ISSRE '98

The Ninth International Symposium on

SOFTWARE

RELIABILITY

ENGINEERING

Paderborn, Germany,
November 4-7, 1998

IEEE
Computer
Society

PERSONNEL IS-EFFORT COLLECTION IN INDUSTRIAL ENVIRONMENTS

Thorsten Spitta[1]

University of Bielefeld / Germany

Abstract: The knowledge of development and maintenance effort is a necessary precondition for all types of software management, IS-controlling, guidance of quality and effort estimation. Data collecting systems seem to be widely used, but the data is said to be unsafe and firm-specific. This paper presents a concept for the collection of personnel effort and for events (e.g. faults). The revised informal specification of this paper is based on experience with such a system used by 40 developers in five locations for many years. Design and practical use of a system based on that concept allow the collection of statistically valid effort data. This would only be possible if developers are sure to be not supervised by their data, and if they don't need to decide upon classifications during data-entry time. Examples show applications of the data for project management, quality control and IS resource management reporting from 6« years with overall 170 PYs.

1 Introduction

A great deal of software engineering techniques is based on the knowledge of effort data: Effort estimation, cost allocation, project management etc. *'The Mythical Man Month'* [3], *'Software Engineering Economics'* [1] are two examples of fundamental books which intensively deal with the amounts and distribution of human effort in software development and maintenance. But until now there has not been published a generalized concept of how to collect the effort.

This paper describes such a concept. It has been empirically developed since 1987 in a medium sized company that had five locations developing and three locations maintaining software with about 40 developers. This means 30,000 to 50,000 data entries per year for those 40 persons, as demonstrated later. The data to be shown here is based on the experience of the first 6« years. At that time the database contained 271,077 hours of effort, distributed on 22 large, 248 small projects and 461 maintenance activities with accounted effort and 106 canceled requirements with no effort (see details in table 4, section 5). Including service, the database contained 169.4 PY (person years)[2] of effort.

As realized by recent empirical investigations of the author [12] there are still few organizations collecting such data. The main arguments against effort collection are the *validity* of the data and the *effort* of its collection. In detail:

- The lines between development, maintenance and service are not sharp enough. There would be false attachments of data. This is the *classification problem*.
- Data has to be collected by programmers. It does not only show effort but also human behavior; therefore it may be falsified. You have to motivate people to collect *correct* data. This will be discussed as the *motivation problem*.

[1] Prof. Dr.-Ing. Thorsten Spitta, University of Bielefeld, Department of Economics, D-33501 Bielefeld, Germany, Fax: 0049/521/1068013. Email: ThSpitta@wiwi.uni-bielefeld.de.

[2] 1 PY = 1,600 [h] with 200 working days of 8 [h].

- The effort of collecting data is too high. It consumes a relatively large amount of programmer's productive time. This is called the *effort problem*.

The following paper first discusses general requirements for the data to be collected (section 2). Section 3 and 4 discuss the data to be collected in detail and the operation conditions of a data collection system with its social implications. They are not sufficiently respected by the pure technical design. The concept in section 3 is a slight revision of the original one, published when starting the system's application (see [11, ch. 12]). The last section demonstrates the multipurpose use of the model by examples of original data. They show that effort collection is not only helpful for cost allocation and long-range IS-controlling but also in the short run for project management.

2 Goals and Requirements

An effort collection system has to support multidimensional goals to justify the effort of the collection process. It must support the information resource management, which contains economic and technical elements. The goals in detail are managerial overviews about:

- Capital invested in software resources [7], [9]
- Structure, cost and trends of maintenance effort [5], [2]
- Project management, especially controlling external developers [8]
- Cost allocation of user department's requirements [6]
- Plan/expenditure variances and other topics of IS-controlling [4], [10].

