nicht publiziert 1970 oder später Vortrag? wo? 1970 oder 1968

Pitfalls of EDP-system-implementation and dissemination - a case study

Wolfgang Giere

First it must be clarified that this paper cannot show a new way out of the so-called software-crisis. It will not end up with a new religion like chief-programmer-team, top-down design, structured programming etc. Furthermore this will not be a scientific paper in the sense of hypothesis and verification by experiments. It is rather a collection of common sense observations about the prerogatives of successful implementation and dissemination drawn from a bad experience.

It is generally accepted (and I feel) that success is a function of user acceptance. Some of it's constituents are listed in fig. 1. However, even when all of these basic requirements are met, failures are frequent. Apparently there are pitfalls. Lets try to identify them in a case study:

The case: In Summer 1969 we designed an online data-collection system for medical record data. Twelve typewriters had to be connected to a small process control-computer. This had to check the identification, structure, and occurence of key-words, had to sequentialize and output page by page on a magnetic tape. (see fig. 2)

In Autumn 69 we ordered this system: Twelve IBM 731 typewriters were connected to a SPC 12 General Automation computer. The system was delivered late, in March 1970. We tested it with the two available secretaries and trained them. The Deutsche Klinik für Diagnostik began operating on the 2nd of April

with ten additional secretaries in the central typing pool, working from the start with the data collection system. The software worked but we couldn't use the data: every time a secretary hit two keys quickly one after another, codemixes occurred in the computer which eventually were recognized as interrupts and caused a system-breakdown. Every time the secretaries typed slowly and rhythmically, the system worked fine. A n-key roll-over-function was not available in those days. The attempt to force the secretaries to better sequentualize via mechanical means (blocking the simultaneous depression of two keys) failed: The workload grew, the secretaries would have had to adapt to a new typing-style and rejected the system. The computernics worked around the clock, the board of directors suddenly and arbitrarily with-drew support. The system was removed, a complete "flop". (By the way, the system of the Deutsche Klinik für Diagnostik as a whole was a success. Otherwise I wouldn't dare to present this material.)

Let's analyse the reasons for this failure. Were the objectives wrong? Today we know they were not. We use routinely text acquisition systems which now even control word by word against thesauruses.

- Was the design of the system wrong; were twelve typewriters too much for that process control computer? Our calculations based on real figures from other typing pools and were on th safe side. The failure was not due to miscalculations.
- Was the selection of the IBM typewriters wrong? We did not choose the simple IBM Selectric but the heavy duty consol-operator machines, the most sturdy design available. They have been used as typewriters without trouble ever since.
- Was the choice of the General Automation SPC 12 wrong? It best met our specifications. The meantime between failures not due to interrupts by codemixes was reasonable, the magnetic tape worked without major difficulties.

These were not the reasons for the failure. But, what were the mistakes I can identify today und which rules could we formulate from this experience?

First: the use of console operator tested machines by secretaries in routine conditions led to difficulties.

- o Rule 1: Never design devices for routine use which have not been tested for exactly that use before ~~in a well controlled~~ pilot projects

Second: the assembly of 12 typewriters led to dynamic problems.

- o Rule 2: Never design complex systems for routine use which have not been tested as a system before!

Third: the tools of the Real Time Operating System were not adequate for the attached data aquisition system.

- o Rule 3: Never rely on operating-system-tools for routine use which have not been tested and demonstrated for that application before(or: Mistrust manuals)!

Fourth: The debugging of the application programs was difficult due to the instability of the operating system.

- o Rule 4: Never program an application - especially not for routine use - using non tested software tools (or: Take an "old" operating system for a new application)!

Fifth: The initially high motivation of the secretaries changed to frustration as soon as they were severly hindered in their daily work.

- o Rule 5: NeVer implement new applications in routine work before they have been demonstrated, tested and trained in a non critical bypass-situation.

Sixth: The EDP staff was frustrated by inadequate criticism and the untimely final end of the development.

- o Rule 6: Never risk the motivation of the development team by interference during development phases, let them go their way ... but

Seventh: The bord of directors had no control. They disconnected the efforts at a 98% stage.
Probably another 2 months of effort would have led to a success.

o Rule 7: Never implement a complex system without an adequate control procedure based on milestones!

