Ian Sommerville Manfred Paul (Eds.)

Software Engineering – ESEC '93

4th European Software Engineering Conference
Garmisch-Partenkirchen, Germany
September 13-17, 1993
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
D-76131 Karlsruhe, Germany

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editors

Ian Sommerville
Computing Department, Lancaster University
Lancaster LA1 4YR, UK

Manfred Paul
Institut für Informatik, Technische Universität München
Orleansstr. 34, D-81667 München, Germany

CR Subject Classification (1991): D.2-3, C.3, K.6

ISBN 3-540-57209-0 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57209-0 Springer-Verlag New York Berlin Heidelberg

Printed in Germany

Typesetting: Camera-ready by authors
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Foreword

Twenty-five years ago, the first of two NATO workshops to discuss the software crisis was held at Garmisch-Partenkirchen. From these workshops, the discipline of software engineering was born. Now, the fourth European Software Engineering Conference returns to Garmisch to celebrate this 25th anniversary and to look forward to the next 25 years.

More than 135 papers were submitted to ESEC'93 out of which the 27 papers in these Proceedings were accepted. Papers were submitted from most European countries, from the US, Canada, Japan and other Asian countries. Quality was the principal selection criterion for selected papers but the Programme Committee also wanted to ensure that the papers presented included both advanced academic research and the industrial use of software engineering technology.

We have therefore a mixture of themes. Some, such as Software Engineering and CSCW, are forward looking and anticipate future developments; others, such as Systems Engineering, are more concerned with the reports of practical industrial applications. Some sessions, such as that on Software Reuse, reflect the fact that some of the concerns first raised in 1968 remain unsolved problems. The increasing importance of requirements engineering is reflected by the inclusion of two sessions on this topic. The long-term research objectives of understanding the software process and the development of effective environmental support are also covered here.

The submitted papers are complemented by presentations from distinguished invited speakers from Europe and North America. Invited papers cover real-time systems, software measurement and metrics, and industrial software engineering practice. We are particularly pleased to welcome our introductory speakers Professors F.L. Bauer and J. Buxton who played a large part in 1968 and 1969 workshops and who have made a major contribution to the development of software engineering.

The organisation of a major conference is impossible without a great deal of help and support. We are grateful to the members of the Programme Committee and to other reviewers for their time. Pere Botella and Johannes Siedersleben the Tutorial and the Tools Fair organisers have made an important contribution to the success of ESEC'93. Particular thanks also to Jacqui Forsyth at Lancaster University and to Uta Weber at the Technical University of Munich who provided invaluable help with the technical and the local organisation.

European work in software engineering is now comparable with the best in the world and in some fields, such as the use of formal methods, it is generally recognised that European work is pre-eminent. We believe that this series of ESEC conferences, first established in 1987, has contributed to this and we are confident that ESEC'93 will continue the high standard of its predecessors.

July 1993

Ian Sommerville
Manfred Paul

Program Committee

Program Chair

Ian Sommerville, Lancaster University (UK)

General Chair

Manfred Paul, Technische Universität München (Germany)

Tutorial Chair

Pere Botella, Universitat Politecnica Catalunya (Spain)

Tools Fair Chair

Johannes Siedersleben, sd &m (Germany)

Members

D. Barstow (France)
J. Bishop (South Africa)
D. Bjørner (Denmark)
P. Botella (Spain)
M. Broy (Germany)
J. Buxton (UK)
R. Conradi (Norway)
B. Dömölki (Hungary)
A. Endres (Germany)
J. Estublier (France)
C. Fernstrom (France)
A. Fuggetta (Italy)
F. Gallo (Italy)
J. Gorski (Poland)
W. Hesse (Germany)
P. Hruschka (Germany)
G. Kaiser (USA)
M. Lacroix (Belgium)
B. Lang (France)
K. Löhr (Germany)
B. Lennartson (Sweden)
N. Madhavji (Canada)
C. Montangero (Italy)
P. Navrat (Czechoslovakia)
G. Oddy (UK)
H. Obbink (Netherlands)
H.D. Rombach (Germany)
E. Rothauser (Switzerland)
A. Schultz (Austria)
M. Tedd (UK)
E. Tyugu (Estonia)
A.O. Ward (UK)
R.C. Welland (UK)
J. Winkler (Germany)