IS-controlling is not focused on the fiscal year. Data has to be observed for much longer periods. Therefore the main classifications of effort data must be stable for a long time. The system has to store *events* and *times* (effort) of all systems over all effort classes. Specific requirements have to serve four purposes:

(1) Project management

- State of each project at any time
- Effort of each project per period
- Pending projects (they have no effort in a period)
- Fault-types in systems during introduction phase, etc.

(2) Maintenance and evolution management

- Number and distribution of user requirements in different departments
- Effort distribution of completed maintenance cases
- Effort of purchased systems, etc.

(3) Cost accounting

- Effort and cost per cost center
- External vs. internal personal effort, etc.

(4) Support

- PC-support effort per department
- Faults in network and PC-software, etc.

3 Data to be collected

In this section the general classifications for an implementation are discussed which will appear there as *object types*, *attributes* or *attribute domains*[3].

We assume a dynamic life cycle model for software which is the generally accepted state of the art (Figure 1). The implications of that model for data to be collected are:

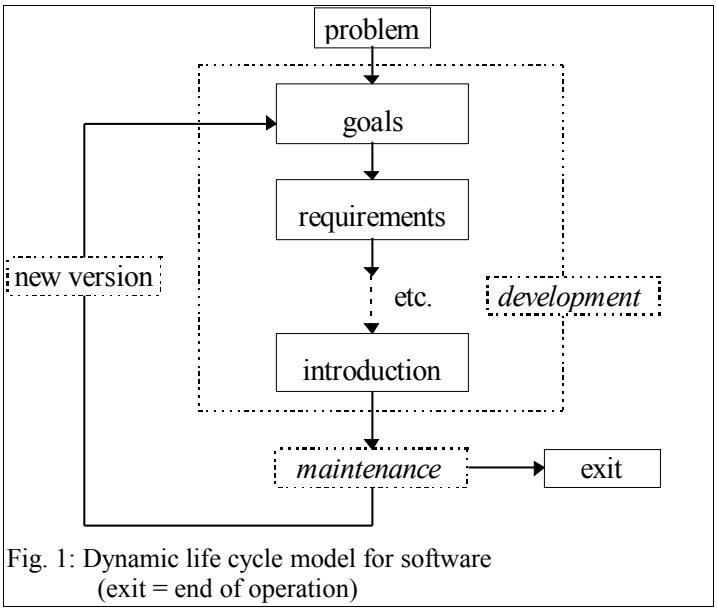

Fig. 1: Dynamic life cycle model for software
(exit = end of operation)

- It describes the life cycle of a *system* or *part of a system* which has two states: *development* and *maintenance*.
- Development is a sequence of *versions* (or *releases*). The development of a version is followed by a period of operation with maintenance activities. During operation time new requirements, based on the next version, are collected. The process to produce a first or new version is a *project*, which must be attached to a *system*. While two versions always follow each other strictly sequentially, it might be useful to divide a project into *part-projects* which can be processed at the same time.
- The effort to be collected is *working hours*. In a project they can be attached to *phases* which generally are (not in any detail!) sequential. There are two quite different attachments to phases: phases of a whole project and phases of an activity. E.g. during the project-phase *introduction* it might be necessary to *specify* some additional functions. Both types of phases should be collected for project management purposes. The first type of phase may only be booked by a project leader or manager while the developer decides upon the second. This avoids classification errors in the area of phases.
- During phases there are types of methodological activities (functional/data design; coding, testing etc.) which are performed simultaneously. The coding of these *activity types* would cause classification errors in the data (and raise the collecting effort!). Although technically possible, activity types should not be coded. Coding of too much detail bothers people with unsolvable classification problems, especially when coding effort, and enlarges errors instead of improving precision.
- Details are written in free text, which contains important information for project management, but not for classification.
- During maintenance, phases must not be coded.
- A project or a system is attached to a *cost center*. Because of today's dynamics of enterprise structures, the attachment to a *system* is more stable than to a cost center. An application system classification should not be firm-specific, but generalized within the subject of the specific type of software dealt with. Table 4 in section 5 is an example for an industrial environment. This is not only important for the comparison of data between firms, but also

[3] See [13], where the exact data model is presented.

for stability against changes in firms' structures. A two-level hierarchy of such a classification is necessary and sufficient. A more than two-level hierarchy makes the data difficult to handle.

