SOFTWARE FOR COMPUTER CONTROL 1986

Edited by
D. FLORIAN and V. HAASE

SOFTWARE FOR COMPUTER CONTROL 1986

Selected Papers from the Fourth IFAC/IFIP Symposium, Graz, Austria, 20–23 May 1986

Edited by

D. FLORIAN

and

V. HAASE

Institut für Maschinelle Dokumentation, Forschungsgesellschaft Joanneum, Graz, Austria

Published for the

INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL

by

PERGAMON PRESS

OXFORD · NEW YORK · BEIJING · FRANKFURT
SÃO PAULO · SYDNEY · TOKYO · TORONTO

U.K. Pergamon Press, Headington Hill Hall, Oxford OX3 0BW, England

U.S.A. Pergamon Press, Maxwell House, Fairview Park, Elmsford, New York 10523, U.S.A.

PEOPLE'S REPUBLIC OF CHINA Pergamon Press, Room 4037, Qianmen Hotel, Beijing, People's Republic of China

FEDERAL REPUBLIC OF GERMANY Pergamon Press, Hammerweg 6, D-6242 Kronberg, Federal Republic of Germany

BRAZIL Pergamon Editora, Rua Eça de Queiros, 346, CEP 04011, Paraiso, São Paulo, Brazil

AUSTRALIA Pergamon Press Australia, P.O. Box 544, Potts Point, N.S.W. 2011, Australia

JAPAN Pergamon Press, 8th Floor, Matsuoka Central Building, 1-7-1 Nishishinjuku, Shinjuku-ku, Tokyo 160, Japan

CANADA Pergamon Press Canada, Suite No. 271, 253 College Street, Toronto, Ontario, Canada M5T 1R5

First edition 1987

British Library Cataloguing in Publication Data
Software for computer control, 1986:
selected papers from the Fourth IFAC/IFIP symposium, Graz, Austria, 20–23 May 1986.
———(IFAC proceedings series; 1987 no. 4).
1. Automatic control———Data processing
I. Florian, D. II. Haase, V. H.
III. International Federation of Automatic Control IV. International Federation for Information Processing V. Series
629.8'95 TJ213
ISBN 0–08–034083–0

These proceedings were reproduced by means of the photo-offset process using the manuscripts supplied by the authors of the different papers. The manuscripts have been typed using different typewriters and typefaces. The lay-out, figures and tables of some papers did not agree completely with the standard requirements: consequently the reproduction does not display complete uniformity. To ensure rapid publication this discrepancy could not be changed: nor could the English be checked completely. Therefore, the readers are asked to excuse any deficiencies of this publication which may be due to the above mentioned reasons.

The Editors

Printed in Great Britain by A. Wheaton & Co. Ltd., Exeter

IFAC

International Federation of Automatic Control

SOFTWARE FOR COMPUTER CONTROL 1986

IFAC Proceedings Series, 1987. Number 4

IFAC PROCEEDINGS SERIES

Editor-in-Chief
JANOS GERTLER, Department of Computer and Electrical Engineering, George Mason University, Fairfax, Virginia, USA

GERTLER & KEVICZKY (*General Editors*): A Bridge Between Control Science and Technology (*Ninth Triennial World Congress, in 6 volumes*)
- Analysis and Synthesis of Control Systems (*1985, No. 1*)
- Identification, Adaptive and Stochastic Control (*1985, No. 2*)
- Large-scale Systems, Decision-making, Mathematics of Control (*1985, No. 3*)
- Process Industries, Power Systems (*1985, No. 4*)
- Manufacturing, Man–Machine Systems, Computers, Components, Traffic Control, Space Applications (*1985, No. 5*)
- Biomedical Applications, Water Resources, Environment, Energy Systems, Development, Social Effects, SWIIS, Education (*1985, No. 6*)

BARKER & YOUNG: Identification and System Parameter Estimation (1985) (*1985, No. 7*)

NORRIE & TURNER: Automation for Mineral Resource Development (*1986, No. 1*)
CHRETIEN: Automatic Control in Space (*1986, No. 2*)
DA CUNHA: Planning and Operation of Electric Energy Systems (*1986, No. 3*)
VALADARES TAVARES & EVARISTO DA SILVA: Systems Analysis Applied to Water and Related Land Resources (*1986, No. 4*)
LARSEN & HANSEN: Computer Aided Design in Control and Engineering Systems (*1986, No. 5*)
PAUL: Digital Computer Applications to Process Control (*1986, No. 6*)
YANG JIACHI: Control Science and Technology for Development (*1986, No. 7*)
MANCINI, JOHANNSEN & MARTENSSON: Analysis, Design and Evaluation of Man–Machine Systems (*1986, No. 8*)
BASANEZ, FERRATE & SARIDIS: Robot Control "Syroco '85" (*1986, No. 9*)
JOHNSON: Modelling and Control of Biotechnological Processes (*1986, No. 10*)

