The Role of Benchmarking in Information System Development

M. Daneva
Institut für Wirtschaftsinformatik
at the University of the Saarland
Altenkesseler str. 17/B2, D-66 115 Saarbrücken, GERMANY
E-mail: maya@iwise1.iwi-uni.sb

Abstract. This paper highlights the importance of benchmarking for fitness and enhancement of quality and productivity in IS development life cycle. We introduce a pragmatic approach to IS improvement based on continuos learning from best-in-class achievements. We also report a benchmarking case study that illustrates the application of the method during the requirements analysis and the development of requirements specification.

1 Introduction

Controlling quality and productivity is a critical success factor for today's software projects [13]. The importance of IS controlling increases when the projects deal with the development or the customization of business/engineering applications, and are essential parts of a business process reengineering initiative. To get some insight in IS-controlling, software engineering people from the industry, the government and the academia have developed numerous measurement approaches and assessment frameworks that should serve as a basis for enterprise-wide metrics programs capable of guiding quality improvement efforts. In a lot of cases, these schemes are too complex to use and difficult to learn; their application has a significant duration and require costly data collection procedures and expensive IS-staff training. Unfortunately, it is very easy to specify an improvement conceptual model that does not work for most practical purposes [14].

The main objective of this work is to advise the IS-development community of how to benefit more from software metrics technology in terms of efficiency and effectiveness. We deliver a pragmatic tool to IS organizations together with practical guidelines how and when to use it.

The remainder of this paper is organized as follows: Section 2 presents how the concept of benchmarking fits with IS-improvement. Section 3 overviews our benchmarking method. Section 4 provides some guidelines for conducting benchmarking studies easily. Section 5 reports a comparative study on IS requirements definitions.

2 Benchmarking: a feasible path to IS-enhancement

Competency and knowledge in controlling IS productivity are considered as a key strategy for dominating the global information technology market in the next century [13]. Surprisingly, recent research on the deployment of quality control mechanisms

revealed that the actual use of controlling procedures is generally poor and lower than organizations claim [10]. It also is surprising that the IS-practitioners and academicians have devoted much efforts to develop new measurement and improvement frameworks, and very few works have dealt with the effectiveness and efficiency of these mechanisms [10]. Although the lack of reliable quantitative information about the usefulness of the current measurement and assessment methods, the existing experience helps us identify some serious drawbacks of these improvement schemes:

1. Quality programs based on Capability Maturity Model [15], Bootstrap method, or the SPICE Standard are too expensive and implementationally difficult [3, 14]. Getting organization's staff involved in the assessment team and training them in assessment technique proved too costly and disruptive for small and middle-sized enterprises.
2. The assessment models completely disregard company's specific improvement objectives [18]. Major efforts are required to customize these frameworks to the organization's circumstances and needs. As the size of the IS-organization shrinks, the assessment methodology becomes increasingly informal. In such situations the existing models fail in formulating improvement advice which maximizes the returns on investments for small companies.
3. The current assessment practices mainly address the question *where* to improve, but not *how* and *how much*. As Zultner writes [20], they are extremely weak on what to do after the assessment. The software metrics technology itself does not add value to a company; its use makes sense only when this leads to implemented improvement actions.

To counteract these difficulties, we propose an approach to IS-enhancement based on the concept of software benchmarking [6]. This is a process of continually searching for the best software experience, indicating software standards to be achieved in day-to-day company's operations, adopting or adapting the best-in-class practices, and implementing them to become the best of the best. We believe, the above three points referring to the current assessment methods can be handled by means of IS benchmarking in the following manner: First, our intention is to provide both large companies and small IS organization with an instrument for deriving comprehensive and effective software metrics programes that contribute significantly to successful improvement initiatives. Second, our approach is context-based what means we regard the fact that software practitioners usually set their own quality priorities according to their own development environments [10]. Moreover, IS-benchmarking is aimed at giving starting point to analyze how well a straight transplant of somebody else's solution fits with the own organization. This ensures the holistic view to the questions *where*, *how* and *how much* to improve.

3 An overview of the approach

3.1 Preliminaries

The present approach is based on the idea that a software organization could and should learn about its own technological and marketing opportunities by learning

about other's similar operations and processes. We do not see benchmarking as a competitive study, „number crunching", spying or stealing, but organizational learning. This hypothesis is directly derived from our previous work on the problem of software quality ranking [5] and software benchmark design and use [6]. We have developed a holistic view of software benchmarking, whose focus was on the key steps of the benchmarking process and how they are to be performed. We also determined systematic procedure for software controlling together with a generic benchmarking model explaining the software enterprise items in benchmarking terms [4].

