Data Collection of Development and Maintenance Effort

- Data Design and Experiences -

Thorsten Spitta

thSpitta@wiwi.uni-bielefeld.de

August 1998

Discussion Paper No. 401

Universität Bielefeld
Fakultät für Wirtschaftswissenschaften
Postfach 10 01 31
33501 Bielefeld

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. The paper presents a revised concept for the collection of personnel effort and for events (e.g. faults) as a generalized data model. Integrity constraints of that model may be fitted to individual circumstances. The model was first published in 1989. This revised version is based on experience with the 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.

Contents

Contents

The following discussion paper is an extended version of Spitta (1998) from ISSRE '98 which is more result oriented. The extension of this paper is the generalized concept of the data model in section 4 of the ISSRE paper.

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'* [Broo75], *'Software Engineering Economics'* [Boe81] are two examples of fundamental books which intensively deal with the amounts and distribution of human effort in software development and maintenance. Despite this great importance for software management there is no generalized concept, how to design and manage the effort collection process itself. The only paper known to the author is by Basili and Weiss [BaWe84], which does not discuss effort collection but event collection about failures. The paper of Basili and Weiss deals with the collection of data during maintenance processes in the US navy, collecting and analyzing fault events during the operation phase of software. An important issue of that paper is the reliability and correctness of the collected data which is mainly organized by supervision of data entry forms. Also in [Boe81] there are some hints about effort collection but they are, according to that time, also based on manual forms which have to be typed in to a batch process.

Because personnel effort comprehends at least 50% of IS-effort of any organization, it is important to know something about its distribution in detail. Only few management processes can be be practiced seriously without knowledge of personal effort data.

This paper describes a general concept of effort and event collection during software development and IS-system maintenance based on online collection of data. It has been 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. 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 cancelled requirements with no effort (see details in table 4, section 5). Including service, the database contained 169.4 PY (person years)[1] of effort.

As realized by recent empirical investigations of the author [Spit98a] 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*.
- 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 will show how to handle these problems in presenting a detailed data model which can easily be implemented. The data collected with our empirically validated implementation (more details see [Spit97]) show the applicability of such data for many managerial purposes.

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

Although the model will show the object types for planned data, the examples in Section 5 will not. The paper concentrates on the collection of real data because nobody is able to create realistic planned data without knowledge of real data. Our model serves as a guideline how to establish a correct ‘corporate memory’ about IS software and personal ressources as demanded by Boehm 17 years ago [Boe81].

2 Goals of effort collection

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 [Nol79]
- Structure, cost and trends of maintenance effort [Lehm80], [BoPa88]
- Project management, especially controlling external developers [Page85]
- Cost allocation of user department’s requirements [Nol77]
- Plan-/expenditure variances and other topics of IS-controlling [Dué89], [SeGr96].

IS-controlling is not focussed 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)
- Support effort for new systems
- 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
- Support effort per unser department
- External vs. internal personal effort, etc.

(4) Support
- PC-support effort per department
- Faults in network and PC-software, etc.

Because there are many more unplanned questions, the technical basis has to be a database system, and the collection of data has to be online and timely.

3 Requirements

In this section the general classifications for the data design are discussed which will appear there as *object types*, *attributes* or *attribute domains* in the next section.

We assume a dynamic life cycle model for software [Floy81] which since Lehman's fundamental empirical results [Lehm80] and Boehm's 'spiral model' [Boeh86] has been the generally accepted state of the art of software development and standard software implementation (Figure 1). The implications of that model for data to be collected are:

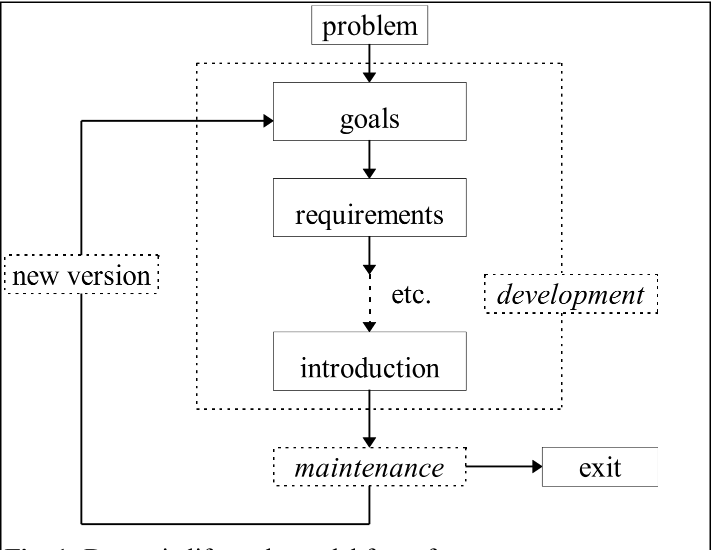

Fig. 1: Dynamic life cycle model for software (exit = end of operation)

- 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 simultanously. 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. The phase is the most detailed coded item in effort attachement (see table 1 next page). 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 6 in section 5 is an example for an industrial

environment. This is not only important for the comparison of data between firms, but also 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!). Two other effort classes are useful for event and cost accounting: *faults* and *purchase*. 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.