TAL': Information Control Problems in Manufacturing Technology (*1987, No. 1*)
SINHA & TELKSNYS: Stochastic Control (*1987, No. 2*)
RAUCH: Control of Distributed Parameter Systems (*1987, No. 3*)
FLORIAN & HAASE: Software for Computer Control (*1987, No. 4*)
MARTOS, PAU & ZIERMANN: Modelling and Control of National Economies (*1987, No. 5*)
GENSER, ETSCHMAIER, HASEGAWA & STROBEL: Control in Transportation Systems (*1987, No. 6*)
ADALI & TUNALI: Microcomputer Application in Process Control (*1987, No. 7*)
WANG PINGYANG: Power Systems and Power Plant Control (*1987, No. 8*)
BALCHEN: Automation and Data Processing in Aquaculture (*1987, No. 9*)
YOSHITANI: Automation in Mining, Mineral and Metal Processing (*1987, No. 10*)
GEERING & MANSOUR: Large Scale Systems; Theory and Applications (*1987, No. 11*)
ROOS: Economics and Artificial Intelligence (*1987, No. 12*)
TROCH, KOPACEK & BREITENECKER: Simulation of Control Systems (*1987, No. 13*)
KAYA & WILLIAMS: Instrumentation and Automation in the Paper, Rubber, Plastic and Polymerization Industries (*1987, No. 14*)

IFAC Related Titles

BROADBENT & MASUBUCHI: Multilingual Glossary of Automatic Control Technology
EYKHOFF: Trends and Progress in System Identification
ISERMANN: System Identification Tutorials (*Automatica Special Issue*)

FOURTH IFAC/IFIP SYMPOSIUM ON SOFTWARE FOR COMPUTER CONTROL

Organized by
Institute for Information Processing, TU Graz
Institute for Machine Aided Documentation, FGJ Graz
Austrian Academy for Advanced Management (ÖAF)

Sponsored by
The International Federation of Automatic Control
Committee on Computers
The International Federation of Automatic Control
Committee on Education
The International Federation for Information Processing
Committee on Computer Applications in Technology
The Austrian Ministry of Science and Research
The Government of Styria
The City of Graz

International Program Committee
G. Bull, UK
B. Cronhjort, Sweden
P. Elzer, FRG
G. Ferrate, Spain
C. Foulard, France
R. Gellie, Australia
J. Gertler, Hungary
T. Harrison, USA
R. N. Henry, UK
H. Holt, Denmark
R. Isermann, FRG
E. Knuth, Hungary (Chairman)
N. Kopacek, Austria
H. Kopetz, Austria
J. Kramer, UK
T. Lalive, Switzerland
R. Lauber, FRG
N. Malagardis, France
M. Mansour, Switzerland
E. A. Puente, Spain
M. Rodd, South Africa
W. Schaufelberger, Switzerland
J. Sedlak, Czechoslovakia
B. Tamm, USSR
J. D. N. van Wyk, South Africa
N. Weinmann, Austria
T. J. Williams, USA

National Organizing Committee
K. Bermann
V. H. Haase (Chairman)
R. Hammer
H. Kraus
E. Pillhofer
E. Sos

FOREWORD

These proceedings of the fourth IFAC/IFIP Symposium on Software for Computer Control contain 4 invited, 6 plenary and 27 submitted papers presented at the event in Graz/Austria in May 1986. The records of 4 panel discussions complete the scientific material. Because of a non-functioning tape recorder we were unable to prepare the panel discussion on distributed control systems chaired by G. Suski.

The main topics are artificial intelligence methods in control-software, real-time languages, CAD-techniques, distributed control systems, control system design, theoretical foundations and industrial applications. Of special importance were papers and discussions dealing with MAP, as well as the closing panel on the position of control engineers in society.

The symposium was attended by approx. 100 experts from more than 20 countries. Many individuals and institutions have contributed to make the event a successful one, especially members of the Organizing Committee and the International Programme Committee chaired by Elöd Knuth, the Austrian Academy for Advanced Management and the Institutes for Information Processing (IIG) and for Machine Aided Documentation (IMD). The symposium was supported by the Austrian Ministry of Science and Research, the Government of Styria, the City of Graz, the Technical University of Graz, the Forschungsgesellschaft Joanneum, by IBM Österreich GesmbH and ITT Austria GesmbH.

The meeting was an event where personal friendships could be established and valuable technical discussions held. The editors hope that this volume will make a major contribution to the scientific community.

We are indebted to Ms. A. Werner for her patience during the preparation of this volume.

