F. L. Bauer J. B. Dennis G. Goos C. C. Gotlieb
R. M. Graham M. Griffiths H. J. Helms B. Morton
P. C. Poole D. Tsichritzis W. M. Waite

Software Engineering

An Advanced Course

Edited by F. L. Bauer

TP31
B8
(2)

7966465

F. L. Bauer J. B. Dennis G. Goos C. C. Gotlieb
R. M. Graham M. Griffiths H. J. Helms B. Morton
P. C. Poole D. Tsichritzis W. M. Waite

Software Engineering

An Advanced Course

Edited by F. L. Bauer

Springer-Verlag
New York Heidelberg Berlin

Editor
Prof. Dr. F. L. Bauer
Mathematisches Institut
TU München
Arcisstraße 21
8000 München 2/BRD

Originally published in the series
Lecture Notes in Computer Science Vol. 30
Springer-Verlag Berlin Heidelberg New York
First Edition 1973
Reprint of the First Edition 1975

AMS Subject Classifications (1970): 68A05
CR Subject Classifications (1974): 4.

ISBN 0-387-08364-2 Springer-Verlag New York · Heidelberg · Berlin
ISBN 0-540-08364-2 Springer-Verlag Berlin · Heidelberg · New York

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

Contents

PREFACE

F.L. Bauer		2

CHAPTER 1: INTRODUCTION

K.W. Morton	WHAT THE SOFTWARE ENGINEER CAN DO FOR THE COMPUTER USER	4
	1. *Introduction*	4
	2. *Program Duplication*	5
	3. *User Images*	8
	4. *Application Program Suites*	1o
	5. *Conclusion*	11
	6. *References*	11
J.B. Dennis	THE DESIGN AND CONSTRUCTION OF SOFTWARE SYSTEMS	12
	1. *Introduction*	12
	2. *Terminology*	13
	2.1. *Computer Systems*	13
	2.2. *Software Systems*	15
	2.3. *Hierarchy*	15
	2.4. *System and Application Software*	17
	3. *Description of Software Systems*	19
	4. *Function, Correctness, Performance and Reliability*	19
	4.1. *Function*	2o
	4.2. *Correctness*	22
	4.3. *Performance*	23
	4.4. *Reliability*	24
	5. *Software Projects*	25
	6. *Acknowledgement*	27
	7. *References*	27

