Annual Review of Automatic Programming Vol. 16, pp. 121-128, 1992 0066-4138/92/$15.00

WORK ORGANIZATION AND SOFTWARE DEVELOPMENT

M. RAUTERBERG and O. STROHM

Work and Organizational Psychology Unit, Swiss Federal Institute of Technology (ETH) Nelkenstrasse 11; CH-8092 Zürich; Switzerland

Abstract

The current state of traditional software development is surveyed and essential problems are investigated on the basis of empirical data and theoretical considerations. The concept of optimisation cycle is proposed as a solution. The relation of several different kinds of local optimisation cycles to the specifications, the communications, and the optimisation problem is integrated into a concept of participatory software development. Software development without integrated work and/or task organizational development is suboptimal. User participation and prototyping decrease significantly cost and time exceedings. The more effort in early development stages, the less effort in the maintenance phase.

Keywords

software development, user participation, prototyping, iterative cyclic process model

1. Introduction

Analysis of current software development procedures brings to light a series of weaknesses and problems, the sources of which lie in the theoretical concepts applied, the traditional procedures followed as well as in the use of inadequate cost analysis models. These point to the significance of participation by all groups affected. Analysis of these cases shows that there are three essential barriers to optimisation: the specification barrier, the communication barrier and the optimisation barrier. Speaking quite generally, one of the most important problems lies in coming to a shared understanding by all the affected goups of the component of the worksystem to be automated — that is to say, to find the answers to the questions of "if", "where" and "how" for the planned implementation of technology, to which a shared commitment can be reached. An optimal total system must integrate the social and the technical subsystem simultaneously.

2. Problems of Traditional Software Development

In order to arrive at the optimal design for the total working system, it is of paramount importance to regard the social subsystem as a system in its own right, endowed with its own specific characteristics and conditions, and a system to be optimised when coupled with the technical subsystem. "Human resources are at the core of future growth and Europe's innovation capability"[4]. The work task plays a central role in this, being as it is the "interface" between the organization and the individual.

In the context of the our research project the general software development processes of different software companies (N=22) were analyzed as field studies by document analysis and interviews. These field studies were carried out in firms that develop information systems for offices and administration. Several other representative software projects were analyzed only by questionnaires (N=83). The analysis was not only done with regard to technical aspects, but also with regard to work organization, use of methods, user participation and the problems connected with different procedures. The whole sample (N=105) consisted of companies with internal software development departments regarding to software development activities alone (74%), to banks or insurance business

(15%), to industrial affairs (7%), or to other issues (4%).

The central problems of all software projects in our sample are organizational or methodological aspects, strategical aspects, technical aspects, social aspects, qualificational aspects, or other aspects[16]. The main important topics are problems, which have organizational or methodical causes. One topic is the difficulty to define requirements, because users are not participated in this stage, the importance of this stage is underestimated and/or there is a lack of adequate methods.

2.1. The specification problem

The "specifications barrier" is a problem which is in the foreground even at a cursory glance. How can the software developer ascertain that the client is able to specifiy the requirements for the subsystem to be developed in a complete and accurate way which will not be modified while the project is being carried out? The more formal and detailed the medium used by the client to formulate requirements, the easier it is for the software developer to incorporate these into an appropriate software system. But this presumes that the client has command of a certain measure of expertise. However, the client is not prepared to acquire this before the beginning of the software development process. It is therefore necessary to find and implement other ways and means, using from informal through semi-formal to formal specification methods. It would be a grave error with dire consequences to assume that clients — usually people from the middle and upper echelons of management — are able to provide pertinent and adequate information on all requirements for an interactive software system.

