NBSIR 87-3559

Annotated Bibliography on Reliable System Design

Wayne McCoy, Kathleen Roessing, and Mary Ruhl

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

May 1987

U.S. DEPARTMENT OF COMMERCE
EAU OF STANDARDS

NBSIR 87-3559

ANNOTATED BIBLIOGRAPHY ON RELIABLE SYSTEM DESIGN

Wayne McCoy, Kathleen Roessing, and Mary Ruhl

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

May 1987

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

ABSTRACT

The difficulty in assuring some level of fault-tolerance, reliability, safety, availability or survivability in large, complex distributed systems has long been recognized. Techniques are now emerging that try to address this issue in system design, including formal description, design tools, automatic implementation and system simulations. This bibliography contains brief summaries of 350 papers from various computer science and engineering journals, books, dissertations and technical reports in the years 1971-1987, on these and related topics.

KEY WORDS: Automatic implementation, Distributed computer system, Fault-tolerance, Formal description, Reliability, System design.

ACKNOWLEDGEMENTS

Special thanks to Ted Zwiesler, Rick Kuhn, and Ken Dymond for their assistance in compiling many articles included in this report.

INTRODUCTION

An accelerating trend toward large complex systems, consisting of diverse collections of processors and hardware resources, operating systems and applications software, and communications fabric, portends difficulty in assuring some level of fault-tolerance, reliability, safety, availability or survivability in particular such systems. Fortunately, this difficulty has been recognized and techniques are now emerging toward addressing it in the system design at the outset. Formal specification techniques, analytical and simulation studies, design tools and automatic implementation are all intense areas of research into the problem. This bibliography represents some of the work that has been done in these and other related areas over the seventeen year span 1971-1987, containing 350 entries from various computer science and engineering journals, books, dissertations, and technical reports. Complete coverage is not claimed, nor is the inclusion of every seminal paper on a particular topic. The intent was to give a broad range of thinking.

Some particular areas are not included. These are: fault-tolerant designs for digital hardware, system maintenance, software metrics and software maintenance, and validation testing. While these areas are important, their focus is either too narrow or not relevant to total system design issues. Another area not specifically included, but for which a small number of entries do appear, is security. These entries have a bearing on issues in reliability and availability and are not as narrow in view as encryption techniques, for example. Several papers on multiprocessor interconnection and control issues are included as well, since some of these issues are general enough to extend to systems with higher distributivity.

The intended users of this bibliography are system designers, system design researchers and design tool builders and users.

The following is a list of the references and their associated key words. It will aid in searching for an article on a specific topic. This method was chosen because the sets of keywords have insufficient overlap for more topical cross referencing. Also, the key words indicate the topic covered more readily than do the titles of the papers. The papers are ordered alphabetically by the author's name.

CROSS REFERENCE

[ABAD86]
Design for testability
Testing
Test scheme

[ADEL85]
Artificial intelligence
Cognitive models
Cognitive science
Software design

[AGGA85]
Network vulnerability
Network articulation level

[ALLC80]
Concurrent languages
Modula
Multiprogramming
Real-time systems
Time dependencies

[ANDE85a]
Real-time systems
Software fault tolerance
Software reliability

[ANDE85b]
Data communication protocols
Attribute grammars
Real-time constraints
Concurrent activities

[ANDR79]
Concurrent systems
Message switching
Modula
Structured multiprogramming

[AVEN85]
Multistate system
Multistate component
s-coherent system
Reliability bound

[AVIZ80]
Computer reliability
Fault tolerance
Graceful degradation
Reliability estimation
Reliability modeling
Transient fault analysis

[AVIZ85]
Design diversity
Fault tolerance
N-version software
Software reliability
Tolerance of design faults

[BABA85]
Byzantine agreement
Distributed computing
Ethernet
Fault-tolerance

[BABI79]
Concurrent program
Correctness
Deadlock
Mutual exclusion
Finite delay and termination
Parallel program
Variant function
Verification

[BAKE80]
Program complexity
Program control flow
Source program transformations
Structured programs

[BALL86]
Network reliability
Computational complexity

[BALZ85]
Automatic programming
Knowledge base
Maintenance
Prototyping
Specification
Transformation

[BANE86]
Checks
Errors
Fault detection
Fault location

[BARS85]
Automatic programming
Programming knowledge
Program transformations

[BASI80]
Axiomatic correctness
Functional correctness
Program deviation
Structured programming

[BASI86]
Controlled experiment
Data collection and analysis
Experimental design
Software metrics
Software technology measurement and evaluation

[BAST85]
Computational correctness possibility
Control flow correctness possibility
Evaluation of design decisions
Program correctness possibility

[BASU80a]
Accumulating programs
Linear data domain
Program verification

[BASU80b]
Functional specifications
Iterative programs
Linear data domain
Program development
Total correctness

[BEAU78]
Computer performance
Computer reliability
Graceful degradation

[BELK86]
Abstract data types
Language translation
Prototyping
Specifications
Specification testing
Transformation rules

[BEND84]
Classification
Queuing system
Reliability models

[BERN85]
Broadcast network
Markov analysis
Redundancy
Reliability
Stable storage

[BIDO85]
Abstract data types
Algebraic specification
Decomposition schemes
Error handling

[BIRM85]
Abstract data types
Availability
Concurrency
Consistency
Distributed systems
Fault tolerance
Recovery
Reliability

[BLIK81]
Assertion-specified programs
Bubblesort procedures
Program correctness
PROMET-1

[BLOO79]
Synchronization
Concurrency
Modularity
Data abstractions
Programming methodology

[BLOO86]
Specification
Analysis
Protection system

[BLUM86]
Authentication
Automated development tools
Communication protocols
Formal description technique
Protocol implementation
Protocol specification
Protocol verification
State transition

[BOCH83]
Communication processes
Design verification
Distributed system design
Parallel processing
Specification consistency
Specification methods

[BOES86]
Network reliability
Connectivity
Vulnerability

[BOOT80]
Abstract data types
Computation structures
Performance analysis
Software design

[BREM81]
Networks
Reliable communication
Graceful degradation protocols

[BYRN85]
Dependency model
System effectiveness
System degradation
Measure of effectiveness
Probability of system failure

[CARC86]
Computer communication standards
Formal specification
LAN
Multicast communication servicer
Protocol verification
Rapid prototyping
Temporal ordering

[CEVA85]
DOD applications
SDI
High confidence software
Software reliability measurement methodology

[CHAN79]
Concurrent processes
Distributed systems
Program proving

[CHAN81]
Distributed systems
Message-passing systems
Communicating sequential processes
Deadlock
Recovery
Parallel algorithms

[CHAT78]
Availability
Computer-aided algebra
Symbol manipulation programs
Markov and semi-Markov processes

[CHEA79]
Automatic programming analysis
First-order recurrence relations
Program optimization
Program verification

[CHEN80]
Branch and bound methods
Distributed database
System design

[CHEN81]
Communication protocols network

[CHER85]
Banyan
Fail-softness
Multistage interconnection
Reliability prediction

[CHER86]
Redundancy
Reliability optimization
Dynamic programming
Knapsack problem

[CHEU80]
Self-metric software
Software reliability
Software reliability model

[CHI85]
Algebraic specifications
Formal specifications
Software design
Specification implementation

[CHOW79]
Job routing
Load balancing
Multiple processor system
Performance analysis

[CHWA81]
Diagnosis algorithms
Fault diagnosis
System diagnosis
T-diagnosable system

[CLAR86]
Network reliability
All-terminal reliability
Series-parallel network

[COLE81]
Attributed translations
Multiprocessing
Pipeline programs
Programming methodology

[COOK79]
Distributed programming
Modula
Processor module
Data abstraction

[COST78]
Availability
Hardware and software modeling
Hardware and software redundancy
Reliability

[COUR85]
Estelle
Protocols

[CRAL85]
Modeling technique of protocol service interfaces
Prolog
ISO/OSI transport layer
NBS Class 2 and Class 4 Transport layer
DoD TCP

[CREM78]
Information structuring
Redundancy
Sequential and concurrent processing

[CRIS85]
Availability
Correctness
Fault-tolerance
Programming logic
Reliability
Stochastic modeling

[CROW84]
Software reliability
Software failure
Clustering
Cyclic trend
Fourier series
Spectral analysis

[CURR86]
Incremental development
Software reliability
certification
Statistical quality control
Statistical testing process

[DAHB86]
Diagnosis
Fault tolerance
Multiprocessor systems

[DALE86]
System reliability
Optimization
Resource allocation
Effort function

[DANN82]
Control constructs
Program proving
Program verification

[DAVI78]
Asynchronous networks
Fault-tolerant computers
NMR
TMR

[DECH86]
Access scheme
Broadcast
Complexity analysis
Distributed algorithm
Parallel algorithms

[DENN77]
Protection
Security
Program certification
Confinement

[DERS81]
Invariant assertions
Program annotation
Program correctness
Verification

[DES078]
Fault-tolerant computing
Modular redundancy
Reliability
Responsive structure

[DES086]
Availability distribution
Dependable computer systems
Reliability
Repairable computer systems

[DEVI77]
Multiprogramming
Time sharing
Resource allocation
Deadlock
Interlock
Deadlock avoidance

[DIAS81]
Crossbar switches
Delta networks
Multi-stage interconnection networks

[DIJK74]
Multiprocessing
Networks
Synchronization
Self stabilization
Robustness
Error recovery
Self-repair

[DOWN85a]
Computer performance modeling
Reliability growth
Software reliability
Software testing
Stochastic models

[DOWN85b]
Software reliability
Fault-tolerance
Concurrent systems

[DOWN85c]
Software reliability
Mills error-seeding model
Jelinski-Moranda model

[DOWN86]
Probability models
Reliability growth
Software reliability
Software testing

[DUNH86]
Life-critical software
Real-time software

Software modeling and measurement
Software reliability

[DUPU85]
Mutual exclusion
Distributed algorithms
Synchronization
Time stamps

[ECKH85]
Coincident errors
Fault-tolerant software
Multiversion software
Reliability of redundant software