In addition to the life cycle of software with three effort classes (development, maintenance, faults), there is a fourth effort class which is more dedicated to (personal) hardware than software. This effort class is *support* which naturally has no phases and at first no cost centers (but surely later!). All effort classes are *events* until something is done with that event (correcting a fault, working on a project, extending purchased software). Events are booked by managers not by developers. Events have small frequencies and may be checked in every case for classification errors. Managers can be expected to be interested in correct classifications because they are observed by users. This is the solution for the motivation problem with events.

Table 1: Collection screen for a day-report after input before submission of the data

```
                        EFFORT-COLLECTION short
MeierC                                                        05/22/89

Eff   Sv    Cost  Phas   Eff-      Activity (short text)
Cls   No    Cent   No    Time _____________________________________.
 f    71    ___    _     1.5_      fault search double book sales___
 f    71    ___    _     0.5_      sort fault deliv. note___________
 m    70    ___    _     1___      gen. probl._____________________
 s    72    812    _     1___      div. probl. consignment note_____
 p    264   ___    3     0.5_      discussion mobile order entry____
 m    315   ___    _     0.5_      div. probl. inventory____________
 m    331   ___    _     2.5_      tour-planning/customer data______
 _    ___   ___    _     ____ _____________________________________
 _    ___   ___    _     ____ _____________________________________

___  ------------------------------------------------------------
 PF1= help       PF3= back        F12= end              Return= NEXT
```

Legend: **bold** : Default data shown by the system. After '**Return'** other default data (*cost center* and *phase number*) are automatically filled in. The redundant input of *effort class* serves as a consistency check if the correct *service number* is used.

Table 1 shows the online form of the normal case of effort collection for one working day with authentic data. The reader can see that very few items have to be collected for one accounting position. This prototypical design is one part of solving the motivation and also the effort problem. Programmers want the data entry to be efficient. Until now (1998) it *cannot* be recommended to collect effort via a graphical user interface because this is ineffective.

It is self-evident that classifications of new staff have to be watched and that each manager discusses faulty classifications with his people.

The classification problem focuses on the point that

- as few classifications as possible have to be made by the staff itself while collecting data.

4 Operation conditions

Because one main problem of data collection systems is validity, everything that can be formally checked should be done so by implemented integrity constraints. Errors within the nor-

mal margins of error only cause a statistical variance. Because there are about 1000 accounted records per PY, the law of the large numbers holds for that type of data. Merely individual differences compensate each other, if more than five developers are booking.

The remaining point to be discussed is the *motivation problem*. What can be done to avoid *systematic* falsification of the mass data (effort) by the developers?

It is evidently not useful and also virtually impossible to supervise each developer. The only way to avoid falsification is to insure that no developer should have any motive to fake the data. If supervision is not a solution then only a trusting working atmosphere can help. There are four conditions to produce this climate that makes falsifications improbable. When starting our system, checks were made which showed the effectiveness of these conditions.

1. **Personal responsibility**. Each person books his/her own data *online*. No other person can create or change personal data. The database is locked against free SQL-updates and can only be manipulated by the booking programs.
2. **Transparency**. Everyone of the staff booking is allowed to read (only online!) all bookings. This promotes an open atmosphere between developers, managers and teams. Not only staff but also middle management collects his/her effort.
3. **No supervision**. Evaluation of programmers should by no means be conducted in this context, neither their effectiveness nor their errors. This applies to employees. Because IS-staff knows that this is technically possible this is a very important point in using an effort database. If the staff realizes that persons are measured by the data (e.g. by checking the productive times per PY) they might fake it. By contrast, the system should be used to *protect* the staff's working places against questions like: *'What are these many expensive people really doing?'* In this context plan-/expenditure comparisons might be mentioned as problematic with respect to the correctness of the expenditure data. If people are measured by the variances, they will try to reach good values.
4. **Instruction and Information.** New staff and new external developers have to be informed about the goals and the rules of the system. This is best done with examples of reports as shown in section 5 below. All booking persons have to be informed what is done with their data.