CHAPTER 2: DESCRIPTIONAL TOOLS

G. Goos — HIERARCHIES 29

0. Introduction

1. Hierarchical Ordering as a Design Strategy 29

1.1. Levels of Abstraction 36

1.2. The Order of Design Decisions 38

2. Hierarchical Ordering and Languages 41

2.1. Abstract Machines and the Production Process 41

2.2. Hierarchies of Languages 42

3. Protection by Hierarchical Ordering 44

4. References 46

G. Goos — LANGUAGE CHARACTERISTICS 47
Programming Languages as a Tool in Writing System Software

0. Introduction 47

1. The Influence of Language Properties on Software Creation 47

1.1. Language Constructs as Models for Program Behavior 48

1.2. Influence on Programming Style and Program Documentation 49

1.3. Machine Independence and Portability 51

1.4. Portability Versus Efficiency 52

1.5. Limitations of Programming Languages 53

2. Requirements for Structured Programming and Program Modularity 54

2.1. Modularity 54

2.2. Hierarchies, Nesting and Scope Rules 56

2.3. Concurrent Processes 58

3. Data Structures in System Programming 59

3.1. Simple Values 61

3.2. Records 62

3.3. Storage-Allocation for Records 64

4. System-Dependent Language Features and Portability 66

5. Some open Problems 67

6. References 69

M. Griffiths	LOW LEVEL LANGUAGES SUMMARY OF A DISCUSSION SESSION	7o
	1. Introduction	7o
	2. Justification	7o
	3. Features	71
	4. Machine Dependence	72
	5. Efficiency	73
	6. Style and Education	73
	7. Conclusion	74
	8. Acknowledgement	74
	9. References	74
M. Griffiths	RELATIONSHIP BETWEEN DEFINITION AND IMPLEMENTATION OF A LANGUAGE	76
	1. Introduction	77
	1.1. Requirements of Different People	77
	1.2. Design of Language for good Programming	8o
	1.3. Design for Testing	82
	2. Language Definition	83
	2.1. Syntax	83
	2.2. Static Semantics	85
	2.3. Dynamic Semantics	85
	2.4. Example taken from ALGOL 6o	85
	2.4.1. Syntax	86
	2.4.2. Static Semantics	88
	2.4.3. Dynamic Semantics	92
	2.4.4. Comments on the Example	95
	3. From Definition to Implementation	96
	3.1. Semantic Functions	96
	3.2. Implementation Languages	98
	3.3. Execution Model	98
	3.4. Final Comments on Implementation	99
	4. A Look at some Definitions	1oo
	4.1. ALGOL 68	1oo
	4.2. Vienna Definitions	1o2
	4.3. Extensible Languages	1o5
	5. Conclusion	1o6
	6. Acknowledgements	1o7
	7. References	1o8

J.B. Dennis	CONCURRENCY IN SOFTWARE SYSTEMS	111
	1. Introduction	111
	2. Petri Nets	112
	3. Systems	115
	4. Determinacy	119
	5. Interconnected Systems	121
	6. Interprocess Communication	125
	7. References	127
CHAPTER 3: TECHNIQUES		128
J.B. Dennis	MODULARITY	128
	1. Introduction Concepts	128
	1.1. Definition of Modularity	129
	1.2. Modularity in Fortran	131
	1.3. Modularity in ALGOL 6o	134
	1.4. Substitution	136
	1.5. References	137
	2. Data Structures in Modular Programming	139
	2.1. Address Space and Modularity	139
	2.2. Representation of Program Modules	14o
	2.3. Linguistic Levels for Modular Programming	144
	2.3.1. PL/I	145
	2.3.2. ALGOL 68	146
	2.3.3. LISP	147
	2.3.4. Discussion	149
	2.4. References	149
	3. Modularity in Multics	151
	3.1. The Model	151
	3.1.1. The File System	151
	3.1.2. Processes and Address Spaces	152
	3.1.3. Making a Segment known to a Process	154
	3.1.4. Dynamic Linking	157
	3.1.5. Search Rules and the Working Directory	16o
	3.2. Accomplishments	161
	3.3. Unresolved Issues	162
	3.3.1. Treatment of Reference Names	162
	3.4. References	165
	4. A Base Linguistic Level for Modular Programming	166
	4.1. Objects	166
	4.2. Structure of a Base Language Interpreter	167
	4.3. State Transitions of the Interpreter	17o
	4.4. Representation of Modular Programs	177
	4.5. Use of the Model	18o
	5. References	182

P.C. Poole W.M. Waite	PORTABILITY AND ADAPTABILITY	183
	1. Introduction	184
	1.1. The Basic Principles	185
	1.2. What we can expect to achieve	185
	2. Portability Through High Level Language Coding	187
	2.1. The Need for Extensions	187
	2.2. Extension by Embedding	188
	3. Portability through Abstract Machine Modelling	192
	3.1. Background	193
	3.2. Relating the Model to Existing Computers	196
	3.3. Relating the Model to the Problem	2o3
	4. Realization of Abstract Machine Models	2o5
	4.1. Translator Characteristics	2o5
	4.2. Obtaining the Translator	2o9
	5. A Case Study of some early Abstract Machines	211
	5.1. Machine and Language Design	211
	5.2. Porting and Adapting	222
	5.3. Review and Evaluation	233
	6. Low Level Languages for Abstract Machines	234
	6.1. The Basic Hardware Model	
	6.2. A Framework for Low Level Languages	239
	6.3. An Example of a Low Level Language	25o
	7. A Hierarchy of Abstract Machines	262
	7.1. Need for the Hierarchy	262
	7.2. A Standard Base for the Hierarchy	267
	7.3. A Case Study	272
	8. References	275
P.C. Poole	DEBUGGING AND TESTING	278
	1. Introduction	278
	2. Planning for the Testing and Debugging Phases	281
	2.1. Documentation	282
	2.2. Debugging Code	284
	2.3. Generation of Debugging Code	287
	2.4. Modularity	289
	2.5. Parameterisation	292
	3. Testing and Debugging Techniques	294
	3.1. Classical Debugging Techniques	295