[EKAN79]
Access control
Keys
Locks
Protection

[ELLA81]
Invariant assertions
Loop predicates
Program validation

[ENGE86]
Mean time between failures
Nonhomogeneous Poisson process
Power-intensity process
Log-linear process

[ESTR86]
Concurrent systems
Hierarchical design
Interactive simulation
Performance models
Queueing models
Reachability analysis

[FAGA86]
Defect detection
Quality assurance
Software development
Software quality

[FARO81]
Networks
Communication protocols
Theory of colloquies
Automata
Petri nets

[FARO83]
Specification Description Language
Calculus of Communicating Systems
Computer communications

[FERN85]
Verification
Specifications
Communicating systems
Semi-automatic tools
Temporal logic

[FICK85]
Knowledge-based software development
Program transformation systems

[FINK80]
Computer networks
Distributed computing
Message routing
Multiprocessor architectures

[FLON77]
Parallel programs
Nondeterminism
Verification
Mutual exclusion

[FLYN80]
Directly executed Languages
Distributed systems
Parallelism

[FOST80]
Program correctness
Program testing
Software errors
Software reliability

[FRAN81]
Banyan networks
Crossbar networks
Space-time product
VLSI

[FREE83]
Expert systems
Executable specifications
Logic programming systems

[FUJI78]
Fault diagnosis
Polynomial time algorithm
Polynomially complete
Self-diagnosable systems
Turing machines

[FUKU78]
Clustering
Density estimation
Pattern recognition
Problem reduction or localization

[GARC82]
Crash recovery
Distributed computing systems
Failures
Mutual exclusion
Reorganization

[GARD80]
Distributed control
Deadlock
Locking
Recovery
Replicated databases

[GARM81]
Software life cycle reliability
Software development
Software maintenance

[GEHA85]
Verification
Specification
Concurrent

[GILB72]
Concurrent programming
Cooperating processes
Mutual exclusion
Parallel processes

[GLAS79]
Software reliability
Military/space applications

[GLAS81]
Complexity
Persistent software errors
Software problem report
Testing rigor

[GLIG79a]
Access privilege
Management policies
Capabilities
Shared objects
Selective revocation

[GLIG79b]
Authentication
Capabilities
Encryption
Hierarchical systems
Object migration
Redundancy

[GLIG80]
Deadlock detection
Distributed systems
False deadlocks
Ostensibly blocked transactions

[GOEL80]
Software reliability models

[GOEL85]
Failure count models
Fault seeding
Model fitting
NHPP

Software reliability
Times between failures

[GOLD86]
Knowledge-based software development
Program optimization
Program synthesis
Program transformation

[GOODM81]
Communication networks
Hypercube
Message traffic
Routing algorithms
Tree structures

[GOODW81]
Computer oriented language
Data abstraction
Dynamic defining of types
LISP
Programming environments

[GOPA81]
Broadcast routing
Packet switching networks
Queueing analysis
Source based forwarding

[GOUD81a]
Protocol correctness
Synchronization
Deadlock free

[GOUD81b]
Deadlock free
Communication protocols
Communicating processes
State deadlocks

[GOUD85]
Synchronous protocol
Communicating finite state machines
Reachability graph

[GREI77]
Formal specifications
Program correctness
Parallel processing
Synchronization

[GRIE77]
Garbage collection
Multiprocessing
Program correctness for multiprocessing tasks

[GRIE81]
Verification
Sequential

[GRIF85]
Atomic actions
Concurrency control
Markov processes
Queueing models
Reliability
Transaction systems

[GUTT78]
Abstract data types
Software validation
Algebraic axioms

[GUTT80]
Abstract data type
Correctness proof
Specification
Software specification

[HAAS81]
Deadline scheduling
Guarded commands
Parallel processes
Real-time programming

[HAC85]
Workload modeling
System failure
System operating mode
Measurement data

[HAIL85]
Protocols for error-prone channels
Finite state machine
Abstract program

[HALS78]
Message passing
Distributed computing
Actor semantics
Networks

[HANS78a]
Concurrent programming
Distributed processes
Process communication and scheduling
Microprocessor networks

[HANS78b]
Concurrent programs
Mutual exclusion
Program specification
Program implementation
Program verification
Guarded regions

[HASS86]
Binary tree
Fault tolerant
Reconfiguration
Redundant
Reliability

[HAYE85]
CICS
Formal specification
Large scale software

[HAYE86]
Abstract data types
Data type invariant
Software reliability
Specification language

[HECH86]
Computer failure models
Computer system reliability
Software management
Software reliability

[HEND86]
Functional programming
Software design
Specification
Validation

[HENI80]
Documentation techniques
Functional specifications
Real-time software
Requirements
Specifications

[HENR81]
Design methodologies
Software metrics
UNIX

[HEWI77]
Parallel processes
Computational analysis
of systems

[HOAR71]
Natural deduction
Axiomatic method
Program correctness

[HOAR85]
Verification
Concurrent

[HOLL87]
Markov models
Multiprocessors
Performance comparison and evaluation
Petri nets

[HOLO80]
Synchronization primitives
Synchronization problems

Synchronization parameters
and variations

[HOLT80]
Programming languages
Parallel programming
Abstract machines

[HOLZ81]
Message passing protocols
Algebraic method
Finite state machines

[HORO81]
Binary trees
Parallelism
VLSI
Multiprocessing
Networks

[HOWD80]
Effectiveness
Reliability
Testing

[HOWD81]
Errors
Design properties
Functional program tests

[HUAN80]
Consistency
Decomposition
Invariant-relation theorem
Program verification

[HUNG85]
Concurrent processing
Correctness proof
APK
Communication tree
Invariants generation

[HUTC85]
Cluster
Coupling
Data binding
Measurement
System structure

[IBAR81]
Complexity
Network diagnosis
NP-complete

[IBAR82]
Algorithm complexity
Deadlock prevention

[IEEE85]
Metrics

[IGAR75]
Specifications
Verification conditions

[IYER85]
Failure analysis
Software reliability
System workload
VM/SP

[IYER86]
Analysis
Markov chains
Moments
Reliability

[JACO86]
User interface
Specification techniques

[JAHA86]
Real-time logic
Safety analysis systems
specification
Time-critical system
Verification

[JALO86]
Atomic actions
Backward recovery
Forward recovery
Software fault-tolerance

[JARD85]
Verification
Protocol specification
Formal description technique
Extended state transition
machines

[JEWE85]
Bayesian Analysis
Program testing
Software reliability

[JOHN86]
Safety
Reconfigurable duplication
Standby sparing

[KANT81]
Automatic programming
Program synthesis
Refinement
Stepwise refinement

[KANT85]
Automatic programming
Automating algorithm design
Human problem solving
Protocol analysis

[KART79]
Dynamic architecture
Dynamic computer group
Multicomputer

[KARU79]
Digital systems
Optimal systems design
Self-diagnosis
System-level diagnosis
Single-loop systems

[KELL76]
Parallel program
Correctness
Verification
Deadlock
Mutual exclusion
Petri net

[KELL85]
Relational databases
Incomplete information
Updates

[KEMM85]
Design and development
Formal verification
Reliable software
Requirements
Specifications

[KENE86]
Failure rate
Sequential detection scheme
False detection rate

[KESS77]
Monitor
Mutual exclusion
Synchronization
Conditional critical region

[KESS81]
Concurrency
Message passing
Synchronization

[KIM84]
Fault tolerance
Reliability
Database concurrency control
and recovery
Relational database

[KING80]
Correctness assertions
Predicate transformers
Program correctness
Relational semantics
Subgoal induction

[KLIG86]
Exception handling
Guaranteed response time
Real-time systems
Software reliability

[KLUG82]
Channel processors
Memory access conflicts
Petri nets
Synchronic distance

[KNIG86]
Design diversity
Fault-tolerant software
Multiversion programming
Software reliability

[KOHL81]
Decentralized system
Access synchronization
Concurrency control
Crash recovery
Atomic action

[KORE79]
Intermittent fault
Modular redundancy
Permanent fault
Reliability

[KORN79]
Parallel processing
Distributed computation
Pattern-directed invocation
Problem solving

[KRAM78]
Distributed processing systems
Finite state machines
Resource sharing

[KRAM85]
Configuration process
Configuration specification
Distributed systems
Flexibility
Reusability
System evolution

[KROL86]
Consistency
Error-correcting codes
Fault tolerance
Hardware redundancy

[KUMA80]
Hierarchical modeling
Performance evaluation
System design
Optimization algorithms

[KUMA86]
Distributed program
Distributed system
Reliability
Spanning tree

[LADN79]
Parallelism
Lockout
Finite state processes
Deadlock
Critical section

[LAMP76]
Distributed systems
Computer networks
Clock synchronization
Multiprocess systems

[LAMP77]
Asynchronous multiprocessing
Multiprocess synchronization
Shared data

[LEE82]
Fault-tolerant
Rollback recovery
Cooperating processes
Reliability

[LEMO80]
Standardization
Formalization methods

[LESS80]
Dynamic architecture
Powerful parallel system
Reconfigurable hardware resource
Reconfigurable memory processor

[LEUN80]
Concurrent processes
Invariant assertion
Control modules
Program specification

[LICH86]
Consistency checking
Program Design Language
Software development
Software quality assurance

[LIES86]
Real-time systems
Scheduling
Software fault tolerance
Software reliability

[LIN83]
Communication protocols
Completeness
Synchronization

[LIN86]
Parallel processing
Reconfigurable multiprocessors
Interconnection networks
Circuit switching
Connection conflicts

[LING79]
Verification
Sequential

[LIPT73]
Synchronization
Primitive process concept
PV
Interprocess communication

[LIPT75]
Deadlock free
Reduction
Interruptible parallel program
Verification method

[LISK81]
Integrated programming language
Distributed programs
Robust programs

[LITT80]
Program error
Reliability growth
Software failure
Software life-cycle cost
Software reliability measurement

[LOCK85]
System reliability
Inclusion-exclusion
Topological reliability
m-out-of-n system
Source-to-multiple terminal reliability

[LU78]
Character recognition
Error transformation
Pattern recognition
Syntactic pattern recognition

[LUCE79]
Correctness of data representation
Program derivation
Program schema
Program specification
Program synthesis