The remaining two points are useful for the correctness and usability of the data too, but do not influence motivation as strongly as the points above.

5. **Completeness**. A person should either collect data for *all* productive work or no data at all. Developers, support staff and project managers belong to the first class; operators, network specialists etc. to the second. This avoids a shifting of effort between activities or effort classes or the omission of effort. It is a means to solve the *classification problem*.
6. **Actuality**. The latest day to collect data is the end of a week. Only recent data are useful for project management and only they are not based on human memory.

The motivation problem summarizes that

- a minimum of effort data items should be required of developers and middle management
- only a trustful atmosphere will avoid *systematic* falsification.

5 Examples of data from 6 years

The following examples of original data show that an effort collection system is useful for more than one purpose, that is *project management* (steering and reporting), *quality* and *reliability engineering* (maintenance effort and system's evolution) and *information resource management*. The multidimensional use of the data is the solution of the effort problem.

5.1 Project management

Table 2 shows three days of effort for a project. The *project* is in phase 5 (implementation), while there are several *activities* in a phase < 5 (3= specification). Only one of the four developers (D) has worked on one day (2/4/92) exclusively for this project (light shadow). The report gives the project leader hints - not more - that the additional specifications should be checked. Are these activities a) necessary and b) agreed upon, or are they c) 'subversive'? A second glance shows that developer B obviously works on extensions, while the system is in test (dark shadow).

Table 2: Booking of a project for several days

Project: P475 ApplSyst: L.LGB CostCtr: 800

Accounting from: 02/04/92 - 02/06/92

Date	User ID	Effort [h]	Phas No	Activity
02/04/92	A	1.0	3	discussion actitivities with PL (=Proj. leader)
	B	1.0	3	disc. activities LGB 1.HY
	B	2.5	5	programmg. automatic stock-change
	C	1.0	5	maint. activity plan
	C	5.0	5	date executive board
	D	1.5	5	LGB2942P test listprint
	D	3.0	5	LGB2105P (online) worked over
	D	3.0	5	actualization list of listprints
02/05/92	B	1.0	5	extend. stock places
	B	3.5	5	new progr. LG-withdraw-voucher
	D	3.0	5	work over LGB2105P (online) and test
02/06/92	A	0.4	5	change prog QBE750A
	B	1.1	3	extending autom. stock-change
	B	1.5	5	extend print DelivNote foreign Order-No
	D	3.0	5	tested LGB2105P and put in TEST
	D	1.0	5	tested LGB2927P: charge missing

Figure 2 shows aggregated effort for external staff in large projects over six quarters. The view of a financial accounting department would be focused on the cost structure: *'External costs are relatively constant (on a high level)'*. From the project management view the situation is quite different: The sales&delivery projects are just before the introduction phase. A rise in the last quarter (integration test) is normal. The production project has an ideal course: It has been in the introduction phase since I/91. The effort is continuously decreasing. However, the logistic project shows an alarmingly ascending course. The curve seems to be a demonstration of Brook's 'law': *Adding people to a late project makes it later.*