Doris Florian

Volkmar H. Haase

CONTENTS

SOFTWARE PROJECT MANAGEMENT

P. F. Elzer

Brown, Boveri & Cie., Central Research Laboratory (ZFL/L3), POB 101332, Heidelberg, FRG

Abstract. The paper gives an overview over methods and tools for the management of the software life cycle and a report on the 'First IFAC/IFIP Workshop on Experience with the Management of Software Projects' in Heidelberg, May 14 - 16, 1986. It mainly emphasizes non-technical aspects, like e.g. cost-estimation, productivity, planning, and human factors. In the technical parts it is tried to draw attention to aspects which are usually neglected, like e.g. test and verification or application dependent selection of tools and languages. Finally an attempt is made to identify technological trends in the development of software tools.

Keywords. Computer organization, Computer applications, Computer software, Human Factors, Programming languages, Programming tools, Software development, Software management, Software specification, Software testing.

1. INTRODUCTION

This paper has a twofold origin. It is a summary of the author's experience and observations in the field of software project management as well as a report on the 'First IFAC/IFIP Workshop on Experience with the Management of Software Projects', which had been initiated by the author and held in Heidelberg from May 14 to 16, 1986 and which yielded some quite interesting results.

Some of the ideas presented here can be traced back to discussions at the first workshop on the Software Environment for Ada at the University of Southern California at Irvine (lit.1). This event was intended to be a source of technical ideas for features and components of a common 'Ada programming environment' for all Ada applications.

But already during that workshop several of the speakers criticized its 'technological' point of view and argued strongly in favour of approaches which should also take into account non-technical aspects of software management. But at that time this remained a minority opinion. The majority of the software specialists at the workshop strongly favoured a purely technological solution and especially one which provided complete coverage of the software life cycle by computer based tools. Nevertheless, the author, who had had some previous experience with the design and application of software tools, used the opportunity of a study trip through the USA in order to investigate further into problems of the interaction between software tools and the economic and social environment in which they were used. He summarized his impressions in a paper: 'Observations on Existing Software Environments' (lit.2). A small part of his findings was also incorporated in the official requirements specification for the Ada software environment: 'Pebbleman' and 'Pebbleman revised' (lit.3, 4). But unfortunately time was not yet ripe for such ideas and therefore the 'official' position had to remain purely technology oriented for some years to come.

In the meantime the author continued to collect material and experience with the subject and, to his satisfaction, saw similar views emerge in many places in the 'software scene'. Finally, IFAC got interested in the subject and agreed to sponsor the first workshop on this matter.

It turned out to be a success. It not only demonstrated the feasibility of assembling enough managers and have them present and discuss their experience, but it also showed that by far the majority of them liked this opportunity and expressed the wish to make this type of conference an established and regular event. The organizers of the workshop have therefore started the necessary steps towards this goal.

2 RESULTS OF THE WORKSHOP

2.1 Invited Papers

There were four invited papers. Their contents had been discussed beforehand with the authors in order to make them fit together and to highlight certain topics out of the broad area of software management. They were:

Heidi Hennenberg, Krupp-Atlas-Electronics, Bremen, FRG:

Software Project Management - there is more to it than just technology.

This paper really 'set the tone' for the

whole workshop. It showed that there is no technological 'catch-all' for the solution of all problems in the development of systems and software. One has to see them in context and to try to solve them by means of a balanced and consciously chosen system of methods and tools.

David J.L.Martin, Brown,Boveri & Cie, Mannheim, FRG:

Practical Improvements in the Management of Real-Time Software Projects.

D.Martin gave a very clear and well worked-out summary of one big project he had lead to success and another one he was just preparing. The paper was so well prepared and presented that it was afterwards rated by the audience as 'best paper' and proposed for publication in 'Automatica'.

Malcolm Key, British Telecom, Ipswich, UK:

The Reasons why Software has a Bad Name.
In this presentation the author gave an overview over various management tools and techniques which are necessary for the conduct of large projects. It was an important paper insofar as it highlighted some of the points which are usually forgotten when a project is prepared.

Per Svensson, Swedish Defense Res. Inst., Stockholm, Sweden:

Creative Research and Product Development in Software Projects - the CANTOR Experience.

This paper was fascinating because it described in a completely honest way the view of a person who actually had to do the production work and who had to live with the management theories and methods about which others talk.

2.2 Submitted Papers

In total there were sixteen submitted papers, covering e.g. the following topics:

- Differences between the mentality of managers and that of engineers
- The reaction of people to new rules and guidelines
- Aspects of embedding a software development organization into a large user organization
- Experience with the use of special tools in real application projects
- Validation of Software
- Summaries of the experience with several completed projects
- Interrelations between technical development and its political and economic environment

2.3 Subgroup Discussions

Triggered by the intensive interest of the participants it was spontaneously decided to organize subgroup discussions on the following topics:

- Re-use of Software, chaired by Dr.P.J.L. Wallis from the University of Bath, UK
- Human Factors, chaired by Dr.W.Bergstroem from Elkraft Power Co.,Denmark
- Future Trends, chaired by the author of this paper

The results of the subgroup discussions were presented to the plenum by the respective chairmen and discussed there. They will be included in the proceedings of the workshop.