[MA82]
Branch and bound
Distributed processing
Interprocess communication
Task allocation

[MALL78]
Incomplete diagnosis
Incorrect diagnosis
Intermittent faults
Self-diagnosable system
Syndrome

[MANC86]
Communicating sequential processes
Fault tolerance
Guarded commands
Nondeterminism
Replicated processing

[MAO80]
Communicating sequential processes
Communication ports
Concurrent programming
Distributed networks
Nondeterminism

[MART86]
Distributed processing system
Graceful degradation
Operational survivability

[MARX81]
Reliable software
Specification language

[MCKE85]
Abstract data types
Concurrency control
Distributed systems
Synchronization

[MCMI82]
Large scale parallel distributed processing systems
Redundancy
Fault tolerant

[MEDI81]
Ada environments
Incremental compilation
Interactive debugging
Syntax-directed editing

[MEKL80]
Abstract process
AP-net
Petri net
Process expression
Software design representation

[MENA79]
Data bases
Deadlock detection
Distributed data bases
Graph theory

[MERK78]
Security
Cryptography
Cryptology
Computer network security
Passive eavesdropping

[MEYE78]
Diagnosis algorithm
Fault syndromes
Modular networks

[MEYE80a]
Degradable computing systems
Fault-tolerant computing
Hierarchical modeling
Performability evaluation
Performance evaluation
Reliability evaluation

[MEYE80b]
Fault-tolerant computing
Performance evaluation
Reliability evaluation

[MEYE81]
Connection assignment
Diagnosis algorithm
Modular architecture
Permanent fault

[MILI85]
Error recovery
Exception handling
Forward error recovery
Program fault-tolerance

[MILL86]
Complete monotonicity
Nonhomogeneous Poisson processes
Probability models
Software reliability

[MULA85]
Real-time systems
Reliability
Safety
Fault tolerant

[MURA80]
Deadlock-freeness
Decision-free concurrent systems
Modular synthesis
Parallel computation model

[MUSS80]
Abstract data types
Algebraic specifications
Equational theories
Program verification

[NARA86]
Connection assignment
Diagnosable systems
Fault diagnosis
Self-diagnosis

[NASA85]
Metrics

[NATA85]
Atomic action
Communication failure
Computing agent
Distributed operating system
Distributed system

[NECH85]
Expert systems
Natural language generation
Software development
Software maintenance

[NEGR84]
Fault tolerance
Distributed processing
System level diagnosis
Error-confinement
Non-hierarchic system

[NEUM86]
Abstraction
Critical requirements
Hierarchical design
Reliability
Safety
Security
Trusted subsystems

[OKUM85]
Additional software test time
Logarithmic Poisson model
Software reliability model
Software quality control

[OSSF80]
Concurrent diagnosis
Diagnostic programs
Fault-tolerant computing
Maintainability
Intermittent fault
Self-checking

[OWIC79]
Structured multiprogramming
Correctness proofs
Program verification
Concurrent processes
Mutual exclusion
Deadlock

[OWIC76]
Parallel programming

[PAPA81]
Attribute grammar evaluator
Attribute grammars
Formal specifications
Semantics

[PARE85]
Algebraic operators
Completeness
Data manipulation language
Entity-relationship model

[PARN79]
Contractibility
Extensibility
Modularity
Software engineering

[PARN85]
Abstract interfaces
Information hiding
Modular structure of software

[PEAC85]
Bradford-Zipf distribution
Clustering
Program behavior
Program restructuring

[PERL81]
Metrics

[PERL83]
Network
ARPANET
Routing broadcast scheme
Self-stabilization

[PERR86]
Byzantine agreement
Distributed computing
Early stopping
Fault tolerance

[PETE79]
Communicating asynchronous processes
Synchronization

[PIAT80]
Distributed data processing
Specification
Validation
Data communications
Computer networking

[POLA79]
Assertion language
Inductive assertions
Permutation
Theorem proving

[PRAD80]
Decoder logic
Error correction
and detection
Multiple errors
Multiple faults
Self-checking
Transient faults
Unidirectional errors

[PROV86]
Network reliability

[RAGH86]
BPC Permutations
Fault-tolerant routing
Multiprocessor systems

[RAMA80]
Asynchronous
Concurrent
Petri net
Real-time

[RAMA81]
DCDS
Assertion
Dual-programming
Path analysis
Process control

[RAMA85]
Software complexity
Metrics
Specification language
Waterfall development model

[RAMA86]
Information abstraction
Knowledge-based systems
Metrics
Reusability
Software life cycle

[RAND78]
Fault tolerance
Fault avoidance
Hardware reliability
Software reliability
System structure

[RAO79]
Computer networks
Cutsets
Distributed computers
Load balancing
NP-complete problems

[RAPP85]
Data flow
Program testing
Test data selection

[RAU79]
Analytical models
Interleaved memories
Memory bandwidth
Memory interference
Multiprocessors
Performance evaluation

[REDD78]
APL implementation
Architectural design
Parallel computation
Parallel languages
Reconstructible computers

[REED78]
Distributed computer systems
Reliability
Synchronization

[RICA80]
Mutual exclusion
Message passing
Deadlock
Starvation

[RICH85]
Software testing
Software verification
Symbolic evaluation

[ROBI77]
Hierarchical structure
Program verification
Formal specification
Abstraction

[ROSE85]
Reliability
Recovery

[ROSEN81]
Network disturbances
ARPANET
Protocols

[ROSS85a]
Estimation
Reliability
Poisson process

[ROSS85b]
Failure rate
Software reliability
Stopping times

[RUBI82]
Protocol validation
Communication systems
Interactions

[RUDI81]
Reception errors

Static deadlocks
Dynamic deadlocks

[RUSS80]
Domino effect
Error recovery
Parallel back tracking
Process communication
State restoration

[RYPK79]
Allocation modes
Deadlock avoidance
Deadlock detection
Logical resource
Resource allocation
Resource sharing

[SAXE86]
Built-in test
Fault coverage
Testing

[SCHA78]
Computer architecture
Data-driven processing
Microprogramming
Multiprogramming
Radar signal processing
Real-time processing

[SCHL85]
Fail-stop processors
Fault-tolerant computing
Markov chains
Performance evaluations

[SCHO86]
Conditional inference
Exponential order statistics
Identifiability
Multinomial trials
Order restricted maximum likelihood estimates

[SCHU79]
Concurrent programming
ILIAD
Multiprogramming
Multiprocessing
Real-time languages
Real-time programming

[SEAQ80]
Synchronization mechanism
Concurrency
Resource guardians
Event sequences

[SEDM80]
Availability
Fault detection
Fault recovery
Fault tolerance
Maintainability
Reliability
Self-checking

[SHAN82]
Abstract model specification
Data abstraction
Procedure abstraction
State machine
Verification

[SHEN85]
Defect density
Error-prone modules
Probability of errors
Software errors
Software metrics

[SHIN86]
Detection time
Fault and error latency
Fault injection
Maximum likelihood estimator

[SHIN87]
Clock synchronization
Fault-tolerant real-time multiprocessors
Malicious faults

[SHOO84]
Software reliability
History

[SIDH81]
Communication protocols
Specification
Verification
Specification techniques
Protocol design

[SIDH82a]
Protocol synthesis
Design rules
Communication protocols
Completeness
Deadlock-freeness
Temporal logic

[SIDH82b]
Communication protocols
Protocol properties
Design rules
Protocol synthesis

[SIDH83]
Communication protocols
Executable specification
Verification
Horn clause logic
PROLOG

[SIDH86]
Transport protocol
Formal description techniques
Automated development tools

[SILB79]
Distributed systems
Guarded commands
Input/output commands
Synchronization

[SING85]
Dynamic linear and
nonlinear models
Kalman Filtering
Likelihood ratios
Predictable distributions
Prequential analysis
Reliability growth
Software reliability

[SING86]
Flow-network
Delta-star transformation
Maximum flow

[SMIT85]
Automatic programming
Knowledge-based system
Program synthesis

[SMOL81]
DCDS
Distributed processing
Fault tolerance
Real-time systems
Reliability

[SMOT86]
Fault coverage
Fault-tolerant computers
Reliability bounds
Sensitivity analysis

[SNYD81]
Capabilities
Grammatical protection systems
Security
Take/Grant models

[SOI81]
Computer communication network
Network topology
Reliability indices

[SPEE79]
Interleaved memories
Interaction model
Performance analysis
Resource utilization

[SPIT75]
Verification
Automatic programming
Correctness

[SPIT78]
Program verification
Specification
Data abstraction
Hierarchical structures

[SPRE85]
Software reliability
Parameter estimation
s-confidence interval
Jelinski-Moranda model

[STAN85]
Bidding
Distributed computing
Stability
Stochastic learning automata

[STEI85]
Algorithm design
Automatic programming
Developmental evaluation
Meta-evaluation

[STEM86]
Capabilities
Functional addressing
Interprocess communication
Port

[STRO86]
Program analysis
Program verification
Security
Software reliability
Type checking

[SUMI86a]
Integrated hardware-software
reliability model
Multiple error generation
and removal
State-dependent general lifetimes
and repair times
Time-dependent compound
performance measures

[SUMI86b]
Software reliability
Multiple-error introduction
Markov models
Performance evaluation

[SUZU86]
Verification
Concurrent
VLSI

[TAI80]
Program testing

Testing complexity
Test criteria

[TAMI80]
Invariants
Partial correctness
Program verification
QLISP
Synthesis of invariants

[TANA78]
Context-sensitive parser
Error-correcting parser
Formal languages
Maximum-likelihood parser

[TAYL80a]
Error correction
Error detection
Software fault tolerance
Software reliability

[TAYL80b]
Compound data structures
Error correction and detection
Robust data structures
Software fault tolerance
and reliability

[TAYL80c]
Concurrent software
Error detection
HAL/S
Process synchronization errors

[TAYL86a]
Atomic actions
Backward recovery
Exception handling
Forward recovery
Software reliability

[TAYL86b]
Crash recovery
Software fault tolerance
Global and local error correction

[THAY82]
Algorithmic state machine
Microprogrammed structures
Transformation of programs

