Abstract. In the last five years research has been actively carried out in an attempt to find structuring principles for large software systems, design principles and methodologies for such systems, and techniques and mechanisms for their efficient and economical implementation. This interest has been generated by the hope of overcoming the problems stemming from the inherent complexity of large software systems. This report proposes the name "software architecture" for this research area, justifies not using the name "software engineering" which is currently in vogue, overviews the content of the area, and presents a keyworded and annotated bibliography of the literature on software architecture problems and their solution.

I. Introduction

architecture: the art or science of building; specifically the art or practice of designing and building structures, esp. habitable structures, in accordance with principles determined by aesthetic and practical or material considerations.


During the three decade history of digital computers, there has been a natural and gradual generalization of the research interests of computer scientists. Initial software research was directed primarily toward the determination of fast, accurate algorithms for numerical computation and efficient heuristics for non-numerical problems. This work was supported by the Atomic Energy Commission, Project SU-326P23.

The author's current address is: Department of Computer and Communication Sciences, University of Michigan, 2076 Frieze Building, Ann Arbor, Michigan.
processing. The need for a "friendly" environment in which the users could carry out their work led to a shift of software research emphasis to the more general domain of symbol manipulation, in particular the areas of language processing and operating systems. Paralleling this research in software, the attention of hardware specialists was initially turned to the increase of computing machinery capabilities, particularly speed and size, and the use of the newly developed transistor and integrated circuit technologies. It then gradually switched to the development of new machine architectures and organizations which were more closely allied to the special processing needs encountered in language processing and operating systems.

Within the area of language processing, the particular topics of programming languages and their translators have been almost fully amenable to solution. Because of the reasonably well developed theoretical foundations provided by linguistics and automata theory and the easily understood nature of symbol manipulation, a wide range of techniques was developed for the translation of programming languages close to natural language and standard mathematical notation. Additionally, most of the techniques required processing facilities which were either already provided by the logical processing capabilities of computers or were merely a natural extension of them. Hardware specialists did not experience any unsolvable problems in producing a stack-oriented computer architecture which naturally facilitated compilation by providing a hospitable environment for both the compilers and the object code produced by them. Finally, even though the language translators were embodied in programs larger than those previously encountered, the well understood theoretical basis for the translation process led to a modularity which permitted one person to understand fully their organization and operation and eventually allowed the development of translator writing tools which partially automated the implementation of translator systems.

Two other topics within the area of language processing — information retrieval systems and information management systems — have not been as fully developed. The large size of the data bases worked on by these systems is still a stumbling block to the development of other than experimental systems — production systems are either uneconomical or do not provide acceptable response times. The problem is a combination of an inability to produce an economical, large, random-access storage unit and an inability to find data structures which provide efficient accessing when the information is stored in current storage devices. Finally, the modularity of the information manipulation systems which have been produced has been very close to the detailed specification level and no general hierarchical structure for an information manipulation system has been developed. This has created a situation in which it is difficult for one person or a manageable number of people to comprehend the total design of the system and carry out its implementation. Additionally, the lack of a general model has hampered the development of automatic tools to help in the making of the many design decisions and in the implementation of the system.

Whereas the problems of scale within the area of language processing have pertained to the volume of data being manipulated, in the area of operating systems the problems of scale have been
manifested in the large amount of processing which must be done when allowing the sharing of resources and the concurrency inherent in systems which have a large number of both independent and inter-dependent processing streams active simultaneously. The problems stemming from the concurrency have received the most technical attention and algorithms and mechanisms have been developed which are sufficient to assure the appropriate, well-orchestrated operation of the system. Hardware has been produced which facilitates the sharing of resources, including the central processor, but much more needs to be done to provide a general computing utility which is economical and reliable. The seemingly unsolvable problems in operating systems have been associated with the sheer size of the programs which embody them and the inability to find general models which allow the description of their organization and operation -- in fact, their general organization and operation is very poorly understood in the abstract terms necessary for the development of a model which would facilitate their description and design specification and allow the development of powerful automatic implementation tools. The large size of the programs has led to a correspondingly large expenditure of man-years for design and implementation -- the Chinese-army approach has been a general failure, sometimes leading to negative progress, and few generally applicable management principles have been developed to help the situation.

The term "software engineering" has been introduced in order to provide a name for the study of problems of large-scale, complex software systems and with the deliberate intent of indicating the general nature of their solution. The suggestion was that only through the application of sound engineering principles could large, correctly-operating software systems be economically produced. But the term has turned out to be somewhat of a misnomer since the engineering of complex systems presupposes a knowledge of the basic material available and its general properties and the ways in which it can and should be used to produce the final system. Most of the work that has been carried out in this area, however, has been directed toward the identification of the basic material (the building blocks) and its properties and the determination of structuring principles for large software systems, particularly operating systems, and methodologies for their design. "Software architecture" is a more appropriate term, reflecting as it does the emphasis upon the demarcation of primitive material and its properties, the determination of structuring principles which take into account practical and material considerations, the development of appropriate design principles and methodologies, the orientation toward the production of a hospitable end-product, and the involvement of aesthetic considerations.

It is the intent of this report to provide an annotated bibliography of the literature covering software architecture. Preliminary to the bibliography, the next sections provide an overview, in outline form, of the field. It serves the dual purpose of structuring the field's content and providing definitions for the terms which are used to classify the items in the bibliography.

The bibliography is not self-contained. Several bibliographies and articles containing extensive bibliographies are cited (they are all keyworded with the term "bibliography") and should certainly be
consulted as extensions to this bibliography. Edited collections of articles are cited and individual articles from these collections are for the most part individually entered. Several articles in the cited collections were not entered, however, since they were deemed only marginally pertinent or their subject matter was covered by other entries in the bibliography.

Two topics are given only minimal coverage in this bibliography. The area of large data base structure and management is not covered at all due to the author's incomplete familiarity with the extensive literature on the subject. Some articles covering the management of software system design and implementation have been included in order to indicate the nature and extent of the work in this area. The author justifies this limited coverage of the important topic of management by advancing the opinion that management practices are valid only if there is a full comprehension of the system being managed -- since the structure of the design and implementation groups is influenced by the structure of the software system, and since that structure is not fully understood, resulting management principles are not necessarily valid. It is well known that the influence goes the other way as well and some may hold that direction of influence to be more important -- it has been the author's experience that, while the use of established management practices may speed the design and implementation of software systems, they sometimes have a disastrously negative effect upon the clarity and maintainability of the system and the ease with which it may be modernized.

The author is aware that some material may well have been overlooked and hopes that readers will bring any omissions, major or minor, to his attention.

II. Complexity

Complex systems have been studied in social and biological contexts as well as in the technical areas of which computer software is a part. The focus of this study has been upon the inherent attributes of complex systems, the benefits to be derived from the appropriate and judicious use of these inherent attributes, the evolution of complex systems, schemes for their succinct description, methods for their formal analysis, and metrics for measuring the amount of complexity. Much of this work on complex systems in general is directly applicable to software architecture.

A. General attributes of complex systems.

1. Modularity -- there are natural units which form the building blocks for construction of the system.

2. Hierarchical structure -- the base building blocks may be grouped into subsystems which may, in turn, be grouped into "higher" subsystems, etc. Most usually, but not necessarily always, the hierarchical structure exhibits strict nesting of the subsystems.

3. Abstractions -- each subsystem in the hierarchical structure provides a point at which an abstraction may be
made of either the system's organization or behavior in
that the internal aspects of the subsystem may be
subsumed within a more precise description of the subsystem
in terms of its interactions with other subsystems.

E. Complexity metrics.

1. General -- there are several metrics which may be
   used to measure the complexity of general systems.
   a. Size -- the most obvious metric.
   b. Connectivity -- this measure accounts for the
      amount and cost of interactions among the
      subsystems with due attention being paid to the
      system's hierarchical structure.

2. Software systems -- more specific metrics arise when
   attention is focussed upon software systems in
   particular.
   a. Large-scale -- rather than have a specific
      definition, the use of this term is rather
      intuitive. The other metrics serve to specify this
      one more concretely.
   b. Understandability -- the rather mushy test here
      would be the length of time for one person to gain
      a working knowledge of the detailed operation
      of the entire system.
   c. Cost -- always a factor, this can be measured
      in dollars, man-hours, or lines of code.
   d. Information processing -- complex software
      systems do not generally carry out numerical
      computations and the nature and structure of the
      symbols manipulated by the system can provide some
      insight into the system's complexity.
   e. Perpetually erroneous -- this attribute is
      all-to-well known by users and can lead to measures
      of complexity such as the number of errors which
      must always be present or the difficulty of
      "getting around" the residual errors.

C. Description techniques -- the modularity and hierarchical
   structure of complex systems leads to an ability to make
   abstractions as noted above. These abstractions can be
   usefully employed when trying to describe the system's design
   or its behavior. A second use of abstract descriptions is to
   allow the formal analysis or synthesis of systems.

1. Documentation -- this is the most used and least
   effective means of describing a complex software system.
   It is generally verbose and ambiguous and is certainly
   not a basis for formal analysis of the system.
2. State descriptions -- these describe the system as an entity which moves from one state to another and generally pertain more to the system's behavior than to its structure.

a. State descriptions of structure -- when the system is not capable of any concurrent activity, its state corresponds directly to some point in the conditional, sequential flow of control among elements in the physical structure of the system. Almost always, these descriptions take the form of directed graphs.

1. Flowcharts -- the traditional means for describing a sequential program.

2. Program schemata -- the extension of flowcharts to situations in which parallel activity may take place.

b. State descriptions of behavior -- these describe the logical steps carried out by the system during its operation and there is little or no attempt to relate this to the physical structure of the system.

1. Directed graphs -- the nodes represent various states and the arcs represent state transitions.

a. State transition diagrams -- used mostly as a means of describing finite state automata.

b. Petri nets -- developed to describe the states achieved by interacting automata.

c. Occurrence and marked graphs -- a modification of Petri nets developed to permit the description of the inherent concurrency of cyclic sequential programs which is masked by a flowchart description.

d. Hierarchical directed graphs -- an extension of normal directed graphs developed to mirror the hierarchical nature of the system's behavior.

2. "Algebraic" equations -- these describe the behavior of the system in terms of a trace of the system's activity during its operation.

a. Regular expressions -- used to describe the input accepted by finite state automata.
b. Message transfer expressions -- an extension of regular expressions which give a trace of the inter-module communication taking place in a system of concurrently operating modules.

3. Process descriptions -- these focus upon the structure of the system by exhibiting the process by which the system may be constructed.

a. Syntactic definition -- the syntax of a language specifies the ways in which sentences of the language may be constructed. In a similar way, the alphabet of a system (the basic building blocks for its construction) may be specified and a syntax given to describe the structure of the system in terms of how it may be put together from elements of the alphabet.

b. Vienna definition language -- the latest in a series of attempts to provide a formal process description of the semantics as well as the syntax of programming languages. The scheme provides for the description of the interpretation that is to be associated with any sentence produced by the syntactic definition.

III. User requirements

The software systems exist in order to satisfy the needs of a user community. Aside from the users' processing requirements, there are three basic requirements levied which create specific problems for the designers, implementors and service personnel.

A. Usability and evolution -- the users require a system which satisfies their needs (the usability requirement) and since these cannot always be fully specified in advance, the system must be easy to modify (the system evolution problem).

1. The usability requirement.

a. Completeness -- provision of a full set of services.

b. Adaptability -- the ability to have the system changed so as to provide a previously missing feature. The changes in this category are permanent, global changes which the users expect to be accomplished in a reasonable amount of time.

c. Extensibility -- the ability to change a private version of the system so as to particularize it to a user's specific needs. These changes are temporary, local ones which the user expects to be accomplished almost immediately upon request.
2. The system evolution problem.

a. Static evolution -- performed by maintenance personnel and results in a permanent change to the system.

1. Modification -- additions and deletions caused by changes in the needs of the users or in the host-machine configuration.

2. Transportation (portability) -- using or producing exportable or importable systems or subsystems which can be easily inserted into a (perhaps radically) different environment.

   a. Generality -- independence of the system from its environment.

   b. Localized environment dependency.

   c. Standardization -- imposition of a uniformity upon the host environments.

   d. Universal intermediary -- the provision of a universally accepted intermediate description scheme so that only a minimum amount of translation need be done, hopefully automatically.

b. Dynamic evolution -- temporary, reversible changes which are, for the most part, performed automatically.

1. Extensibility -- the system may be specifically designed so that private versions of it may be tuned to the particular needs of a subset of the user community.