2.2. The communication problem

The communications barrier between applier, user and end-user on the one hand and the software developer on the other is essentially due to the fact that "technical intelligence" is only inadequately imbedded in the social, historical and political contexts of technological development[8]. Communication between those involved in the development process can allow non-technical facts to "slip through the conceptual net of specialised technical language, which therefore restricts the social character of the technology to the functional and instrumental"[8]. Every technical language not only dominates the concrete process of communication in the speciality concerned, but also determines the cognitive structures underlying it. The application-oriented jargon of the user flounders on the technical jargon of the developer. This "gap" can only be bridged to a limited extent by purely linguistic means, because the fact that their semantics is conceptually bound makes the ideas applied insufficiently sharp. To overcome this fuzziness requires creating jointly experienced, perceptually shared contexts. Beyond verbal communication, visual means are the ones best suited to this purpose. The stronger the perceptual experience one has of the semantic context of the other, the easier it is to overcome the communications barrier.

2.3. The optimisation problem

As a rule, software development is a procedure for optimally designing a product with interactive properties for supporting the performance of work tasks. Software development is increasingly focussing attention on those facets of application-oriented software which are unamenable to algorithmic treatment. While the purely technical aspects of a software product are best dealt with by optimisation procedures attuned primarily to a technical context, the non-technical context of the application environment aimed at requires the implementation of optimisation procedures of a different nature. Optimisation means utilising all (even only limitedly available) means within the context of an economical, technical and social process in such a way that the best result is achieved under the given constraints. It would be false indeed to expect that at the outset of a larger reorganization of a work system any single group of persons could have a complete, pertinent and comprehensive view of the ideal for the work system to be set up. Only during the analysis, evaluation and planning processes can the people involved develop an increasingly clear picture of what it is that they are really striving for. This is basically why the requirements of the applier sometimes seem to "change" — they do not really change but simply become concrete within the anticipated boundaries. This process of concretisation should be allowed to unfold as completely, as pertinently and as inexpensively as possible. Completeness can be reached by making sure that each affected group of persons is involved at least through representatives. Iterative, interactive progress makes the ideal concept increasingly concrete.

3. Analysis of Traditional Software Development Processes

3.1. The development process: an overview

Software projects are often realized with structured, linear stagemodels and defined milestones. The different tasks of software development can be assigned to the following stages: problem analysis, conception, specification, programming, test and implementation. The mean effort of each stage is estimated by the percentual portion to the whole project effort without the maintenance phase (see Figure 1).

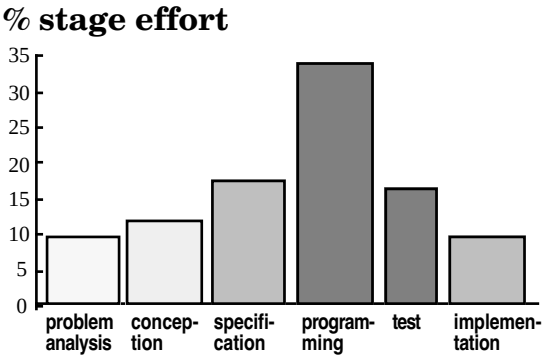

Figure 1: Mean effort in the different steps of software development process (N=79)

3.2. The early stages: analysis, conception and specification

The early stages are frequently the most neglected activities. This is essentially due to the fact that methods and techniques need to be used primarily the way occupational and organizational sciences have developed and applied them[7]. Inordinately high costs incur from the troubleshooting required because the analysis was less than optimal. The time has come to engage occupational and organizational scientists at the analysis stage who have been specially trained for optimal software development! Introducing task orientation within the framework of socio-technological system conception makes the following conditions indispensible[17]: 1. The employees must have control over the work process as well as the necessary means. 2. The structural features of the task must be such as to release in the working person the energy for completing or continuing the work. Task planning is therefore the focus of attention in the analysis phase. The five features "completeness", "variety of tasks", "opportunity for social interaction", "personal autonomy" and "opportunity for learning and development" must be striven for in order to suitably plan the tasks[17].

Once the analysis of the work system to be optimised has been completed, the next stage is to mould the results obtained into implementable form. Methods of specification with high communicative value are recommended here.