[TJAD76]
Concurrency
Hierarchy of tasks

[TRAC85]
Software reliability model
Software testing
Jelinski-Moranda model
Shooman model
Musa model

[TROY85]
Model comparisons
Software reliability

[TRST84]
Telecommunication network
Network topology
System effectiveness

[TSUB86]
Ada
High level language architecture
Software reliability

[TYRR86]
Communicating sequential
processes
Concurrent processes
Distributed systems
Fault-tolerant software
Occam
Petri-nets

[UPAD86]
Error detection
Error latency
Recovery time
Rollback recovery
Transient errors

[VANC86]
Verification
Concurrent

[VANE79]
Control structure
Correctness-oriented programming
Invariant assertions
Verifications

[VENK85]
Estelle
Specification techniques
Network systems
Petri nets
Temporal logic

[VISS85]
Protocol designers
Architecture
Specification
Verification
Service concepts

[VOGE80]
Automated test systems
Dynamic analysis
Program testing
Static analysis

[VOSS80]
Concurrency
Deadlock-free

Decentralized control
Distributed databases
Petri nets
Predicate/transition nets

[WALT81]
Decomposition for diagnosability
Design for diagnosability
Fault detection

[WASS85]
Executable specifications
Interactive information systems
Rapid prototyping
Software development methodology

[WASS86]
Interactive information systems
Software development methodology
Software reliability

[WEBE86]
Data abstraction
Informal and procedural and algebraic specifications
Software development
Software structuring

[WEIS80]
Formal specification languages
Structured programming methodology

[WEST86]
Protocol validation
State exploration
Protocol specification
Session layer
Specification
Random exploration

[WEYU80]
Program testing
Software error detection
Software reliability
Theory of testing

[WHIT80]
Control structure
Domain errors
Software reliability
Software testing

[WITT81]
Bus topologies
Cube-connected cycles
Dual-bus hypercubes
Hypercube spanning buses

[WOOD80]
Allegations
Infeasible paths
Path testing
Test metrics

[WU81]
Interconnection network
Topological optimization

[WUU85]
Distributed system
False deadlock
Transaction wait-for-graph
Two-phase locking

[YAMA85]
Error detection rate
Maximum-likelihood estimation
Nonhomogeneous Poisson processes
Software reliability analysis

[YANN86]
Distributed recovery
Fault tolerance
Fault-tolerant multiprocessor systems
Reconfiguration

[YAO79]
Distributive computing
Complexity
Probabilistic models

[YAU80]
Capabilities
Concurrency
Control errors
Control flow checking
Program design

[YAU81]
Codd relations
Graph grammar
Hierarchical graph
Recursive graph

[YAU85]
Design stability measures
Program modifications
Software maintenance

[YAU86]
Design methodologies
Design representation
Design verification and validation
Distributed software system design
Error-resistant software design
Software metrics

[YONE77]
Multiprocessor information processing systems
Distributed systems
Artificial intelligence
Modelling systems

[YUAS85]
Data abstraction
Formal specification
Program verification

[ZAFI80]
Reliable communications
Error-free protocols
Error detection and correction

[ZAVE76]
Shared variables
Concurrent processes
Formalization
Synchronization

[ZAVE86]
Distributed systems
Executable specifications
Functional programming
Operational approach to
software development
Parallelism
Real-time systems

BIBLIOGRAPHY

[ABAD86] Abadir, Brever, "Test Schedules for VLSI Circuits Having Built-In Test Hardware," IEEE Transactions on Computers, pp. 361-367, April 1986.

Introduces a test scheme which describes how a test methodology is to execute. A theory of test plan execution overlap is presented, and is used as the basis for constructing test schedules with optimal execution times.

KEY WORDS: Design for testability, Testing, Test scheme.

[ADEL85] Adelson, Soloway, "The Role of Domain Experience in Software Design," IEEE Transactions on Software Engineering, pp. 1351-1360, November 1985.

Explains a designer's experience and skills in a domain and what happens to his skills and knowledge when the domain changes. Emphasis is placed on a expert designers creating a design context, how familiar they are with it and the result of their work.

KEY WORDS: Artificial intelligence, Cognitive models, Cognitive science, Software design.

[AGGA85] Aggarwal, "Reliability Indices for Topological Design of Computer Communication Networks," IEEE Transactions on Reliability, p. 523, December 1985.

Corrects the definition of network articulation level from [SOI81].

KEY WORDS: Network vulnerability, Network articulation level.

[ALLC80] Allchin, "Modula and a Question of Time," IEEE Transactions on Software Engineering, p. 390, July 1980.

Reviews the programming language Modula for its ability to support the wide variety of time usages required in a process control system. It is explained that in many instances concurrent languages supporting device control and multiprogramming may need additional facilities for the support of time.

KEY WORDS: Concurrent languages, Modula, Multiprogramming, Real-time systems, Time dependencies.

[ANDE85a] Anderson, Barrett, Halliwell, and Moulding, "Software Fault Tolerance: An Evaluation," IEEE Transactions on Software Engineering, pp. 1502-1510, December 1985.

Provides an overview of a project conducted at the University of Newcastle upon Tyne where techniques were developed for, and applied to a realistic implementation of a real time system.

Results of three phases of experimentation are presented. An analysis of these results shows that use of the software fault tolerance approach yielded a substantial improvement in the reliability of the real time system used.

KEY WORDS: Real-time systems, Software fault tolerance, Software reliability.

[ANDE85b] Anderson, Landweber, "A Grammar-based Methodology for Protocol Specification and Implementation," Proceedings of the Ninth Data Communications Symposium, pp. 63-70, IEEE Computer Society Press, Whistler Mountain, BC, Canada, September 10-13, 1985.

Presents a new methodology for specifying and implementing communication protocols, based on a formalism called 'Real-Time Asynchronous Grammars' protocols to be specified at a highly detailed level. As an example, an RTAG specification is given for part of the Class 4 ISO Transport Protocol. RTAG allows protocols to be specified at a highly detailed level, and thus, major parts of an implementation can be automatically generated from a specification.

KEY WORDS: Data communication protocols, Attribute grammars, Real-time constraints, Concurrent activities.

[ANDR79] Andrews, "The Design of a Message Switching System: An Application and Evaluation of Modula," IEEE Transactions on Software Engineering, pp. 138-147, March 1979.

Illustrates the use of Modula (programming language for implementing parallel systems) for the design of a message switching communication system. A message switching system poses a number of interesting problems: 1) a high degree of concurrent activity exists; 2) a variety of I/O devices need to be controlled; 3) messages can have multiple destinations; 4) messages can be preempted. The strengths and weaknesses of Modula with respect to these specific problems and its utility as a general purpose language are evaluated.

KEY WORDS: Concurrent systems, Message switching, Modula, Structured multiprogramming.

[AVEN85] Aven, "Reliability Evaluation of Multistate Systems with Multistate Components," IEEE Transactions on Reliability, pp. 473-479, December 1985.

Presents two efficient algorithms for reliability evaluation of monotone multistate systems with s-independent multistate components. Computer programs for implementing the algorithms are given. Computational-times are presented, and compared with the 'Inclusion-Exclusion Method' and the 'State Enumeration Method'. Results clearly demonstrate the superiority of the algorithms to the two other methods.

KEY WORDS: Multistate system, Multistate component, s-coherent system, Reliability bound.

[AVIZ80] Avizienis, Ng, "A Unified Reliability Model for Fault-Tolerant Computers," IEEE Transactions on Computers, pp. 1002-1011, November 1980.

Summarizes the results of an extended effort to develop a unified approach to the reliability modeling of fault-tolerant computers which strikes a good compromise between generality and practicality. The developed unified model encompasses repairable and non-repairable systems and models, transient as well as permanent faults and their recovery. Based on the unified model, a powerful and efficient reliability estimation program, ARIES, has been developed.

KEY WORDS: Computer reliability, Fault tolerance, Graceful degradation, Reliability estimation, Reliability modeling, Transient fault analysis.

[AVIZ85] Avizienis, "The N-Version Approach to Fault-Tolerant Software," IEEE Transactions on Software Engineering, pp. 1491-1501, December 1985.

Presents evolution of the N-version approach to the tolerance of design faults. Principal requirements for implementation, DEDIX distributed supervisor and testbed for execution, goals of current research, and potential benefits are all discussed.

KEY WORDS: Design diversity, Fault tolerance, N-version software, Software reliability, Tolerance of design faults.

[BABA85] Babaoglu, Drummond, "Streets of Byzantium: Network Architectures for Fast, Reliable Broadcasts," IEEE Transactions on Software Engineering, pp. 546-554, June 1985.

Considers the reliable broadcast problem in distributed systems with broadcast networks as the basic communication architecture. It is shown how properties of such network architectures can be used to effectively restrict the externally visible behavior of faulty processors. These techniques are used to derive simple protocols that implement reliable broadcast in only two rounds, independent of the failure upper bounds.

KEY WORDS: Byzantine agreement, Distributed computing, Ethernet, Fault-tolerance.

[BABI79] Babich, "Proving Total Correctness of Parallel Programs," IEEE Transactions on Software Engineering, pp. 558-574, November 1979.

Presents steps to proving parallel programs correct: 1) model the parallel program; 2) prove partial correctness; 3) prove absence of deadlock, livelock, and infinite loops. The main contributions are techniques for proving the absence of deadlock and livelock. It is shown how variant functions may be used to prove finite termination, and how finite termination can be used to prove absence of livelock. Handling finite delay is discussed. An example is included which occurred in a commercial environment and a classic synchronization problem is solved without the aid of special synchronization primitives.

KEY WORDS: Concurrent program, Correctness, Deadlock, Mutual exclusion, Finite delay and termination, Parallel program, Variant function, Verification.

[BAKE80] Baker, Zweben, "A Comparison of Measures of Control Flow Complexity," IEEE Transactions on Software Engineering, pp. 506-511, November 1980.

Focuses on three measures of control flow complexity: McCabe's cyclomatic complexity, Halstead's software effort, and Woodward's complexity measure based on the number of crossings of arcs in a linearization of the flowgraph. Their major properties as measures of control flow complexity are established and particular attention is directed at the behavior of these in structured programming environment. Weaknesses of these measures are exposed individually.