	3.2. Online Debugging	3o1
	3.3. Testing Strategies and Techniques	31o
	4. References	317
D. Tsichritzis	RELIABILITY	319
	1. Design and Construction of Reliable Software	319
	1.1. Introduction	319
	1.2. Influence of the Language	32o
	1.3. Semantic Checking	322
	1.4. Programming Style	323
	1.5. Influence of Protection	325
	1.6. Program Correctness	325
	1.6.1. Informal Proof	326
	1.6.2. Formal Proof	327
	1.7. Design for Reliability	328
	1.8. Reliability during the Life Cycle of the Software	329
	1.9. Summary and Conclusions	33o
	2. Protection	332
	2.1. Introduction	332
	2.2. Domains and Objects	333
	2.3. Protection Walls and Monitors	335
	2.4. Identity Cards and Capabilities	336
	2.5. Policing	338
	2.6. Describing the Protection Status of a System	34o
	2.7. Implementation	342
	2.8. A Capability Based File System	344
	2.8.1. Introduction	344
	2.8.2. Capability Format	345
	2.8.3. Packing Capabilities	346
	2.8.4. Kernel System Facilities	348
	2.8.5. Passing Capabilities	349
	2.8.6. Outline of the File System	351
	2.8.7. Facilities of the File System	351
	2.8.8. Organization of the File System	354
	3. Security	357
	3.1. Introduction	357
	3.2. Information System Approach	359
	3.2.1. Integrity of Personnel	359
	3.2.2. Authentication of Users Identity	36o

3.2.3. Protection of Data Off Line and in Transmission 360
3.2.4. Threat Monitoring 361
3.3. Data Depend nce and Data Transformations 362
3.3.1. Data Transformations 362
3.3.2. Data Dependent Access 363
3.3.3. Program Certification 363
3.4. Summary of Current Practices 364
4. References 371

CHAPTER 4: PRACTICAL ASPECTS 374

D. Tsichritzis — PROJECT MANAGEMENT 374
1. Introduction 374
2. Project Communication, Organization and Control 376
3. Project Phase 378
3.1. Proposal 378
3.2. Survey Phase 379
3.3. Design and Implementation Phase 381
4. Managing "Large" Projects 382
5. References 383

G. Goos — DOCUMENTATION 385
0. Introduction 385
1. The Needs for Documentation 386
1.1. The User's Guide 387
1.2. The Conceptual Description 389
1.3. Design and Product Documentation 390
2. Special Problems 391
2.1. Description of Data and Algorithms 391
2.2. Crossreferencing between Documentation and Program 392
2.3. Maintaining the Documentation 393

R.M. Graham — PERFORMANCE PREDICTION 395
1. Performance: Definition, Measurement and Limitations 396
1.1. What is Performance? 396
1.2. Measurement of Performance 397
1.2.1. Performance as a Function of Input 397
1.2.2. Metrics 398
1.2.3. Steady State, Transient, and overload Behavior 400

	1.3. *Limitations of Performance*	401
	1.3.1. *Inherent Limitations*	401
	1.3.2. *Economic Limitations*	402
	1.4. *Summary*	403
	2. *System Modeling*	403
	2.1. *Types of Models*	404
	2.1.1. *Analytical Models*	405
	2.1.2. *Directed Graph Models*	407
	2.1.3. *Simulation Models*	412
	2.2. *Problems in Modeling*	416
	3. *Use of models in Performance Prediction*	418
	3.1. *Problems in using Models*	418
	3.2. *Prediction using an Analytical Model*	422
	3.3. *Prediction using a Directed Graph Model*	427
	4. *Simulation*	437
	4.1. *Major Methods*	437
	4.2. *Specification of Job Properties*	439
	4.3. *Data Collection*	443
	4.4. *Simulation Languages*	444
	4.5. *An Example Simulation Model*	452
	5. *Integrated Performance Prediction, Design, and Implementation*	455
	5.1. *The Problems with Non-Integrated Prediction*	456
	5.2. *Single Language Approach*	457
	5.3. *Interaction with the Designer-Implementer*	460
	5.4. *Aids to Project Management*	461
	6. *References*	462
C.C. Gotlieb	PERFORMANCE MEASUREMENT	464
	1. *Introduction*	464
	2. *Figures of Merit*	464
	3. *Kernels, Benchmarks and Synthetic Programs*	467
	4. *Data Collection and Analysis*	470
	5. *Hardware Monitors*	471
	5.1. *One Computer Monitoring Another*	472
	5.2. *Monitor Logic*	472
	5.3. *Examples of Currently Available Hardware Monitors*	474
	5.4. *Analysis of Output of Hardware Monitors*	475