This was the case. Let us now assume the system was successfull implemented and accepted by the users. Could it then according to our present knowledge have been disseminated? Lets answer this question on the basis of rules drawn from actual experienc with dissemination. (Details see fig. 3.) We can summarize this experience in rule 9 and 10.

o Rule 9: Never try dissemination without an experienced marketing organization. If you don't have it yourself, cooperate with a company and/or set up a user group!
o Rule 10: Never try dissemination without an exellent service. Make sure it reacts readily to user input!

Neither a marketing organization nor service were available at the DKD. Therefore,the answer to the hypothetical question whether a successful system could have been disseminated is no.

But again, lets assume, it was yes. Could then the dissemination be successful? In other words: Is it sufficient to have an operational system and dissemination support? Here is a list of the accepted requirements for success in the medical environment (fig. 3).

Back to the analysis of our failure:
A single compulsive programmer - an intelligent young man with burning eyes (according to Weizenbaum) - of course used the SPC 12 assembly language to optimize straight forward the applicatio
The result:
- documentation - a problem,
- adaptability without him - none,
- use of a better operating system - never,
- transfer to a new hardware - impossible.

Lets summarize the experience from these mistakes in the rules 11 and 12.

o Rule 11: Never let a single programmer implement a system. Have at least a separate test-programmer, testing the actual system on the basis of specifications!

- Rule 12: Never allow programming before system components are isolated, interfaces are described and provision for changes is made. Take the highest-level standardized language appropriate for the application!

We developed these twelve rules from the internal point of view of the EDP-department on the basis of our experience. From the same experience we can formulate advice for the external point of view of financing and managing agencies for their strategy!

Advice 1: Distinguish between development stages

- hardware-module (lab-model)
- hardware-systems (serie zero)
- operating systems (tools)
- application systems, tools (pilot use)
- applications (routine use)
- multiplication (dissemination).

Never allow the mix of different levels, even if the implementation process is slowed down!

Remember the typical procedures, people and environment connected to the levels!

The technical modul is developed in a lab oriented toward basic research by an engineer as a demonstration module. It never is ready, the engineer continous to strive for a better solution.

The system is typically made by the technical laboratory of a big company as series zero, promoted by the head of the development team aiming for a successful Pilot installation.

The operating system is (should be) designed "top down" by computer scientists, e.g. bright people from MIT, Stanford, Bell laboratories and the like.

The application system is built "bottom up" and refined step by step. The author must understand both the applications and the tools, operating systems in order to design and promote an application system. He normally has an inferiority complex against the computer scientist and the physician as well.

The application itself is "invented" by the user, the physician, guided and (hopefully) helped by the software tools of the underlying layers (abstract machines).
He is not interested in the nice functional design of the system, but in the smooth operation of his application in his daily environment. The system is best accepted if flexible and not noticable. The design is nor top down nor bottom up but by trial and error, iterative, adapting to the growing experience of the user.

The dissemination is not done by the inventor of an application. He is always interested in further refinement. The sales representative in turn must offer a stable product. There is only one solution: An definition of levels (releases, versions). They must be milestones in the continous flow of improvement. This is an organizational problem. Managers are needed for the task, not egg headed computer scientists or spirited pilot users.

As a result of this differentiation we come to:

Advice 2: Distinguish between different contractors

- fundamental research lab (hardware module)
- technical lab (hardware system)
- basic software lab (operating system)
- application software lab (application system)
- pilot user (application)
- dissemination agency (distribution)

In respect to management we come to:

Advice 3: Distinguish between milestones and working phases. Control at milestones, provide sufficient support during development phases. Adjust funding and control to the inherent dynamics of each development stage. Make sure that control structures do work!

Hardware-, system- and software-tool development are more or less internal tasks, while design and implementation of application systems need external control. (For this purpose we designed in Germany together with R.W. Schuster procedures, used for every federal grant in our field, the "Dokumentations- und Verfahrensrichtlinien für medizinische DV-Projekte, DVmed")

The multiplication of applications must be initiated by the demand of the environment. New users have to engage themselves to ensure thorough design and feed back. (In my department today no application is re-implemented after the successful pilot installation without financial and personal support of the interested user.)

Multiplication does not need programmers; it needs service. Make sure you are able to control its efficiency.

No application program should be distributed without guaranteed feed-back. The information of user groups is helpful. Periodic evaluation should be installed.