Reviewers

Z. Banaszak
D. Barstow
W. Bartsch
N. Belkhatir
R. Bentley
J. Bishop
D. Bjørner
G. Blair
L. Blair
P. Bochmann
P. Botella
A. Bradley
M. Broy
X. Burgues
J. Buxton
S. Chobot
R. Conradi
G. Dean
A. Dix
B. Dömölki
H. Ehler
A. Endres
J. Estublier
C. Facchi
C. Fernstrom
P. Forbrig
M. Fuchs
A. Fuggetta
F. Gallo
J. Gorski
T. Gritzner
R. Grosu
J. Handke
W. Hesse
P. Hruschka
H. Hussmann
Y. Ismailov
G. Kaiser
M. Lacroix
B. Lang
K. Löhr
F. Long
B. Lennartson
H. Loeper
N. Madhavji
A. McGettrick
C. Montangero
P. Navrat
D. Nazareth
H. Obbink
H. Oberguelle
G. Oddy
A. Olive
F. Orevas
M. Paul
J. Rowlands
G. Riedewald
T. Rodden
H.D. Rombach
E. Rothauser
K. Sandahl
T. Schafer
A. Schultz
E. Sharkawi
J. Sistac
O. Slavkova
I. Sommerville
K. Spiers
A. Spillner
R. Steinbruggen
K. Stoelen
M. Tedd
E. Tyugu
B. Uologh
T. Uustalu
A.O. Ward
R.C. Welland
J.Winkler

Table of Contents

Invited Papers

Requirements Specification 1

Requirements Specification 2

Formal Aspects of Software Engineering

On the Decline of Classical Programming

Professor J.N. Buxton

Chairman, Buxton-Douglas Ltd
Chairman, Room Underwriting Systems Ltd
Professor of Information Technology, King's College

Abstract. The paper puts forward the view that we are in a period of fundamental change in the nature of the I.T. business, in that the driving pressures are shifting from the supplier side to the applications user side. In part this is a long-term consequence of the unbundling of software pricing (discussed at the NATO conferences 25 years ago), and in part the consequence of the recent spectacular reductions in hardware costs. The paper discusses the consequences for business, education and research and it indicates the danger of rediscovering the problem of the "software crisis", but at a new level of programming by the applications owners.

1 Introduction

In this paper I shall put forward the view that we are in a period of far-reaching and fundamental change in the practice of Information Technology. This has been a gradual though accelerating process during the 1980s and some of its effects are now becoming apparent in the changes in the structure and activities of industry. The change involves more than a development of maturity after some 40 years since the first use of computers - a central aspect is the increasing shift to the design of computing systems by non-specialists in IT and indeed by those with little interest in the technology per se. The driving pressure on IT development is shifting from the supply side to the user side.

Necessary precursors for this change to come about have been, firstly the unbundling of software prices from associated hardware and secondly, the spectacular drop in hardware costs over the years. This revolution in electronics has given us the PC level of equipment at prices accessible to individuals. In part also the change is based on the development of new programming concepts which are nearer to the application field and which give the user direct access to the services of the machine

In my view a useful and helpful analogy can be drawn with the development of the motor car industry. In the first era before, say 1914, cars were expensive, their operation required the services of specialist engineers, they were far from reliable and their possession was a mark of status and prestige. The "owner-driver" was a wealthy

eccentric or hobbyist. All this was changed in the 1920s by the advent of the Model T Ford in the USA and of the Austin 7 in the UK. The car went from being a status symbol for the privileged and the domain of experts to a necessity for ordinary life. Costs became highly competitive and user-friendly standardised interfaces were a necessity for survival in the market. The crucial steps in this transition were an order of magnitude reduction in cost of the hardware and improvements in reliability and usability which made the car simple enough for a non-engineer to learn to use.

In the case of IT, the introduction of spreadsheet systems provides a good example of the advent of technology which made the computer accessible to owners of problems in an application field. These systems are "programmed", but in the language of the accountant and without the services of professional computer programmers. Their use has transformed computing practice.

However, this new wave of applications which are programmed by the application owners has brought its own problems. In essence the activity involved is still programming - though in a very high level language - and in the longer term the critical issue is still that known as "maintenance". All is well while applications are small and personal, but when they become large, complex and even integral to the conduct of the business, and then the author leaves, we have real problems.
There is an urgent requirement to bring to this new field of user-programmed PC applications the same principles of good software engineering as those we need in the more classical programming field. Even with spreadsheet applications and the like, we need to develop well-engineered solutions, well documented and capable of enhancements and maintenance for the full working life of the requirement, by people other than the original author.

In this paper we look first in somewhat more detail at the general features of this new era. Then we turn to an analysis of some specific developments in computer programming and the role of the "professional programmer". Finally, we embark on some speculation as to the future of programming support environments and of research in the field.