	6. Software Monitors	478
	6.1. Monitoring form Job-Accounting Data	478
	6.2. Packaged Software Monitors	48o
	6.3. Special Monitor and Trace Programs	481
	6.4. Estimating Monitor Statistics from the Observations	486
	7. References	488
C.C. Gotlieb	PRICING MECHANISMS	492
	1. The Rationale of Pricing	492
	2. Determining Factors	493
	3. Costs	493
	4. The Factory Model	495
	5. Pricing a Service	495
	6. Software Requirements	497
	7. Examples for Pricing Mechanisms	498
	7.1. Rate Schedule for the University of Toronto, 1 Jan 1972	498
	7.2. Disk Pack Rental (Off-Line)	5oo
	7.3. Disk Pack Storage	5oo
	7.4. Disk to Tape Backup	5oo
	7.5. Tape Rental	5oo
	7.6. Tape Storage	5oo
	7.7. Tape Cleaning and Testing	5oo
	7.8. Negotiated Contract Services	5oo
	7.9. Calcomp Plotting	5o1
	7.1o. Card Processing	5o1
	8. References	5o2
H.J. Helms	EVALUATION IN THE COMPUTING CENTER ENVIRONMENT	
	1. Introduction	5o3
	2. The User and his Needs	5o5
	3. Software and the Computing Center	51o
	4. Installation and Maintenance of a Piece of Software	517
	5. Conclusion	52o
	6. References	521
APPENDIX		
F.L. Bauer	SOFTWARE ENGINEERING	522
	1. What is it?	523
	1.1. The Common Complaint	523

1.2. *The Aim* 524
1.3. *The Paradox of Non-Hardware Engineering* 524
1.4. *The Role of Education* 525
2. *Software Design and Production is an Industrial Engineering Field* 528
2.1. *Large Projects* 528
2.2. *Division into Managable Parts* 529
2.3. *Division into Distinct Stages of Development* 53o
2.4. *Computerized Surveillance* 531
2.5. *Management* 532
3. *The Role of Structured Programming* 532
3.1. *A Hierarchy of Conceptual Layers* 532
3.2. *Communication between Layers* 534
3.3. *Software Engineering Aspects* 537
3.4. *Flexibility: Portability and Adaptability* 538
3.5. *Some existing Examples* 539
3.6. *The Trade-Offs* 541
4. *Concluding Remarks* 541
Acknowledgements 543
References 543

SOFTWARE ENGINEERING

An Advanced Course

by	J.B.Dennis	(Cambridge, Mass.)
	G.Goos	(Karlsruhe)
	C.C.Gotlieb	(Toronto)
	R.M.Graham	(Berkeley, Cal.)
	M.Griffiths	(Grenoble)
	H.J.Helms	(Copenhagen)
	B.Morton	(Reading, England)
	P.C.Poole	(Abingdon, England)
	D.Tsichritzis	(Toronto)
	W.M.Waite	(Boulder, Colo.)
edited by	F.L.Bauer	(Munich)

The Advanced Course took place February 21 - March 3, 1972, organized by the Mathematical Institute of the Technical University of Munich and the Leibniz Computing Center of the Bavarian Academy of Sciences, in cooperation with the European Communities, sponsored by the Ministry of Education and Science of the Federal Republic of Germany.

PREFACE

It is not necessary to start with a definition of Software Engineering: the present book, a consolidated effort of a group of experts, carefully prepared in a two-week seminar in Garmisch, Dec. 71/Jan. 72, and presented at a EEC sponsored course in Febr.-March 72, illustrates the use of the term.

In 1967 and 1968, the word 'Software Engineering' has been used in a provocative way, in order to demonstrate that something was wrong in the existing design, production and servicing of software. The situation has considerably changed since then; many people show concern about the problems of software engineering and some of the manufacturers, to which the provocation was mainly addressed, claim that they already obey the principles of software engineering, whatever this may mean. Soon 'software engineering' will turn up in the advertisements. But although the problems are indeed much better understood, the material is still not concentrated and systematized. The reports of the NATO Science Committee sponsored conferences of Garmisch and Rome are a useful collection of material, but not much more. In order to have teaching material available, more has to be done. This book brings a first step in this direction.