Last not least: Every implementor should dare to evaluate his systems performance on the basis of the specifications, no dissemination should be tried without this and an assessment of "marketing" results.

I hope, that these thoughts lead to discussion and constructive criticism in the light of other experiences.

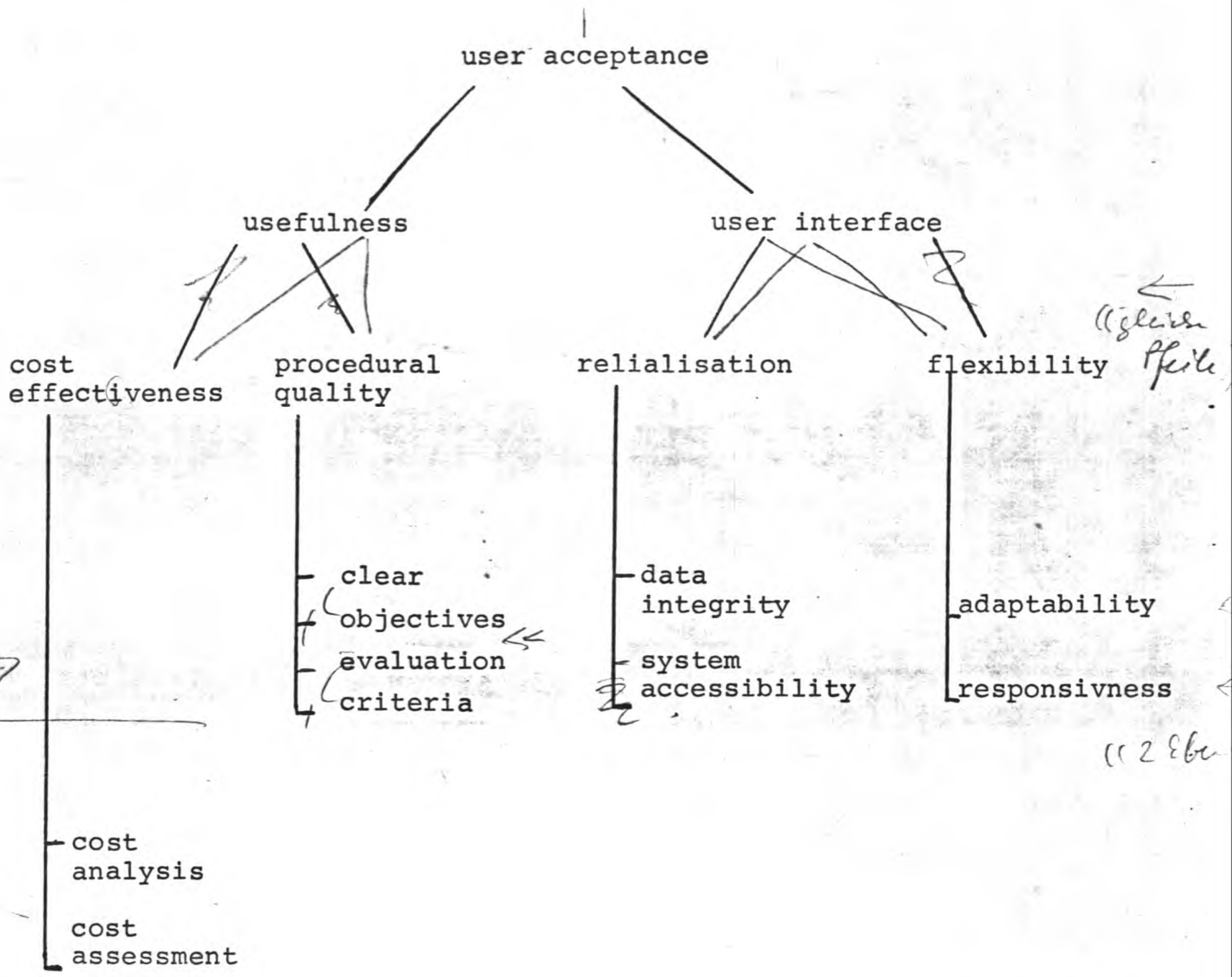
user acceptance
usefulness
user interface
cost effectiveness
procedural quality
relialisation
flexibility
clear objectives
evaluation criteria
data integrity
system accessibility
adaptability
responsivness
cost analysis
cost assessment