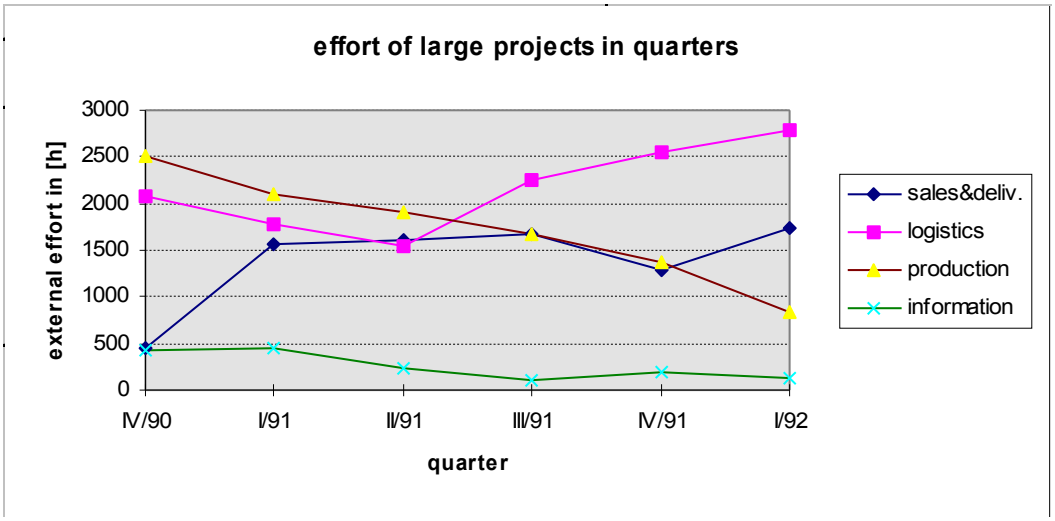

Fig. 2: External personal effort over 6 quarters for large projects

The second field of the database is quality and reliability engineering.

5.2 Quality and reliability engineering

Maintenance effort can tell more about quality than development effort. Figure 3 shows the evolution of some systems and their accumulated maintenance efforts.

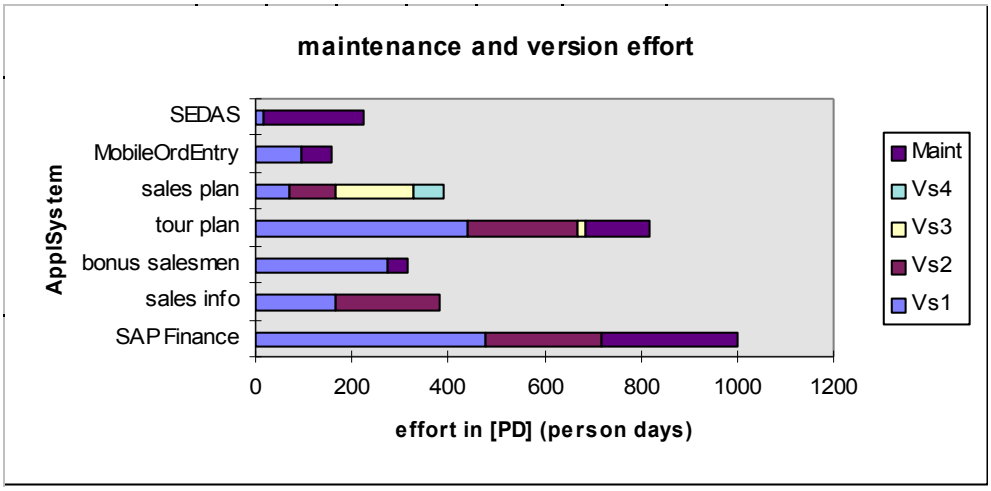

Fig. 3: Maintenance effort and effort of evolutionary versions of application systems

In order to interpret each effort value correctly it might be necessary to know the time distribution of the projects and the organizational context of the data. There are only a few interpretations of three of the systems in figure 3:

SEDAS is a preliminary national German version of the international standard EDIFACT. The system was implemented under high pressure of an important customer in a 'quick and dirty' manner in 1987. The very large maintenance effort proves that there must be quality problems, although the system is a rather simple batch interface. On the other hand, *MobileOrderEntry* was a complex and ambitious project for 140 salesmen, introduced in 1988. The maintenance effort of about 70% of the development effort within 5 years seems to be a sign of good quality. *Sales plan* was evolutionarily developed. The four versions without stable

maintenance can be an indicator for either a very 'creative' user department or a 'creative' developer who follows each technical fashion[4].