The approach we present here, is anchored on management by metrics [7], which uses high-level quantitative indicators for the manager to control software projects and products. We also follow the AMI tradition [1] and Basili's framework [2] to goal-driven derivation of software metrics, whose focus on goals leads to focus on what actions can be taken.

3.2 The benchmarking process

This section summarizes our benchmarking method. The approach includes the following steps:

1. Clarify the goal. Formulate the software bottleneck and identify decisions which would be made on the base of IS-benchmarking results. Derive a *benchmarking goal* that refers to the specific aspects of company's competitive agenda: increasing customer satisfaction, shortening time-to-market, improving service quality.
2. Operationalize the benchmarking goal. Identify interest groups, i.e. the IS-staff-members who are interested the most in benchmarking information. Express the benchmarking goal in a hierarchy of measurement goals. These help us determine that the benchmarks reflects what they are presumed to measure. Begin with an analysis of what information the benchmarking study has to reveal, e.g. current productivity level, current schedule ranges, probabilities of achieving significant recognition (European Quality Award, ISO 9000 Certification, CMM Level 3, 4 and 5 [15]). This analysis delivers a top level measurement goal that is further broken down into a set of primitive measurement goals which are easily expresses by software attributes.
3. Select competitors. These are software objects from one of the following groups: (i) w*ork products* - any artifact, produced as a part of defining, maintaining, or using software process, which may or may not be intended for delivering to the customers or end users [15], (ii) *processes* - any means by which people, procedures, methods, equipment, and tools are integrated to produce a desired end results, and (iii) *resources:* i.e. teams, hardware, CASE tools, etc. The software objects candidates for benchmarking should be prioritized by experts or by using some heuristic procedures or formal decision making techniques.

   Establish best-in-class object which one can learn from. This can be either an object which embodies somebody else's practice recognized as a best-in-class one, or a preliminary defined model whose values are numerical quality or productivity thresholds.

4. Specify IS benchmarks. Define two major benchmark components - the domain specification and the measurement (or assessment) specification. The first one describes completely the IS-quality or productivity aspects the benchmark should deal with and the second specification establishes the relevant IS-characteristics to be measured or assessed.
5. Collect benchmarking data. Implement the measurement specification by constructing a set of measured functions. Carry out the measurement, i.e., assign actual values to characteristics involved in benchmarking. The benchmarking data can be classified as follows:
   - *expert judgments.*
   - *elementary metrics:* these are numbers derived from facts [1]. For example, marketing indicators, such as market share, sales volume, or engineering indicators, such as lines of code, elapsed time, cyclomatic complexity.
   - *assessments:* these can be done using either predefined calculation rules (Halstead's software science, Function Point Analysis [13], etc), or questionnaires in the software enterprise (CMM, BOOTSTRAP).
   - *profile charts:* these are sets of expert judgments and assessments relevant to a certain software characteristics which can not be quantified directly (for example, maintenance).
   - *competitive figures:* these are sets of software characteristics and their competitive levels revealing the position of the company with respect to its competitors.
6. Compare software objects. Given objects' measurements/assessments, determine some superiority and inferiority criteria that help the benchmarker rank objects according to their capability to support the benchmarking goal. The ranking points out where a company is in both developmental and quality terms.
7. Decide on learning strategy. Project future quality and productivity levels. Check to what extent it is possible to implement a solution already used in another IS-organization. Prioritize strategies how to implement the best-in class experience in home. Make decisions on course of improvement actions.
8. Develop and implement an action plan. Establish functional improvement goals and identify steps that systematically lead to goal achievement.
9. Re-evaluate. Check how well the best-in-class practice is incorporated into home. Continue through conducting a new benchmarking study.

It is worth stressing that IS-benchmarking is rarely a strictly linear process. It is possible to re-define benchmarking goal after defining precisely measurement goals, or after identifying the best-in-class achievements.