fig. 1

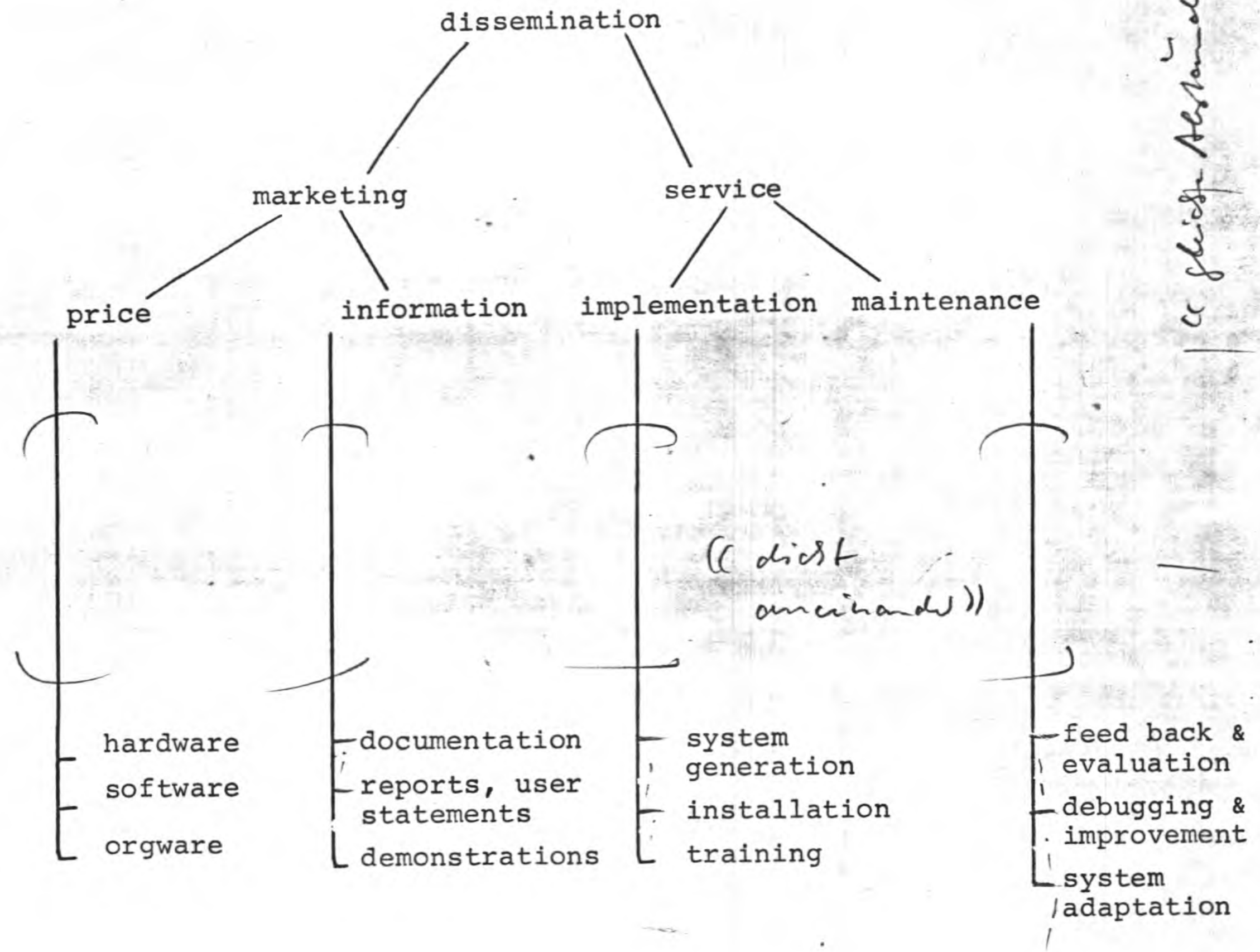
dissemination
marketing
service
price
information
implementation
maintenance
hardware
software
orgware
documentation
reports, user statements
demonstrations
system generation
installation
training
feed back & evaluation
debugging & improvement
system adaptation