KEY WORDS: Program complexity, Program control flow, Source program transformations, Structured programs.

[BALL86] Ball, "Computational Complexity of Network Reliability Analysis: An Overview," IEEE Transactions on Reliability, pp. 230-239, August 1986.

Presents an overview of results related to the computational complexity of network reliability analysis problems. It is shown how these problems are related to the more familiar computational network problems of recognizing certain subnetworks, finding optimal subnetworks, and counting certain subnetworks. These relationships are used to show that the k-terminal, the 2-terminal, and the all-terminal network reliability analysis problems are at least as hard as the renowned set of computationally difficult problems, NP-Complete. The impact of these results on how one should approach problem solving in this area is discussed.

KEY WORDS: Network reliability, Computational complexity.

[BALZ85] Balzer, "A 15 Year Perspective of Automatic Programming," IEEE Transactions on Software Engineering, pp. 1257-1267, November 1985.

Discusses an approach to the problem of acquiring and determining high-level specification and translating high-level specifications to lower level for automatic compilation. The success of automatic programming is shown. Emphasis is placed on a paradigm for automated software and the Gist specification language.

KEY WORDS: Automatic programming, Knowledge base, Maintenance, Prototyping, Specification, Transformation.

[BANE86] Banerjee, Abraham, "Bounds on Algorithm-Based Fault Tolerance in Multiple Processor Systems," IEEE Transactions on Computers, p. 296, April 1986.

Presents a graph-theoretic model for determining upper and lower bounds on the number of checks needed for achieving concurrent fault detection and location. The objective to estimate the overhead in time and the number of processors required for such a scheme is indicated.

KEY WORDS: Checks, Errors, Fault detection, Fault location.

[BARS85] Barstow, "Domain-Specific Automatic Programming," IEEE Transactions on Software Engineering, pp. 1321-1336, November 1985.

Emphasizes the need for domain knowledge in an automatic programming system. Investigation for the ONIX project at Schlumberger-Doll Research has led to a framework for specifying domain knowledge in automatic programming which involves transformation in the description of programming showing the interaction between programming and domain knowledge.

KEY WORDS: Automatic programming, Programming knowledge, Program transformations.

[BASI80] Basili, Noonan, "A Comparison of the Axiomatic and Functional Models of Structured Programming," IEEE Transactions on Software Engineering, pp. 454-464, September 1980.

Discusses axiomatic and functional models of the semantics of structured programming. The models are presented together with their respective methodologies for proving program correctness and for deriving correct programs. Examples using these methodologies are given. Finally, the models are compared and contrasted.

KEY WORDS: Axiomatic correctness, Functional correctness, Program deviation, Structured programming.

[BASI86] Basili, Selby, and Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, pp. 733-743, July 1986.

Presents a framework for analyzing most of the experimental work performed in software engineering over the past several years. A variety of experiments in this framework and their contributions are described.

KEY WORDS: Controlled experiment, Data collection and analysis, Experimental design, Software metrics, Software technology measurement and evaluation.

[BAST85] Bastani, "On the Uncertainty in the Correctness of Computer Programs," IEEE Transactions on Software Engineering, pp. 857-864, September 1985.

Develops an approach to the correctness of a computer program by viewing it as a set of hierarchically structured fuzzy equivalence classes. This method is applicable during the design phase and permits program proving whenever possible. For each equivalence class, the computational correctness possibility and control flow correctness possibility is determined. This theory can be used to guide the design process.

KEY WORDS: Computational correctness possibility, Control flow correctness possibility, Evaluation of design decisions, Program correctness possibility.

[BASU80a] Basu, "A Note on Synthesis of Inductive Assertions," IEEE Transactions on Software Engineering, pp. 32-39, January 1980.

Investigates a class of programs (accumulating programs) for which inductive assertions can be mechanically generated from I/O specifications. Accumulating programs are iterative realizations of problems in which the required output information is accumulated during successive passes over the input data structures. Obtaining invariant assertions for such programs is shown to be equivalent to the problem of generalizations of specifications to that over an extended closed data domain.

KEY WORDS: Accumulating programs, Linear data domain, Program verification.

[BASU80b] Basu, "On Development of Iterative Programs from Function Specifications," IEEE Transactions on Software Engineering, pp. 170-182, March 1980.

Investigates a systematic approach to the development of totally correct iterative programs for the class of accumulation problems. The development of iterative programs for accumulation problems is shown to involve successive generalizations of the data domain and the corresponding function specifications. The problem of locating these generalizations is discussed. A linear data domain is defined in terms of decomposition and finiteness axioms, and the property of a loop body being well behaved over a linear data domain is introduced. An abstract program for an accumulation problem is developed using these considerations.

KEY WORDS: Functional specifications, Iterative programs, Linear data domain, Program development, Total correctness.

[BEAU78] Beaudry, "Performance-Related Reliability Measures for Computing Systems," IEEE Transactions on Computers, pp. 540-547, June 1978.

Discusses measures which reflect the interaction between the reliability and performance characteristics of computing systems. These measures can be used to evaluate traditional computer architectures, such as uniprocessors and standby redundant systems; gracefully degrading systems, such as multiprocessors, which can react to a detected failure by reconfiguring to a state with a decreased level of performance; and distributed systems. The analysis method, which provides quantitative information about the tradeoffs between reliability and performance, is demonstrated in several examples.

KEY WORDS: Computer performance, Computer reliability, Graceful degradation.

[BELK86] Belkhouche, Urban, "Direct Implementation of Abstract Data Types From Abstract Specifications," IEEE Transactions on Software Engineering, p. 649, May 1986.

Describes an alternative approach for the development of specifications. Approach relies on a specification language for abstract data types and a synthesis system. System is capable of translating abstract data type specifications into an executable program. This provides the necessary tools for the early testing of the specifications and for the development of prototypes and implementation models.

KEY WORDS: Abstract data types, Language translation, Prototyping, Specifications, Specification testing, Transformation rules.

[BEND84] Bendell, "A Classification System for Reliability Models," IEEE Transactions on Reliability, pp. 160-164, June 1984.

Argues the case for a classification system analogous to that in use for queues; identifies necessary features of such a system; and proposes a partial classification system. To be successful such a system must not only be a relatively convenient summary of diverse models, but must have the general support of the reliability community.

KEY WORDS: Classification, Queuing system, Reliability models.

[BERN85] Bernstein, "A Loosely Coupled Distributed System for Reliably Storing Data," IEEE Transactions on Software Engineering, pp. 446-453, May 1985.

Proposes an algorithm for storing information redundantly on the nodes of a broadcast network. A voting technique is used to increase reliability. Since multiple votes are cast only when copies of a data item disagree, the algorithm has the property that communication overhead is minimal. Nodes storing erroneous copies are automatically resynchronized. A Markov analysis is performed which relates parameters of the algorithm to the mean time to failure.

KEY WORDS: Broadcast network, Markov analysis, Redundancy, Reliability, Stable storage.

[BIDO85] Bidoit, et al, "Exceptional Handling: Formal Specification and Systematic Program Construction," IEEE Transactions on Software Engineering, pp. 242-251, March 1985.

Presents an algebraic specification language (PLUSS) and a program construction method. Programs are built systematically from an algebraic specification of the data with which they deal. The method was tested on a realistic problem and in these experiments, it turned out that error handling was the difficult part to specify and to program. It is shown how to cope with this problem at the specification level and during the program development process.

KEY WORDS: Abstract data types, Algebraic specification, Decomposition schemes, Error handling.

[BIRM85] Birman, Joseph, Raeuchle, and Abbadi, "Implementing Fault-Tolerant Distributed Objects," IEEE Transactions on Software Engineering, pp. 502-508, June 1985.

Describes a technique for implementing k-resilent objects--distributed objects that remain available, and whose operations are guaranteed to progress to completion, despite up to k site failures. An implementation is derived from the object specification automatically, and does not require any information beyond what would be required for a nonresilient nondistributed implementation.

KEY WORDS: Abstract data types, Availability, Concurrency, Consistency, Distributed systems, Fault tolerance, Recovery, Reliability.

[BLIK81] Blikle, "On the Development of Correct Specified Programs," IEEE Transactions on Software Engineering, pp. 519-527, September 1981.

Describes a method of program development which guarantees correctness. Programs consist of instruction and a specification and a program is correct if: 1) the operational part is totally correct; 2) the precondition guarantees nonabortion; and 3) the local assertions are adequate for proof of 1) and 2). The paper contains a description of an experimental programming language PROMET-1. The method is illustrated by the derivation of a bubblesort procedure.

KEY WORDS: Assertion-specified programs, Bubblesort procedures, Program correctness, PROMET-1.

[BLOO79] Bloom, Synchronization Mechanisms for Modular Programming Languages, MIT, Laboratory for Computer Science, January 1979. Report # MIT/LCS/TR-211.

Examines synchronization constructs from the standpoint of language design for reliable software. The criteria a synchronization mechanism must satisfy to support construction of reliable, easily maintainable concurrent software are defined. A definition of the range of problems considered to be synchronization problems is provided by describing the possible types of constraints that may be access to shared resources. This taxonomy of synchronization constraints to develop techniques for evaluating how well synchronization constructs meet the discussed criteria. The techniques are applied to three existing synchronization mechanisms: monitors, path expressions, and serializers. Evaluations are presented and the three mechanisms are compared.

KEY WORDS: Synchronization, Concurrency, Modularity, Data abstractions, Programming methodology.

[BLOO86] Bloomfield, Froome, "The Application of Formal Methods to the Assessment of High Integrity Software," IEEE Transactions on Software Engineering, pp. 988-993, September 1986.

Presents a case study in which the Vienna development method (VDM), a formal specification and development methodology, was used during the analysis phase of the assessment of a prototype nuclear reactor protection system. VDM specification was translated into the logic language Prolog to animate the specification and to provide a diverse implementation for use in back-to-back testing. It is claimed that this technique provides a visible and effective method of analysis which is superior to the informal alternatives.

KEY WORDS: Specification, Analysis, Protection system.