3.3 The instrumentation

An IS-benchmarking study has to be aware of the software objects' characteristics submitted as benchmarks as well as the possible ways the benchmarks are quantified and interpreted. To provide the benchmarker with guidance to successful benchmarking studies we propose an instrumented scheme that consists of a benchmarks model and operations permitted on it. The scheme ensures a common understanding of benchmarking and provides a mechanism for structuring the

information about benchmarking practices. The scheme is based on the ISO Standards as well as the STEP standard [19] that suggest to organize software characteristics according to an hierarchical quality model flexible enough to be customized to the needs of any benchmarker. We refine this idea by developing three level scheme for IS-benchmarks. It consists of categories, indicators and metrics. The categories provide a high level view to the types of benchmark measurements performed on software objects. The indicators are measurable aspects of the quality/productivity for any given category. The indicators are quantified by assessing one or more facts about an software object. Each fact to be assessed is called a metric. The benchmarker can investigate objects at any of these levels.

Generally, a benchmarking procedure defines six operations over a quality model:

- derivation: given a set of categories, generate a set of indicators, and a set of metrics to be considered, trying to minimize the measurement costs.
- integration: given a set of measurements, generate relevant set of indicators and categories, and construct objects' benchmarking profiles.
- auditing: given a preliminary established model recognized as a best-in-class one, compare a set of target objects against the best-in-class assessments.
- visioning: given a benchmarking goal, rank target objects with respect to their capability of supporting the predefined objective.
- interpreting: given a model benchmarking profile, map objects' values (e.g. the function point number is 107) onto merit values (e.g. 1.5 on a scale from 0.0 (low complexity) to 10.0 (highest complexity)).
- reasoning: given a model benchmarking profile, explain why the measurements produced this profile.

4 Benchmarking case study

This section summarizes some results of a benchmarking initiative undertaken at the University of the Saarland. Our benchmarking approach was applied to the life cycle phase of requirement analysis. The objective with this study was twofold: (1) to control the process of requirement gathering, and to ensure the quality of IS-requirement specifications in efficient and effective way, and (2) to anticipate pitfalls in integration of submodels and/or customization of IS reference models.

The benchmarked IS-requirement definitions were represented in form of business process models by using the specification language of event-driven process chains (EPC) [17]. Any EPC specifies details of the modes of operation which a system can be in, as well as the interface between the system and entities beyond the scope of the system. The formalism comprises three basic elements: functions, events, and logical operators (*and, or*, and *exclusive or*). The functions are considered as time-consuming activities that create or modify objects. The events are both conditions for the execution of functions, and results. The events refer to points in time and are defined as data items that should be available at the start and at the end of functions' execution. In the EPC-schemes, the functions are represented by rounded boxes and the events as hexagons. An example of an EPC-scheme is given in Fig. 1.

We operationalized the benchmarks model so as to use it in benchmarking studies dealing with the complexity. A set of appropriate complexity indicators has been defined as well. For any indicator we explained how it helps us improve the IS-requirements specifications and how it would be more difficult to make improvements without the indicator. Finally, the indicators were specified by following AMI metrics description guidelines [1].

In the next section, we review some complexity indicators. We suggest to present the benchmark definitions together with an illustrative example.

4.1 An illustrative example

To show how metrics data are collected we use the EPC in Fig. 1 which depicts the high level definition of the process „Bid preparation" developed by the ESPRIT BIDPREP Consortium [11]. Due to space limitations we present a part of benchmark specifications only (namely, the reference object and the integration rules).

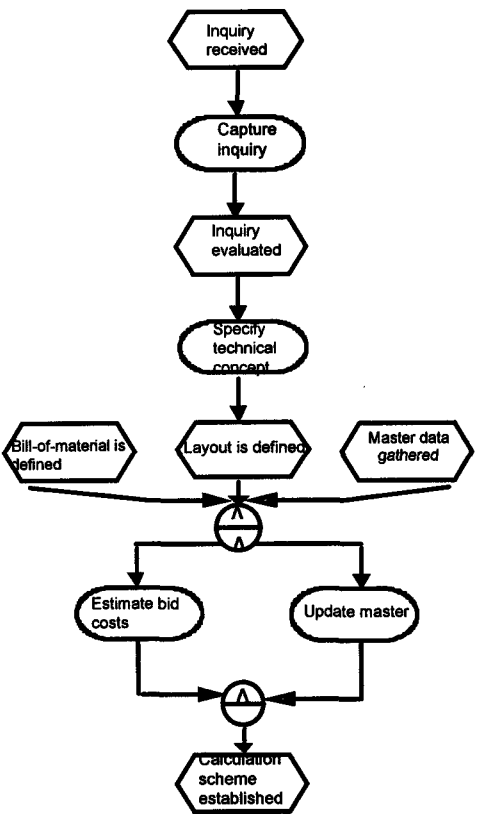

Figure 1. A part of BIDPREP Specification.