2. Adaptation -- the system may be designed to adjust its behavior to the dynamically changing load induced by the users.

3. Reconfiguration -- the system may be designed to tolerate the dynamic deletion and recovery of pieces of host-machine hardware.

B. Efficiency and evaluation -- the users require a system which does not waste their time or money (the efficiency requirement) and thus it must be possible to obtain valid measurements of the system's performance (the system evaluation problem).

1. The efficiency requirement.

   a. Efficiency -- measures of efficiency and their applicability.
b. Optimality -- can it be achieved and is the effort really justified in a dynamic system?

2. The system evaluation problem.

a. Evaluation -- the measurement of the actual performance of the system.

1. Analysis -- a priori determination of the system's performance characteristics.

a. Modeling -- the use of mathematical models.

b. Simulation -- the use of stochastic simulation models.

2. Monitoring -- determination of the performance characteristics during the operation of the system. Monitors are used to collect statistics which are subsequently analyzed.

a. Hardware monitors -- hardware devices used to collect the statistics.

b. Software monitors -- pieces of software inserted into the system, either during its implementation or dynamically while it is running, to collect the statistics.

b. Testing -- the comparison of the system's performance against some established criteria.

1. Criteria -- the development of meaningful acceptance, quality or performance criteria.

2. Test cases -- the development of test loads for submission to the system.

3. Worst case analysis -- the special considerations concerning test case development when critical performance limits may be delineated.

C. Integrity and validation -- the users require a system which consistently produces valid results (the integrity requirement) and thus the validity of the system must be amenable to analysis and it must be possible to correct errors (the system validation problem).

1. The integrity requirement.

a. Availability -- the users want a system which is available when they want to use it.

b. Reliability -- the users want a system that is
not subject to transient, erratic behavior.

c. Correctness — the users want a system that conforms to some generally accepted expectations. They have a variable tolerance level, however, being able to live with some errors (such as in program compilation) as long as there is a reasonably easy way to get around them.

2. The software validation problem.

a. Formal validation — the use of effective procedures to analyze a system's correctness or the determination of structuring techniques, mechanisms or principles which will preclude invalid behavior by the system.

1. Axiomatic approaches — methods of analysis which depend on established or specially developed axiom systems.

   a. Models of computation — the use of directed graphs or program schemata to develop sufficient and/or necessary conditions for algorithms (usually ones containing some parallelism) to terminate or be determinate or the use of these description techniques to show equivalence between programs.

   b. Predicate calculus — the expectations about the system's behavior are stated as predicates over the values of the variables manipulated by the program.

   c. Message transfer expression analysis — the expectations about the system's behavior are stated as traces of the communication which takes place among the concurrently operating parts of the system.

2. Non-axiomatic approaches — the analysis is carried out with the aid of a process description of the system rather than a state description.

   a. Generative approaches — generative definitions of the structure (syntax) or behavior (semantics) of the system are used as the process descriptions.

   b. Bottom-up approaches — the hierarchical decomposition of the system is employed to guide the analysis, with the validity of the base modules being proven first and then the
IV. Design

The design process can be viewed as the translation of a rough specification of the functional characteristics of the system into a detailed description of the system in terms which are executable by the host environment. This overall translation breaks down into a series of sub-translations which reflect the traditional design cycle. Several principles have been delineated which tend to ease the translation process and several methodologies have been advanced as efficient paths through the series of translations. Some tools have been developed to automate parts of the overall translation.

A. Design cycle -- the traditional design cycle may be stated in terms of a series of successively more elaborate and detailed descriptions.

1. Functional level -- the descriptions at this level specify the external requirements and the general functional organization of the system.

2. Process level -- the descriptions at this level exhibit the structure of the system in terms of the interconnections among asynchronous, concurrent modules and the behavior of the system in terms of inter-communication among the modules.

a. Unrestricted resources -- the descriptions at this level are developed without regard for limitations as to the number or capabilities of the physical resources available in the host environment except as these limitations may be required by the functions to be performed.

b. Restricted resources -- at this level, the descriptions still are couched in terms of

validity of the next higher level of modules being proven with the base modules being assumed correct, etc.

b. Error detection -- techniques for the detection of the presence and location of errors. Special problems which must be considered are the generation of test cases, the detection of errors which are global in their effect, irreproducibility of errors in an asynchronous environment, and the validity of using environment simulators so that a perhaps faulty system need not be run in a dedicated, actual environment.

c. Tolerance -- the design of systems so that errors will not affect the operation of the system or will affect it only within a small segment of the total system.

d. Debugging -- techniques and system extensions which facilitate the removal of errors.
inter-communication among concurrent modules, but extra modules or communication paths are incorporated in order to effect restrictions required by efficiency considerations or limitations of the host-environment. The restrictions may be necessitated, in part, by a limited number of processors, processors which have only limited capabilities, the required use of re-entrant or re-usable code, or communication handling mechanisms which have a bounded queue length.

3. Sequential programming level -- at this level, the descriptions present the detailed internal logic of each module, in terms of a sequential program, which the module must carry out in order to perform the message handling required in its communication with other modules.

   a. Procedure oriented (high-level) specifications -- the description is in a high-level programming language.

   b. Machine oriented (low-level) specifications -- the description is in a low-level programming language for some (perhaps only virtual) machine.

4. Computer organization level -- the descriptions at this level specify the logic and organization of machines which carry out the primitive operations used in stating the machine oriented specifications. These descriptions need not specify hardware logic but may rather define a virtual "machine" which provides an environment for the execution of the system specified in the description at the sequential programming level.

b. Design principles.

1. Modularity -- the systems should be segmented so that there are modules which embody one logical subpart of the system. The segmentation may be determined, in part, by the functions to be carried out or in order to isolate design decisions.

2. Conventions and standards -- rules should be established as to techniques and mechanisms to be used in performing certain standard operations.

3. Late-binding -- details about the exact form and operation of parts of the system should be delayed as long as possible.

4. Restricted information distribution -- the external specification of a module should specify only enough of its operation to allow successful interaction on the part of other modules.

5. Integrated simulation -- the feedback obtained by
simulating parts of the system should be available while the design is being carried out. Waiting until the design is complete does not allow the feedback to have maximal effect.

6. Iteration -- the design process should be so formulated as to allow iteration back through previous steps with a minimum of effort.

C. Design methodologies.

1. Bottom-up -- in this methodology, primitive building blocks are used to construct modules and then these are used to construct other modules, etc., until the final system is arrived at.

2. Top-down -- this methodology follows a path of decomposition rather than the path of composition followed in the bottom-up methodology. A design is produced by forming successively more detailed descriptions of the system by a series of elaborations of previous descriptions.

3. Others -- neither the top-down or bottom-up methodologies are strictly used in practice since it is fruitful to employ the information known about both ends (the functional specification of the system and the capabilities of the host environment in which it will run). Other methodologies are a mixture of top-down and bottom-up -- inside-out, outside-in, top-down with look-ahead, bottom-up with look-ahead.

D. Design tools -- tools have been developed to help and partially automate the design of software systems. Since the design process is basically one of elaboration or abstraction of descriptions, the tools are oriented toward the automatic translation of descriptions.

1. Near-decompositility analysis -- this is a graph theoretical analysis technique which uses connectivity measurements to delimit the subsystems within a system of modules.

2. Concurrent activity constructs -- these are mechanisms and programming language constructs for allowing the expression of parallel activity and the synchronization and communication among concurrently operating modules within the framework of a sequential program.

3. Sequencing analysis -- these are procedures for automatically uncovering the parallel activity which can be carried out by the modules in a system which has been described by a sequential program.

4. Automatic flowcharters -- programs which produce flowcharts for systems which have been described in some programming language.
5. Systems programming languages -- special programming languages developed for the implementation of large software systems, usually operating systems.

6. Translators and translator writing systems -- compilers, assemblers and interpreters and systems which aid in their implementation.

7. Decompilation -- systems which perform the reverse translation from that performed by compilers.

8. Sysgen systems -- systems designed to allow the binding of a system, which has been designed to operate in a multitude of host environments, to one particular host environment.

V. System design and implementation languages

Since the emphasis in design, implementation and maintenance is on the description of the system, particular attention has been paid to ways of stating the description succinctly and unambiguously.

A. Desirable attributes -- the descriptions obviously must reflect the behavior of the systems but should also reflect the structure of the system. Therefore there are attributes of languages for describing the design or implementation of a system which reflect the structuring and design principles related above.

B. Structured programming -- particular attention has been paid to the design of programming languages so that the implementation of a system may be carried out in a natural, top-down elaboration process.

1. Go-to-less programs -- overuse of the goto statement can destroy the structure and clarity of a program.

2. Control constructs -- new constructs have been added to programming languages, such as the case construct, to reduce or eliminate the need for using the goto statement.

C. System programming languages -- special languages have been developed for the implementation of systems. Most have been extensions of generally available programming languages, but new and radically different ones are starting to appear.

D. Translator writing systems -- if new languages are developed then compilers must be prepared to translate programs in the languages into executable code. It is the role of translator writing systems to partially automate the preparation of compilers, assemblers and interpreters.

VI. Computer organization
This term is used to describe the logical organization of a computing facility -- it is what the user "sees" when he uses the machine -- and is in opposition to computer architecture which pertains to the physical organization. Very seldom does the logical structure directly reflect the physical structure, although it is definitely influenced by it. Within this part of software architecture is the research carried out with the aim of providing computer organizations which are the natural ones needed for the implementation of software systems.

A. Language oriented machine design -- this phrase has been coined to describe the development of computer architectures which facilitate logical organizations required for software system implementation, particularly the implementation of language translators.

B. Microprogramming -- microprograms are (usually unchangeable) programs which control the gating and inter-register information transfer so as to produce a specific organization.

1. Emulation -- this is the use of microprogramming within one machine so that it mimics the operation of a machine of a different architecture.

2. Language machines -- the use of microprogramming so that a machine may almost directly execute statements in some programming language.

3. Control constructs -- microprogramming may be used to provide a wide variety of control constructs not only for sequential processing but also for processing which involves parallel activity.

VII. Management

The design and implementation of large software systems requires more than one person and this creates a management problem.

A. Planning.

1. Language choice -- the language in which the system will be implemented must be chosen.

2. Programmer assignment -- programmers must be assigned to tasks which they are capable of completing.

3. Responsibility hierarchy -- a hierarchy of responsibility must be established so that the progress of the project may be monitored.

4. Cost estimation -- before starting the project it must be decided whether or not sufficient monetary and human resources are available.

5. Scheduling -- a schedule must be established which specifies the times at which check-points will be made
and milestones should be reached.

B. Control.

1. Documentation standards -- a uniform vocabulary and syntax must be established so that effective communication can take place among the implementors and designers and so that the workings of the final product may be easily discerned by the users.

2. Progress monitoring -- the progress of the implementation must be monitored.

3. Quality control -- the final product must live up to some absolute standards and intermediate checks must be made as to whether this will be achieved.

4. Change control -- indiscriminate changes to a partially implemented system can cause serious problems.

C. Pricing -- a price for the final product must be determined.

VIII. Miscellaneous

There are two topics within the software architecture field which do not fit into the outline presented above.

A. Education -- computer scientists must be educated in the principles and techniques of successfully designing and implementing large software systems. This pertains to the general knowledge which must be conveyed rather than the technical specifics of particular systems such as operating systems. The knowledge may be conveyed by a traditionally organized course or by means of a software laboratory course.

B. Software design and implementation laboratories -- these are facilities in which the tools, mechanisms, and methodologies of software architecture are made available for the design and implementation of software systems and the experimentation with the systems.

IX. Acknowledgements

Acknowledgement should be given to William F. Miller and Victor R. Lesser whose association and conversations with the author have directly and indirectly contributed to this report. The author would also like to acknowledge the conscientious efforts of Jorge Bruguera which have made the Stanford University Computer Science Department library one of the most complete and well-managed collections of computer science literature.
X. Bibliography


Topics: Sequential programming level

Keywords: automatic flowchart production


Topics: Description techniques

Keywords: computation model, parallel computation, directed graph, properties of computation, hierarchical

A computation is modeled by a directed graph built hierarchically from a set of nodes with pre-defined interpretations. The arcs between the nodes represent FIFO data queues and all sequencing of the nodes in a graph model is effected by the flow of data among the nodes. It is shown that any computable function may be represented and that any computation represented by a graph model is determinate.