(1) The Specification of the Organizational Interface: The first thing is to determine "if" and "where" it makes sense to employ modern technology. Although the view is still widely held that it is possible to use technology to eliminate the deficiencies of an organization without questioning the structures of the organization as a whole, the conclusion is nevertheless usually a false one. It is important to understand the work system as a living organization, as a self-sustaining organism, which must develop and change in order to reach the organizational aims. The purpose of defining the organizational interface, from this point of view, is to improve the viability of the organization with the help of modern technology. An unavoidable consequence is, that the necessary measures must be taken in such a way that the ease with which the employees can assimilate and adapt to the type of the organization is maximised. The effects of the organizational measures taken can be assessed, for example, by means of the "Activity Evaluation System"[17] or "Activity Evaluation System for 'Intellectual Work'."[17].

(2) Specification of the Tool Interface: The intended division of functions between man and machine is decided during the specification of the tool interface. The tasks which remain in human hands must have the following characteristics[18]: 1. sufficient freedom of action and decision-making; 2. adequate time available; 3. sufficient physical activity; 4. concrete contact with material and social conditions at the workplace activities; 5. actual use of a variety of the senses; 6. opportunities for variety when executing tasks; 7. task related communication and immediate interpersonal contact.

(3) Specification of the Input/Output Interface: Once those concerned are sufficiently clear about which functions are amenable to automation, the next step which should be taken is to test the screen layout on the end-

users with the extremely inexpensive hand-drawn sketches. The use of prototyping tools is frequently inadvisable, because tool-specific presentation offers a too restrictive range of possibilities. The effect of the design decisions taken can be assessed with the help of discussion with the end-users, or by means of checklists.

(4) Specification of the Dialogue-Interface: The use of prototypes to illustrate the dynamical and interactive aspects of the tools being developed is indispensible for specifying the dialogue interface. But prototypes should only be used very purposefully and selectively to clarify special aspects of the specification, and not indiscriminately. Otherwise there looms the inescapable danger of investing too much in the production and maintenance of “display goods”. A very efficient and inexpensive variation is provided by simulation studies, for example, with the use of hand prepared transparencies, cards, etc. which appear before the user in response to the action taken[6].

More and more companies try to practice user participation in the software development process. Statements of software engineers like "we must participate the users" or "without users it does not work" and the selection of the users for the participation in the project with regard to their professional background show, that user participation as one necessity becomes more and more accepted. First of all this concerns to projects in which the system implementation is combined with greater organizational changes of the work system.

We differentiate user participation into three categories: active or passive participation, or without any participation. Further and more detailed analysis of our data[15,16] show the following results. In 29% of the projects (N=83) *active* user participation was practiced. This means, that the users had have decision possibilities, were frequently asked to problems of task design, functionality, dialogue design, etc., and were involved in the early stages. In 57% of the projects a *passive* form of user participation was practiced. This means, that the users gave informations, evaluated the ideas of the software engineers, but did not be so deeply involved in the early stages. 14% of all projects were realized without any kind of user participation.

Software developer put prototyping as one useroriented method more and more into practice. The prototyping method was used in 55% of the projects (N=83). Of all projects with prototyping (N=46) the most software engineers evaluate this procedure as "very useful" (59%) or "useful" (33%). These software engineers say, that prototyping is first of all a good method to support the cooperation and communication with the users.

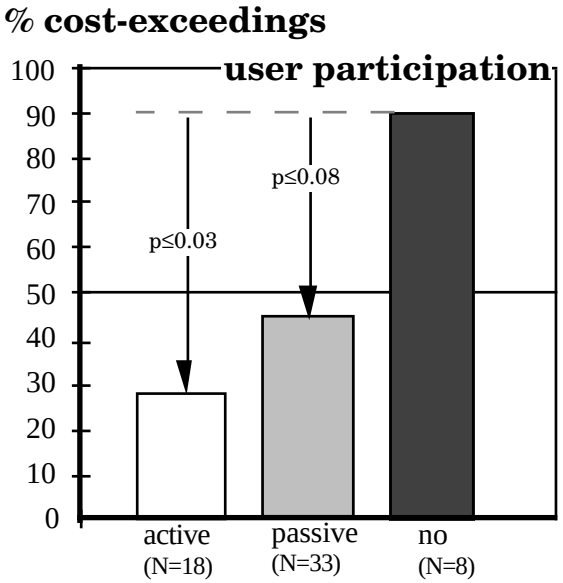

Figure 2: The relation of cost exceedings with user participation; "cost exceeding" is the percentual cost portion of total project budget.