4.2 Function Cohesion (FC)

Reference Object: Function

Integration Rules: $FC = (FC_{inp} + FC_{out})/2$

where FC_{inp} und FC_{out} are the input and output function cohesion measures. These are defined as follows:
$FC_{inp}(0) = k_0 N_0$
$FC_{inp}(i) = k_i.(FC_{inp}(i-1) + N_i)$ $i=1,..,n$
$FC_{out}(0) = k_0 N_0$
$Fc_{out}(i) = k_j.(FC_{out}(i-1) + N_j)$ $i=1,..,l$
where i and j are levels of nesting, k_i and k_j are weights, and N_i and N_j are the numbers of the input and output events with respect to the function under consideration.

The numbers k_i and k_j are defined by modifying Rechenberg's weighting scheme [10]:

Type of Event	Weight	Type of Event	Weight
AND-Event	3	OR-Event	2
XOR-Event	3	Single Events	1

Table 1.

Examples:
FC (Capture inquiry) = (1+1)/2 = 1
FC (Estimate bid costs) = (3x3 + 1x1)/2 = 5

4.3 Data Item Cohesion (DIC)

Reference Object: Event

Integration Rules: $DIC = DIC_{inp} + DIC_{out} + c.NF$

where DIC_{inp} and DIC_{out} are the input and output data item cohesion cohesion measures, c is an event cohesion class, and NF is the number of functions affected by the event under consideration. These are defined as follows:
$DIC_{inp}(0) = k_0 NE_0$
$DIC_{inp}(i) = k_i.(DIC_{inp}(i-1) + NE_i)$ $i=1,..,n$
$DIC_{out}(0) = k_0 NE_0$
$DIC_{out}(i) = k_j.(DIC_{out}(i-1) + NE_j)$ $j=1,..,l$
where i and j are levels of nesting, k_i and k_j have the above meaning, NE_i is the number the of co-items, with which a data item acts together as an input to a function, and NE_j is the number the of co-items, with which a data item acts together as an output from a function.

The cohesion class c defines the role of an event. It characterizes the decision power of the event and reflects the way in which the event affects the functions related to it. We established the following rules for defining the cohesion class of an event:

1. An event has a cohesion class 1 when it is produced by only one function and is an input to only one function.
2. An event has a cohesion class 2 when it is produced by more than one functions but is an output to only one function.
3. An event has a cohesion class 3 when it is produced by only one function and is an input to more than one functions.
4. An event has a cohesion class 4 when it is produced by more than one functions and is an output to more than one functions.

Example:
DIC (Inquiry is evaluated) = 0+0+2x1= 2
DIC (Layout is defined) = 0 + 2x3 +3x3 = 6 + 9 = 15

4.4 Validation of complexity benchmarks

To investigate the expressive power and the adequacy of the developed benchmarks, we followed Fenton's guidelines [8] for software metrics validation. The operationalized benchmarks models were applied to representative sample of EPC-schemas produced. The results were compared with expert evaluations referring to the same models. We refined benchmark calculation rules every time when our method did not come to the same conclusion as the experts. Finally, the conformity between expected and effective benchmark values has been checked through estimating Kendal's concordance coefficient. This study pointed out the strong correlation between expert opinions and the quantified indicators.

We have also studied the statistical independence among the benchmarks. The benchmark values exhibited a low correlation, what pointed out that they characterize distinct aspects of business process models.

4.5 Benchmarking profiles

To illustrate how our solution strategy works, we report here a summary of a comparative study on the model complexity. We evaluated and benchmarked a model produced by IDS Prof Scheer GmbH, Saarbrücken, Germany, for the process „Human resource administration" at the enterprise X, an electric company in Eastern Europe, with an optimized model that represents the same process. The latter was recognized as a best-in-class one. The considered benchmarks are: Function Cohesion, Data Item Cohesion and Cohesion of Logical Connectors. Figure 2. represents the average benchmark values. The complexity evaluations of the model of company X exceed those of the best-in-class model.