5.3 Information management and IS-controlling

The database also allows for a long range view on IS software resources. Table 3 shows a reporting of projects for the sales&delivery department. In this case an attribute *strategicRelevant?* is used to separate large and important from small projects. The reader can see that there is a large ratio of abandoned small projects. The same report for maintenance or faults would show the 'pipeline' of events with PhaseNo = 0.

The list of projects also shows some aspects of strategic information management. In addition to planned projects new problems appear during the analysis of strategic projects, which should be handled as separate part-projects. E.g. the base article data are in a wrong state and have to be renewed (ProjNo 323) and to be re-specified (ProjNo 260), in order to make the strategic project (No 371) a success. On the other hand, there are *reactive* projects like the 'green point' (ProjNo 680) forced by a new bill passed in Europe in 1991. Other reactive projects are forced by important customers.

Table 3: Project status for one user department (sales&delivery)

date: 03/31/1992

IsServ -Nr	Subject	Effort [h]	Phase
	strategic projects		
154	sales info article/salesman	714	+
281	tour planning	1,558	+
340	customer orders/delivery	11,300	5
348	integration old system food sales	6,477	+
579	accounting food sales new	1,259	+
680	*'green point'*	69	3
*** **sales&delivery**		**21,308**	
260	definition article structure	1,777	+
323	restoring article data	1,071	+
371	article/BOM/working schedules	19,031	6
*** **base data**		**21,879**	
	misc. projects		
301	detachment accounting nonfood	65	+
335	*contribution margins*	97	!
355	*shipment-change division B*	158	+
361	gross demand	742	+
439	restoring accounting old system	140	!
518	*change region-/district -key*	18	!
*** **sales&delivery**		**1,220**	
374	integration coding tables	280	+
405	color-bundle 3 pieces	21	!
*** **base data**		**301**	

legend: underlined: additonal ~, *italic*: reactive project

Table 4 shows an aggregation of the complete database for the first 6« years as a distribution of *IsService* events over the application structure. The application system classification was stable over long periods although in this case the firm's organizational structure completely changed twice, also causing a change of the cost center structure. Table 4 shows that already the classification of *IsService* events is a good tool of information and project management, without regarding effort values. Two facts in table 4 will be discussed: The large number of canceled and abandoned activities and the number of actual projects in work in different application areas. The indicator function of such aggregated views for IS-controlling is evident.

Table 4: Total number of IsService events over 6« years

[4] The technical base of the system was a PC. The 4th version was not yet finished at the time of extracting the data.

Number of IsService events in 6,5 years: 01/01/1987 - 06/30/1993													
SvCls	*fault+supp+maint*				*projects*				*total*				Sum
Application	c	!	i.w.	"+"	c	!	i.w.	"+"	c	!	i.w.	"+"	
administration	12	4	6	64	4	4	8	25	16	8	14	89	127
information	5	0	9	48	4	2	5	23	9	2	14	71	96
logistics	7	1	7	48	11	15	19	24	18	16	26	72	132
production	3	1	6	26	6	2	31	19	9	3	37	45	94
sales&deliv.	31	20	19	202	23	14	28	51	54	34	47	253	388
total	58	26	47	388	48	37	91	142	106	63	138	530	**837**

details for IS-controlling:						(last half year)
production	(+)	(-)	"1-6"	"0"	*totl*	
maint.	3		3	3	6	
project	4	3	25	3	31	
total	7	3	28	6		

sales&deliv.	(+)	(-)	"1-6"	"0"	*totl*
maint.	19		3	16	19
project	9	2	8	18	28
total	28	2	11	34	

legend:	c: cancelled, ! : broken
(+):	finished since 01/01/93
(-):	in introduction
"1-6"	in w ork (i.w .)
"0"	registered ('pipeline')

Especially in medium-sized enterprises, there is a culture of very flexible reactions on the market. This causes spontaneous requirements in the IS-area. If this dynamic process is not guided, there could be a waste of IS-capacities and an erosion of strategic projects.