2.4 Questionnaires

During the workshop two questionnaires were distributed in order to sound the opinion of the participants and to collect some data on tools used, productivity factors etc. This action also turned out to be a remarkable success. The general criticisms were very positive and constructive and the data collected were so numerous that it will take some time to evaluate them properly. Therefore only part of them will be included in the proceedings. However, a few of them have already been evaluated for this report in order to give an impression of the results.

2.5 Conclusions

In the closing session of the workshop it was tried to formulate some conclusions. The following positive findings were generally agreed upon by the participants:

- Good management of software projects has to take into account more aspects than just tools and technology
- During the past years there has been a significant evolution in the technology and it is still going on.
- The motivation of the people who are involved in a project is extremely important.
- With a balanced system of tools, management and motivation successes have been achieved and can be proved.

But there were also negative results:

- As far as tools are concerned, still more evolution is necessary; the current tools all are not quite sufficient. In particular there are problems with interfaces between tools.
- Overly rigid (top-down) phase models have turned out to be counterproductive.
- There is a lack of quantitative data on productivity in software development and too little knowledge of the factors which influence it.

In the following sections it shall now be tried to give a very condensed overview over the whole software development process. As, of course, a thorough discussion of the subject would easily fill a book, the author had to concentrate on a few specific points which he thought to be of prime importance. Thus, this presentation has a strongly subjective character, but many discussions have convinced the author that his ideas are a quite good approximation of the opinion of many practitioners in the field.

3 ESSENTIAL ASPECTS OF SOFTWARE DEVELOPMENT

3.1 The Ingredients of a Successful Software Project

The following list comprises some of the important aspects of software development, which have to be considered if a project is to be successful:

Technology

The Design

- Adequacy
- Modularity
- Adaptability

The Software Development Environment

- Specification Tools
- Programming Languages
- Test and Verification
- Documentation

Support Hardware

- Development Machines
- Special Test Environment

Management

Organization

- Project phases
- Planning
- Cost Estimation
- Teams and Structures
- Influencing Factors

Human Factors

Support Tools

- Documentation
- Reporting
- Checkpoints

It should always be borne in mind that the technological and the management aspects are of equal importance. It is also a fact that most of them have been individually investigated and are well known and covered by literature, courses, etc. But obviously it is not yet common knowledge that they all together have to form a 'management system' and that in general they are even interrelated.

This can e.g. be exemplified by the development of the phase model:

3.2 The Phase Model

The phase model was originally derived from management considerations and later turned out to be a useful framework for the construction and classification of tools. For some time it was even considered as a technological dogma. People now begin to understand that both aspects are interwoven and interdependent.

During the workshop it was confirmed that an overly rigid phase planning was counterproductive, but that a reasonably phased structure of a project is necessary and useful. It was stressed that the usual phase plan has to be extended by a phase of thorough planning. The recommended phase model therefore looks approximately as follows (cf.lit.5):

- Planning and establishment of the management structure
- Establishment of quality assurance mechanisms
- Definition of the requirements
- Design specification
- Design and coding
- Unit test
- Integration
- System test and validation
- Maintenance

3.3 The Importance of Good Design

With good reasons the design has been mentioned first on the list in section 3.1. In principle it should be a matter of self-understanding that for the development of computer and software systems good design is as important as for any other technical product. The best tools and the most capable manager can not save a project in which the design of the product is bad or even wrong. But until now software development has mainly been regarded under the aspect of the development (or production) environment, i.e. programming languages, specification tools, test and verification, and, maybe, a little documentation. Such a view would appear utterly strange to e.g. a mechanical engineer. Of course he also has to think about the tools with which to produce e.g. a car, but primarily he is concerned with designing a good and affordable car. The production facilities are then constructed according to the requirements of the product and financial considerations.

But unfortunately very little is known about what is really good software design! Besides, the author believes that this problem can not be solved by software specialists. In the first place the quality of software is determined by the properties and the requirements of the application. To stay within the abovementioned example, knowledge about production methods may help to make a car cheaper or, maybe, less prone to rust, but it will certainly not improve e.g. its roadholding or the fuel consumption. So the manager will have to apply a few principles against which to check the quality of a design. The most important ones seem to be those listed in section 3.1.

In the first place the design has to be adequate to the problem. It must neither be too futuristic nor overly conservative. One must not take unnecessary development risks by trying an unknown problem solution on, let's say, a new generation of computers. But one must also avoid 'obsolescence on delivery'.