fig. 3

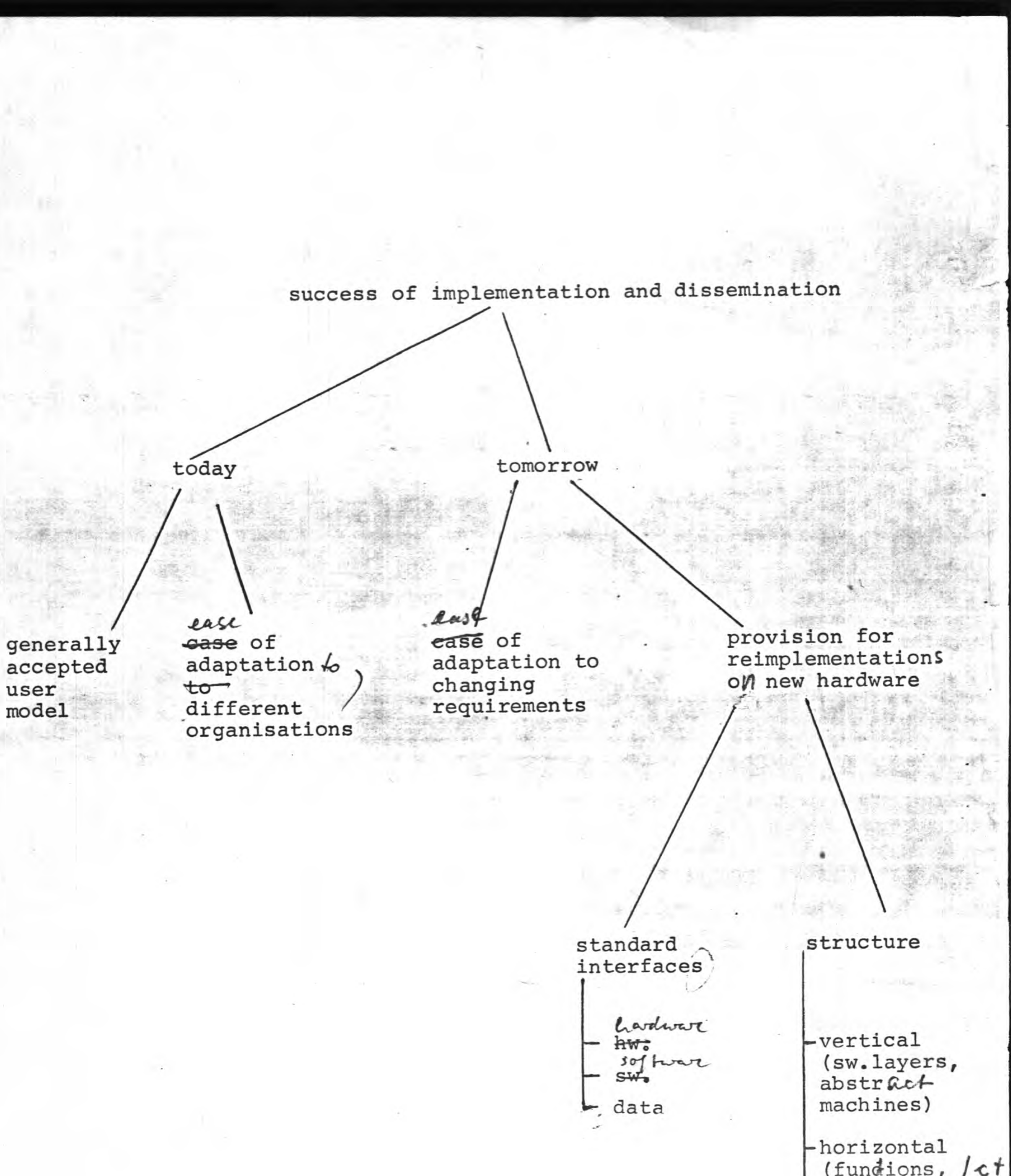
success of implementation and dissemination
today
tomorrow
generally accepted user model
~~case~~ ease of adaptation ~~to~~ to different organisations
~~case~~ easy of adaptation to changing requirements
provision for reimplementations on new hardware
standard interfaces
hardware
~~hw.~~
software
~~sw.~~
data
structure
vertical (sw. layers, abstract machines)
horizontal (functions, subsystems)

fig. 3

Author:

Prof.Dr.med. Wolfgang Giere
Abteilung für Dokumentation und Datenverarbeitung
Zentrum der Medizinischen Informatik
Klinikum der J.W. Goethe-Universität
Theodor-Stern-Kai 7

D-6000 Frankfurt/M. 70