2 The transition to the new era

The climate of the first era in computing was set above all by supply side dominance and in particular by the cost of hardware. The early field of technical applications was quickly overtaken by DP applications which became the principal area of use. In general terms, computing was taken on board as a new and expensive technology which would at some time in the future be of great importance. Companies bought computers to reserve their future positions and in the short term there was a passive acceptance of their unreliability, lack of real cost benefits and user-hostile interfaces.

As hardware costs began to fall dramatically in the 1970s and beyond, software began

to be seen as the dominant cost and also the major problem. We can quote a few global estimates which have been widely circulated.

- in typical corporate applications, software costs are some 3x hardware costs;
- some 70% of systems cost is in "maintenance" - ie in repair and in enhancement of the software;
- in the US DoD, some 80% of delivered software becomes "shelfware" and is never put into use;
- the Japanese produced a national projected estimate of the need for programmers which suggested that in Japan alone, some 970,000 were needed by the year 2000.

These and many other observations combine to suggest that in aggregate terms, a scarce resource in programming skill is being utilised to a large extent ineffectively, in building expensive software which is of limited utility. This became generally known as the "software crisis".

In the 1980s a view was widely adopted that the software problem could best be addressed by a transformation of the industry, seen then as craft-intensive with inadequate basis in scientific theory or in engineering practice, into a more soundly based and capital-intensive industry. This was to be brought about by the introduction of better methods and by investment in the use of software tools -hence the research into CASE tools, IPSEs, software factories, formal methods and so on, all combined under the general title of "software engineering research".

This period of development, when hardware costs became less significant than software, was one when the supply side dominance of the market continued; though the dominant supply technology was now that of software development.

It is the central thesis of this paper that we are now undergoing a fundamental shift in the nature of the business: dominance of the supply side is now in effect superseded. On the hardware side, the major engine of change has been the development of the PC into the versatile and inexpensive professional workstation, together with the software to enable the user to make effective and direct use of the remarkable level of available computational power. The combination of the modern PC together with networking and with software packages for wordprocessing, spreadsheets, databases and so on has transformed the appearance of computing and has created a new and powerful user community throughout industry and commerce.

A further driving force for change in the current economic climate especially, is the growing realisation of the poor results in cost-benefit terms from much current IT use.

A recent report by the consultants A.T. Kearney states that 89% of British firms do not use technology successfully, and the greatest waste is in computing administration systems.

In other areas of industry the use of IT is an integral and necessary part of the business, where computers have been in use for decades and form integral components of products or production processes, eg in aerospace, banking and the process industries. Here also a process of change is going on, though less spectacular and clear-cut in nature. The process is that of gradual acquisition, by industries where IT is a necessary part of the business, of their own IT skilled personnel. In earlier days such companies relied heavily on the skills of their hardware suppliers to install their systems and on the systems/software consultants to carry out their requirements analysis and systems design. The result was the phenomenon of "locking-in" to a specific set of hardware and to the continuing services for maintenance of a specific systems house.

The rise of in-house skills, together with the spectacular development of a highly competitive market in hardware components means that users are less content now to be locked in. The pressure for open systems supply is now widespread and hardware suppliers are responding to it. Systems houses begin to see their future business as being increasingly as experts in "systems integration", ie in helping users to configure their systems from the best components available regardless of supplier. Users now expect to be able to run their selected packages or their own bespoke systems on new hardware platforms. So, to sell new hardware one must both offer transparent support to old systems and also give freedom for the user to add more software and to combine with hardware from other vendors.

The consequences in the service industry of systems, IT consultancy and software houses is also beginning to be considerable. Variations are wide, for instance between countries, but there seems to be evidence for a drift from traditional areas such as bespoke systems, with continuing supplier maintenance, towards software products which are parameterisable, user enhanceable or indeed programmable, in a sense which we discuss later, by the user.

3 Models of computer programming

The traditional model of software development was that of a multi-stage process often illustrated by a "waterfall model". Firstly, the analysis of user requirements was done by dialogue between the user and the systems analyst. The analyst then specified the system and designed the data structures and algorithms required for the solution. The results included documentation in flow charts, data layouts and so on. In the third stage, the "coding" was done by a real programmer in assembly code at or close to the level of the hardware machine.

The first great breakthrough in improvement of the process, was the development of high level languages in the late 1950s. The adoption of HLLS such as FORTRAN, Cobol and their successors took many years but led to an order of magnitude increase in productivity of the technical staff. In effect the analyst could now programme directly from the level of algorithms and data structures. The grades of systems analyst and programmer tended to merge and the domain of the "real programmer" or machine level expert is mainly now in the specialisms of operating systems or resource critical applications design.

Despite the productivity gains of high level language use, both per se in speeding up coding and indirectly in enabling increased populations to learn to "program" without having to understand detailed machine architecture, nevertheless the "software bottleneck" problem remained endemic in the 1970s and 80s.