Then a design must be modular. This has technical reasons as well as organisational ones. From a technical point of view it is well known that a modular system is easier to design, to understand and to maintain than a monolithic one. Under managerial aspects it is necessary to prepare for the necessity to develop a system in a team, i.e. to be able to assign well separated work modules to different people or subgroups.

Finally a design has to be adaptable to change. This does not only relate to changes 'after delivery', which Parnas may have had in mind when he postulated his 'Design for Change', but also with changes which will, inevitably, occur already in the development phase. Inevitably, because software projects usually take much longer than people expect. Everybody talks about an 'innovation rate' which is supposed to be between 2-3 years, but statistics teach us that the average software project of nontrivial size takes between 5-8 years! And - that was the remark of one of the speakers at the workshop - the lifetime of a government is usually four years. So one may well face drastic changes of the environment in the middle of the de-

velopment phase.

4 ORGANISATIONAL ASPECTS

4.1 Planning

Everybody agrees that planning is necessary, and in every project it is done in some way. But some mistakes are quite common.

Firstly, planning is obviously not taken seriously enough. This observation has already been described in the book 'The Mythical Man-Month' by F.Brooks (lit.6). Brooks describes how project teams are usually built up too fast and planning is regarded as a kind of 'side-activity' for the manager during the early project phases. But the bulk of his time is consumed by instructing all the new people and assigning work packages to them, which consequently are only partially thought out and often uncoherent because planning has not been completed. From this an important rule can be derived: Do not start a software project of non-trivial size with a fully staffed team, but allow for a planning phase, in which a few - but very good - people prepare the project by thorough planning and architectural design!

Secondly, planning tools are not used properly. They are either not used at all or - to the contrary - adhered to too strictly, which, in turn, leads to inevitable frustrations and to abandoning them after some time. It was generally agreed that it was better to use planning tools than to work without them, but that they should only be loosely connected to the project and used as 'guidelines' and 'early warning systems'. So e.g. PERT-diagrams were not rated very highly, because they led too much into detail and were difficult to adapt, but bar-charts were generally regarded as very helpful.

4.2 Cost Estimation

4.2.1 Productivity and Cost Models

This was generally regarded as one of the most important issues in the management of software projects. In the USA it has been discussed in conferences for several years and there is a lot of literature dealing with this subject. A number of 'cost-models' have been developed which try to take into account as many influencing factors as reasonably possible and therefore sometimes have become quite complex. But obviously none of them has succeeded in really giving precise and reliable forecasts.

On one occasion the author has undertaken a comparison of such cost-models and found out that a 'rule-of-thumb', i.e. 'estimate the possible size of the code in your project and divide it by the productivity of our team', yields nearly exactly the mean value of the forecasts of a number of more or less complicated cost-models. The rule-of-thumb performed even better if one applied the usual statistical error boundaries to the estimates of code size. Of course it is common knowledge by now that the linear relation between code size and project cost does not hold any more for really big software, but the cost models did not do any better there. But this difficulty can be overcome by a modular design with loosely coupled components. The explanation for this can be found in the work of Halstead (lit.7), who had discovered that the effort - which is expressed by cost - for the development of a piece of software is not a linear function of the size of the software, but grows according to some exponential relation. The reason for this is that the true cause for the effort necessary is the internal complexity of the software, which also grows exponentially with the size. But he also showed that modularization can drastically reduce the effort, because the total effort necessary for some big software system can be computed as the sum of the effort necessary for all the modules instead of the exponential result one would get from applying his formulas to the whole piece of software.

Another important principle, which has first been described in great detail by Brooks (lit.6), but which is forgotten every time a project becomes critical, is: 'Adding manpower to a late project makes it later', or more generally: there is an optimal team size which must not be exceeded if the project is to be completed in a reasonable time. The reason for this is that humans, who work together in a team, have to communicate in order to get the common work properly done. This communication takes time and this time decreases the 'productivity' (e.g. measured in lines of code per man-year). But as it is clearly impossible to realize a 100 man-year project by one person who is allowed to work 100 years, one has to allow for these 'communication losses'. But one has to know that they exist and to organize the team in such a way that they do not exceed a tolerable amount of the total time budget. Modern cost models obviously take this into account, as Fig.1 shows. This figure is taken from (lit.5) and has been computed using the SLIM model (lit.8).