To define corrective actions, the modellers needed a roadmap and guideline where to begin from. Therefore, we focused our further investigation on each particular benchmark. To identify which objects contribute the most to the average benchmark

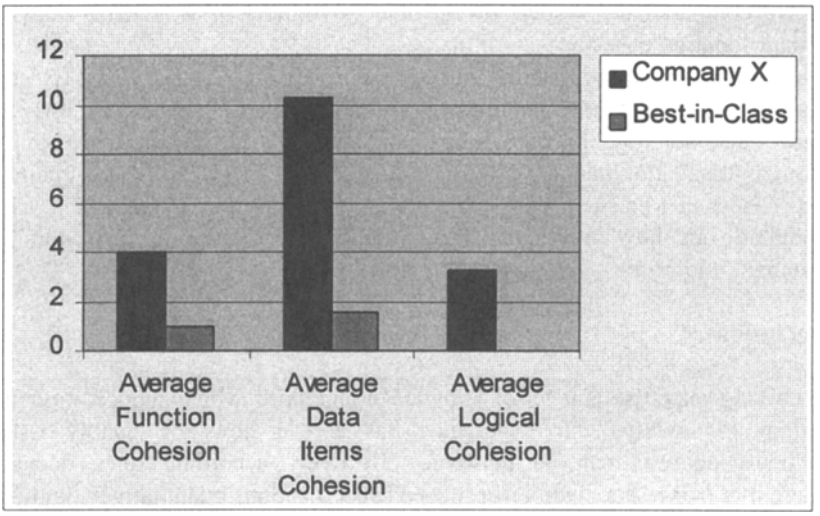

Figure 2. Benchmarking profiles.

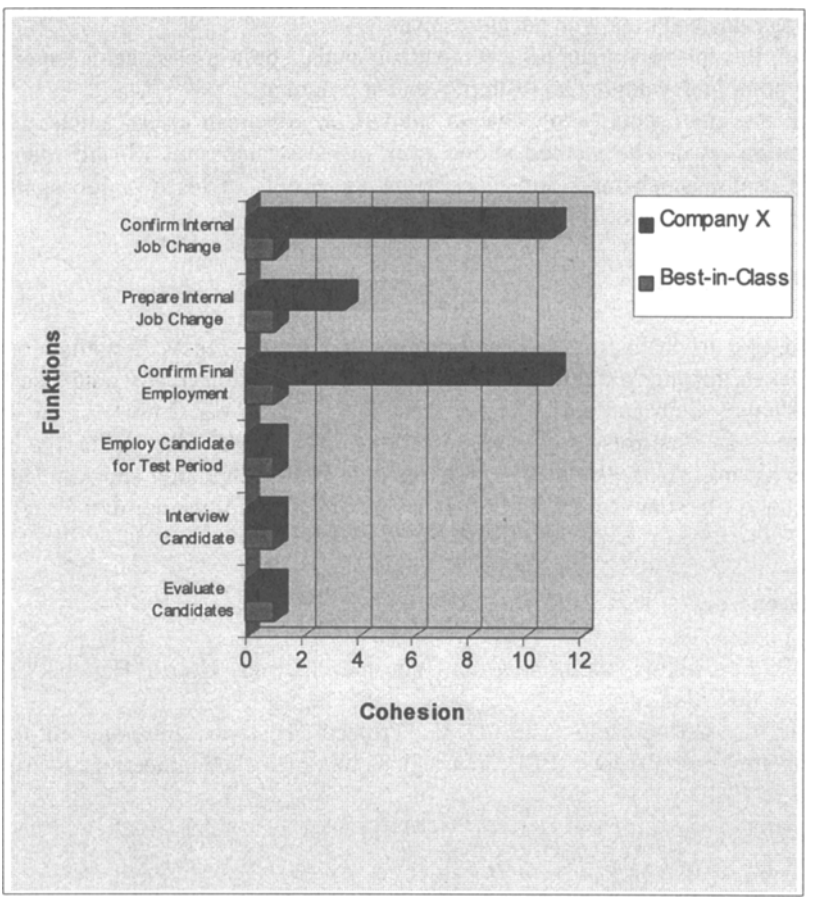

Figure 3. A comparison on function-by-function basis.

values we compared the models on an object-by-object basis. Let us analyze the benchmark function cohesion.

Figure 3 represents a function-by-function comparison. The functions, which both models have in common are compared with respect to their FC values. As it is evident from the data, the high average function cohesion is attributed to the functions „Confirm internal job change", „Confirm final employment", „Prepare internal job change". These can be submitted to the modellers for further enhancement. Thus, it was pointed out how a benchmarking study can deliver a basis for model improvement initiatives.