Topics: Validation, Management

Keywords: debugging, cost estimation

'...by carefully studying a general flow chart during the detailed designing of software, we can predict the number of bugs in the system, and decide on the number of checkpoints to be made during the process, and, consequently, the man-hours necessary.' (author)


Topics: Design methodologies

Keywords: bottom-up, hierarchical decomposition

Proposes bottom-up design after obtaining a general tree-structure of the interdependencies among the parts of the system.

Altm69 S. M. Altman and A. W. Lc. Systematic design for modular

Topics: Design methodologies

Keywords: primitive modules, bottom-up, asynchronous digital systems

This paper presents a systematic procedure for the design of the logical structure of a set of asynchronous control modules which may serve as the basic building blocks for the construction of the control structure of an asynchronous digital system.


Topics: Computer organization

Keywords: machine language design

Argues in favor of a traditional machine language (opcode, index register, and indirect addressing) and against using Polish notation - see criticism of Barton in "Ideas for computer system organization".


Topics: Sequential program level

Keywords: programming languages, concurrent processes, multiprogramming

A set of supposedly necessary commands are proposed as an extension to Algol in order to permit the specification of the possible concurrent operation of subparts of a procedure. The commands are intended to allow easy translation of the source program for a multiprogramming environment.


Topics: Evaluation

Keywords: monitoring, statistics

Topics: Education

Keywords: system design, system implementation

This article gives a description and evaluation of a system design and implementation laboratory course for advanced graduate students.


Topics: Management

Keywords: estimating cost

The paper presents a simple guideline for estimating the manpower requirements for large scale software production efforts. The guideline is in the form of a quantitative method and the paper reviews (quickly) some alternatives. It is concluded that the method offers aid when attacking systems in unfamiliar areas and verification of estimates obtained by other methods, but the method is not as good as reliance on prior experience.


Topics: Computer organization

Keywords: parallel computers, multiple functional units, concurrent processes

The authors indicate how a computer architecture which more directly maps the conceptual organization implied by concurrent processes can enhance the system throughput.


Topics: Sequential programming level

Keywords: go-to-less programming languages

"In this paper we show that every flowchart program can be written without go to statements by using while statements. The main idea is to introduce new variables to preserve the values of certain variables at particular points in the program; or alternately, to introduce special boolean variables to keep information about the course of the computation." (authors)

Topics: Design methodologies

Keywords: top-down elaboration, integrated simulation, iterative


Topics: Descriptive techniques

Keywords: formal definition of syntax, BNF


Topics: Sequential programming level --> Process level

Keywords: desequencing, algorithm


Topics: Descriptive techniques

Keywords: graphs, computation


Topics: Validation, Optimization

Keywords: equivalence preserving transformations


Topics: Validation

Keywords: debugging, extensible system, high-level languages

Topics: Description techniques, Sequential programming level, Process level

Keywords: procedure concept, implied processing, process specification, dynamic modification of process specification


Topics: Process level

Keywords: delayed binding, communication, modularity, semaphores, conditional P-operation, concurrent processes


Topics: Evaluation

Keywords: hardware/software interface, analytical


Topics: Sequential programming level

Keywords: language design

The author argues that programming languages should not be influenced by the characteristics of a machine, but rather by the ease of teaching it, the need to have program libraries, the need for representation of structure in the data and the algorithm, and the need to produce translators and systems.


Topics: Computer organization

Keywords: sequencing, storage partitioning, machine language design, data descriptors, structured programming

In this cogent article, Barton reviews his philosophy of
computer organization which has been embodied in the
Burrough's computing machinery. In discussing the areas of
storage space partitioning and processing time partitioning,
he argues that natural boundaries can be used beneficially.
The viability and need for natural expression of processing
within the machine's language are argued by reviewing the
benefits of polish notation, seconding the arguments of
Dijkstra for structured programming and noting the
structured programming aspects of polish notation, and
reviewing the possibility of eliminating specialized
sequence control hardware by extending the order code to
structured operands. An argument for machine generality is
set forth by a discussion of using data descriptors as
parameters to data processing routines.

Rask72 H. R. Raskin, B. F. Borgerson and R. Roberts. PRIME -- A
modular architecture for terminal-oriented systems. Proc.

Topics: Computer organization

Keywords: modularity, parallel subsystems, fault
tolerance, integrity

Rauer71 F. I. Rauer. Software engineering. Proc. IFIP Congress 71,
Ljubljana, August 1971, pp. 1267-1274.

Topics: General

Keywords: hierarchy, structured programming,
modularization, top-down, bottom-up, portability,
adaptability

In the author's opinion, it is difficult to achieve the
aim of software engineering -- 'to obtain economically
software that is reliable and works efficiently on real
machines' -- because there is no physical object being
manipulated and there is tremendous pressure to "produce
something." Rauer then cites three major techniques --
division of labor by hierarchical decomposition of the
system, iteration among levels during the elaboration of the
design, and computerized surveillance of the course and
progress of the design. He argues the applicability of
structured programming in facilitating the use of these
techniques and concludes by indicating that the primary
problem is the determination of layers within the system's
structure.

Rela71 L. A. Relady and M. M. Lehman. Programming System Dynamics
of the Meta-Dynamics of Systems in Maintenance and Growth.
FC 1546, IBM Research Lab, Yorktown Heights, N. Y.,
September 1971.

Topics: Description techniques, Complexity
Keywords: maintenance, growth of systems, complexity metrics

'A macro-model of the growth of large programming system projects is developed. The model reflects the consequences of continuous repair and functional enhancement activities always associated with such systems. It is shown that these activities become increasingly fragmented as the number of system releases increases. The resultant stratification of the maintenance effort is modelled by an explicit complexity term that arises from the communication needs of both the maintenance organization and the programming system. The term indicates the inevitability of ultimate exponential increase in the total effort (men, machines, time and dollars) required to maintain and grow the system.

'The consequences of increasing complexity represented by the macro-model are strongly related to the spread of change and of faults over the system. A micro-model that, in turn, relates the spreading to system structure, is also outlined.

'The models suggest changes in programming methodology, project management and system structure so as to delay the exponential growth effort required for maintenance. Their development is largely independent of specific programming content. Hence the models apply equally to other large systems.' (authors)


Topics: Descriptive techniques

Keywords: information flow, register transfer level

This article overviews the PMS and IPS systems which provide descriptions of the information flow and register-transfer levels of computing systems.


Topics: Sequential programming level

Keywords: structured programming, assembler languages, system implementation languages

'In a classic paper Wirth describes a language which combines the readability of Algol 60 with the flexibility and degree of control of a conventional assembly language. This paper gives an outline of a similar language for a small 16-bit computer -- the Honeywell DDP-516. Implementing the compiler in its own language by recoding an Algol version of the compiler has shown that the language is
suitable for large systems." (authors)


Topics: Evaluation
Keywords: software monitors, inefficiencies

This paper discusses the problem of software performance measurement and, by example, indicates some of the inefficiencies which may be detected only by appropriate instrumentation. A case is made for software measurement mechanisms and implications for the design of hardware and software are summarized.


Topics: Management
Keywords: production, implementation tools

This article gives an outline for the area of software implementation methods, including both management considerations and available tools.


Topics: Management
Keywords: implementation, design methodologies, system organization

This article lists some 250 questions which indicate desirable practices to be followed in the production of marketable software systems. The questions pertain mostly to record keeping and management practices but a few pertain to design methodologies and system organization.


Topics: Portability
Keywords: program transferability
Presents an outline of the principles to be followed in order to achieve transferability, the information required to transfer a program, and mechanical tools which can provide aid.


Topics: Sequential programming level

Keywords: automatic documentation, machine language programs


Topics: Sequential programming level --> Process level

Keywords: concurrent processes, sequencing analysis

Conditions are developed such that if two contiguous portions of a sequential program satisfy the conditions then the program segments may be executed in parallel without destroying the determinacy of the computation. The author points out that the parallelism is a property of the algorithm and may be precluded by some program realizations of it. He also addresses commutivity of two program segments. Finally, he proves that it is impossible to determine, in general, the concurrency of two program segments.


Topics: Process level

Keywords: process creation, process destruction, inter-process communication, levels of control


Topics: Process level, Adaptation

Keywords: resource-sharing, adaptive scheduling, semaphore

File: 273

'This paper describes the main design principles of the
multiaccess system ESOFE. ... The main features of the system include the ability given to any user to schedule his own parallel process, using system primitive operations, and the allocation/scheduling policy, which dynamically takes into account recent information about user behavior. (authors)


Topics: Process level

Keywords: concurrent processes, system design


Topics: Validation

Keywords: debugging systems, extensible


Topics: Evaluation

Keywords: Simulation, time-sharing, optimization


Topics: Description techniques

Keywords: flow charts, automatic simplification

Brad68 Paul T. Brady. Writing an online debugging program for the experienced user. Comm. ACM, 11, 6 (June 1968), 423-427.

Topics: Validation

Keywords: debugging, online


Topics: Sequential programming level
Keywords: composition aid, editing


Topics: Description techniques

Keywords: flow table, binary signals, finite queues.


Topics: Validaticn, Process level

Keywords: sequential processes, concurrent processes, flow table model


Topics: Validation, Process level

Keywords: flow table model, finite delays, concurrent processes, state transition


Topics: Descripticn techniques

Keywords: Adams graph, Rapp and Miller, Luconi, Rodriguez, control structures, properties of computation


Topics: Evaluation

Keywords: testing, generating test data, matrix inversion


Topics: Computer organisation --> Sequential programming level, Sequential programming level
Keywords: concurrent processes, hierarchical systems, kernal, communication

This presents a detailed description of the system designed under the philosophy explained in Brinch Hansen's article "The nucleus of a multiprogrammed system."


Topics: Computer organization \rightarrow Sequential programming level, Sequential programming level

Keywords: concurrent processes, hierarchical systems, kernal, communication

The kernel proposed in this paper provides services for process creation and destruction and inter-process synchronization and communication.


Topics: Process level

Keywords: synchronization, semaphores, conditional critical sections, validation


Topics: Portability

Keywords: general

A general discussion of software portability precedes a report on the ALTRAN-P system's design as a portable system. ALTRAN is a symbolic algebra language built upon Fortran and the system gains its portability through the use of primitives and macros which contain the machine dependencies and the use of a multi-layered boot-strapping process.


Topics: Evaluation, Adaptation

Keywords: analytic model, monitoring, dynamic tuning to system load
Bryant66  Peter Bryant. Levels of computer systems. Comm. ACM, 9, 12 (December 1966), 873-876.

Topics: Design principles
Keywords: levels of control, levels of communication


Topics: Sequential programming level
Keywords: common modules, reorganization


Topics: Descriptive techniques
Keywords: formal definition of syntax, BNF

This article gives a specification of ALGOL 60 and FORTRAN in a tabular metalanguage based on BNF, but which has additional metacoperators to denote limited repetition and card fields.


Topics: Sequential programming level, Process level
Keywords: systems programming language


Topics: General
Keywords: description, specification, evaluation, large-scale systems, education

A report on the second NATO sponsored conference on software engineering in which the emphasis was on techniques -- both their theoretical foundations and their application in the world of practical system development. The first part of the report collects together comments on the subjects of theory versus practice, description and specification, quality assurance, flexibility, the problems introduced by large-scale systems and the education of software designers and implementors. The working papers
reproduced in this report are presented elsewhere in this bibliography.


Topics: Evaluation

Keywords: general, criteria, tools


Topics: Evaluation


Topics: Process level

Keywords: communication, networks


Topics: Sequential programming level --> Process level, Sequential programming level

Keywords: programming languages, sequencing analysis


Topics: Sequential programming level --> Process level, Sequential programming level

Keywords: programming languages, sequencing analysis


Topics: Evaluation

Keywords: debugging

Chap70 Ned Chapin. Flowcharting with the ANSI standard: A

Topics: Descriptive techniques

Keywords: flow charts


Topics: Sequential programming level

Keywords: validation, measurement, description, system programming languages, structured programming, abstractions

Cheatham predicts that the topics which will influence the future design of programming languages include the need to easily implement large systems, the need to analyze and validate large systems, and the need to describe abstractions of large system organization and behavior.


Topics: Sequential programming level

Keywords: composition tools, validation tools, measurement tools

This article discusses the design of a system for the computer-aided composition, validation, and measurement of computer programs.


Topics: Sequential programming level

Keywords: machine instruction set


Topics: Computer organization

Keywords: microprogramming, bibliography

Topics: Sequential programming level

Keywords: simplification, improvement


Topics: Computer organization

Keywords: bottom-up design, primitive modules

This chairperson's introduction to a session at the 1967 SJCC sets forth the general reasons for approaching hardware system organization from a building block point of view. Basically, the viewpoint permits a bottom-up system design which permits experimentation and facilitates system evolution and modernization.