[BLUM86] Blumer, Sidhu, "Mechanical Verification of Automatic Implementation of Communication Protocols," IEEE Transactions on Software Engineering, pp. 827-843, August 1986.

Discusses an automated technique for protocol development and its application to the specification, verification, and semi-automatic implementation of an authentication protocol for computer networks. An overview of the specification language, implementation method, and software tools used with this technique is given. The authentication protocol is described, along with an example of its operation. The reachability analysis technique for the verification of some protocol properties is discussed, and protocol verification software that uses this technique is described. The results of the mechanical verification of some properties of the protocol and a partial implementation generated automatically from the protocol specification are presented.

KEY WORDS: Authentication, Automated development tools, Communication protocols, Formal description technique, Protocol implementation, Protocol specification, Protocol verification, State transition.

[BOCH83] Bochmann, Raynal, "Structured Specification of Communicating Systems," IEEE Transactions on Computers, pp. 120-133, February 1983.

Discusses specification methods for distributed systems. A model of communication processes with rendezvous interactions is assumed as a basis for the discussion. The basic concepts of the specification method are discussed and then applied to some complex examples. The stepwise refinement of ports and interactions is demonstrated by a hardware interface for which an abstract specification and a more detailed implementation is given. Proof rules for verifying the consistency of detailed and more abstract specifications are also discussed.

KEY WORDS: Communication processes, Design verification, Distributed system design, Parallel processing, Specification consistency, Specification methods.

[BOES86] Boesch, "Synthesis of Reliable Networks--A Survey," IEEE Transactions on Reliability, pp. 240-246, August 1986.

Presents a deterministic model for network reliability called network vulnerability. Many different vulnerability criteria and the related synthesis results are examined. These synthesis problems are all graph external questions. Certain reliability synthesis problems can be converted to a vulnerability question. Several open problems and conjectures are discussed.

KEY WORDS: Network reliability, Connectivity, Vulnerability.

[BOOT80] Booth, Wiecek, "Performance Abstract Data Types as a Tool in Software Performance Analysis and Design," IEEE Transactions on Software Engineering, pp. 138-151, March 1980.

Extends the concept of abstract data types to associate performance information with each abstract data type representation. The resulting performance abstract data type contains a functional part which describes the functional properties of the data type and a performance part which describes the performance characteristics of the data type. The methods for determining the necessary information to specify the performance part of the representation are discussed.

KEY WORDS: Abstract data types, Computation structures, Performance analysis, Software design.

[BREM81] Bremer, Drobnik, "A New Approach to Protocol Design and Validation," INWG Workshop "Protocol Testing--Towards Proof?", National Physical Laboratory, Teddington, England, May 1981.

Proposes a new approach for the integrated design and validation of protocols on the basis of formal service specifications. It extends the known concepts of abstract data types, abstract machines, and stepwise refinement to a distributed environment and uses the assertional technique of program validation. Its basic features are illustrated by the validation of the ISO-standardized data link control procedure HDLC.

KEY WORDS: Networks, Reliable communication, Graceful degradation protocols.

[BYRN85] Byrnes, Angell, "The Dependency Model: A Tool for Calculating System Effectiveness," IEEE Transactions on Reliability, pp. 17-24, April 1985.

The Dependency Model (DM) is a mathematically unsophisticated but useful and practical tool for measuring the effectiveness (reliability, availability, maintainability, efficiency, etc.) of a complex system. Description, comparison to other models, and benefits of DM are included. It has been successfully applied to performance evaluation of the TRIDENT Command and Control System.

KEY WORDS: Dependency model, System effectiveness, System degradation, Measure of effectiveness, Probability of system failure.

[CARC86] Carchiolo, Faro, Mirabella, Pappalardo, and Scollo, "A LOTOS Specification of the PROWAY Highway Service," IEEE Transactions on Computers, pp. 949-968, November 1986.

Presents a LOTOS specification of the PROWAY interface for process control applications, defined by IEC. LOTOS is shown to be tailored for the specification of asynchronous systems. In particular, it proves suitable for the specification both of the services which define an interface and of the protocol which implement it. It is shown how LOTOS supports formal reasoning aimed at establishing consistency between service and protocol specifications. Two examples of such a verification are developed, and advantages and limitations of this approach are outlined.

KEY WORDS: Computer communication standards, Formal specification, LAN, Multicast communication servicer, Protocol verification, Rapid prototyping, Temporal ordering.

[CEVA85] Cevano, "Toward High Confidence Software," IEEE Transactions on Software Engineering, pp. 1449-1455, December 1985.

Shows how software measurements can affect the confidence and trust the DoD places in its software. A management approach to achieving high confidence software is presented, highlighting software reliability as a key factor. A software reliability measurement methodology is described that: 1) specifies software reliability goals and processes needed to achieve these goals; 2) predicts and tracks progress during development; 3) estimates reliability based on testing effort and processes; 4) assesses achieved software reliability during operational use. A concept for a workable set of software reliability measures is discussed.

KEY WORDS: DOD applications, SDI, High confidence software, Software reliability measurement methodology.

[CHAN79] Chandy, Misra, "Distributed Simulation: A Case Study of Design and Verification of Distributed Programs," IEEE Transactions on Software Engineering, pp. 440-452, September 1979.

Proposes a distributed solution where processes communicate only through messages with their neighbors; there are no shared variables and no central process for message routing or process scheduling. Deadlock is avoided in this system despite the absence of global control. The correctness of a distributed system is verified by proving each of its component processes and then using inductive arguments.

KEY WORDS: Concurrent processes, Distributed systems, Program proving.

[CHAN81] Chandy, Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," Communications of the ACM, pp. 198-206, April 1981.

Presents an approach to carrying out asynchronous, distributed simulation on multiprocessor message-passing architectures. This scheme differs from other distributed simulation schemes because: 1) the amount of memory required by all processors together is bounded and is no more than the amount required in sequential simulation; and 2) the multiprocessor network is allowed to deadlock, the deadlock is detected, and then the deadlock is broken. Proofs of correctness of this approach are outlined.

KEY WORDS: Distributed systems, Message-passing systems, Communicating sequential processes, Deadlock, Recovery, Parallel algorithms.

[CHAT78] Chattergy, Pooch, "Analysis of Availability of Computer Systems Using Computer-Aided Algebra," Communications of the ACM, pp. 586-591, July 1978.

Presents analytical results related to the availability of a computer system constructed of unreliable processors. Results are obtained by using various computer-aided algebraic manipulation techniques. The purpose is to demonstrate that the difficulties of obtaining analytical solutions to Markov processes can be reduced by the application of symbol manipulation programs. Since many physical systems can be modeled by Markov and semi-Markov processes, the potential range of

application of these techniques is much wider than the problem of availability analyzed here.

KEY WORDS: Availability, Computer-aided algebra, Symbol manipulation programs, Markov and semi-Markov processes.

[CHEA79] Cheatham, Holloway, and Townley , "Symbolic Evaluation and the Analysis of Programs," IEEE Transactions on Software Engineering, pp. 402-417, July 1979.

Describes a symbolic evaluator for part of the EL1 language, with particular emphasis on techniques for handling conditional data sharing patterns, the behavior of array variables, and the behavior of variables in loops and during procedure calls. Symbolic evaluation is a form of static program analysis in which symbolic expressions are used to denote the values of program variables and computations. An expression simplifier, which is the heart of the system, is described in some detail. Potential applications of the symbolic evaluator to problems in program validation, verification, and optimization are mentioned.

KEY WORDS: Automatic programming analysis, First-order recurrence relations, Program optimization, Program verification.

[CHEN80] Chen, Akoka, "Optimal Design of Distributed Information Systems (DIS)," IEEE Transactions on Computers, pp. 1068-1079, December 1980.

Develops a model for the optimization of distributed information systems. This model considers simultaneously: 1) the distribution of processing power; 2) the allocation of programs and databases; and 3) the assignment of communication line capacities. It also considers the return flow of information, and dependencies between programs and databases. An algorithm has been developed to obtain the optimal solution of the model. The model can be used for the allocation of resources in DIS and to help managers decide whether to centralize or decentralize their information systems. An example of the use of the model in assisting a large bank to decide the optimal configuration of a proposed DIS is included.

KEY WORDS: Branch and bound methods, Distributed database, System design.

[CHEN81] Chen, Hoare, "Partial Correctness of Communication Protocols," INWG Workshop "Protocol Testing--Towards Proof?", National Physical Laboratory, Teddington, England, May 1981.

Discusses a chain of linearly connected processes in which each process can communicate only with its neighbour to the left or to the right. These chains can be used in the design of multi-level communications protocols, and an example of such is given.

KEY WORDS: Communication protocols network.

[CHER85] Cherkassky, Malek, "Reliability and Fail-Softness Analysis of Multistage Interconnection Networks," IEEE Transactions on Reliability, pp. 524-528, December 1985.

Presents a method for reliability analysis of multistage interconnection networks implemented with crossbar switching elements. Analytic estimates for the network reliability are derived, and the existence of a switching element with optimal fanout ensuring maximal network reliability is shown. General quantitative measure for fail-softness evaluation of multistage interconnection networks is introduced. It is shown that under single-fault assumption, the fail-softness improves with an increase in network size.

KEY WORDS: Banyan, Fail-softness, Multistage interconnection, Reliability prediction.

[CHER86] Chern, Jan, "Reliability Optimization Problems with Multiple Constraints," IEEE Transactions on Reliability, pp. 431-436, October 1986.

Presents a class of reliability optimization problems with multiple-choice constraints. A two-phase solution method is presented for solving these problems. In phase 1, the problem is decomposed into n subproblems. These subproblems can be solved by dynamic programming, independently (parallelism). In phase 2, a 0-1 multiple-choice knapsack problem, which is generated from phase 1, is solved. A combinatorial tree which always satisfies the multiple-choice constraints is used. The method is illustrated with a numerical example.

KEY WORDS: Redundancy, Reliability optimization, Dynamic programming, Knapsack problem.

[CHEU80] Cheung, "A User-Oriented Software Reliability Model," IEEE Transactions on Software Engineering, pp. 118-125, March 1980.