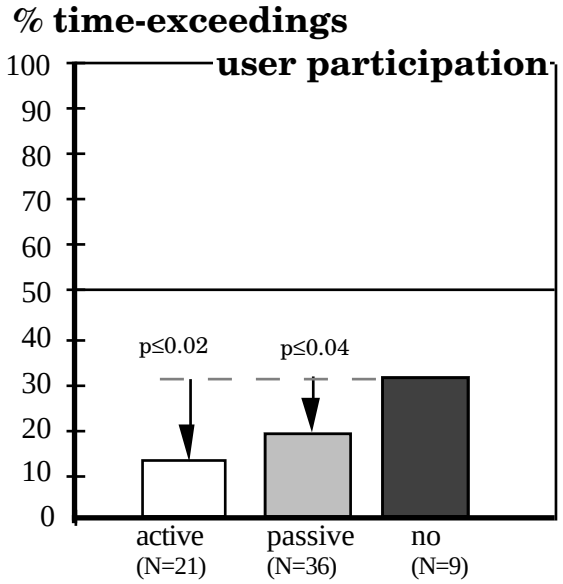

Figure 3: The relation of time exceedings with user participation; "time exceeding" is the percentual elongation part of the total project time.

Active user participation decrease significantly the cost exceeding portions (active vs. no p≤0.03, one-tail T-test; N=59; see

Figure 2), as well as the time exceedings portion (active vs. no p≤0.02, one-tail T-test; N=59; see Figure 3). The usage of prototyping decrease significantly cost exceedings (p≤0.04, one-tail T-test; N=59; see Figure 4), as well as time exceedings (p≤ 0.05, one-tail T-test; N=66; see Figure 5).

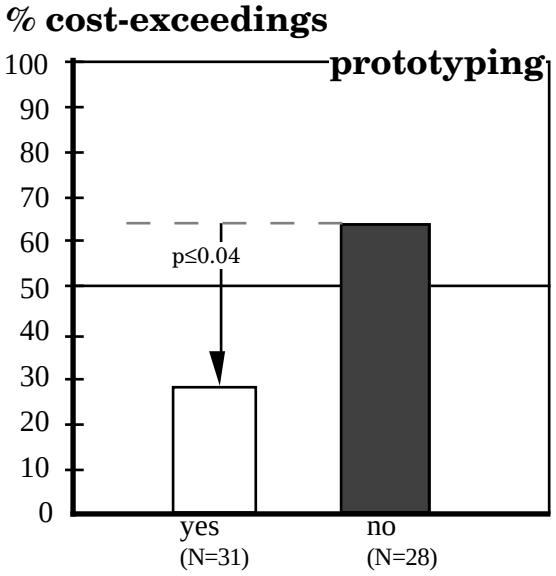

Figure 3: The relation of cost exceedings with prototyping; "cost exceeding" is the percentual cost portion of total project budget.

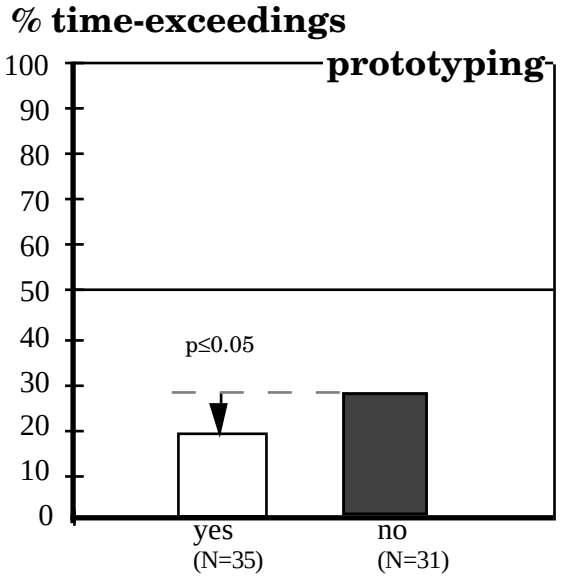

Figure 5: The relation of time exceedings with prototyping; "time exceeding" is the percentual elongation part of the total project time.