Topics: Validation

Keywords: programs with jumps and simple functions, formal proofs


Topics: Process level, Description techniques

Keywords: algebraic

'The purpose of this work is to supply a simple method for the definition and efficient control of a network for asynchronous parallel processing. The system is able to compile a definition of a network of processes which run partially in parallel and partially sequentially, into a set of control instructions for some monitoring process, to be executed at run time.' (author)


Topics: Sequential programming level

Keywords: logical analysis, reformatting


Topics: General
Keywords: design methodologies, generalization, modularity, hierarchical systems, conventions, validation, optimization

This small collection of papers covers a basic set of program and data processing systems design concepts, as indicated by the titles of the articles: Towards a Theory of Program Design, The Design of Programs, Generalization: Key to Successful Electronic Data Processing, Modular Data Processing Systems Written in COBOL, The Philosophy of Conventions, Some Comments on the Design of a System, Design and the Reduction of Bugs, System Level Optimization, The Future of Programming and Programmers.


Topics: Sequential programming level, Process level

Keywords: concurrent processes, programming languages

This paper proposes a set of high-level programming language constructs which generalizes the normally subordinate relationship between program modules. The sending and receiving (of both data and control) are separated and a natural expression of intermodule constraints on parallelism ensues.


Topics: Sequential programming level, Computer organization

Keywords: concurrent processes, programming languages

This article covers the basic architecture of a multiple processor computing facility, and introduces the low-level instructions FORK and JOIN for the specification of the initiation and termination of concurrent activity.


Topics: Description techniques

Keywords: directed graphs

Coop68 D. C. Cooper. Some transformations and standard forms of graphs, with applications to computer programs. In Dale and Michie (eds.), Machine Intelligence 2, Am. Elsevier, N.
Topics: Description techniques
Keywords: directed graphs, transformations


Topics: Validation
Keywords: assertions, use of program structure


Topics: Process level
Keywords: limited resources, concurrent processes

The authors distinguish complexity arising from non-interdependent sharing of resources (multiplexing) and interdependent sharing of resources (sharing). They then suggest various viewpoints (such as reversibility of binding) and mechanisms (such as segmentation) which facilitate insight into the operation of the system as a whole.


Topics: Sequential programming level
Keywords: systems programming languages


Topics: Process level, Computer organization --> Sequential programming level, Education
Keywords: limited resources, software laboratory, system

Keywords: bibliography, general programming, system kernel, bottom-up construction, concurrent processes, message switching, bounded mailboxes

This report discusses the kernel for a multiprogramming system designed to allow the construction of experimental systems for educational and research purposes. The kernel
provides an environment in which asynchronously operating modules may be flexibly interconnected through a mailbox buffering system.


Topics: Education

Keywords: concurrent processes, design methodologies, reliability, performance monitoring, bibliography

Provides an excellent, quick overview and shows the place of software engineering concerns within the general area of operating systems.


Topics: Process level

Keywords: types of concurrent processes, synchronization, semaphores


Topics: Evaluation

Keywords: software monitor


Topics: Description techniques

Keywords: simulation, processes, dynamic environment

Dani64 Almon E. Daniels. Some observations concerning large programming efforts. Prcc. AFIPS 1964 SJCC, Vol. 25, pp. 231-238.

Topics: Design methodology

Keywords: principles

Among others, the author develops the following conclusions: the users must be involved with the design, construction and testing; a full design is desirable, but an
implementation course should be followed which produces a usable sub-system early; changes and modifications should be planned for; documentation should not be sparse; an environment simulator will be needed to fully exercise the system.


Topics: Computer organization
Keywords: microprogramming, survey, annotated bibliography


Topics: Process level
Keywords: synchronization, semaphores


Topics: Description techniques
Keywords: paging behavior, process


Topics: Evaluation
Keywords: modeling, statistical, external events


Topics: Description techniques
Keywords: behavior, working set model

'This is a paper about the history of the working set model for program behavior. It traces briefly the origins and bases for the idea and some of the results subsequently obtained.' (author)

In this page an interpreted graph description of algorithms is developed which facilitates program generality (independence of the algorithm description from the environment in which it is to be used) and the description of deterministic asynchronous behavior. The implications of the description scheme for computer organization are discussed.

Our work will discuss a particular measurement tool, called the informer, now being used to measure the time-sharing operating system of the SDS-940 at Berkeley. The informer provides an environment in which user programs may serve as measurement routines. Measurement routines may be inserted dynamically to be called when control reaches arbitrary locations within the measured program. An important quality of this measurement facility is that no error in its use can cause a system failure. Also, system degradation due to measurement programs can be automatically controlled. (author)

Dijkstra proposes a solution to the mutual exclusion problem and verifies that it is correct.

E. W. Dijkstra. Letter to the editor: Go to statement

**Topics:** Sequential programming level

**Keywords:** go-to-less languages, control constructs, program clarity

---


**Topics:** Design principles, Process level

**Keywords:** concurrent processes, semaphores, levels of abstraction

---


**Topics:** Verification

**Keywords:** bottom-up, a priori proofs

---


**Topics:** Process level

**Keywords:** synchronization, semaphores

A treatment of the problems of synchronization among asynchronously operating sequential processes and their solution by the use of semaphore variables and operations.

---


**Topics:** Design principles

**Keywords:** hierarchical, bottom-up

This article reports on some of the reasons for structuring the T.H.E. system as was done -- it doesn't offer any principles for determining the layers ("levels of abstraction") but rather indicates some of the (hand-)analysis of interdependencies which led to the final layer organization.

---

The Netherlands, April 1970.

Topics: Sequential programming level

Keywords: structured programming


Topics: Sequential programming level

Keywords: structured programming, validation, string of pearls, modification, adaptation

In this article, Dijkstra reports on some experience and insight gained from programming experiments with small to modest sized systems and hypothesizes about their applicability to large scale system problems. The conclusions which he relates all pertain to the relationship between program structure and the possibility and ease of demonstrating program correctness. First, since the number of instances in a family of computations carried out by a program is generally very large, validation by sampling is impossible and any proof of correctness must rely wholly upon the text of the program. This coupled with the high probability that proof difficulty will explode with increasing program size implies that program structuring mechanisms must be used which allow validation without tremendous effort. Second, structured programming is required in order to impose sequencing disciplines which facilitate the statement of assertions concerning possible computations. Finally, the result of a design is not only a single program but rather a "string-of-pearls", where each pearl embodies the consequences of a single (independent) design decision (which jointly refines the specification of program behavior and data structure) and which is a natural element for change during program modification or adaptation.


Topics: Sequential programming level

Keywords: concurrent processes, hierarchical, bottom-up validation

The paper initially discusses a spectrum of synchronization problems and proves the correctness of the proposed solutions. This initial discussion is motivated by a desire to provide a base layer in a hierarchical system which "does away with interrupts" and provides an environment for synchronization of concurrent processes. Then it is noted that 1) in implementing this layer, processes within their critical sections should receive
preferential scheduling treatment, 2) reproducibility of external timing (which is one factor in accounting) is not guaranteed because of the critical section conflict, and 3) the common semaphore variables cause confusion in trying to delimit the state space of a process. To solve these problems it is proposed that all critical sections be grouped into a "secretary" process which the other processes, now called "directors," can call upon for services. It is pointed out that a hierarchy of directors, secretaries, subsecretaries, etc., impose an ordering which precludes deadlock as far as mutual exclusion is concerned.


Topics: Validation

Keywords: hardware, levels of description

'The experimental three-level modeling technique discussed in this paper can be used during the design stage of a system for identifying mismatches among the architectural, microprogramming, and hardware logic levels. Compatible switching between modeling levels is emphasized. Execution of an application program by the architectural and microprogramming level models with switching between levels is illustrated.' (authors)


Topics: Design principles

Keywords: operating systems

'Basic Time-sharing is an attempt to systematize the computing principles which should underlie the design of operating systems. These principles emphasize the fundamental separateness of work and resources and the parallelism inherent in the computing process. ... BTS provides a vocabulary of system design, not a design template.' (authors)


Topics: Description techniques

Keywords: pictorial, interactions, analysis

Eckh71 Richard H. Eckhouse, Jr. A High-Level Microprogramming

Topics: Computer organization

Keywords: microprogramming languages


Topics: Validation

Keywords: error detection, testing


Topics: Description techniques

Keywords: models of computation, program schemata, validation, equivalence, bibliography

An excellent review of the work in modeling and analysis of sequential computation.


Topics: Validation

Keywords: fault tolerance, verification

'Dynamic verification of a decision implies that every time the decision is made there is a consistency check performed on the decision using independent hardware and software. This technique, properly applied, allows the construction of an operating system which can be guaranteed not to make certain decisions improperly even in the presence of a single hardware or software failure.' (author)


Topics: Description techniques

Keywords: computer hardware, APL language

Falk70 A. D. Falkoff. Working paper: Criteria for a system design language. In Burton and Randell (eds.), Software
From the premise that a system is a collection of programs, Falkoff proposes that a formal programming language is the appropriate medium for the specification of a system design. He then discusses, with respect to Iverson's APL language, eight more concrete criteria: easy to use, permits suggestive formulations, allows formal manipulation, leads to aesthetic criteria for quality judgements, convenient for documentation, non-specific as to technology and problem area, applicable to systems of any complexity and executable on the machine. In the thinking of this reviewer, some of the criteria in this list are questionable (especially the last one) and the list is incomplete. Two missing criteria are the ability to make and elaborate abstractions and the ability to describe asynchronous behavior -- these would not be satisfied to sufficient degree by most current formal programming languages. Falkoff seems to confuse design and implementation which are distinct activities (with distinct description scheme requirements) even though they progress simultaneously for the most part.


An excellent survey of the pre-1968 developments in systems for the writing of programming language translators. Contains an extensive bibliography.


Topics: Sequential programming level

Keywords: assembler languages, regular expressions, structured programming

'A generalized assembler language with possible applications in the areas of computer networks and software support packages is developed and discussed. ... The use of regular expressions for branching has interesting effects on the structure of programs. The most notable effect is that it requires programs to be segmented into subprograms which consist entirely of sequentially processed instructions. This lends to easier documentation of assembler languages and to simpler programming of lookahead machines.' (authors)


Topics: Evaluative

Keywords: bibliography


Topics: Sequential programming level

Keywords: control structures


Topics: Validation

Keywords: assertions, predicate calculus


Topics: Validation

Keywords: interactive, iterative, assertions
This paper presents an imagined dialogue between a user and a computer in which the computer checks for syntax errors in the program being developed and checks for inconsistency of the operation of the program with respect to stated expectations.


Topics: Evaluation

Keywords: analytical model


Topics: Process level

Keywords: programming languages


Topics: Evaluation

Keywords: prediction, evaluating alternatives, performance


Topics: Validation

Keywords: automatic error detection, automatic error correction, programs


Topics: Design methodologies

Keywords: general, model, bottom-up, top-down

A model of the design process is developed in two stages, corresponding to the task environment of design and the activity of posing and solving design problems. Use of the model with top-down and bottom-up disciplines is discussed. An example of the design of an object using a semi-automated design system based on the model is
presented. Several issues raised by the model's qualitative aspects, its suitability to automated design, and lines for further development are discussed. (authors)


Topics: Sequential programming level
Keywords: System programming languages, translators


Topics: Integrated simulation
Keywords: Mechanism, use in debugging, use in program expansion


Topics: Validation
Keywords: Debugging techniques

A very brief review of the actual presentation, composed of the abstract of Gaines' doctoral thesis.


Topics: Computer organization
Keywords: Concurrent processes, levels of control, supervisory services


Topics: Description techniques
Keywords: Hierarchical decomposition, program structure


Topics: Design principles
Keywords: planning, testing, documentation


Topics: Sequential programming languages

Keywords: coroutine constructs


Topics: Design principles, Implementation tools

Keywords: systems programming languages, interprogram communication conventions, programming conventions, precompilers


Topics: Description techniques, Validation, Process Level

Keywords: concurrent processes, state transition

The behavior of a system of asynchronous processes is modeled as a sequence of transitions from one state (set of data values and process configurations) to another. A mechanical procedure is provided for deriving, from the individual steps in the sequential program for each process, transition rules for the state change behavior of the entire system. The transition rules may then be used to generate state change sequences so that the existence of a desirable state change or the absence of an undesirable state may be mechanically verified.