The difference between sequential and parallel activities of lower staff is important for the potential validity of data: It is relatively easy to discriminate sequential events, but nearly impossible to distinguish parallel events clearly. Software developers work on at least two activities (development, maintenance, fault correction, support) every day as shown in table 1. Even in very large enterprises and very large projects the one-project-working day would be an exception. According to our data, a maintenance programmer works on up to twelve activities per day with an average of five.

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_____
 d   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 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.

The next section shows the data model which fulfills our requirements.

4 Generalized data design

This section shows the data design of a generalized effort collection system as an object type structure, object types with attributes and integrity constraints for the attributes. The model is based on the experiences with the application of the system for more than 6 years. The original design was made and published in 1989 [Spit9, ch. 12]. The design presented here had only slightly to be revised in some details of object types' hierarchy (*ApplSystem* and *IsService;* see fig. 2) and in some attribute domains (*purchase* and *support*). On base of the relatively detailed attribute-type model it will be easy to implement it upon an arbitrary database system. The last topic is operation conditions, which promote a correct data collection with the proposed design in supporting the motivation problem.

4.1 Object type structure

The object type structure is shown as an entity relationship model.

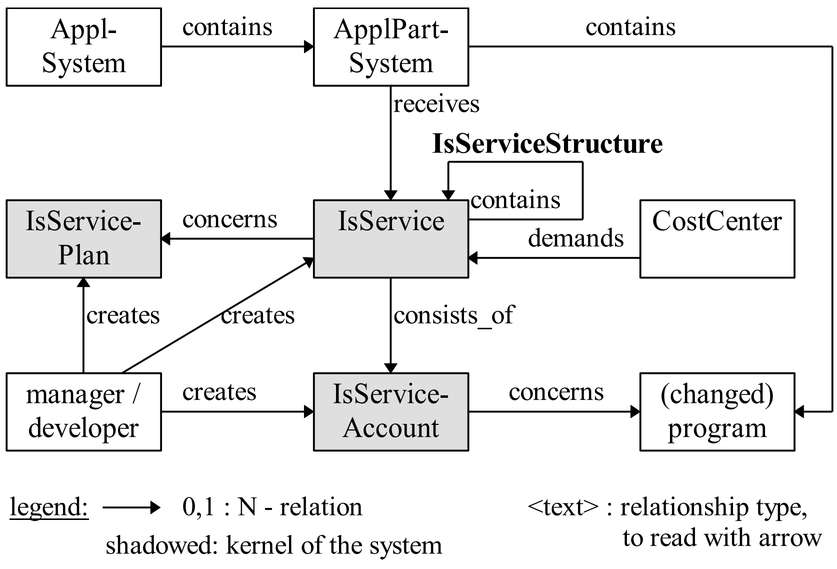

Fig. 2: Object type model for time and event collection

4.2 Object type model

The object type model with object types and their attributes is relationally expressed with only the kernel of the system shown in figure 2. The end of the names of non-key attributes express their types[2], which are shown in section 4.3. ‘Effort’ is the data type *Time*.

Table 2: Object type model (relations with attributes)

IsService	(<u>IsService#</u>, EffortCls, CostCenter#, ApplPartSystemCode#, Description, estimatEffortTime, realEffortTime, PhaseNo, strategRelevance?, EntryDate, desiredDate, promisedDate, CompletionDate)
IsServiceStructure	(<u>Upper.IsService#</u>, <u>Lower.IsService#</u>)
PhaseProgress	<u>IsService#</u>, <u>BeginDate#</u>, PhaseNo)
IsServiceAccount	(<u>IsService#</u>, <u>DeveloperName#</u>, <u>Date#</u>, IsServAcct#A, CostCenter#, PhaseNo, EffortTime, ActivityDesc,)
changedProgram	(<u>IsServAcct#</u>, <u>Program#</u>)
IsServicePlan	(<u>IsService#</u>, <u>beginDate#</u>, <u>endDate#</u>, EffortTime)

<u>Syntax:</u>

<u><string>#</u> : primary key	<string># : foreign key	<string>#A : alternate key

As discussed in section 3, there are two different roles of *PhasNo*: *IsService.PhaseNo* is an attribute of a *project, IsServiceAccount.PhaseNo* is an attribute of an *activity*. The ‘redundant’ attribute *CostCenter#* also has a non-redundant semantic: As shown in table 1, *IsServiceAccount.CostCenter#* has only to be assigned by the developer if *IsService.EffortCls* has the value ‘<u>s</u>upport’. This concerns the motivation problem and to the effort problem too.