Figure 2, 3, 4 and 5 show, that there are significantly advantages in projects with active user participation and with usage of prototyping. We can not find a significant correlation between the category of user participation ["active", "passive", "no"] and the project size ["small": "≤5 man years", "big": "≥6 man years"] (N=77; CHI²-Test, df=2, p≤ 0.18); there is also no significant correlation between prototyping ["yes", "no"] and project size (N=77; CHI²-Test, df=1, p≤ 0.84).

We distinguish between four project types of software developments[15]: type A is a customized software product by inhouse development for a specific inhouse department; type B is a customized software product for external clients; type C is a trade product for external companies; type D is standard software for unknown users. There is a significant correlation between project type ["A","B","C","D"] and project size ["small": "≤5 man years", "big": "≥6 man years"] (N=77; CHI²-Test, df= 3, p≤ 0.01). Big projects are above average type A projects, and small projects of type B. Projects of type C and D seem to be independent of project size.

Active user participation seems to be typically for project type A, passive user participation for project type C and no participation for project type B and D (N=83; CHI²-Test, df=6, p≤ 0.07). Prototyping ["yes", "no"] does not correlate with project type ["A","B","C","D"] (N=83; CHI²-Test, df=2, p≤ 0.41).

3.3. The programming and implementation stages

The programming stage is made up of the following three steps[1]: 1. design of the programme architecture; 2. design of the individual programme modules (object classes, etc.); 3. coding and debugging. The distinction between design and specification is important. During specification, all relevant properties of the technical subsystem are defined as precisely as possible. In the programming stage all care must be taken to ensure that the technical subsystem being developed has these properties to the greatest possible extent. It is pure software expertise which is of primary importance here. The implementation phase is characterized by the the first tests of the software system in the concrete working context.

Once a working version is available, it can be put to test in usability studies (“use-oriented benchmark tests”[11]) in concrete working situations. This is the first place where it is possible to clarify the problems with the actual organizational and technical environment. By contrast to laboratory stu-

dies, field studies take into account the aspect of “ecological validity”[6]. Trials with real work tasks make it possible to check and assess the degree to which the planned organizational ideal has been reached. Although video is the data recording medium preserving the most information, a combination of log-files and direct protocolling makes a good compromise between performance and economy.

3.4. The maintenance phase

The mean effort for maintenance is 20% (N=55). 33% of the maintenance effort is spent for debugging, 67% of the maintenance effort therefore is needed for changing the systems (e.g. changed requirements caused by users). The cumulated effort for the early stages "problem analysis", "concept" and "specification" correlates significantly negative ($r= -0.32$, $p<0.05$, N=55) with the effort for maintenance. This means, that adequate effort in the early stages reduces the often cost- and timeintensive repair and correction tasks in the maintenance phase (see Figure 6).

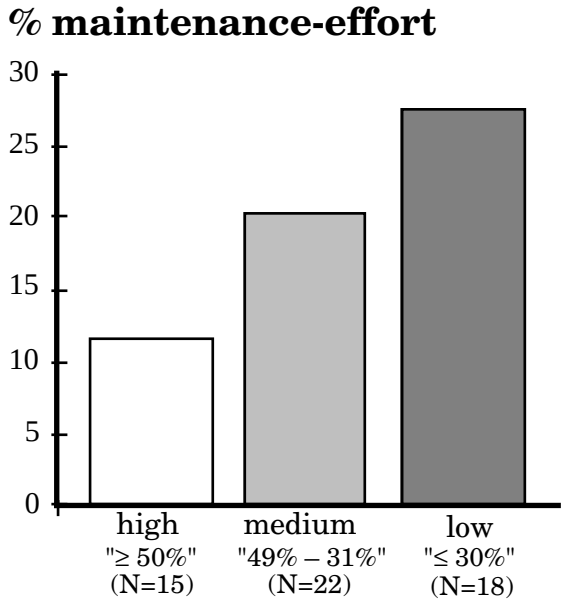

Figure 6 The relation of maintenance effort with cumulated effort of early software development stages "problem analysis", "conception" and "specification".

