http://www.isworld.org/Researchdesign/drisISworld.htm>
- Weick, K.E. (1995). *Sensemaking in Organizations*, Sage Publications: London.
- Wikipedia (2008). Hawthorn effect. Retrieved February 3rd 2008, from http://en.wikipedia.org/wiki/Hawthorne_effect
- Øgland, P. (2006). "Using internal benchmarking as strategy for cultivation: A case of improving COBOL software maintainability". *29th Information Systems Research in Scandinavia (IRIS 29): "Paradigms Politics Paradoxes"*, 12-15 August, 2006, Helsingør, Denmark.
- Øgland, P. (2007a). "Improving Research Methodology as a part of doing Software Process Improvement", *30th Information Systems Research in Scandinavia (IRIS 30): "Models, Methods and Messages"*, 11-14 August, 2007, Tampere, Finland.
- Øgland, P. (2007b). "Designing quality management systems with minimal management commitment", *Systemist*, vol. 29, no. 3, 101-112.