Defines a user-oriented software reliability figure of merit to measure the reliability of a software system with respect to a user environment. The effects of the user profile, which summarizes the characteristics of the users of a system, on system reliability are discussed. A simple Markov model is formulated to determine reliability of a software system based on the reliability of each individual module and the measured intermodular transition probabilities as the user profile. Sensitivity analysis techniques are developed to determine modules most critical to system reliability. The applications of this model to develop cost-effective testing strategies and to determine the expected penalty cost of failures are also discussed. Some future refinements and extensions of the model are presented.

KEY WORDS: Self-metric software, Software reliability, Software reliability model.

[CHI85] Chi, "Formal Specification of User Interfaces: A Comparison and Evaluation of Four Axiomatic Approaches," IEEE Transactions on Software Engineering, pp. 671-685, August 1985.

Presents a comparison for four axiomatic approaches which have been applied to the specification of a commercial user interface--the line editor for the Tandy PC-1 Pocket Computer.

Techniques are shown to result in complete and relatively concise descriptions. A number of useful and non-trivial properties of the interface are formally deduced from one of the specifications. A direct implementation of the interface is constructed from a formal specification.

KEY WORDS: Algebraic specifications, Formal specifications, Software design, Specification implementation.

[CHOW79] Chow, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," IEEE Transactions on Computers, pp. 354-361, May 1979.

Presents, analyzes, and compares queueing models for a simple heterogeneous multiple processor system. Each model is distinguished by a job routing strategy which is designed to reduce the average job turn around time by balancing the total load among the processors. The job routing strategies are divided into two classes: deterministic and nondeterministic. The nondeterministic policies are described by state independent branching probabilities. For the deterministic policies, the next processor is chosen to minimize or maximize the expected value of a performance related criterion function. The models with nondeterministic policies are analyzed using standard queueing network techniques. An approximate numerical method is introduced for analyzing two-processor heterogeneous models with deterministic routing policies and the performance of sample two-processor systems is compared.

KEY WORDS: Job routing, Load balancing, Multiple processor system, Performance analysis.

[CHWA81] Chwa, Hakimi, "On Fault Identification in Diagnosable Systems," IEEE Transactions on Computers, pp. 414-422, June 1981.

Gives a characterization of t1/t1-diagnosable systems. Class of t0 diagnosable systems is considered. It is shown for any member of this class that: 1) necessary and sufficient conditions for t1/t1-diagnosability are greatly simplified, 2) optimal diagnosis algorithms of time complexity O(nt0) exist, and most importantly, 3) given the test results, any set F of faults with |F| <= t1 can be identified to within a set F', with F a subset of F', and F' <= min{t1,|F| + 1}.

KEY WORDS: Diagnosis algorithms, Fault diagnosis, System diagnosis, T-diagnosable system.

[CLAR86] Clark, Neufeld, and Colbourn, "Maximizing the Mean Number of Communicating Vertex Pairs in Series-Parallel Networks," IEEE Transactions on Reliability, pp. 247-251, August 1986.

Indicates that a communication network can be modeled as a 'probabilistic graph' where each of 'b' edges represents a communication line and each of 'n' vertices represents a communication processor. Each edge 'e' (vertex 'v') functions with probability pe (p sub e). If edges fail independently with uniform probability 'p' and vertices do not fail, the probability that the network is connected is the 'probabilistic connectedness' and is a standard measure of network reliability.

It is shown that for large 'p', the most reliable series-parallel network must have the fewest minimum edge cutsets and for small 'p', the most reliable network must have maximum pairs of adjacent edges. A construction is presented which incrementally improves the communicating vertex pair mean for many networks and demonstrates that a 'fan' maximizes this measure over maximal series parallel networks with exactly two edge cutsets of size two.

KEY WORDS: Network reliability, All-terminal reliability, Series-parallel network.

[COLE81] Coleman, "A Method for the Syntax Directed Design of Multiprograms," IEEE Transactions on Software Engineering, pp. 189-196, March 1981.

Describes a method of program design which leads to the expression of a program as a pipeline network of simple processes. Starting from the problem statement the valid inputs and outputs are specified by grammars, which can be combined to define the requisite translation. A notation for translation grammars is described informally which allow a translation to take into account semantic as well as syntactic information. The notation is capable of direct compilation but it is shown how it may be used to derive a program into a conventional high level language. It is shown that more complex problems can be solved by simple pipeline structures of simple translations. The method is illustrated by examples from data processing.

KEY WORDS: Attributed translations, Multiprocessing, Pipeline programs, Programming methodology.

[COOK79] Cook, *MOD--A Language for Distributed Programm ing, University of Wisconsin-Madison, Mathematics Research Center, October 1979. Technical Summary Report #2008.

Discusses a high level language called *MOD. *MOD is a high-level language system which attempts to address the problems of high communications costs and the absence of shared variables and procedures as synchronization tools, by creating an environment conducive to efficient and reliable network software construction. The concept of a processor module is introduced as well as a methodology for distributed data abstraction and process communication. In addition, a VHLN (virtual, high-level language network) is proposed for system development.

KEY WORDS: Distributed programming, Modula, Processor module, Data abstraction.

[COST78] Costes, "Reliability and Availability Models for Maintained Systems Featuring Hardware Failures and Design Faults," IEEE Transactions on Computers, pp. 548-559, June 1978.

Studies the modeling of systems featuring hardware and software faults as a means of evaluating the availability and reliability characteristics. Case of a nonredundant computer is studied and it is shown that the unavailability presents an overshoot with respect to its asymptotic value whose height and length are functions of the failure rates associated with

the different design errors. Fault-tolerant systems that include protective redundancies both at the hardware level and at the software level are studied; the importance of homogeneous solutions on both levels is shown.

KEY WORDS: Availability, Hardware and software modeling, Hardware and software redundancy, Reliability.

[COUR85] Courtiat, "A Simulation Environment for Protocol Specifications Described in Estelle," Proceedings of the IFIP WG 6.1 Fifth International Workshop on Protocol Specification, Testing and Verification, North-Holland Publishing, Amsterdam, Holland, May 1985.

Presents the design of a dedicated simulation environment for protocols and services specified by using the Estelle formal description technique. The definition of the simulator global underlying model and the derivation of this model from an Estelle description are particularly emphasized.

KEY WORDS: Estelle, Protocols.

[CRAL85] Crall, Sidhu, "Executable Logic Specifications for Protocol Service Interfaces," Proceedings of the IFIP WG 6.1 Fifth International Workshop on Protocol Specification, Testing and Verification, North-Holland Publishing, Amsterdam, Holland, May 1985.

Discusses a general, formal modeling technique for protocol service interfaces. An executable description of the model using a logic programming based language, Prolog, is presented. The model is applied to the service interfaces of protocol standards developed for the transport layer of the ISO/OSI architecture. Protocols studied are NBS Class 2 and Class 4 and DoD TCP. The author suggests that Prolog is a useful formal language for specifying protocol interfaces.

KEY WORDS: Modeling technique of protocol service interfaces, Prolog, ISO/OSI transport layer, NBS Class 2 and Class 4 Transport layer, DoD TCP.

[CREM78] Cremers, Hibbard, "Orthogonal Information Structures," Proceedings of a Conference on Theoretical Computer Science, pp. 182-190, University of Waterloo, Waterloo, Ontario, Canada, 1978.

Develops, in the mathematical framework of data spaces, some important general principles of information structuring. The principles are related to the notions of redundancy of information, completeness of a set of access paths, information sharing and compounding, and virtual access to information. The results are relevant to both sequential and concurrent processing.

KEY WORDS: Information structuring, Redundancy, Sequential and concurrent processing.

[CRIS85] Cristian, "A Rigorous Approach to Fault-Tolerant Programming," IEEE Transactions on Software Engineering, pp. 23-31, January 1985.

Investigates the design of programs that are tolerant of hardware fault occurrences and processor crashes. Using a stable storage management system as a running example, a new approach for specifying, understanding, and verifying the correctness of fault-tolerant software is suggested. The approach extends previously developed axiomatic reasoning methods to the design of fault-tolerant systems by modeling faults as being operations that are performed at random time intervals on an computing system by the system's adverse environment. A clear separation is made between software correctness and system reliability. The combined correctness and reliability verifications establish that under given fault and reliability hypothesis, a system behaves according to its functional specifications with a probability greater than that required by its reliability specifications.

KEY WORDS: Availability, Correctness, Fault-tolerance, Programming logic, Reliability, Stochastic modeling.

[CROW84] Crow, Singpurwalla, "An Empirically Developed Fourier Series Model for Describing Software Failures," IEEE Transactions on Reliability, pp. 176-183, June 1984.

Proposes an empirically developed Fourier series model which can adequately describe data, and which under certain circumstances can be used to predict future failures. Analysis is informal, and the key tool used to develop the approach is a spectrogram of data. Emphasis is placed on data analysis rather than statistical inference.

KEY WORDS: Software reliability, Software failure, Clustering, Cyclic trend, Fourier series, Spectral analysis.

[CURR86] Currit, Dyer, and Mills, "Certifying the Reliability of Software," IEEE Transactions on Software Engineering, pp. 3-11, January 1986.

Presents a procedure for certifying the reliability of software before its release to users. Development of certified software products and the derivation of a statistical model used for reliability projection is discussed.

KEY WORDS: Incremental development, Software reliability certification, Statistical quality control, Statistical testing process.

[DAHB86] Dahbura, "An Efficient Algorithm for Identifying the Most Likely Fault Set in a Probabilistically Diagnosable Systems," IEEE Transactions on Computers, pp. 354-356, April 1986.

Gives an O(n**3) algorithm for determining the most likely set of faulty processors in a class of systems introduced by Maheshwari and Hakimi, known as probabilistically diagnosable systems. This technique uses the a priori probability of each unit combined with the results of tests which the processors administer to one another to perform diagnosis. The algorithm uses the well-known results on network flow and minimum weight vertex cover sets.

KEY WORDS: Diagnosis, Fault tolerance, Multiprocessor systems.

[DALE86] Dale, Winterbottom, "Optimal Allocation of Effort to Improve System Reliability," IEEE Transactions on Reliability, pp. 188-191, June 1986.

