OPTIMISATION CYCLE: A CONCEPT FOR OPTIMAL SOFTWARE DEVELOPMENT.

MATTHIAS RAUTERBERG[*]

Work and Organisational Psychology Unit
Swiss Federal Institute of Technology (ETH)
Nelkenstr. 11; CH-8092 Zürich

Abstract

The current state of traditional software development is surveyed and essential problems are investigated on the basis of system 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 barrier is integrated into a concept of participatory software development.

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. Barriers in the Framework 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 subsytem. “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 organisation and the individual.

2. 1. The Specifications Barrier

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 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.

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 Communications Barrier

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 Barrier

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 reorganisation 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. Overcoming the Barriers

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 three barriers mentioned above be overcome?

3. 1. The Optimisation Cycle

Systems theory distinguishes between “control” (i. e. “feed forward” or “open loop” control systems) and “regulation” (i. e. “feedback” or “closed loop” control systems). The following are minimum conditions in the case of “control” for the output to adjust as intended to the reference input: “(1) precise knowledge of the response of the system being controlled on the one hand and the output and interference on the other; (2) precise knowledge of those quantities whose effect on the system is detrimental to the intended influence (interference or perturbance, such as technical feasibility, etc.); if the system has a response delay, then a prognosis is needed for these interferences at least for the period of the delay; (3) knowledge of procedures for deriving controller output from such information. These conditions are hardly ever met in practice! That is why it is constantly necessary to supplement or replace control by regulation.”[13]. The application of the highly effective “regulation” control principle actually only requires a knowledge of those controller outputs of the test component which steer the provisional ouput of the action component in the desired direction (see Fig. 1). “In regulation, the control apparatus takes into account the results of the control procedure. There is thus a feedback from the output of the regulated system to the input of the regulating one.”[13].

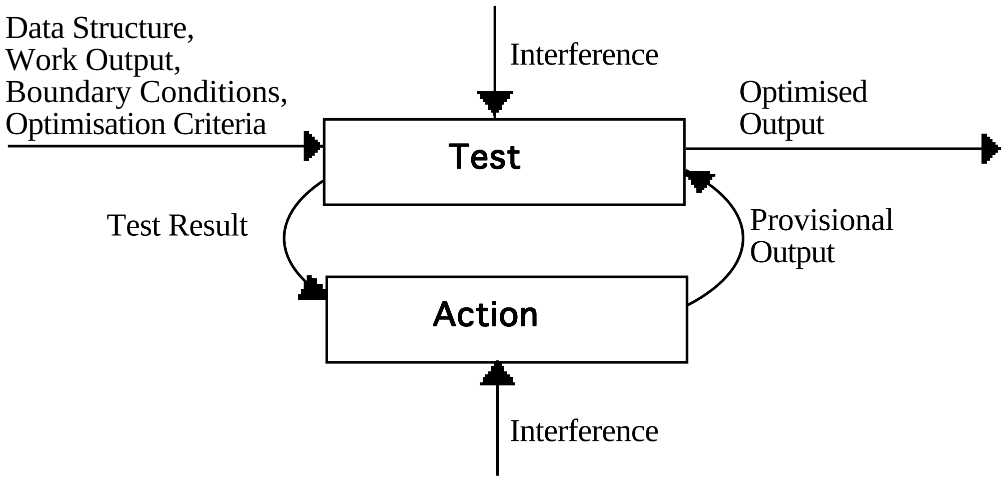

Figure 1 The optimisation cycle in the context of an iterative cyclic concept for software development[10, 12].

The distinction between boundary constraints on the one hand and interference on the other is that the boundary constraints are deliberately and purposefully set (e. g. development costs, duration of project, etc.), whereas interference is unintentional and unanticipated. We designate the “Test-Action- Cycle” as *optimisation cycle*. An important dimension of the optimisation cycle is its *length*, i. e. the time required to complete the cycle once. Depending on the nature of the activity and the testing, the length can be anything from a matter of a few seconds to up to possibly several years. The longer this period, the more costly the optimisation cycle. It is the aim of user-oriented software development to incorporate an as efficient optimisation cycle as possible into software development procedures[6, 9].