Topics: Design methodologies

Keywords: top-down, bottom-up, others

Gill argues that, in practice, strict top-down or bottom-up methodologies are never used -- rather the process starts at both ends and meets in the middle. This results from the need to assure that aspects introduced at either end (system features, hardware capabilities) do not propagate too far toward the other end; that ambiguity can
he tolerated in progressing top-down but not bottom-up; that
commonality can be discerned during a top-down design but
only guessed at during a bottom-up design; and variability
of the target environment cannot be incorporated with a
bottom-up approach whereas variability of the produced
system cannot be handled in a top-down approach. This last
point leads the author to distinguish a middle-out
methodology -- based on the definition of a universal
intermediate system -- for use in designing a system which
has multiple sets of features (multiple intents) and which
is to run in several differing environments.

Glas71 E. L. Glaser, et. al. The LOGOS Project. Jennings
Computing Center Report, Case Western Reserve Univ.,
Cleveland, Ohio, October 1971.

Topics: Software engineering laboratory

Keywords: description techniques, validation, hardware and
software, data base management

This is a collection of papers concerning the design and
implementation of a computer-aided design system for
integrated hardware and software systems.

Good70 D. I. Good. Toward a man-machine system for proving program

Topics: Validation

Gorn59 S. Gorn, P. Z. Ingerman and S. B. Crozier. On the
construction of micro-flowcharts. Comm. ACM, 2, 10
(October 1959), 27-31.

Topics: Description techniques

Keywords: instruction logic, microprograms, automatic
generation, flowcharts

Gosd66 J. A. Gosden. Explicit parallel processing description and
control in programs for multi- and uni-processor computers.

Topics: Sequential programming level, Computer
organization, Sequential programming level --> Process level

Keywords: concurrent processes, programming languages,
sequencing analysis

This paper reviews the evolution of parallel processing
and constructs for denoting parallelism within an algorithm.
The author then proposes a simple set of commands which
facilitate solutions to the problems of locating,
specifying, and sequencing concurrent processes.

Topics: Portability

Keywords: data exchange

An outline of the problems of data exchange.


Topics: Evaluation

Keywords: general

The authors organize the area of software evaluation tools into 1) modeling (composed of analytic and simulation techniques) which is used mainly during the design and implementation phases and has the intent of predicting system performance and selecting quantities to be monitored and 2) (hardware and software) monitoring which is used mainly during the implementation, optimization and maintenance phases and has the intent of measuring system performance and selecting design parameters for modeling. In tabular form, they compare the various types of tools and review some currently used monitors.


Topics: Management

Keywords: personnel, programmers

'The problems of managing programmers have been growing rapidly in recent years. The specific issues referred to are the increasing costs of programming and the failure to complete projects when scheduled or needed. This paper proposes more effective supervision of programmers together with improved management policies and procedures.' (author)


Topics: Design principles

Keywords: integrated simulation
This is a preliminary design report on a system to facilitate the design and evaluation of software systems oriented around the embedding in PL/1 of constructs for simulation.


Topics: Sequential programming level

Keywords: compilers, translator writing systems, validation, error detection, error correction, bibliography

A complete treatment of the area of compiler implementation.


Topics: Validation

Keywords: debugging


Topics: Validation

Keywords: debugging systems, interactive

This article discusses the facilities needed in a debugging system for a high-level language and some techniques for their efficient implementation. A specific system, AIDS, is used for examples.


Topics: Description techniques

Keywords: sequential processes, concurrency, sharing


Topics: Validation

Keywords: concurrent processes, semaphores, bottom-up

Formalization of a well-defined synchronization
A mechanism can be used to prove that concurrently running processes of a system communicate correctly. This is demonstrated for a system consisting of many sending processes which deposit messages in a buffer and many receiving processes which remove messages from that buffer. The formal description of the synchronization mechanism makes it very easy to prove that the buffer will neither overflow nor underflow, that senders and receivers will never operate on the same message frame in the buffer and that they will not run into a deadlock." (author)


Topics: Description techniques

Keywords: flowcharts, automatic generation

This article describes a program for the automatic generation of flow charts from IBM 704 assembly code. The value of a series of flow charts at different levels of detail was realized and the production of the series was incorporated into the program.


Topics: Description techniques

Keywords: directed graphs


Topics: Functional level

Keywords: description, system externals, validation

Discusses, in the context of a Fortran compiler, what needs to be specified concerning the external characteristics of a piece of software and the tests which will be applied to check its validity.


Topics: Computer organization

Keywords: software design, hardware/software compatibility, system programming languages
This paper explores some of the methods and implications of a software design viewpoint which holds that 1) software should be implemented in a conceptually rich high-level programming language and 2) machines should be designed so as to reduce the amount and difficulty of translating the programs.

Head 63  

Topics: Description techniques, Functional level

Keywords: functional specification, file specifications, flow charts, complexity

"Problems in the implementation of large real-time applications are treated, and suggested guidelines for both program and file specification are developed. The problems delineated also concern in systems programming." (author)

Heart 70  

Topics: Process level

Keywords: communication, networks

Hell 66  

Topics: Sequential programming level -- Process level

Keywords: algorithm, sequencing analysis

Hen 72  

Topics: Sequential programming level, Validation

Keywords: structured programming, assertions, top-down validation

The top-down development of a simple program is traced. Then an error in the program is located by an analysis which is guided by the successive descriptions of the program. The authors point out several difficulties with the top-down approach: since the elements being worked with are not fully specified, there is a problem in conveying their meaning formally; the data structures must be elaborated in conjunction with the elaboration of the programs and this concurrent elaboration has pitfalls; the method is simple and its simplicity lulls the user into a false sense of
security.

**Hig68**  

Topics: Reliability

Keywords: error recovery

This article discusses recovery management support for the IBM System 360/65 -- a piece of software which attempts to handle recovery from errors due to hardware malfunctions.

**Hoar71**  

Topics: Validation

Keywords: invariants, top-down, program structure, iterative

'This paper has illustrated a methodology for systematic construction of program proofs together with the programs they prove. It uses a "top-down" method of analysis to split the process into a number of stages, each stage embodying more detail than the previous one; the proof of correctness of the program at each stage leads to and depends upon an accurate formulation of the characteristics of the programs to be developed at the next stage.' (author)

**Holt68**  

Topics: Description techniques, Process level

Keywords: concurrent processes, synchronization, occurrence graphs, petri nets

**Hopk70**  

Topics: Sequential programming level --&gt; Computer organization

Keywords: performance, binding, system generators

After developing the conclusion that local decisions made for improving performance tend to increase the complexity of the system and actually reduce the overall system performance, it is proposed that aids be provided in the form of 1) compilers and interpreters which allow several
iterations of design and implementation, concentrating on
the completeness and validity of the system, to be
accomplished before the system is tuned to a specific
environment, and 2) system generation processors which
facilitate the binding of the system to a particular
environment in a way which enhances the system's
performance.

Hopp69 G. M. Hopper. Position paper for a panel on Software
Transferability: Standardization of high-level languages.

Topics: Portability

Keywords: standardization

This is an outline of a discussion of the mechanism,
problems, and benefits of standardization.

Horn69 J. J. Horning and E. Randell. Structuring Complex
Processes. RC 2459, IBM Research Lab., Yorktown Heights, N.
Y., 1969.

Topics: Description techniques

Keywords: concurrent processes, levels of description,
communication

Report Series 31, Computing Lab., Univ. of Newcastle Upon

Topics: Description techniques

Keywords: design principles, design methodologies,
concurrent processes

'The general concept of a 'process' has become
increasingly important both as a tool for gaining an
understanding of complex computer systems, and as a basis
for organizing their design. This has occurred despite the
lack of general agreement on a definition of the concept.

'The present paper provides a framework for the
definition and extended discussion of processes and
techniques for structuring them. It uses the concept to
unify a variety of hardware and software tools and
methodologies.' (authors)

Huss70 Samir S. Hussan. Microprogramming: Principles and

Topics: Microprogramming

Topics: Evaluation

Keywords: modeling, hierarchical, analysis

A modeling scheme is proposed to allow configuration independent performance criteria to be established and a case study is given, introducing a method of analysis which is based on the hierarchical decomposition of the system.


Topics: Descriptive techniques


Topics: Process level

'in conjunction with an earlier paper, this step toward machine design based on the structure of nested-declaration pure-procedure multiple-activity algorithms describes: the declaration, flow-control, and some data-addressing mechanisms of such algorithms; the dynamic Saguaro-Garden data-access-providing structure of their Records of Execution; and implementation in a segmented environment.' (author)


Topics: Descriptive techniques

Keywords: sequential programs, semantics

'there appears to be a need for an intuitive implementable cell-based model of the semantics of algorithm execution. It is contended that for block structured processes, such a model must be based on the concepts of nested blocks, access environments, labels, and cell retention. A model so based, the contour model, is introduced by illustrating the execution of an Algol 60 program in terms of the model. The
model is informally specified and some of its major features are pointed out.' (author)


Topics: Computer organization
Keywords: microprogramming, bibliography


Topics: Design principles
Keywords: hierarchy, structure

This article quickly reviews three existing systems (Atlas Supervisor, CDC Scope system, and OS/360) and then culls some structuring principles pertaining to files, simplicity, dynamicity, protection, and hierarchy.


Topics: Validation
Keywords: debugging, graphic display, light pen


Topics: Modularity


Topics: Validation
Keywords: assertions, graph model, interpreted nodes

'First, the problem of proving the correctness of an operating system is defined. Then a simple model is presented. Several examples are given to show how this model allows derivation of proofs about small systems.' (author)

Kapi71 Uy. K. Kapitonova and A. A. Letichevskii. Design problems of software special systems. Proc. IFIP Congress 71,
In this article are considered some problems and difficulties arise in the process of developing a complex system of programs for practical realization of computer-aided digital system design.


...we very briefly survey what results have been achieved in the areas of equivalence, termination and correctness.


A directed graph model is defined in order to study termination, determinacy, and queue boundedness properties of parallel computations. Since data dependent decisions, structured data and procedures are not part of the modeling scheme, it is not easy to express programs directly in the modeling scheme.


A modeling scheme is introduced for the study and description of algorithms containing concurrent activity. Results are given concerning: necessary and sufficient conditions for determinacy; decision procedures for determinacy, termination, etc.; and the unsolvability of the
general equivalence problem.


Topics: Evaluation
Keywords: "stimulus" testing, monitors


Topics: Design principles
Keywords: general, bibliography, operating systems

'The purposes of this paper are threefold: (1) to present the basic properties with which operating systems can be grouped, classified, and evaluated; (2) to identify the major categories into which operating systems can be classed and to give concrete examples of each; and (3) to discuss resource management in operating systems with an emphasis on storage allocation and processor scheduling.' (author)


Topics: Management
Keywords: documentation standards, documentation techniques


Topics: Management
Keywords: planning, personnel, programmers, evaluation


Topics: Evaluation
Keywords: software monitors, integration into system


Topics: Validation
Keywords: assertions, checking logic during computation


Topics: Validation
Keywords: automatic, assertions


Topics: Validation
Keywords: assertions, formal correctness proofs

A review article on King's compiler which incorporates predicate calculus proof techniques proposed by Floyd to check the consistency between the actual program behavior and assertions describing the expected behavior in terms of relations among the values of the program's variables.


Topics: Validation
Keywords: testing

This article gives a very brief overview of a debugging tool, available to TSS/360 system programmers, developed for the testing of supervisory and system routines.


Topics: Evaluation
Keywords: networks, analytic, simulation


Topics: Process level
Keywords: synchronization, semaphores
Knuth points out that Dijkstra's solution to the mutual exclusion problem permits the lockout of one of the concurrent processes. He proposes the use of a queueing scheme with a first-in-first-out discipline as a solution which does not permit lockout.


Topics: Sequential programming level
Keywords: go-to-less programming languages


Topics: Sequential programming level --> Process level
Keywords: sequencing analysis

'A model "system-program" for parallel computation is introduced in this report and a set of concepts and objects associated with it is considered. In the framework of the model the problem of transformation of sequential programs into parallel asynchronous programs is considered.' (authors)


Topics: Description techniques
Keywords: algebraic, flowchart, automatic generation of flowcharts.