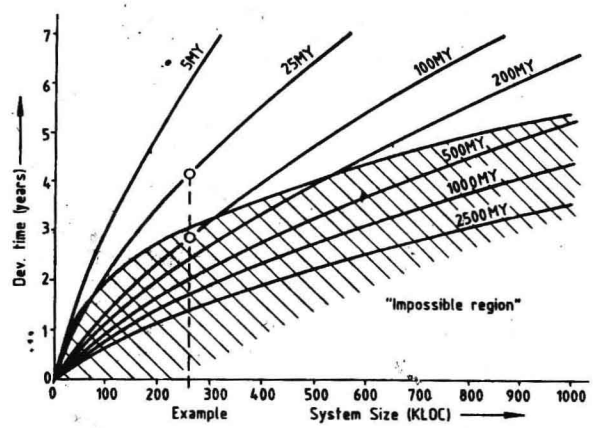


Fig.1: SLIM - Diagram (from: M.Key (lit.5))

An important aspect of this figure is described by M.Key as follows: 'It also shows an Impossible Region. Faced with this evidence it is more difficult for the senior manager to say: "Well, if you can't do it, I will find someone who can!". Clearly, management must attempt to achieve an end date determined by a market window; what it must not do is go into the 'impossible' region of the graph! Therefore, the plans must be realistic in their timescales and have a degree of flexibility which can accommodate slip.'

But Fig.1 also shows another, very important, aspect. From the manpower curves one can see how to do the same work with much less effort just by allowing for a little

more time! E.g. one can produce 250 K of software using 25 man-years or 100 man-years. In the latter case one has even slipped slightly into the impossible region, i.e. it will be a very tough project. And the saving of time is less than 30%, whereas from a naive point of view one would expect 75%. This observation is confirmed by Fig.2, which has been taken from a study by IBM (lit.9).

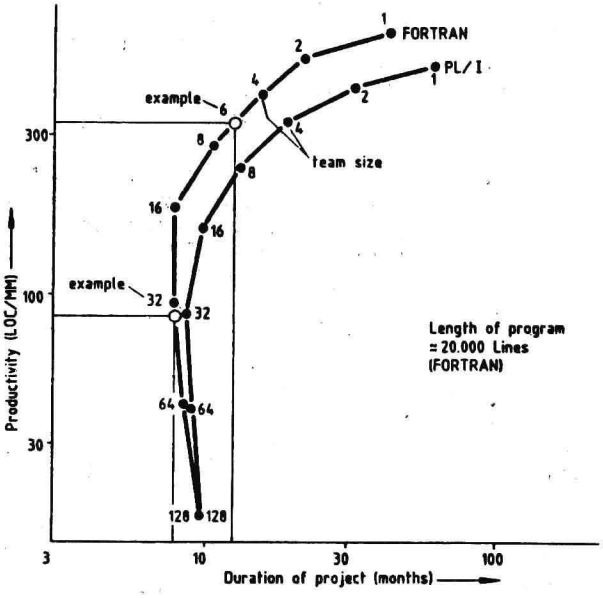
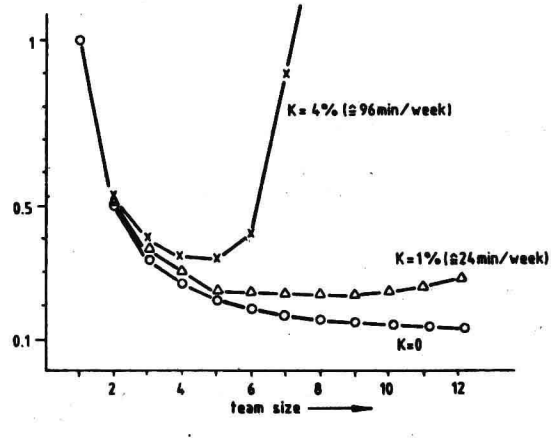


Fig.2: Dependence of Productivity on team size (adapted from (lit.9))

If in Fig.2 one locates the team sizes which result from the above figures, i.e. approx. 6, resp. 33 people, on the curve for FORTRAN (empty circles), and looks at the resulting values for productivity and project duration the values of Fig.1 are confirmed: the productivity of an individual in team of 6 is four times that of one in a team of 33, and the gain in project time is approx. 30%. Thus one obviously has detected a rather solid 'law of nature'. This law has first been described by Brooks, too, who also found an explanation for it: It is the communication of people which is necessary in a team! He also gave a formula describing this effect in quantitative terms. This formula and some of its results are plotted in Fig.3, which illustrates in a nearly dramatic way one of the central problems of management of programming teams.



"Brooks's Formula"

$$N_T = \frac{n^2 - n}{2}$$

n : Number of team members
N : Number of un-organized communication channels within team
t : Normalized duration of a project
K : Percentage of communication

Fig.3: Dependence of project duration on team size

Even with the modest amount of 1% of her or his time for one team member talking to another one the optimal team size is as small as 6 - 8! Even with this team size there are 15 - 28 'communications' per person and week, which consume 6 - approx.11 hours per week of each person's time. Brooks concludes that, as you can neither forbid communication completely nor have every project been conducted by just one person, one has to organize communication. He describes several methods for this purpose, but it would by far exceed the framework of this paper to describe them here. Thus the author would really encourage the reader to study the book of Brooks and to try to map the solutions described there on the own management problem. Of course a general principle should be applied in this case as well as in every other one: Do not try to adapt other people's methods or experience to your problems without reflection and proper adaptation!