If the general scheme of regulation is applied to software development, then the individual components of regulation are assigned as illustrated in Figure 1. The *optimisation criteria* are all relevant technical and social factors. Testing ascertains the extent to which the optimisation criteria are met, subject to the boundary constraints. The action taken could come from a range of extremely different procedures, methods or techniques. All of this depends on the nature of the work output. Interference could come from the three barriers discussed above as well as from technical and/or social problems in realising the project. It is frequently the case that control systems operate more economically than regulation systems for the same purpose — but only if the conditions mentioned above prevail! This is one of the most important reason why the attempt is made to come as close as possible to a particular control system, namely the "Waterfall Model". If, however, the barriers discussed above, are taken seriously, then we must determine those places in development processes at which optimisation cycles are indispensible.

3. 2. The Analysis Phase

The analysis phase is frequently the one most neglected. This is essentially due to the fact that methods and techniques need to be used primarily the way occupational and organisational 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 ocupational and organisational 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[15]: 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[15]. Two levels can be distinguished for the analysis of a given work system.

(1) Task Analysis The following concept of task and conditions analysis describes a step-by-step deepening analysis of tasks[15]. Task analysis is designed to help the draughting of organisational design proposals and is made up of seven steps: 1. Structuring of the production process and the plant environment. 2. Identification of the work process within the production process. 3. Listing the characteristics of the product to be produced or process to be controlled. 4. Analysis of the division of labour among the employees. 5. Rough description of the structure of the work tasks. 6. Determination of the objective degrees of freedom in performing tasks. 7. Determination of the frequency of repeated and of rare tasks per shift. The techniques applied include: document analysis, the observation interview, and the expert interview[7].

(2) Analysis of Work Activities: Draughting recommendations for job (re-)design often makes an activity analysis unavoidable[15]. Such an analysis provides essential information on the modus, frequency of occurrence and time taken for the individual components of the activity. The following three steps must be observed: 1. Analysis of the course of the activity to determine the components involved. 2. Developing a system of categories for determining precisely the component activities. 3. Determination of the nature, frequency of occurrence and time taken for each individual activity component throughout the entire shift. While traditional software development partly includes a

global analysis of the tasks in the work system, analysis of work activities and their effects are largely excluded from consideration.

3. 3. The Specification Phase

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 Organisational 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 organisation without questioning the structures of the organisation as a whole, the conclusion is nevertheless usually a false one. It is important to understand the work system as a living organisation, as a self-sustaining organism, which must develop and change in order to reach the organisational aims. The purpose of defining the organisational interface, from this point of view, is to improve the viability of the organisation 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 organisation is maximised. The effects of the organisational measures taken can be assessed, for example, by means of the "Activity Evaluation System"[15] or "Activity Evaluation System for 'Intellectual Work'."[15].

(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[16]: 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].

3. 4. The Implementation Phase

The implementation phase 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 implementation phase 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.

3. 5. The Trial and Assessment Phase

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 organisational and technical environment. By contrast to laboratory studies, 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 organisational 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.

4. A Participatory Concept for Software Development

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 Fig. 2 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 Fig. 2) 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.