There was an increasing strain on the scarce technical staff resource of people - now usually called "software engineers" - who are capable of thinking in IT terms of data structures and algorithms and who can transform a user requirement into an IT solution. The resource continues to be scarce because:

a) under this model of the process, the step of translation of requirements into the design of algorithms is still essential;

b) the increasing numbers of existing systems in use need "maintenance" - ie they must evolve with changing requirements, and they can only be maintained by software engineers.

There is limited scope for training more programmers. Though some largely untapped labour pools exist, it may well be that most of those with the ability to think in algorithms are already in the IT profession. Current directions of development in the area of capital-intensive software factories represent a response to the increase of demand for software skills together with the scarcity of engineers able to respond to the demand. The rationale for this response is that in other branches of engineering capital investment in the production line together with more rational approaches to the process have transformed industries from craft status with substantial success. We will suggest below that this strategy perhaps now looks somewhat outdated and overtaken by events. Indeed, one could take the view that it is based on a false analogy in that building software is a design issue; production is already largely automated by use of a programming language and compiler.

4 The next breakthrough

The development of the spreadsheet is the archetypical example of a next-generation approach. The approach is to automate the closed language of a specific application domain together with a generic algorithm for "solutions" of problems expressed in that

domain - with the spreadsheet, the language is that of the rows and columns of figures familiar to accountants, together with arithmetical relationships amongst them, and the system ensures these always balance. So, the user designs and maintains his own "program" expressed directly in the language of the application domain and the intermediate step of designing problem-specific IT algorithmic solutions goes away. When combined with hardware cheap enough for all to afford, the result is a real order of magnitude change in computer use.

The spreadsheet can be regarded as an extreme position on a spectrum of computing systems. At the other end is the classical single applications system, designed and built by IT professionals to meet a user requirement. In between are more flexible approaches which in essence involve reuse of existing software intended to be of wider use than one application - for example, parameterisable systems or systems built around reusable kernels. Such approaches are now the norm for building customised systems where the new design and coding is performed by the IT engineer.

Returning to the real breakthrough area, typified by the spreadsheet, in which IT professionals are not involved as intermediaries in specific applications, we can cite other areas where closed domains have been automated and the user expresses his own problems in domain-specific forms. Wordprocessing is perhaps the widest used example. Text processing in the desk-top publishing area is not far behind. There are further example systems in engineering design and in architecture and other fields.

We maintain that any field in which real IT knowledge is essential to build a computer application is not an example in this class. It is interesting as an intermediate case to consider the currently fashionable issue of "object-orientedness" and to enquire as to whether or not this constitutes a breakthrough. The concepts of classes of objects with sets and attributes were invented as a means of representing real world entities such as machines and vehicles in the field of discrete event simulations in O.R; an early example of the entity - attribute-set model can be found in [2].

The further and important step of the concept of class inheritance was added by Dahl and Nygaard in SIMULA 67. Though objects, attributes, sets and inheritance provide powerful modelling tools for describing, in language close to the users terms, the entities in a problem domain, their rules of behaviour still have to be expressed in algorithmic terms and so object-based applications still need IT designers.

Another major area in which a good case can be made for maintaining that the programming approach of the new era has already arrived is that of DP applications, using 4th generation languages and database construction and enquiry languages. The advance of 4th generation languages has gradually weakened the grip of traditional Cobol programming in the DP field; and this has gone largely unobserved by more technical groups and in particular by academic computer scientists. As an example of the shift to a higher level style of application language domain programming,

however, the 4th generation language area is still to some extent an intermediate case as IT knowledge is still essential to construct applications.

We do not propose to go on to consider further points on this spectrum of computer applications such as hypertext, building and vehicle structure design systems etc - suffice it to say that we think the point made that there is indeed a spectrum of a wide and important range of applications where the generation of IT solutions involves less than fully professional IT knowledge and indeed, at the spread sheet and word processing end, needs very limited IT skills.

Conclusions

A centrally important consequence of these developments concerns the projected demand for increased numbers of SW Engineers. A decade ago the rate of demand was increasing steadily and implausible predictions as to the eventual size of the programming population were made. this apparent demand has been eroded as the perceived need to develop ever more specific applications programs has been replaced in many areas by the use of domain-specific packages and user-programmable software. The quasi-potential increase in numbers has occurred: but it is in numbers of problem owners programming their own solutions directly rather than in numbers of software engineers acting as intermediaries between the problem owners and their computers.