But even if one has thoroughly understood the problems connected with the management of sizeable teams and on top of this is a gifted 'leader' who really can get his people to work, the problem of a reliable original estimate of the costs of a project remains. Obviously there do not exist really reliable data on programmer productivity, etc. The manager normally has to take recourse to his own experience. But there are possiblities to check this experince for plausibility and to compare the performance of the own team or company with the outside world. First one can browse through published literature for figures, can talk to colleagues and calculate backwards from competitor's prices and/or project durations. But there is also compiled material: Nearly everything which Barry Boehm publishes (e.g (lit.11,12)) contains valuable figures and reference information. But a less widely known book has turned out to be a really invaluable source of raw data: Montgomery Phister: Data processing, Technology and Economics (lit.9). This book contains innumerable statistics, collected over a period of approx. 15 years and covers all aspects of data processing from computer production to program development. Besides it is apparently updated in regular intervals.

Workshops like that in Heidelberg serve two purposes in this context: They allow rapid collection of data and they provide a forum for organized communication. How well this works, is shown in Fig.4* which is a plot of productivity figures collected by means of a questionnaire during the Heidelberg workshop.

The bandwidth of the results corresponds very well with values obtained from other sources:

2500 - 3500	LOC/MY	Workshop average, 1986 (*)
4000	LOC/MY	author's own experience, difficult FORTRAN code, 1984
2000	LOC/MY	author's own experience, difficult Assembler code, 1982
3200	LSC/MY	(lit.12), 1977

*see page 11

(* : A later, more thorough evaluation of the workshop results showed a wider distribution: 2700 +/- 900 LOC/MY)

4.2.2 Influencing Factors

But these figures cannot be applied uncritically and schematically. One also has to take into account the most important factors which influence the productivity of program designers. The most complete collection and evaluation of such factors can also be found in (lit.12). There 30 influencing factors have been listed and their effect evaluated. Those with the highest values have been listed below:

1 Complexity of customer interface 4.0/1.0
2 Experience with programming language 1.0/3.2
3 General qualification of personel 1.0/3.2
4 Experience with application 1.0/2.8
5 Designer participation in specification 1.0/2.6
6 User participation in requ.def. 2.4/1.0 (!)
7 Experience with computer used 1.0/2.1
8 Complexity of application algorithm 2.1/1.0
9 Percentage of delivered code 1.0/2.1/1.7 (!)
10 Limitations of working memory 2.1/1.0

Some other factors, which usually enjoy high estimation among theorists are of less influence than expected, e.g.:

29 Complexity of control flow 1.4/1.0(!)
30 Module size 1.25/1.0/1.35

The figures indicate productivity and are to be read as follows: The first figure holds if the respective factor is smaller than normal, the last one, if it is greater than normal. The middle figure (if given) describes productivity under normal conditions. So e.g. can very complex relations to a customer, i.e. something which depends on talent for negotiations and on the quality of contracting department, decrease productivity to a quarter of a good value! On the other hand, if one can assemble a team of qualified people who are familiar with the application and the programming language (factors 2, 3 and 4), one theoretically has a chance to complete a given project 25 times as fast as under adverse conditions. The purely technological factors, e.g. 29 and 30 are of remarkably small influence. The effect of the factors 6, 9 and 29 is counterintuitive, i.e. experience and statistics show different results than what has always been expected from theoretical discussions.

In general this list easily explains why the reported productivity figures of programmers can vary by a factor of 20! And for a manager, who wants to do reliable planning, this means: Observe your team, keep your own statistics, monitor your influencing factors and apply a reasonable safety margin in your estimates!

D.Martin (lit.13) also described and evaluated the influencing factors which had been relevant to his projects. Though he did this only qualitatively, the results confirmed the values given in the above list.

4.2.3 Distribution of Effort over Project Duration

As already mentioned above, one should never start a sizeable project with a fully staffed team. But what, then, is a reasonable distribution of manpower over the duration of a project?

One curve actually observed in a successful project is shown in Fig.5* which is taken from (lit.13). It shows that a successful Project of over 100 man-years has actually been prepared by two people over one year! A more qualitative approach is used in Fig.6*. This diagram, however, illustrates the reasons for such a curve by indicating the order and the overlapping of activities in a software project. Less detailed, but supported by good statistics, are the values given in (lit.9):

Program design:	26	(% of total
Coding:	24	project
Testing:	36	effort)
Documentation:	14	

The author found that this subject is quite well covered by statistics from various sources, but to mention them all would also exceed the framework of this paper.