PhaseProgress is a derived relation which is automatically booked, *realEffortTime* is a derived attribute with the constraint:

$$\forall \text{i} \in \text{IsService}$$

$$\bullet\ \text{i.realEffortTime} = \sum_{a \in IsServiceAccount | a.IsService\# = i.IsService\#} a.EffortTime$$

It is obvious that the derived attribute *realEffortTime* cannot be booked manually. The hierarchy ApplSystem $\rightarrow$ ApplPartSystem is part of the generic key of *ApplPartSystemCode#.*

4.3 Attribute type model

A precise attribute type model is very important for the validity of data. Every automatic consistency check during data entry, made by the source of the mass data - the developer -, avoids faults.

[2] type abbreviations: Cls= class, No= number, Desc= description, ?= data type BOOL.

Table 3: Domains and constraints of attributes

Type	constraint
EffortCls	: (development, fault, maintenance, purchase, support)
	where d, f, m : [**time**][3], f, p : [**event**].
ApplSystemCode#	: (administration, information, logistics, production, sales& delivery)
ApplPartSystemCode#	: **array** of <dynamic enumeration type in a database table>
PhaseNo	: (0 .. 6, ‘+’, ‘c’, ‘*’)[4]
	where (PhaseNo = (1..6) : *development*) **xor**
	(PhaseNo = ‘+’ : *maintenance*) **xor**
	(PhaseNo = ‘c’ : *cancelled project*) **xor**
	(PhaseNo = ‘*’ : *support*) **xor**
	(PhaseNo = 0 : *registered requirement*)
	and $\forall$i $\in$ IsService $\bullet$
	i_t.PhaseNo $\geq$ i_{t-1}.PhaseNo
	and $\forall$i $\in$ IsService-Rel[5] $\bullet$ i.PhaseNo $\in$ (0 , ‘c’ , ‘+’)
	$\Rightarrow$ $\neg\exists$a $\in$ IsServiceAccount-Rel $\bullet$
	i.IsService# = a.IsService#[6]
strategRelevance?	: **Bool**
	where $\forall$i $\in$ IsService-Rel $\bullet$
	i.strategRelevance? $\Rightarrow$ i.EffortCls $\in$ (d , p)[7]
CostCenter#	: **[CostCenter]**
	where $\forall$i $\in$ IsService-Rel $\bullet$
	(i.EffortCls = ‘s’ $\Rightarrow$ i.CostCenter# = **Null**
	and $\forall$a $\in$ IsServiceAccount-Rel $\bullet$
	i.IsService# = a.IsService# $\Rightarrow$ k.CostCenter# $\neq$ **Null**)[8]
Name	: (a..z, A..Z, -)
DeveloperName	: **[UserID]**
Description	: **Text** $\bullet$ Description $\neq$ **Null**
Time	: **[hours]** $\bullet$ Time $\geq$ **Null**[9]
Date	: **[calender date]**[10]

Syntax:

underlined: abbreviation;	$\bullet$: it holds;	**Null**: null value;
bold: base type;	index: state at a certain time	

While the data has to be specified rather exactly, the functional part need not be described in detail because it is relatively trivial. The conditions for the operation phase of the collection software are more important because they influence the correctness of the data significantly.

[3] [ ] = problem specific base type
[4] enumeration types are *ordinal*: (0 < 1 < .. < 6 < ‘+’ <‘c’ < ‘*’). Special characters serve for readability of reports; see table 5
[5] a *relation*, that is a subset of the type’s domain
[6] *PhaseNo* may remain equal or may be raised, but never reduced. *PhaseNo* = 0 is an event without any account. A finished or cancelled project cannot be accounted.
[7] *strategic relevant* are only development projects and purchased software.
[8] *CostCenter* is only booked by developers if *effortCls* = support
[9] **Null** values temporarily permitted
[10] booked automatically by system

4.4 Operation conditions

Because one main problem of data collection systems is validity, everything that can be formally checked should be done by implemented integrity constraints. Errors within the normal 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 accountings. 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 which is the only dangerous point
➔ sufficient information about the application of the data promotes this trust.