4. An Iterative-Cyclic Software Process Model

Sufficient empirical evidence has accumulated by now to show that task and user oriented procedures in software development not only bring noticeable savings in costs, but also significantly improve the software produced[1, 6, 10, 11, 14]. How then, can the problems mentioned above be solved?

4.1. Embarking on the global optimisation cycle

The type of software to be developed has proved to be one of the essential factors governing software development. The global optimisation cycle begins at Start-A of Figure 7 when developing completely new software and at Start-B in the case of further development and refinement of existing software. Different concept-specific local optimisation cycles are used to optimise specific work tasks, depending on the particular type of the project at hand. It is up to the project management to settle on the actual procedure and this decision is reflected in the development form chosen.

4.2. Global and local optimisation cycles

The use of optimisation cycles in software development procedures depends on the following conditions being met[10]: “1. A modified project management model, which guarantees above all communication between those concerned and the developers. 2. Computer supported version and documentation management, which includes also the results of evaluation and current criticism. 3. Informing all those involved about the project’s aims and the peculiarites of the procedure, as well as training the employees concerned. 4. The fundamental willingness of the developers to produce incomplete software and to accept critique of it. 5. The expansion of the expertise of the developer beyond purely data processing technology as regards measures in work structuring. 6. The use of a largely integrated software tool environment, which supports the developer in repeated preparation and modification of the software. 7. The preparedness of all persons involved to learn throughout the course of the project.”

Even if we assume that all the conditions listed are more or less fulfilled, there still remains the question of how to actually carry out the software development project. In order to reach the goals of a work-oriented design concept the first project phases (requirements analysis and definition; Quadrant-I in Figure 7) should be replete with a range of optimisation cycles.

Simple and fast techniques for involving users are discussion groups with various communication aids (metaplan, layout sketches, “screen-dumps”, scenarios, etc. [14]), questionnaires for determining the attitudes, opinions and requirements of the users, the “walk-through” technique, as well as targeted interviews aimed at a concrete analysis of the work environment. Very sound simulation methods (e. g. scenarios, “Wizard of Oz” studies) are available for developing completely new systems without requiring any special hardware or software.