However, two aspects shall be emphasized here:

Firstly, it is important to know about the average values of such distributions, because one needs them for reliable estimates. The reason for this is that reliable productivity figures can usually only be obtained for certain phases of the development cycle. One therefore has to extrapolate the total project costs from known figures for certain phases by using the statistical values for the other phases given in such distribution curves.

Secondly, the usual curves illustrate the maintenance problem. In whatever statistics one consults, maintenance costs usually amount to 50 % of the total lifecycle cost of software. So, one has to be prepared to set aside a group of 10 people for the maintenance of a software system which cost 100 man-years to develop! Of course this situation is by no means acceptable and therefore every effort should be made to reduce the maintenance costs of software by better design and good tools.

4.3 Human Factors

4.3. General

It may look surprising to find this subject in a paper which started out with cost models and statistics. But it is one of the most important points a manager has to observe. All his planning and statistics will be utterly in vain if he does not succeed in keeping his team together and in maintaining a reasonable degree of job satisfaction and productivity.

There are many factors which influence this. H. Hennenberg already described some of them in her overview paper (c.f. 2.1). The 'Human Factors' subgroup then discussed the matter in greater depth and came up with a series of recommendations, the most important of which

*see page 11

shall be explained in the following paragraphs.

4.3.2 Motivation of the Team

To achieve this, the following factors were judged to be most important:

- The team has to have a fair chance of success. That means that plans and schedules have to be feasiple and realistic.

- The individual team member has to have a feeling of importance. Never let the feeling arise that she or he is just regarded as a cogwheel which can be thrown away and replaced at any time.

- The manager has to show an adequate response to the needs of his team. This means in the first place a proper working environment, but also includes the necessity to be able and willing to help people with their private problems as far as reasonably possible.

- Always maintain a slight overload. This aspect was first emphasized by a participant from Japan, but then it was generally accepted that people perform better and feel more satisfied if the manager makes them achieve a little more than they orininally expected by themselves. The author, too, has observed this effect over more than a decade and therefore can support this view.

4.3.3 Team Building

The manager should perform a thorough interest analysis of his (future) team-members. In a profession like program development, which mainly depends on ideas and organization of thoughts, the performance of an individual obviously can vary by a factor of 10 - 20, depending if she or he is employed in the right place or not. And thus job satisfaction becomes an economically much more relevant factor than in many 'traditional' professions.

- Professional ethics and morality are more important than usually. Because complete testing and traditional quality control are not very well developped as far as software is concerned and even simply impossible in big systems, the commitment of the individual to do the very best job she or he can do, becomes an extremely important economic factor. This simply follows from the fact that a thoroughly developped program costs less in maintenance and damages caused by malfunctioning.

- On the other hand, the manager has to maintain visibility of the work of his team members in order to be able to properly perform his control functions and to start corrective actions in time.

4.3.4 Dealing with Conflicts

- Firstly, identify and solve conflicts soon. This seems to be an old and well-known rule for team-leaders. But software people and managers mostly have a predominantly technical background with little training in management and human factors, and therefore traditional rules of leadership are not very well known among them.

- Secondly, be prepared to create pain. Technical conflicts can very rarely be resolved by a compromise and somebody has to lose.

- But, also do not try to avoid conflicts at any price. Conflicts are good for evolution (this has long been discovered by philosophers) and, if handled properly, may even help those who lose one. They may win the next time.

4.3.5 Keeping Balance

One of the findings of the Human Factors Subgroup, which was found most interesting by the audience, is illustrated in Fig.7. The manager has to be aware that humans are controlled by a field of tension in which they try to maintain a kind of equilibrium. It should be an interesting exercise for the reader to interprete this diagram for himself.

Fig.7: The psychological equilibrium

Another interesting observation is illustrated by Fig.8. There seems to be a correlation between the skill-level of team members and the number of meetings held. The consequences of this observation are not clear, because on one hand meetings are good for communication, problem solving and conflict resolution but on the other hand too much communication degrades productivity, as described in 4.2.1.

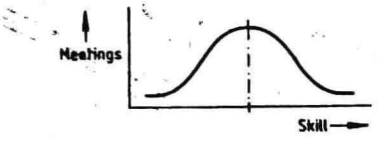


Fig.8: Dependence of number of meetings on skill level

4.3.6 Special Properties of Software Teams

Since decades a discussion is going on among software professionals whether program development is a production activity like any other or whether it is something special to which traditional rules of management do not apply. The author holds that program development is comparable to traditional planning activities and that therefore software managers can learn a lot from architects who plan large buildings, or from administration in civil service, railroads or military logistics.

A particular aspect of this has been put forward by the Human Factors Subgroup at the Heidelberg Workshop:
The majority of Software professionals (at least in Europe) holds university degrees, if although mostly not in software. This means that they have been educated in a tradition where they are judged for obtaining unique results. Usually university graduates also have never learned the necessity of strict rules. Both backgrounds make it difficult to build sizeable teams out of such people.