Topics: Validation
Keywords: debugging


Topics: Validation
Keywords: debugging systems, interactive, extensible

Topics: Validation, Design principles

Keywords: text manipulation, testing, fallback

'Testing and development of elaborate systems requires the assistance of sophisticated techniques and software aids at several stages in the process. These include systems for managing and manipulating the source text, producing and testing new versions in a realistic but 'safe' environment and arranging for easy and compatible fallback to a previous version in case of unforeseen difficulties.' (authors)


Topics: Sequential programming level

Keywords: systems programming languages

'The purpose of this article is to explain the thinking behind SAL rather than to expound on the finer details of the language itself. We feel that this type of language which combines the freedom and flexibility of assembly code with many of the facilities normally associated with high-level languages, could be useful to many other workers.' (author)


Topics: Sequential programming level

Keywords: system programming languages, attributes

After reviewing the advantages (efficient coding, machine dependency, flexibility, etc.) and disadvantages (difficulty in comprehension, poor conceptual framework, machine dependency) of assembler languages as regards systems implementation, the author discusses several attributes which a viable alternative must possess: clarity of expression, efficient compilation of efficient code, high power-to-complexity ratio of the constructs in the language, permission of machine dependent programming if desired by the user, no run-time support package, instructions for which it is apparent how they are realized, and the ability to work with several languages within one system.

Topics: Evaluating
Keywords: analytic models


Topics: Description techniques
Keywords: syntax, semantics, Vienna definition language


Topics: Sequential programming level
Keywords: control extensibility

'A proposal is made for defining new control structures in a high level language. This is done by writing state transforming functions of the abstract machine which interprets the language. The paper describes the programming language, its interpreter, and a primitive function which gives the programmer access to the current state of the machine.' (author)


Topics: Computer organization
Keywords: microprogramming, emulation, control structures


Topics: Computer organization
Keywords: microprogramming, concurrent processes, control data structure, emulator

This thesis develops a new computer organization which is based on dynamic reconfigurability through the use of a data structure for describing control and a set of control data structure manipulation instructions. The power of the organization is shown by presenting an emulator for a
machine organization based on Adams' graph model in which parallel activity is the norm. The feasibility of the organization is demonstrated by presentation of an architecture based on multiple, parallel microprogrammable processors and bit-addressable memories connected by a bussing structure.


Topics: Management

Keywords: specifications


Topics: Evaluation

Keywords: acceptance testing, performance, validation

The authors present a general approach to the acceptance testing of software. The problems of defining the scope of the test and the acceptance criteria are pointed out and four main aspects of the testing process -- documentation, availability, detailed verification of facilities, performance -- are demarcated.


Topics: Sequential programming level, Sequential programming level -> Computer organization

Keywords: structured programming, hierarchical, sequencing

This paper proposes that the internal form of a user's (Algol) program be organized to directly reflect the source-level structure built hierarchically from the statement as the basic unit. The benefits which accrue are cited to be incremental compilation without tremendously high overhead due to interpretation, sequencing controlled by the structure of the program, and facilitation of inter-user protection.


Topics: Validation
Keywords: case analysis, assertions, mathematical induction, bottom-up


Topics: Validation

Keywords: bibliography


Topics: Validation

Keywords: programs, bibliography


Topics: Sequential programming level

Keywords: segmentation


Topics: Evaluation

Keywords: survey, bibliography

Lucas68 P. Lucas, P. Lauer and H. Stiglmeitner. Method and Notation for the Formal Definition of Programming Languages. IFM Lab., Vienna, Austria, June 1968.

Topics: Description technique

Keywords: syntax, semantics, Vienna definition language


Topics: Description technique

Keywords: correctness, non-axiomatic approach, bibliography

Lucas presents a review of existing definitional systems,

Topics: Process level

Keywords: concurrent processes, properties, directed graph, simple data types

'The "computational schema" is introduced as a means for describing asynchronous multi-process computations. Communication constraints which guarantee determinacy of schema execution are developed, and an effective procedure for determining the equivalence of deterministic schemata is presented.' (author)


Topics: Evaluation

Keywords: hardware monitoring

Report on the hardware mechanisms for monitoring the operation of the 1108 operating system. The technique used was the counting of significant events and the presentation of graphical and statistical analyses after offline data reduction.


Topics: Computer organization

Keywords: multiprocessor, functional units, communication


Topics: Software structure, Design principles

Keywords: hierarchical, modular

Dijkstra's "levels of abstraction" framework is applied to file systems.

Topics: Validation, Description techniques

Keywords: model of computation, graphical, properties of algorithms, termination


Topics: Validation


Topics: Validation


Topics: Validation, Description techniques

Keywords: model of computation, graphical, properties of algorithms, termination


Topics: Sequential programming level, Description techniques

Keywords: automatic synthesis, input/output relations, mathematical induction


Topics: Description techniques, Evaluation

Keywords: directed graph, model of computation, performance evaluation

Maly reviews several considerations which are managerial in nature but which he feels lead to an incompetent design if they are not considered. He breaks the design cycle into several steps: determination of available manpower and monetary resources, determination of the objectives as warranted by the scope and demands of the community of potential users, cross design including the functional and inter-module communication specifications, the establishment of a plan for detailed design which demarcates checkpoints and milestones, and the detailed design. The pitfalls which are present are held to be, in part: unstated or controversial objectives, design oversights, design decision procrastination, design decision prematurity, design decision reversal, infrequent review, and lack of control of the design process and the making of changes in the system.


Topics: Complexity
Keywords: hierarchical decomposition


Topics: Management
Keywords: personnel

A personal view of how to organize a group of systems programmers based on observations about what is wrong with the software produced by manufacturers and what is right about the software produced by users, mainly those connected with a university.


Topics: Evaluation
Keywords: testing, generating test data


Topics: Descriptive techniques
Keywords: documentation, automatic control of generation


Topics: Sequential programming level, Validation

Keywords: structured programming, debugging, error detection

An introduction to structured programming and the ways in which it can reduce the number of errors by localizing them and the help it can give in detecting errors by guiding the testing process.


Topics: Process level

Keywords: limited resources, concurrent processes, communication

The authors propose a system organization viewpoint in which all programs are asynchronously operating data transformers, communicating via data buffers which effect the necessary transmission coordination. They discuss the advantages of the concommitant late binding of inter-module communication paths.


Topics: Process level

Keywords: concurrent processes, communication


Topics: Portability

Keywords: programs, data exchange

Discusses the problem and some possible solutions.

Morg70a Howard L. Morgan. Spelling correction in systems programs.
Corollary. ACM, 13, 2 (February 1970), 90-94.

Topics: Validation

Keywords: debugging, detection, automatic correction


Topics: Process level

Keywords: event-driven programming, interrupts


Topics: Computer organization, Process level

Keywords: virtual processor, concurrent processes, process creation, process activation, communication

The authors review the design of a virtual processor in which concurrently operating processes are the primitive elements. Specifically covered are the mechanisms for process creation and deletion and inter-process communication.


Topics: Computer organization

Keywords: language-oriented machine design

Presents an Algol-like language designed to be an 'aid to the programmer in thinking about his problem and the processes he is using to solve the problem,' and a computer organization designed specifically to support the features of the language.


Topics: Validation

Keywords: equivalence

'Interpreters for programming languages may be defined in terms of the information structures (data structures)
generated during program execution. This paper develops a class of proof techniques for proving equivalence between two classes of programming languages interpreters referred to as associative interpreters and sequential interpreters.' (authors)


Topics: Design techniques

Keywords: modularity, modifiability, standardization

It is proposed that the techniques of mass-production -- subassemblies, interchangeable parts, machine tools, standard parts -- have direct applicability to the solution of the software problem. Discusses the possible parameter scales along which the individual members of a family of software components may vary -- precision, mode, robustness, size, interface characteristics, time-space behavior, etc. Also notes that a components supplier must possess techniques for generating the components and must handle testing, distribution, documentation, etc.


Topics: Computer organization

Keywords: language-oriented machine design

'The obvious attack for programmers and hardware people together is to devise a language that reflects what we want to do and how we do it (for instance, in parallel) and machine structures effective in handling that language. Let us call this method "language directed computer design."


Topics: Sequential programming level

Keywords: translator writing system, system structure


Topics: Sequential programming level, Portability

Keywords: translator writing systems, localized environment dependencies, compiling techniques

Topics: Management

Keywords: cost estimation


Topics: Validation

Keywords: assertions

'A constructive approach to the question of the proof of algorithms is to consider proofs that an object resulting from the execution of an algorithm possesses certain static characteristics. ... The stepping stone of this approach is what is called General Snapshots, i.e., expressions of the static conditions existing whenever the execution of the algorithm reaches particular points. General Snapshots are further shown to be useful for constructing algorithms.' (author)


Topics: Sequential programming level

Keywords: structured programming, validation

'The paper describes a programming discipline, aiming at the systematic construction of programs from given global requirements. The crucial step in the approach is the conversion of the global requirements into sets of action clusters (sequences of program statements), which are then used as building blocks for the final program. The relation of the approach to proof techniques and to programming languages is discussed briefly.' (author)


Topics: General

The initial part of this report relates much of the conference discussion concerning software engineering in general and the design, production and service of software in particular. Much of the interest at this conference was on management and the discussion reflects this. Invited
addresses and reproduced working papers are entered separately in this bibliography.

**Need69**


**Topics:** Design principles  
**Keywords:** theoretical insight, horse sense

**Need70a**


**Topics:** Sequential programming level  
**Keywords:** operating systems, debugging, design principles, test-mode

The author discusses four principles which he feels should be observed during the construction of operating systems and the software aids which are required: it should be convenient for several people to work on a common version of the source code (requiring a file manipulation system); it should be easy to diagnose faults (implying a reasonably powerful debugging facility); it should be possible to execute perhaps faulty versions of the system (implying a fall-back facility and a test-mode for system operation); and no code should be written which is consciously intended to be re-written.

**Need70b**


**Topics:** General  
**Keywords:** definition

In their mind the difference is clear: the computer scientist is a theoretician and the software engineer a pragmatist, and each cares only secondarily about the other's area.

**Need71**


**Topics:** Tolerance
'It is commonplace to build facilities into operating systems to handle faults which occur in user-level programs. These facilities are often inadequate for their task: some faults or incidents are regarded as so bad that the user cannot be allowed to act on them and this makes it difficult or impossible to write subsystems which give proper diagnostics in all cases, or which are adequately secure, or which are adequately robust. This paper looks into why there is a need for very complete facilities and why there is a problem about providing them, and proposes an outline structure which could be used.' (author)


Topics: Evaluating

Keywords: monitoring

'A general discussion of the measurement of software systems is followed by a description of a hardware and software scheme for measuring user programs in a time-shared environment.' (authors)


Topics: Design principles

Keywords: general

'This paper attempts to summarize the most significant principles, to evaluate their applicability in the real world of large multi-access systems, and to assess how they can be used more effectively.' (author)


Topics: Sequential programming level, Description techniques

Keywords: system model, directed graph, state transition, non-hierarchical

This paper describes a language for the programming of interactive, graphical, problem-oriented languages. The language permits the description of the system as a directed graph where the nodes represent states of the system and the arcs indicate the output produced and the operation done
when a state transition is elicited by the actions of the user.


Topics: Validation

Keywords: simplification, equivalence

'We have studied the problem of automatic program simplification by describing a number of equivalence-preserving transformations which are suitable for mechanization and yield an obvious improvement.' (author)


Topics: Sequential programming level

Keywords: uniform control constructs, structured programming, debugging

'A new technique for directing the flow of control in block structured programs is described. It allows the programmer to delay using the outcome of a test indefinitely, and to make the entry and exit of begin-end blocks conditional upon the last test performed. This technique is the main tool for directing the flow of control in the programming language Nucleol, and is considered to be particularly useful for program debugging.' (authors)


Topics: Description techniques

Keywords: Petri net


Topics: Evaluation

Keywords: software monitor, simulation

Topics: Design methodologies

Keywords: automated, optimization, SODA

This article describes a system for the automatic design of "report-oriented" information processing systems. Using algorithms from graph theory and mathematical programming and heuristics, the system carries out a multi-level decision making process with respect to performance criteria and data transformation objectives for a particular system. The output consists of specifications for hardware selection, program module organization, file organization, and scheduling algorithms. The system is limited and may be used only for a batch environment where there is a single processor, linear data structures, and sequential auxiliary storage devices.


Topics: Sequential programming level

Keywords: concurrent processes, programming languages, parallel processing

The author proposes some high-level programming language control constructs, motivated by a desire to facilitate translation of the program for an environment having parallel computers, for the description of the sequencing among concurrent processes.


Topics: Complexity

Keywords: software, handling

The author recognizes the problems of large-scale systems as ones of organization, of both the system and the personnel, and descriptions. He argues that some of the solutions are a universal machine-oriented language, standard "parts" of systems, and hierarchical organization.