5 Conclusions

Benchmarking expertise is a set of knowledge and skills which support modellers in controlling IS quality. Our experience has shown how the quality issues in requirements analysis can be mastered effectively according to a documented procedure that makes the quality of produced specifications quantitatively known. Our approach is very different from the existing improvement schemes in the following three ways: First, its focus is on identifying after-assessment actions. The main point in IS-benchmarking is how to add value to the IS-organization by using software metrics technology, instead of how to get assessment.

Second, the approach places the comparison against best-in-class achievements as a starting point in developing an IS-improvement program.

Third, the approach is applicable to support improvement efforts initiated at any organization level. The method allows even the least ambitious IS-staff member to develop useful benchmarks and to compare the results of his own job against the excellent ones.

6 Acknowledgements

I would like to thank the colleagues from the Institut für Wirtschaftsinformatik, Saarbrücken, for many stimulating conversations on the subject. I would also like to thank Thomas Geib and Karl Wagner from IDS Prof Scheer GmbH for the fruitful discussions on enterprise-model-benchmarking. My thanks also go to the friends Mathias Krömker from Bremen Institute for Industrial Technology and Applied Work Science, and Christian Rose from Software Union GmbH, for the use of their reference models.

7 References

1. AMI: Application of Metrics in Industry, Metrics Users' Handbook, AMI Consortium, 1992.
2. Basili,V., H.Rombach, The TIME Project: Towards Improvement-oriented Software Environments, IEEE Transactions on Software Engineering, 14(6), 1988, pp 758-773.

3. Brodman, J.G., D.L.Johnson, What Small Business and Small Organizations Say About the CMM, Proc. of the 17th International Conference on Software Engineering, Sorento, 1994, pp. 331-340.
4. Daneva, M., Benchmarking Practice for Software Quality Achievement, Proc. of the 5th International Conference on Software Quality, Austin, 1995, pp. 443-457.
5. Daneva, M., Knowledge-based Approach to Software Marketing Modelling and Support, Ph.D. Dissertation, BAS, Institute of Mathematics, 1994.
6. Daneva, M., Software Benchmark Design and Use,.Reengineering the Enterprise, Galway, Ed. J. Brown and D. O'Sullivan, Chapman and Hall, New York, 1995, pp. 20-29.
7. ESPRIT Project 2151: „SCOPE - Technology for Evaluation and Certification of Software Product Quality", Project Brochure, November, 1992.
8. Fenton, N., Software Metrics: A Rigorous Approach, Chapman&Hall, 1991.
9. Fenton, N.E., S. Pfleeger, R. Glass, Science and Substance: a Chalange to Software Engineers, IEEE Software, July, 1994, pp. 86-95.
10. Hall, T., N. Fenton, Software Practitiones and Software Quality, Proc. of the 5th International Conference on Software Quality, Austin, 1995, 313-324.
11. Hirsch, B.E, K.-D., Thoben, M. Krömker, A. Wickner, Benchmarking in BidPreparation for Capital Goods, Proc. of IFIP WG 5.7 Workshop on Benchmarking - Theory and Practice, Trondheim, 1994.
12. Jones, C., Global Software Quality in 1995, Proc. of the 5th International Conference on Software Quality, Austin, 1995, pp. 283-290.
13. Jones, C., Patterns of Software Systems Failure and Success, International. Thomson Computer Press, Boston, 1996.
14. Paulk, M.C., A Perspective on the Issues Facing SPICE, Proc. of 5th International Conference on Software Quality, Austin, 1995, pp. 416-425.
15. Paulk. M., C.Weber, B. Curtis, M.B.Chrissis, The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley Publ Co., Redings, MA, 1995.
16. Rechenberg, P., Ein neues Mass für die Softwaretechnische Komplexität von Programmen, Informatik Forschung und Entwicklung, 1986, pp. 26-37.
17. Scheer, A.-W., Wirtschaftsinformatik, Referenzmodelle für industrielle Geschäftsprozesse, 6.Ed., Springer-Verlag, Berlin et all, 1995.
18. Strigel, W., Assessment in Small Software Companies, Technical Report, Software Productivity Centre, Vancouver, 1995.
19. Tyler, J., Models, Managing Modekls, Quality Model: an Example of Quality Management, U.S. Department of Commerce, National Institute of Standards and Technology, NISTIR 4738, December, 1991.
20. Zultner, R., The Deming Way to Software Development, Proc. of the 5th International Conference on Software Quality, Austin, 1995, pp. 457-470.