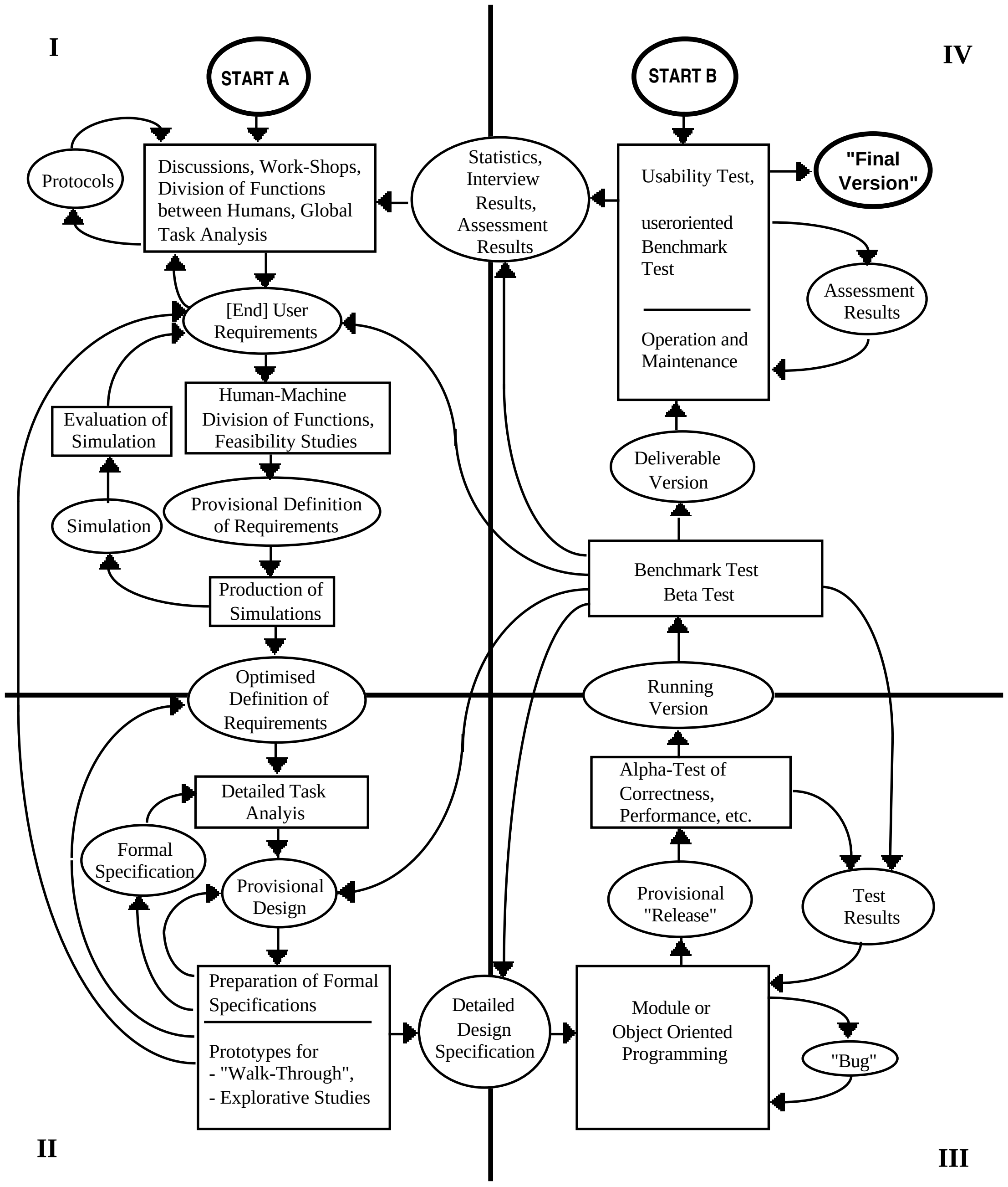
I
START A
Protocols
Discussions, Work-Shops, Division of Functions between Humans, Global Task Analysis
Statistics, Interview Results, Assessment Results
START B
IV
Usability Test,
useroriented Benchmark Test
Operation and Maintenance
"Final Version"
Assessment Results
[End] User Requirements
Evaluation of Simulation
Human-Machine Division of Functions, Feasibility Studies
Deliverable Version
Simulation
Provisional Definition of Requirements
Production of Simulations
Benchmark Test Beta Test
Optimised Definition of Requirements
Running Version
Detailed Task Analyis
Alpha-Test of Correctness, Performance, etc.
Formal Specification
Provisional Design
Provisional "Release"
Test Results
Preparation of Formal Specifications
Prototypes for - "Walk-Through", - Explorative Studies
Detailed Design Specification
Module or Object Oriented Programming
"Bug"
II
III

Figure 2 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 organisational planning. Were software development to be approached from such a perspective, it would be planned from the beginning to engage experts in occupational and organisational 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, organisation 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.

References

1. B.W. Boehm, T. Gray and 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 and 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 and 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, "Participatory Concepts, Methods and Techniques for Optimising Software Development", Work Design and Participatory Systemdevelopment , (P. Brödner, G. Simonis and H-J. Paul, ed.) in press (1992).
13. B. Schiemenz, "Kybernetik", Handwörterbuch der Produktionswissenschaft (W. Kern, ed.), Stuttgart, 1022-1028 (1979).
14. P. Spinas and 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. E. Ulich, "Arbeitspsychologie", Stuttgart, Poeschel (1991).
16. M. Zölch and 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).

CYBERNETICS AND SYSTEM RESEARCH '92

Vol. 1

Proceedings of the Eleventh European Meeting on
Cybernetics and System Research,
organized by the Austrian Society for Cybernetic Studies,
held at the University of Vienna, Austria, 21 - 24 April 1992

Edited by

ROBERT TRAPPL
University of Vienna
and Austrian Society for Cybernetic Studies

World Scientific
Songapore • New Jersey • London • Hong Kong