5 For what purposes are the data useful?

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 4 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 4: 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 3 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.*

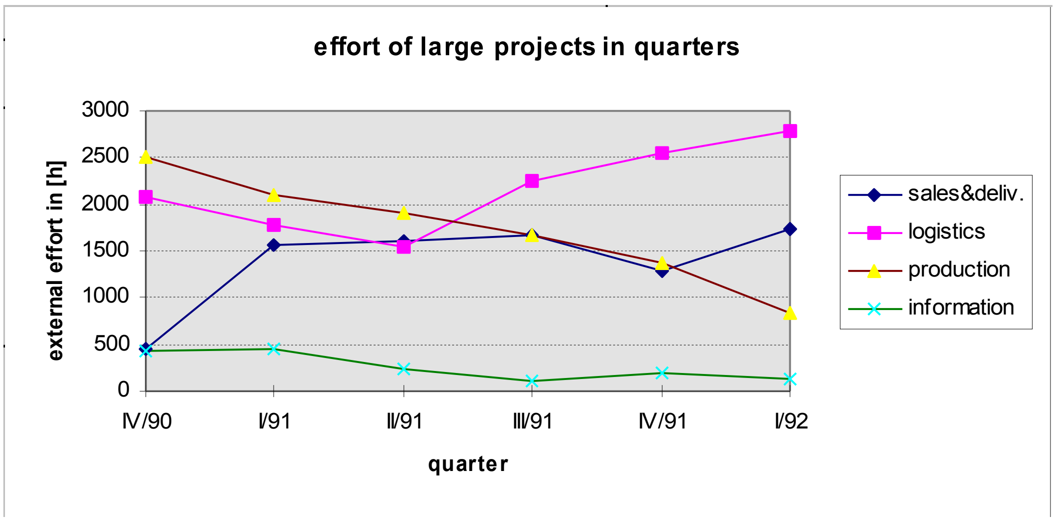

Fig. 3: External personal effort over 6 quarters for large projects

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 4 shows the evolution of some systems and their cumulated maintenance efforts.

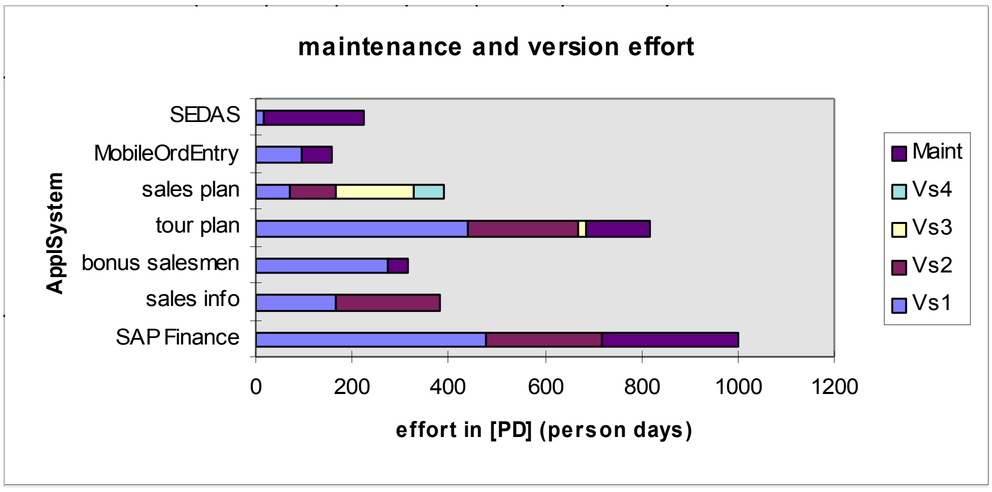

Fig. 4: Maintenance effort and effort of evolutionary versions of application systems

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 4:

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, *MobileOrder-Entry* 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[11].

5.3 Information management and IS-controlling

The database also allows for a long range view on IS software resources. Table 5 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. The strategic projects should be planned and well known in the whole enterprise as part of an information strategy paper [LePo97]. 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 5: 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

In addition, reactive and strategic projects are examples for special attributes in the central relationship type *IsService*. It is easy to extend the database at that point, but it is nearly impossible to extend the effort accountings (relationship type *IsServiceAccount*) because of the very large number of entries.

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

Table 6 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 6 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 6 will be discussed: The large number of c̲ancelled and abandoned activities and the number of actual projects i̲n w̲ork in different application areas. The indicator function of such aggregated views for IS-controlling is evident.

Table 6: Total number of IsService events over 6↔ years

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.