Our intention in the planning of this course was to cover as much as we can at the moment of all the aspects of the theme, and to contribute further to the systematization of the field. We do not actually debate whether there is a need for software engineering. Instead, we think it is essential to point out where the ideas of software engineering should influence Computer Science and should penetrate in its curricula. Thus we will try to find out as much as possible whether a topic of software engineering is something you can mention as a kind of a theme to your students in an academic environment.

In this respect, my major concern was that today one still finds it extremely difficult, as many people told to me, to digest the material at hand so that it could be used in a course. Therefore, we envisaged publication of the lecture notes despite their somewhat tentative nature.

In selecting the participants we took some effort to assure that whatever they may learn here is spread out, in particular is propagated in the universities and the major manufacturers.

It is not quite accidental that efforts on 'Software Engineering' have been carried on to a large extent outside the United States. The poverty of the computer situation in Europe, at least on the continent, which is in sharp contrast to the affluent US computer community, leads to the demand for the most economical solution. But the roots of the software misery go deeper. It comes from the fact that people are forced to live with machines that they do not want. They have not constructed them, they simply receive them and have to make the best out of it. Sometimes, with the chance of buying a new machine, there is some hope that the situation will improve, but for simple market consideration, the manufacturer does everything he can do to make the customer stay with the product, and this usually ends all hopes for improvements. Thus, software engineering, for the time being, is partly a defense stratagem. But I hope that some day this situation will turn around, I hope one day software engineering considerations will dictate how machines are to be built and then to be used. Thus, what we have to work for is also preparing the ground for our future life. On the other hand, failure in mastering the software crisis may lead to strangulation of scientific users that depend on the computer today, in particular in 'Big Science', and may thus do harm also to science and economy in a rich nation.

In the preparation of the Advanced Course, I enjoyed the advice and help of colleagues and friends. I owe thanks to the co-director, Prof.L.Bolliet, and to the lecturers for their encouraging support. In particular, I am obliged to the German representative in the subgroup for education in informatics of group PREST of the EEC, Dr.R.Gnatz, for his help; in this connection the moral support from Mr.J.Desfosses (EEC) and the financial support from the Ministry of Education and Science of the Federal Republic of Germany should be gratefully acknowledged.The Conference Staff will forgive me for not mentioning all of them, my thanks to them go by the name of Mr.Hans Kuss of the Mathematics Institute of the Technical University Munich, who also was the responsible redactor of this publication.

Munich, June 1972

Friedrich L.Bauer

CHAPTER 1.A

WHAT THE SOFTWARE ENGINEER CAN DO FOR THE COMPUTER USER

Prof. Dr. K. W. Morton
Culham Laboratory, Abingdon, Berkshire
Great Britain

1. *INTRODUCTION*

There can be little doubt that there is at present an air of disillusion in the computer community. Computers are not living up to their potential and, in particular, the promises of the so-called third generation systems have been largely unfulfilled. As a result users have become more conservative and critical and are less ready to invest in new equipment. The reason does not lie with the computer hardware which continues to show a remarkable capacity to advance by orders of magnitude. But how often do we find software becoming ten times more reliable, ten times cheaper, ten times more efficient? It is more likely that it is ten times more complex both to maintain and to use and these more desirable qualities have been sacrificed as the sophistication of concept has outstripped the capacity for practical implementation. In short, software in general shows all the signs of poor and inadequate engineering.

While computer science has flourished in the 1960's with the establishment of journals, degree courses in universities, etc., the software engineering aspects of the subject have struggled for support, what techniques exist have been poorly disseminated and there is very little software in the hands of users which has been built on the best available engineering principles. In fact, many people are still arguing about what is software engineering and how is it related to computer science. As a mathematician, I am struck by the similarity of both the controversy and the actual relationship with that existing between mathematics in general and applied mathematics: in my view, it is not the subject matter itself that forms the important distinction but rather the use made of it and the attitude adopted toward it. Computer science gave us Algol 60: it also gave us the prospect of time sharing. But when we sit down at a console to write an Algol program, it is software engineering which determines how easy it is to