Topics: Portability

Topics: Computer organization

Keywords: bottom-up design, primitive modules

The authors describe the primitive building blocks which form a basis for the bottom-up design of computer system discussed by W. A. Clark. The modules operate asynchronously with coordination being effected by both data flow and control signals.


Topics: Description techniques

Keywords: timing, asynchronous systems, propositional calculus.

"After defining precisely what we mean by a process and a system of dependent processes, we describe an algorithm to obtain a formula (the SSP) using the connective V (OR), A (AND) and justaposition (follows) which expresses dependence properties of the system. We then convert the SSP into a logical proposition called the SSP and show how this proposition can be used to determine the probability of system success from the stochastic properties of the processes. We also describe an updating procedure for the SSP which is used to determine whether each process is an essential, merely useful, or useless part of the system at any time during its asynchronous operation." (author)


Topics: Description techniques

Keywords: function, synchronous systems, simulation, synthesis

Parn66a D. L. Parnas. A language for describing the functions of synchronous systems. Comm. ACM, 9, 2 (February 1966), 72-76.

Topics: Description techniques

Keywords: function, synchronous systems, simulation, synthesis

Topics: Descriptive techniques

Keywords: synchronous systems, input/output transformation, state transition description


Topics: Design methodologies

Keywords: top-down, integrated simulation, hierarchical, recursive

The Structure Oriented Description And Simulation language allows the description of a system as it is hierarchically composed of modules operating asynchronously. An example is considered and "our work on the example has indicated that the use of SODAS forces a discipline upon the system designer that could greatly improve the quality of his work and reduce the time needed to complete a project."


Topics: Evaluation

Keywords: concurrent processes, simulation


Topics: Descriptive techniques, Functional level

Keywords: transition diagrams


Topics: Design methodologies

Keywords: integrated simulation, multi-level, sequential process, hardware/software decisions

This paper extends the methodology presented in the paper by Parnas and Darringer to multiple level systems.
This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time. ... We have tried to demonstrate by these examples that it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose, instead, that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others. Since, in most cases, design decisions transcend time of execution, modules will not correspond to steps in the processing. To achieve an efficient implementation, we must abandon the assumption that a module is one or more subroutines, and instead allow subroutines and programs to be assembled collections of code from various modules." (author)
The author discusses an approach to the specification of the function of software modules which was developed with the intent to allow specifications which are precise and complete enough to allow other modules to be written to interact with the module and which give no more information than required. For each module is given a set of "functions" which are each specified by giving its possible values, its initial value, the types of its parameters, and its effect. The effect of a function is stated as a sequence independent series of statements each of which exhibits the change in value of other functions in terms of the values of the other functions and of the parameters. An initial segment of the effect specification gives the error conditions which can occur upon invocation of the function -- the user of the function must provide error handling routines.


Topics: Process level
Keywords: synchronization


Topics: Process level
Keywords: synchronization, semaphores, Petri nets

Noting that semaphores were introduced in order to eliminate the need for busy-waiting (the continuous checking of some condition until it turns true), Patil imposes the strict limitation that there be no conditional checking and shows that, as a result, there is a synchronization problem which cannot be solved by using semaphores. An alternate synchronization mechanism, incorporating condition testing without busy waiting, is proposed and shown to provide a solution. Petri nets are used in the proof.


Topics: Sequential programming level
Keywords: translator writing systems
This article concerns a translator writing system called COMASS which the author indicates might be useful for writing other pieces of software but no justification for that claim is given.


Topics: General

Keywords: complexity, education, hierarchical, virtual machines

Perlis first attributes the software problem to 1) the change of scale required by the increased demands of the user community and 2) the uneven training of the software practitioners. He then proposes that the problem may most naturally be handled by the design of a hierarchy of virtual machines.


Topics: Education

Keywords: curricula

A position paper for a panel discussion on the education of the computer professional. Perlis discusses the general orientation of courses for software engineers.


Topics: Description techniques

Keywords: state descriptions, Petri nets


Topics: Description techniques

Keywords: flowcharts

Proposes an extension to flowcharting techniques for the description of real-time events such as interrupts.

Pili67 M. S. Pilgian and J. L. Pokcrney. Air Force concepts for

Topics: Management, Validation
Keywords: control, acceptance testing


Topics: Evaluation
Keywords: monitoring, software monitor


Topics: Evaluation
Keywords: performance measurement, software monitors

The author places continuous software monitoring within the spectrum of methods for collecting data pertaining to system performance and relates that it can be provided easily and without introducing too high an overhead.


Topics: Sequential programming level, Validation
Keywords: compilers, translator writing systems, error detection, error correction, bibliography


Topics: Sequential programming level, Validation
Keywords: compilers, translator writing systems, error detection, error correction

A collection of important papers in the areas of compiling techniques and translator writing systems.


Topics: Portability
Keywords: abstract machine, macros, boot-strapping


Topics: Design principles

Keywords: interactive development, integrated simulation, validation

'COTAN is a multi-access system... The development of COTAN relied heavily on the continuing availability of online access from an early stage in the project. Access was first obtained through the monitor console on the KDF9 and this was then used to 'bootstrap' the implementation of the multi-access facilities. The new software was checked out in a realtime, online tested called an 'environmental simulator'. This provided a close approximation to the ultimate operating conditions and enabled the system to be brought to a high degree of reliability before being placed in service.' (author)


Topics: Descriptive techniques

Keywords: hierarchical, graphical, program semantics


Topics: Sequential programming level

Keywords: translator writing systems, description, evaluation


Topics: Evaluation, Computer organization

Keywords: performance measurement, compilers, influence on computer organization

In this paper we shall discuss a specific programming language translator and a measurement control center, and associated software artifact, incorporated into the translator at implementation time in order to gather, optionally, information about the translation process.
Utilizing a number of benchmark programs, measurements are collected to determine the relative effort spent in the various sections of the translation process by this particular translator. The type of information so obtained contributes to a better understanding of how the translation of programming languages may influence machine language design. (author)


Topics: Validaticr, Evaluation

Keywords: flow analysis, debugging, optimization

This paper develops the manipulation of boolean connectivity matrices as a tool in the analysis of the structure of programs, shows its applicability to the debugging of programs, and suggests its use for optimization by automatic reorganization at the subroutine level.


Topics: Sequential programming level --> Process level

Keywords: review, sequencing analysis, algorithm


Topics: Computer organization

Keywords: microprogramming


Topics: Design methodologies

Keywords: top-down, bottom-up, SODAS, T.E.E., iterative, hierarchical, integrated simulation

This paper compares and contrasts the work done by Dijkstra, et al., on T.E.E., by Parnas and Darringer on SODAS, and by Zurcher and Randell on iterative multi-level modeling. Randell states that the most important problem for future work is the development of techniques for
presenting partial design so that the design decisions are clearly indicated and the consequences of a particular decision are more easily evaluated.


Topics: Reliability, Validation

Keywords: Performance measurement, hierarchical systems

This article provides an overview of achieving high performance and high reliability in computing systems and the relationship of this to operating system design. The author first notes that the problems stem mostly from the complexity of operating systems, that the two aspects are generally inversely related, and that it is typically impossible to achieve high absolute standards of performance and reliability. With respect to performance it is pointed out that most analysis is done after-the-fact in order to tune an already implemented system -- the proposal is made to integrate this at all levels of design and implementation and use design principles which enhance modularity and structure so as to facilitate any required changes. After dismissing the possibility of achieving totally reliable systems it is argued that while efforts to locate and remove errors in hardware or software modules before the modules are incorporated into the system are valuable, it is equally important to include in the system components which prepare for the occurrence of an error (protection mechanisms, redundant recording of information), detect errors by inspection of redundant information (such as bi-directional references, unique identifiers, etc.), and recover from errors to any extent possible.


Topics: Process level

Keywords: Event-driven sequencing

This paper argues that, in an environment with severely limited resources, the handling of input/output must be centralized and sequencing must be event-driven rather than time-driven.


Topics: Sequential programming level
Keywords: system programming languages

'The language BCEL (Basic CPL) was originally developed as a compiler writing tool and ... is closely related to CPL. ... In order to achieve the efficiency necessary for system programming its scale and complexity is far less than that of CPL. The most significant simplification is that BCPL has only one data type -- the binary bit pattern. ... We will first outline the general structure of BCPL and later discuss how well it is suited to applications in the fields of compiler writing and system programming.' (author)

Topics: Portability
Keywords: general

'Methods of achieving program portability are discussed, with particular reference to the mobility of compilers. The methods of transferring the BCPL compiler is then described including the specification of CCODE which is the language used as an interface between the machine independent and machine dependent parts of the compiler.' (author)

Topics: Description techniques
Keywords: programs, computation, directed graphs

Topics: Description techniques, Validation
Keywords: concurrent processes, message transfer expressions, algebraic, analysis, hierarchical descriptions, communication

The first part of this thesis develops a modeling scheme oriented around the viewpoint of a software system as a group of asynchronously operating processes which are synchronized by inter-process communication. The modeling scheme naturally allows descriptions of structure and behavior at several different levels of detail. Message transfer messages are then developed as an algebraic means for describing the behavior of such systems in terms of a trace of the inter-process communications. An effective procedure is developed for deriving message transfer expressions and it is shown how this may be used to analyze
the correctness of a system in terms of whether or not certain communication paths and behaviors exist in the system.


Topics: Descriptive techniques

Keywords: concurrent processes, communication, synchronization, levels of abstraction, organization, operation


Topics: Sequential programming level, Description techniques

Keywords: flowcharts, interactive and iterative composition


Topics: Descriptive techniques

Keywords: computation model, parallel computation, directed graph, properties of algorithms

This thesis presents a scheme for modeling parallel computation with directed graphs. The nodes represent computational elements and the arcs represent either single-item data transmission elements or control signal transmission elements. The modeling scheme is best suited for low-level, hardware oriented modules.


Topics: Descriptive techniques

Keywords: abstractions

The author points out the value of changing the language of description in terms of the abstractions which can result.
The author points out what we have as of the time of the article and what he feels we additionally need. We have microprogramming, emulation, interrupt mechanisms, dynamic storage allocation, hardware supervisory functions, and compilation. We need a total system design philosophy, large cheap fast peripheral storage, hardware supervisors for time-sharing, graphical terminals, hardware aids for software analysis, debugging and documentation aids, hardware compatibility, and hardware standardization.

This is a collection of papers presented at the first Courant Computer Science Symposium - each paper is entered separately in this bibliography.
Keywords: synchronization, semaphores, communication

Semaphores are extended to allow the passage of messages and it is indicated that the clarity of the system increases and the probability of errors and bottlenecks decreases.


Topics: Portability
Keywords: program transferability, data exchange

The author proposes a hierarchy of data structures (cells, trucks, trains, strips, streams, items) and a data description language as a possible basis for a solution to the portability problem.


Topics: Evaluation
Keywords: monitoring

An array of measuring tools devised to aid in the implementation of a prototype computer utility is discussed. These tools include special hardware clocks and data channels, general purpose programmed probing and recording tools, and specialized measurement facilities.' (authors)


Topics: Management
Keywords: choice of languages


Topics: Sequential programming level
Keywords: decompilation

**Topics:** Validation

**Keywords:** debugging


**Topics:** Validation

**Keywords:** error detection, testing


**Topics:** Sequential programming level

**Keywords:** translator writing systems

While not exactly disagreeing with the view that translator writing systems are the product of the most serious attempt to automate software production, the author points out that 1) the traditional translator writing system does not provide adequate aid in the handling of error diagnostics, data structures and storage management, optimization, code generation, and output editing, and 2) almost no attention has been paid to the software engineering aspects of storage allocation, interfacing to the operating system, documentation and maintenance, and design of subroutine libraries.


**Topics:** Management

**Keywords:** personnel

The problems attendant to the development of large scale systems are held to be primarily "people problems": no one has a full understanding of the full set of facilities to be provided; person-to-person communication is notoriously bad; the variance in the capabilities of the personnel is hard to analyze; good programmers are inherently creative and creativity can slow down progress; non-programming managers cannot always make proper judgments; the actual status of the project at any point in time is hard to perceive; there are very few viable methodologies for carrying out the development and testing of large scale systems; and no one person possesses a knowledge of the details of the entire system.

Topics: Validation

Keywords: error classes, debugging tools, formal correctness proofs

This article surveys the types of errors found in software and some of the tools, both available and needed, for debugging.


Topics: Evaluation

Keywords: operating systems, simulation


Topics: Evaluation

Keywords: software monitors


Topics: Management

Keywords: documentation standards

Gives a list of typical entries which should appear in a documentation folder of a public routine.