While table 5 shows the phenomenon of large and small projects with a very wide variance of effort, table 6 induces the important question to the classification problem:

What comprises the difference between a small project and maintenance?

This difference strongly depends on the size of an organization. In our case an effort limit of 5 PD (persond days) was set for a requirement to be classified as *maintenance.* Another criterion is the number of persons involved. So the following definition can be given:

Maintenance is either fault correction or the work on a requirement which can be done by *one* person in a limited time.

It is my impression that the popular development-maintenance ratios (see e.g. [Zelk79]) rather often are not asked, which distinguishes maintenance from small projects. In our case the effort-ratio over the whole 6↔ years was development : maintenance = 66% : 34%. This was no neccessary outcome but an effect of the firm's strategic decision, aiming at better flexiblility on the market, to renew most of the software systems. The 22 large projects (vs. 248 small) had 60% of project effort (= 66 PY).

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 4.4, 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 its collection is worth it. If it is used only for financial purposes, the effort is rather high. Technical and financial views on the data have to be integrated.

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.

References

[BaWe84] Basili, V.R., Weiss, D.M.: A Methodology for Collecting Valid Software Engineering Data. IEEE Transactions on Software Engineering 10(1984) 6, 728-738.

[Boe81] Boehm, B.W.: Software Engineering Economics. Prentice Hall, Englewood Cliffs, 1981.

[Boe86] Boehm, B.W.: A spriral model of software development and enhancement. ACM Sigsoft Software Engineering Notes 11(1986) 4, ???

[BoPa88] Boehm, B.W.; Papaccio, P.N.: Understanding and Controlling Software Costs. IEEE Transactions on Software Engineering 14(1988) 10, 1462-1477.

[Broo75] Brooks, F.P.: The Mythical Man Month, Addison-Wesley, Reading/Mass. 1975.

[Dué89] Dué, R.T.: Determining Economic Feasibility: Four Cost/Benefit Analysis Methods. Journal of Information Systems & Management 6(1989) 4, 14-19.

[Floy81] Floyd, C.: A Process-oriented Approach to Software Development. In: Systems Architecture, Proceedings of the 6th European ACM Regional Conference, Westbury House 1981, 285-294.

[Lehm80] Lehmann, M.M.: Programs, Life Cycles and Laws of Software Evolution. IEEE Proceedings 68(1980) 9, 1060-1076.

[Nol77] Nolan, R.L.: Controlling the Costs of Data Service. Harvard Business Review 55(1977) 4, 114-124.

[Nol79] Nolan, R.L.: Managing the crisis in data processing. Harvard Business Review, March-April (1979), 115-126.

[LePo97] Levy, M.; Powell, P.: Assessing the value of information systems planning at Heath Springs. International Journal of Technology Management 4(1997), 426-442.

[Page85] Page-Jones, M.: Practical Project Management. Restoring Quality to DP-Projects and Systems. Dorset, New York 1985.

[PfRo94] Pfleeger, Sh., L.; Rombach H. D.: Measurement Based Process Improvement. Editorial to: IEEE Software, 11(1994) 4, 9-11.

[Putn91] Putnam, L.H.: Trends in Measurement, Estimation and Control. IEEE Software 8(1991) 2, 105-107.

[SeGr96] Segars, A.H.; Grover, V.: Designing Company-wide Information Systems: Risk Factors and Coping Strategies. Long Range Planning 29(1996) 3, 381-392.

[Spit89] Spitta, Th.: Software Engineering und Prototyping. Springer, Berlin - Heidelberg et al. 1989 (in German).

[Spit97] Spitta, Th.: Die Gewinnung korrekter Daten aus manuellen Aufschreibungen: Empirische Ermittlung eines Erfassungssystems (How to Gain Correct Data from Individual Collection). In: Grün, O.; Heinrich, L.J. (Eds): Empirische Forschung in der Wirtschaftsinformatik, Springer, Wien - New York 1997, 105-118 (in German).

[Spit98a] Spitta, Th.: IV-Controlling in mittelständischen Industrieunternehmen - Ergebnisse einer empirischen Studie. WIRTSCHAFTSINFORMATIK 40(1998) 5, 424-433 ('IS-Controlling in SME's - Results of an Empirical Study', in German).

[Spit98b] Spitta, Th.: Personnel IS-Effort Collection in Industrial Environments. 9th Intern. Symp. on Software Reliability Engineering (ISSRE '98), Paderborn, Nov 1998, Industrial Practices, 111-120.

[Zelk79] Zelkowitz, M.V.; Shaw, A.C.; Gannon J.D.: Principles of Software Engineering and Design. Prentice-Hall, Englewood Cliffs, 1979.