As a corollary to this development, the market for new equipment suitable for direct use by problem owners has increased very substantially in order to provide the owners with direct access to their own machines. Sales of PCs and work stations now exceed by orders of magnitude in numbers the sales of main frames, and the PC business is now probably dominant over the mainframe business in financial measures.

Meanwhile, the need for bespoke software systems has not gone away in all application areas. There are many application domains not yet served by user-programmable packages. In particular in the high technology industries the need for large and highly application-specific software, for use in products or in production continues to grow and to be crucial to business success and even to survival.

I think we have not yet taken on board the consequences of this changed pattern of demand for software engineers in, for example, the education system. It may well be that current provisions for training engineers at the higher professional levels will turn out to be of the right order of magnitude, though at present one can only guess as statistics are not yet available. It is reasonably clear, however, that we will need to train large sections, if not most of the population in simple levels of computer familiarisation to aid them in access to the application specific systems they will need.

A further area of interest in speculating on the consequence of these recent changes

is that of the development of PSEs, integrated toolsets and methods for support of software production. These were put forward during the 1980s as the essential technical way ahead in turning software production into a capital intensive industry, thereby solving the problem of the projected shortage of software engineers. If the above analysis is correct then the problem for which PSEs were perceived as the solution has in substantial measure gone away. Incidentally the development of PSEs to date has not proceeded as rapidly as was hoped - after some twelve years since the publication of "Stoneman", [3] one sees very few real examples of fully integrated toolsets, though there are very useful CASE tools on the market.

A more promising line of development in support systems and toolsets would now seem to be in the support of flexible and rapidly configurable CASE toolsets to offer support closely aligned to local needs within an organisation which is capable of rapid adaptation as businesses change. We need to pay more attention to the engineering process and to the central requirement for flexibility in that process to meet changing demand.

The need for support systems, and indeed for a more systematic approach to building software, clearly is still a fundamental part of the business. The custom built systems to be supported now include not only the high-tech areas discussed above but also the relatively new and unexplored field of software package production. In this context we include classical parametrisable packages and also the new product ranges of user programmable systems, spreadsheets, user driven database systems and so on. The producers of these systems compete in a market place which demands high quality, fast response to client wishes, low price (which depends on high volume sales) and the market is not unaffected by the dictates of fashion. Here is an area where the concept of the "software factory" really has relevance, but it may require a different sort of factory from those envisaged in earlier years.

Let us now turn to the consequences for research in the IT field. The changes we face now are due primarily to the development, in a few areas, of domain-specific ways of enabling the problem owner to utilise computers in implementing solutions to problems, expressed to the system directly by the user without the intermediate steps of algorithmic solution and bespoke programming. Urgent questions arise as to whether other areas can be found amenable to such approaches - this perhaps is the prerogative of the IT entrepreneur rather than the academic. A proper question for research however is to determine what can be said at a more generic level about domain-specific high level programming. What if any, are its general characteristics, what hypotheses can one make and what theories might be discovered to bring order to this area?

Again and at the level of engineering development, the area of construction of large bespoke systems will continue - both for high-tech fields and in the production of package software product lines - but these areas are also not immune from the need for much deeper involvement of their "users" - albeit technically very aware and

competent users - in ensuring that such systems really meet their requirements and can be modified in step with changes to those requirements. We have here a major and relatively unexplored area for software engineering research: how do we build systems which can be maintained, modified and enhanced by their users? Some current work, notably concerned with study of the software process, is already moving in this direction.

Perhaps the most important conclusion, however, is the realisation that the level of success which we have achieved in the new area of very high level domain specific programming techniques has itself opened up a whole new area of emerging problems. Even programming a spreadsheet is still programming in the sense that an IT system is designed and built to fulfil a business purpose which will inevitably evolve and change in time. The spreadsheet model will need to be developed and maintained, and it will tend to grow both increasingly complex and increasingly essential to the business. In a word, we are about to embark on a rediscovery of all the problems of the software crisis - but at a new level of programming.

It is clear that application domain programming by users is still subject to the ailments of classical amateur programming. We observe that unwieldy, opaque and non-maintainable systems could now be even easier to produce than with classical programming. Our most urgent requirement is to bring some grasp of the elements of software engineering to this application world. We need to see the design of these systems carried out by effective engineering techniques and recorded in accessible documentation, so that such systems can be maintained and enhanced into the future. The requirements to bring this about, for instance on the educational system at all levels, are formidable indeed.

References

1. Malcolm, R., private comm., 1991
2. Buxton, J.N. and Laski, J., "Control and Simulation Language", Computer J. vol 5 no.3 (1962)
3. Buxton, J.N., "Requirements for an Ada Programming Support Environment", Department of Defence, 1980