A close look on *activities in work* of the departments *production* and *sales&delivery* (shadowed) shows some interesting details. Production has 25 projects in work. This seems to be a sign of a chaotic project management. It is highly probable that most of these 'projects' will be broken with their effort lost without any benefit. By contrast to that, sales&delivery gives a very disciplined image. 19 maintenance activities and 9 projects have been finished within the last half year. There is also a long pipeline of 19 maintenance and 28 development activities.

The *effort problem* can be answered from our data like this: 1.5 - 2% of the effort is collection effort. Part of this effort is the online collection by the developers themselves. This is, as discussed in section 4, a good investment into the correctness of the data (responsibility and actuality). Moreover, it can be supposed that the effect of transparency and cost responsibility is higher than the expenditures for the collection process.

The effort problem solves like this:

- If effort data is used for several purposes, as shown here, its collection is worth it. If it is used only for financial purposes, the effort is rather high.

6 Conclusion

It was shown how to specify a universal software for data collection which gains valid data that is usable for managerial, technical and financial purposes. The technical design of the system has to be combined with suitable operational conditions. These two parts of a data collecting environment promote a climate of trust for the staff in order to obtain correct data.

With this data many IS management jobs can be done much better, as there are project management, cost accounting, quality engineering and IS-resource management. This justifies the effort of the data collection itself.

To gather expenditure data is a means of building up a 'corporate memory' for IS resources which was claimed by Boehm 17 years ago [1]. This will enable realistic planned data which will have to be discussed elsewhere.

References

[1] Boehm, B.W.: Software Engineering Economics. Prentice Hall, Englewood Cliffs 1981.
[2] Boehm, B.W.; Papaccio, P.N.: Understanding and Controlling Software Costs. IEEE Transactions on Software Engineering 14(1988) 10, 1462-1477.
[3] Brooks, F.P.: The Mythical Man Month, Addison-Wesley, Reading/MA 1975.
[4] Dué, R.T.: Determining Economic Feasibility: Four Cost/Benefit Analysis Methods. Journal of Information Systems & Management 6(1989) 4, 14-19.
[5] Lehmann, M.M.: Programs, Life Cycles and Laws of Software Evolution. IEEE Proceedings 68(1980) 9, 1060-1076.
[6] Nolan, R.L.: Controlling the Costs of Data Service. Harvard Business Review 55(1977) 4, 114-124.
[7] Nolan, R.L.: Managing the Crisis in Data Processing. Harvard Business Review, March-April (1979), 115-126.
[8] Page-Jones, M.: Practical Project Management. Restoring Quality to DP-Projects and Systems. Dorset, New York 1985.
[9] Pfleeger, Sh., L.; Rombach H. D.: Measurement Based Process Improvement. Editorial to: IEEE Software, 11(1994) 4, 9-11.
[10] Segars, A.H.; Grover, V.: Designing Company-wide Information Systems: Risk Factors and Coping Strategies. Long Range Planning 29(1996) 3, 381-392.
[11] Spitta, Th.: Software Engineering und Prototyping. Springer, Berlin - Heidelberg et al. 1989 (in German).
[12] Spitta, Th.: IV-Controlling in mittelständischen Industrieunternehmen - Ergebnisse einer empirischen Studie. To appear: Wirtschaftsinformatik 40(1998) 5, ('IS-Controlling in SME's - Results of an Empirical Investigation', in German).
[13] Spitta, Th.: Data Collection of Development and Maintenance Effort - Data Model and Experiences. University of Bielefeld/Germany, Department of Economics, Working Paper No 401, August 1998. See http://www.wiwi.uni-bielefeld.de/~spitta/index.html.