Topics: Description techniques

Keywords: BNF, computer systems

In this article, BNF is applied to the description of computer hardware and memory structure.


Topics: Description techniques
Keywords: occurrence graphs


Topics: Description techniques
Keywords: graphical

This thesis develops a technique for the formal description of line drawings and shows its applicability to the analysis of such pictures.


Topics: Education, Design methodologies
Keywords: software lab, hierarchical systems, virtual machine, concurrent processes

'A novel multiprogramming case study which has proven useful in operating system education and research is presented. The operating system and its unusual hypothetical machine are specified. The paper then describes experience with the case study as a class project ... and as a research tool for testing new ideas in operating system primitives and design methodologies.' (authors)


Topics: Evaluation
Keywords: analytical


Topics: Description techniques, Sequential programming level
Keywords: flowcharts, automatic derivation of program

Topics: Sequential programming level
Keywords: systems programming languages


Topics: Validity
Keywords: assertions, program correctness


Topics: Complex systems
Keywords: general, hierarchical decomposition, abstraction, evolution, near decomposability, description


Topics: Sequential programming level
Keywords: automatic synthesis


Topics: Description techniques
Keywords: interactions, algebraic


Topics: Computer organization
Keywords: language-oriented machine design

A multiprocessor system is discussed which almost
directly executes a reasonably traditional programming language.


Topics: Sequential programming level, Validation

Keywords: structured programming, assertions, hierarchical systems, modeling, iterative

This report gives the design criteria and present status of the Program Elaboration And Refinement Language project. This language is designed to allow the interactive, top-down, iterative construction of structured programs with the computer checking syntax and logic, generating and interpreting code, and serving as general assistant. The logic validation techniques are essentially those of Floyd. The system is currently only partially designed (there are no facilities for cooperative design by teams of designers or for the expression of control constructs involving parallelism) and only partially implemented (data types may not be refined and assertions are compiled into runtime error checks) but the appendices give adequate justification for this important software engineering tool.


Topics: Process level

Keywords: communication


Topics: Design principles

Keywords: modularity, concurrent processes, kernal, activity

'A framework is described on which it should be possible to construct general purpose systems suitable for the 70's. ... The first section of this paper postulates the requirements of the 70's concluding with the need for a useful method of modularizing, which both recognizes that code is seldom perfect, and allows hitherto mandatory and untouchable components to be optional and alterable. The second section develops an approach to modular structure.
It takes as the three basic ingredients: Activities (or processes), tools (cde, processors, etc.) and the organization of applying the tools to the activities. The structure modularizes both what happens (into activities) and the code that is used (into logical and into physical units). By developing the role of the central organizing element, as a caretaker of management decisions taken by activities, a means of modularizing the management function is arrived at.' (author)


Topics:  Sequential programming level --> Process level

Keywords:  algorithm, sequencing analysis


Topics:  Design principles, Management

Keywords:  implementation tools, specification

'...this paper examines an imaginary -- but archtypical -- problem whose solution requires a very large program. With this program as background, the remainder of the paper explicates various technical and managerial tools employed in the implementation of such programs.' (author)


Topics:  Evaluation

Keywords:  performance criteria


Topics:  Sequential programming level --> Process level

Keywords:  algorithm, sequencing analysis


Topics:  Validation

Keywords:  debugging, small computer, environment simulation

Topics: Sequential programming level, Description techniques

Keywords: procedure level, computation model, two-dimensional programming, directed graph, hierarchical model

Sutherland's system permits the specification of programs in a directed graph representation very similar to flowcharts. The specification takes place at a graphics terminal and the user may interactively create, modify and execute his program.


Topics: Validation


Topics: Modularity


Topics: Sequential programming level, Validation

Keywords: debugging, interactive programming, interface modification, error correction

The author proposes a program composition facility in which the routine tasks are performed by the computer leaving the human user free to attend to the creative tasks of program design and debugging. What he considers as "routine tasks" is very wide in scope -- the Do-What-I-Mean feature incorporates extensive automatic error correction features mostly centered around spelling correction based on the context of the error. The task of modifying the program during debugging and program elaboration is greatly facilitated by a feature called advising in which the user may easily change the interface between two program modules without changing (or even knowing about) the internals of the modules.

Topics: Sequential programming level, Sequential programming level \(\rightarrow\) Process level

Keywords: concurrent processes, programming languages, sequencing analysis

Rather than propose control constructs for the expression of concurrent processing, this paper defines a class of programming languages in which the sequencing and concurrency characteristics may be determined by tracing during compilation.


Topics: Description techniques

Keywords: concurrent processes, state description, extensibility

'This dissertation investigates the problem of representing groups of loosely connected processes and develops a model for process representation useful for synthesizing complex patterns of process behavior. There are three parts to the dissertation. The first part isolates the concepts which form the basis for the process representation model by focusing on questions such as: What is a process; What is an event; Should one process be able to restrict the capabilities of another? The second part develops a model for process representation which captures the concepts and intuitions developed in the first part. The model presented is able to describe both the internal structure of individual processes and the "interface" structure between interacting processes. Much of the model's descriptive power derives from its use of the notion of process state as a vehicle for relating the internal and external aspects of process behavior. The third part demonstrates by example that the model for process representation is a useful one for synthesizing process behavior patterns. In it the model is used to define a variety of interesting process behavior patterns. The dissertation closes by suggesting how the model could be used as a semantic base for a very potent language extension facility.' (author)

Topics: General

Volume 1 of this book contains articles relating to the design, implementation and maintenance of software systems and Volume 2 basically covers systems for the manipulation of information, such as information retrieval systems. Many of the articles appearing in Volume 1 and some of the articles from Volume 2 are cited individually in this bibliography.


Topics: General

Keywords: review, definition

Tou demarcates the task of the software engineer as that of making a virgin general-purpose digital computer into something which satisfies the needs of the user community. Within this broad area, he distinguishes the computer architect whose responsibility it is to use firmware to build conveniently organized computer systems from the pieces of hardware provided by the computer engineer; and the software programmers who deal with the design and implementation of software support systems and information manipulation systems.


Topics: Management, Design methodologies

Keywords: design cycle, principles

The author proposes some principles and practices, oriented around detailed and explicit planning, pertaining to the design of large scale systems. The design cycle consists of several steps: external function specification, demarcation of functional blocks, design of the form and content of interlock communication, program module specification, and determination of program module logic. For implementation, the author proposes an incremental building process with frequent testing and he discusses the attendant problems of integration, regression, and design of tests.


Topics: Validation

Topics: Reliability
Keywords: techniques

This article briefly reviews the Electronic Switching System control program developed at Bell Telephone Laboratories. The characteristics and data structures are outlined and it is reported that the reliability techniques employed consist of on-line data consistency checks (used sparingly because of overhead), routine memory audits (essentially large scale consistency checks) and a hierarchy of emergency audits.


Topics: Complexity
Keywords: large-scale systems, hierarchical decomposition


Topics: Validation
Keywords: debugging, input recordability, input specificability, asynchronous reproducibility of output


Topics: Sequential programming level
Keywords: modularization

This article presents an algorithm for partitioning a program into some number of pieces (called pages) such that none of the pages exceeds segment size and the number of interpage (inter-segment) references is reduced. This algorithm operates solely on connectivity and size data describing the program, data readily available from a compiler or assembler with relatively modest modification. Interpage references can be a fetch or store of data or a transfer of control. For simplicity of presentation, only the latter are considered in this article. However, in the research project on which the article is based, both
constant and variable data were treated in like manner.'

\(\text{(author)}\)


\text{Topics: Sequential programming level}

\text{Keywords: translator writing systems}

\text{This article discusses a translator writing system based on Zarnke's COMASS system.}

\text{Wagn67} Fric G. Wagner. \text{On the structure of programming languages, or, six languages for Turing Machines. IEEE Conference Record of the 1967 Eight Annual Symp. on Switching and Automata Theory, October 1967, pp. 45-54.}

\text{Topics: Sequential programming level}

\text{Keywords: structured programming, directed graph}

\text{Six programming languages for Turing Machines are presented which differ in syntactic classification and structure. The structural variations are unconstrained, block structure, rested looping, transfers limited to contiguous instructions, and single transfer per program. It is shown that all of the languages are universal -- can compute any computable function -- and some analysis is given of the difficulty of compiling the languages.}

\text{Wait70} W. H. Waite. \text{The mobile programming system: STAGE2. Comm. ACM, 13, 7 (July 1970), 415-421.}

\text{Topics: Portability}

\text{Keywords: bootstrap, macros}

\text{'STAGE2 is the second level of a bootstrapping sequence which is easily implemented on any computer. It is a flexible, powerful macro processor designed specifically as a tool for constructing machine-independent software.' (author)}

\text{Wald72} David C. Walden. \text{A system for interprocess communication in a resource sharing computer network. Comm. ACM, 15, 4 (April 1972), 221-230.}

\text{Topics: Process level}

\text{Keywords: concurrent processes, resource sharing, communication, networks}

Topics: Evaluation
Keywords: hardware, analytic


Topics: Documentation


Topics: Portability
Keywords: general, program transferability


Topics: Sequential programming level
Keywords: debugging, composition, extensibility

'The ECL programming system has been designed to provide an environment conducive to effective programming. To this end, it contains a language with comprehensive data types, operators, control structures, and storage management facilities. It allows interactive program composition and debugging with smooth transition to efficient compiled code. Most important, it allows the programmer to tailor this environment to suit his needs.' (author)


Topics: Description techniques
Keywords: syntax, semantics, programs


Topics: Design principles, Design methodologies
In this article, the author argues that the apparent complexity of a system may be reduced by modularization but that this increases the total complexity of the system because of the introduction of additional interfaces (an argument which depends upon the size metrics of complexity which the author assumes). He also notes that the modularization facilitates the discovery of some errors by localizing their effect. The author then concludes that the hierarchical modularization process is limited by the ability to comprehend the total system since that determines the ability to uncover non-localized errors. A limitation is also reached when errors are allowed to remain in the system, since the user community will have a threshold of tolerance on the complications and decreased throughput resulting from the residual errors.

This article discusses a KDF-9 time-sharing and file manipulation system which was built using the idea of constructing the system from modules which interact through a uniform, pre-defined interface.

A literature survey. Comp. Surveys, 1, 3 (September 1969) 139-145.

Topics: Computer organization
Keywords: microprogramming, survey, bibliography

Topics: Sequential programming level
Keywords: concurrent processes, programming languages

Wirth argues that the constructs proposed by J. P. Anderson for the control of parallel processing reflect the organization of the computer and do not fit in with the algorithmic orientation of Algol-like languages -- he then proposes some which do. Wirth also distinguishes two types of concurrency -- that which is not required by the algorithm but is none-the-less lurking in its sequential program representation, and that which is required by the algorithm in the sense that different parts of the algorithm must be executed by different parts of a non-homogeneous set of processing and storage elements.


Topics: Sequential programming level
Keywords: assembler languages, structured programming

One of the first attempts to introduce structured programming concepts at the assembler programming language level.


Topics: Sequential programming level
Keywords: specification of concurrent processes, interrupts, semaphores


Topics: Sequential programming level
Keywords: structured programming

This article introduces a programming language which is based on Algol 60 but has greatly enhanced data structuring and control facilities. It was designed to facilitate the teaching of programming and the writing of large programs.


Topics: Sequential programming level
Keywords: structured programming, top-down elaboration, programs, data


Topics: Sequential programming level

Keywords: structured programming


Topics: Management

Keywords: systems implementation

Programming management is probably the most neglected of programming techniques: yet, without its proper application, much of the value of other, more ingenious and attractive techniques is lost. This paper draws on the management experience gained in a large contract programming service unit. ... The disciplines discussed include ... procedure standards, the system-program interface, documentation standards, progress control, ...' (author)


Topics: Evaluation, Adaptation

Keywords: analysis, monitoring, dynamic tuning


Topics: Sequential programming level

Keywords: systems programming languages


Topics: Sequential programming level

Keywords: go-to-less programming languages, BLISS, structured programming

Topics: Sequential programming level

Keywords: systems programming language, expression language, go-to-less language, concurrent execution


Topics: Design methodology, Design principles

Keywords: hierarchical, top-down, iterative, integrated simulation

A methodology is described, applicable to the design of both hardware and software, which permits several descriptions at various levels of elaboration to exist at one time. It is suggested that the benefits are better understanding of system structure and behavior, earlier validation, and facilitation of maintenance. The methodology allows for the integration of simulation within the design elaboration process.