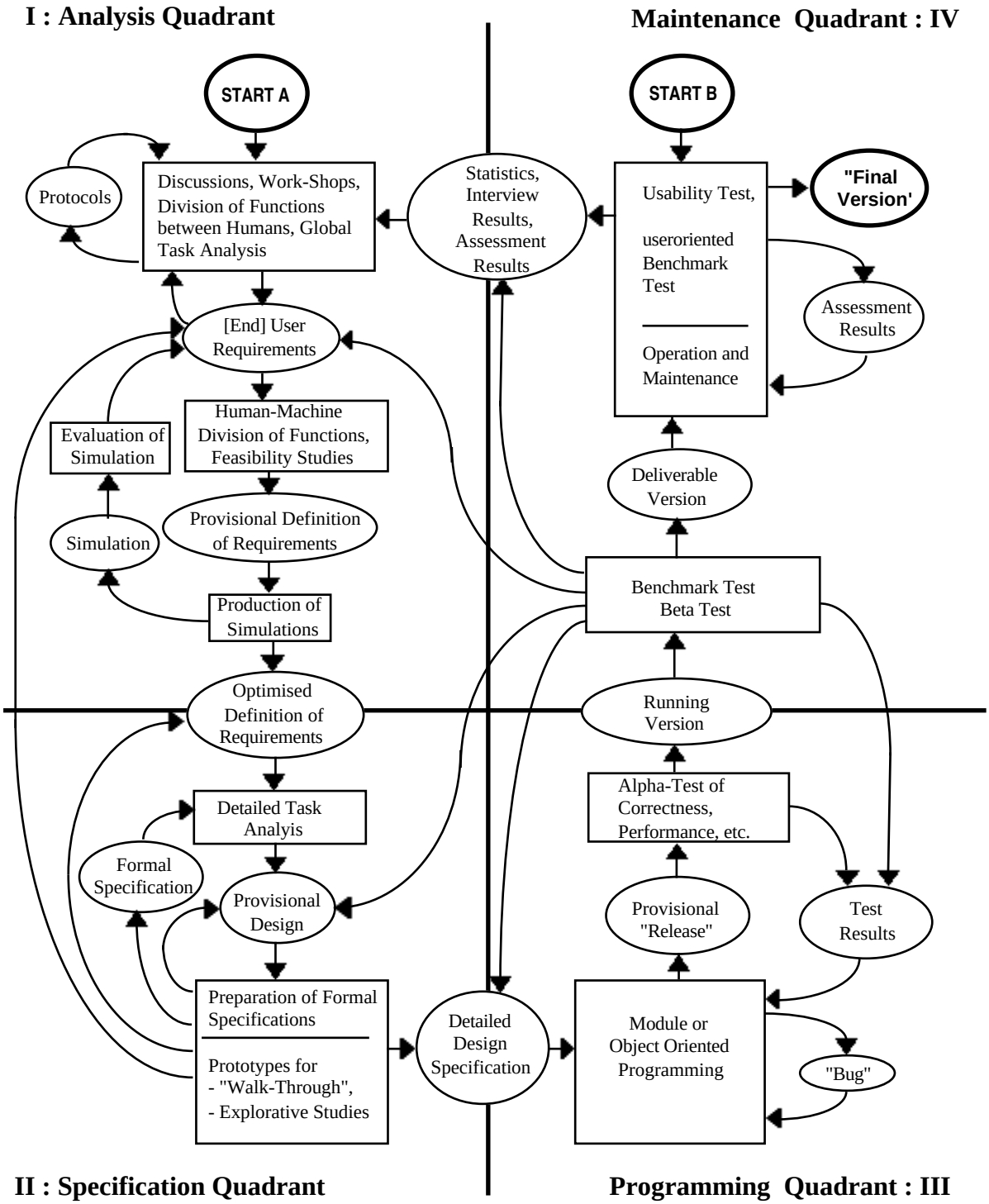

Figure 7 Flow chart for a participatory software development model showing the local optimisation cycles within and between individual quadrants (I - IV)[3, 5].

5. Conclusion

One of the principal problems of traditional software development lies in the fact that those who have been primarily involved in software development to date have not been willing to recognise that software development is, in most cases, mainly a question of task, job and/or organizational planning. Were software development to be approached from such a perspective, it would be planned from the beginning to engage experts in occupational and organizational planning in the process of software design. The global optimisation cycle can be subdivided into four regions: the region where requirements are determined (Quadrant I), the region of specification (Quadrant II), the region of implementation (Quadrant III) and the region of application and maintenance (Quadrant IV). An appropriate investment in optimisation in Quadrants I and II not only helps to reduce the total cost (development costs and application costs), but also leads to optimally adapted hardware and software solutions. This is due to the fact that all subsequent users are involved at least through representatives, and can therefore incorporate their relevant knowledge into the design of the work system. As more effort is expended on optimisation in the first quadrants, so less is needed in Quadrant IV. But first and foremost, we must start learning to plan jointly technology, organization and the application of human qualification. Technology should be viewed as one way of providing the opportunity to organise our living and working environments in a manner which is better suited to human needs.

Acknowledgements

The preparation of this paper was supported by the BMFT (AuT programme) grant number 01 HK 706-0 as part of the BOSS "User oriented Software Development and Interface Design" research project.

References

1. B.W. Boehm, T. Gray, T. Seewaldt, "Prototyping versus specifying: a multiproject experiment", IEEE Transactions on SE 10(3), 224-236 (1981).
2. B.W. Boehm, "Software Engineering Economics", Englewood (1981).
3. B.W. Boehm, "A spiral model of software development and enhancement", Computer (May) 61-72 (1988).
4. C.E.C. Commission of the European Communities, "Science, Technology and Societies: European Priorities. Results and Recommendations of the FAST II Programme", Summary Report. Directorate-General Science, Research and Development, Brussels (1989).
5. J. Grudin, S.F. Ehrlich, R. Shriner, "Positioning Human Factors in the User Interface Development Chain", Proceedings of CHI + GI (Toronto, 5th - 9th April 1987). New York, 125-131 (1987).
6. C-M. Karat, "Cost-Benefit Analysis of Iterative Usability Testing", Human-Computer Interaction - INTERACT '90. (D. Diaper et al., ed.) Amsterdam, 351-356 (1990).
7. L. Macaulay, C. Fowler, M. Kirby, A. Hutt, "USTM: a new approach to requirements specification", Interacting with Computers **2(1)**, 92-118 (1990).
8. M. Mai, "Sprache und Technik", Zeitschrift des Vereins Deutscher Ingenieure für Maschinenbau und Metallbearbeitung **132(7)**, 10-13 (1990).
9. J. Nielson, "Big paybacks from 'discount' usability engineering", IEEE Software **7(3)**, 107-108 (1990).
10. H. Peschke, "Betroffenenorientierte Systementwicklung", Europäische Hochschulschriften Reihe XLI Informatik Bd./Vol.1, Frankfurt Bern New York (1986).
11. M. Rauterberg, "Benutzungsorientierte Benchmark-Tests: eine Methode zur Benutzerbeteiligung bei Standardsoftwareentwicklungen", Reports of the German Chapter of the ACM, vol. 33 "Software-Ergonomie '91", (D. Ackermann and E. Ulich, eds.) Stuttgart, 96-107 (1991).
12. M. Rauterberg, "Optimisation Cycle: a Concept for Optimal Software Development", Cybernetics and System Research, vol.1 (R. Trappl, ed.), Singapore London, 279-286 (1992).
13. B. Schiemenz, "Kybernetik", Handwörterbuch der Produktionswissenschaft (W. Kern, ed.), Stuttgart, 1022-1028 (1979).
14. P. Spinas, D. Ackermann, "Methods and Tools for Software Development: Results of Case Studies", Man-Computer Interaction Research MACINTER-II, (F. Klix, N. Streitz, Y. Waern and H. Wandke, eds.) Amsterdam, 511-521 (1989).
15. O. Strohm, "Arbeitsorganisation, Methodik und Benutzerorientierung bei der Software-entwicklung", Software für die Arbeit von morgen, (M. Frese, Chr. Kasten, C. Skarpelis and B. Zang-Scheucher, eds.) Berlin Heidelberg New York, 431-441 (1991).
16. O. Strohm, E. Ulich, "Arbeitsteilung und Benutzerorientierung bei der Software-Entwicklung", Multidimensionales Software-Projektmanagement, (F. Elzer, ed.) Hallbergmoos, 261-289 (1991)
17. E. Ulich, "Arbeitspsychologie", Stuttgart, Poeschel (1991).
18. M. Zölch, H. Dunckel, "Erste Ergebnisse des Einsatzes der 'Kontrastiven Aufgabenanalyse'", Reports of the German Chapter of the ACM, Vol. 33 "Software-Ergonomie '91", (D. Ackermann and E. Ulich, eds.) Stuttgart, 363-372 (1991).

Annual Review in Automatic Programming
Volume 16, Part II

EXPERIENCE WITH THE MANAGEMENT OF SOFTWARE PROJECTS 1992

Proceedings of the Fourth IFAC/IFIP Workshop, Schloss Seggau, Austria, 18–20 MAy 1992

Edited by

P. ELZER

Technical University of Clausthal, Institute for Process and Production Control, Leibnitzstraße 28, 3392 Clausthal-Zellerfeld, Germany

and

V. HAASE

Institutes for Information Processing, Graz University of Technology

A-8010 Graz, Austria

Published for the
INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL
by

1992

PERGAMON PRESS

OXFORD • NEW YORK • SEOUL • TOKYO

[ISSN 0079-1946]