Discusses optimal allocation of development effort to improve reliability for systems of general, but fixed, structure from a deterministic standpoint also when there is uncertainty in component reliabilities at various stages of development. A computation algorithm is given for implementing the optimal allocation, called the optimal policy, and the form of solution is related to the special case of a series system when development effort functions are the same for all components. It is suggested that this methodology is a useful aid to decision-making about reliability improvements.

KEY WORDS: System reliability, Optimization, Resource allocation, Effort function.

[DANN82] Dannenberg, "Formal Program Verification Using Symbolic Execution," IEEE Transactions on Software Engineering, pp. 43-51, January 1982.

Introduces a notation which allows a concise presentation of rules of inference based on Symbolic execution. Symbolic execution provides a mechanism for formally proving programs correct. This notation is used to develop rules of inference to handle a number of language features and an attribute grammar is used to formally describe symbolic expression evaluation.

KEY WORDS: Control constructs, Program proving, Program verification.

[DAVI78] Davies, "Synchronization and Matching in Redundant Systems," IEEE Transactions on Computers, pp. 531-539, June 1978.

Introduces a novel mutual feedback technique, called from vulnerability to common-point failures. Application of a fault-tolerant crystal-controlled clock is described. It is shown in the presence of certain sensor failure, there is no signal selection rule that can pick a common input value, and it is also shown how this problem can be circumvented by using multiple levels of voters.

KEY WORDS: Asynchronous networks, Fault-tolerant computers, NMR, TMR.

[DECH86] Dechter, Kleinrock, "Broadcast Communication and Distributed Algorithms," IEEE Transactions on Computers, pp. 210-219, March 1986.

Addresses ways in which one can use 'broadcast communication' in distributed algorithms and the relevant issues of design and complexity. An algorithm is presented for merging 'k' sorted lists of n/k elements using 'k' processors and proving its worst case complexity to be 2n. In a variation of the algorithm, it is shown that by using an extra local memory of O(k) the number of broadcasts is reduced to 'n'. The cost incurred by the channel access scheme is discussed and it is proved that resolving conflicts whenever 'k' processors are involved introduces a cost factor of at least log k.

KEY WORDS: Access scheme, Broadcast, Complexity analysis, Distributed algorithm, Parallel algorithms.

[DENN77] Denning, Denning, "Certification of Programs for Secure Information Flow," Communications of the ACM, pp. 504-513, July 1977.

Presents a certification mechanism for verifying the secure flow of information through a program. Appropriate semantics are presented and proved correct. The mechanism can prove that a program cannot cause supposedly nonconfidential results to depend on confidential input data.

KEY WORDS: Protection, Security, Program certification, Confinement.

[DERS81] Dershowitz, "Interference Rules for Program Annotation," IEEE Transactions on Software Engineering, pp. 207-222, March 1981.

Presents methods whereby an Algol-like program given together with its specifications can be documented automatically. The program is incrementally annotated with invariant relations that hold between program variables at intermediate points in the program test and explain the actual working of the program regardless of whether it is correct. Thus, this documentation can be used for proving correctness or as an aid for debugging. The annotation techniques are formulated as Hoare-like inference rules that derive invariants from the assignment statements, from the control structure of the program, or, heuristically, from suggested invariants. The application of these rules is demonstrated by examples that have run on an experimental implementation.

KEY WORDS: Invariant assertions, Program annotation, Program correctness, Verification.

[DESO78] de Sousa, "Sift-Out Modular Redundancy," IEEE Transactions on Computers, pp. 624-628, July 1978.

Proposes and designs a fault tolerance technique for digital systems. An appropriate number of identical channels are provided for each module. The number of channels depend upon the particular application, and all channels are active as long as they are fault-free. Upon the failure of a channel, its contribution to the module output ceases. The configuration tolerates up to (L - 2) channel failures, if L is the initial number of channels.

KEY WORDS: Fault-tolerant computing, Modular redundancy, Reliability, Responsive structure.

[DESO86] de Souza e Silva, Gail, "Calculating Cumulative Operational Time Distributions of Repairable Computer Systems," IEEE Transactions on Computers, pp. 322-332, April 1986.

Considers computer systems for which repair can be performed to put the system back in operation. Behavior is assumed to be modeled as a homogeneous Markov process. The distribution of cumulative operational time is calculated numerically. The main advantages include the ability to specify error tolerances

in advance, numerical stability, and simplicity of implementation.

KEY WORDS: Availability distribution, Dependable computer systems, Reliability, Repairable computer systems.

[DEVI77] Devillers, "Game Interpretation of the Deadlock Avoidance Problem," Communications of the ACM, pp. 741-745, October 1977.

Suggests that when each process specifies its future needs by a flowchart of need-defined steps, a global approach to the phenomenon and its interpretation as a game between the operating system and the processes allows formalization of risk and safety concepts. The bipartite graph representation of this game may then be used to construct explicitly the set of safe states and to study their properties.

KEY WORDS: Multiprogramming, Time sharing, Resource allocation, Deadlock, Interlock, Deadlock avoidance.

[DIAS81] Dias, Jump, "Analysis and Simulation of Buffered Delta Networks," IEEE Transactions on Computers, pp. 273-282, April 1981.

Presents analytic and simulation results for the performance of delta networks in a packet communication environment. The performance of buffered delta networks is compared with unbuffered delta networks and crossbar switches. It is demonstrated that buffering produces considerable improvement in the performance of these networks, making their performance comparable to that of crossbar switches.

KEY WORDS: Crossbar switches, Delta networks, Multi-stage interconnection networks.

[DIJK74] Dijkstra, "Self-stabilizing Systems in Spite of Distributed Control," Communications of the ACM, pp. 643-644, November 1974.

The synchronization task between loosely coupled cyclic sequential processes can be viewed as keeping the relation 'the system is in a legitimate state' invariant. As a result, each individual process step that could possibly cause violation of that relation has to be preceded by a test deciding whether the process in question is allowed to proceed or has to be delayed. A system is presented that is kept in a legitimate state and is distributed over processes. Problems are discussed and solutions with 'k', four, and three-state machines are given.

KEY WORDS: Multiprocessing, Networks, Synchronization, Self-stabilization, Robustness, Error recovery, Self-repair.

[DOWN85a] Downs, "An Approach to the Modeling of Software Testing with Some Applications," IEEE Transactions on Software Engineering, pp. 375-386, April 1985.

Allows the assessment of the effects of different testing (and debugging) strategies in different situations. Techniques developed can be used to estimate, prior to the commencement of testing, the optimum allocation of test effort for software which is to be nonuniformly executed in its operational phase. Applications of statistical models where the data environment undergoes changes are discussed. Two models for the assessment of the effects of imperfections in the debugging process are investigated.

KEY WORDS: Computer performance modeling, Reliability growth, Software reliability, Software testing, Stochastic models.

[DOWN85b] Downs, "A Review of Some of the Reliability Issues in Software Engineering," Journal of Electrical and Electronics Engineering, Australia, pp. 36-48, March 1985.

Examines problems with software testing, statistical models of errors in software, Littlewood's and Sukert's models of the software failure process, formal methods of proving programs, operational reliability of software, fault tolerant software, concurrent systems, and complexity measures. Software vs. hardware reliability is investigated through an example from Musa of how having redundant processors does not increase reliability if operating systems are not redundant.

KEY WORDS: Software reliability, Fault-tolerance, Concurrent systems.

[DOWN85c] Downs, "Software Unreliability and Some Engineering Implications," Conference on Computers and Engineering 1985, pp. 95-100, Hobart, Tasmania, Australia, September 1985.

Reviews efficacy of testing by executing every statement at least once (doesn't execute all logic paths) and every logic path (finding paths very complex, may reduce to finding arbitrary input subset). The following statistical models are reviewed: Mills error-seeding model (can't insert pseudo-random errors randomly); Jelinski-Moranda model (assumes failure rate is proportional to the number of errors; bad because of heavily-used, lightly-used sections of code); white box model (desirable but not in existence yet) and formal methods are suggested to be bad when program is already coded and good when used as a development aid.

KEY WORDS: Software reliability, Mills error-seeding model, Jelinski-Moranda model.

[DOWN86] Downs, "Extensions to an Approach to the Modeling of Software Testing with Some Performance Comparisons," IEEE Transactions on Software Engineering, pp. 979-987, September 1986.

Shows how a major assumption underlying a previously reported approach to the modeling of software testing can be relaxed in order to provide a more realistic model. Under the assumption of uniform execution the new model is found to perform only marginally better than the previous model, indicating that the uniform execution assumption is a poor one. The results obtained point the way to further developments which are likely to lead to models whose performance is superior to that of the

nonuniform execution models presented. Some attention is devoted to the problem of comparison of performance of different models and some difficulties in this area are pointed out.

KEY WORDS: Probability models, Reliability growth, Software reliability, Software testing.

[DUNH86] Dunham, "Experiments in Software Reliability: Life-Critical Applications," IEEE Transactions on Software Engineering, pp. 110-123, January 1986.

Discusses four reliability data gathering experiments which were conducted using a small sample of programs for two problems having ultrareliability requirements, n-version programming for fault detection, and repetitive run modeling for failure and fault rate estimation.

KEY WORDS: Life-critical software, Real-time software, Software modeling and measurement, Software reliability.

[DUPU85] Dupuis, Hebuterne, "On the Use of Quantitative Evaluation To Assess and Study Distributed Algorithms Properties," Proceedings of the IFIP WG 6.1 Fifth International Workshop on Protocol Specification, Testing and Verification, North-Holland Publishing Company, May 1985.

Compares the quantitative performances of mutual exclusion distributed algorithms. Emphasis is placed on the importance of synchronization mechanisms provided by time stamps. A bad synchronization lengthens waiting times, provokes asymmetrical and non-FIFO ordering. These considerations lead naturally to the need of stronger concepts for fairness and symmetry, necessarily defined from an user point of view, and based on quantifiable and measurable parameters.

KEY WORDS: Mutual exclusion, Distributed algorithms, Synchronization, Time stamps.

[ECKH85] Eckhardt, Lee, "A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors," IEEE Transactions on Software Engineering, pp. 1511-1516, December 1985.

Develops a theoretical basis for the study of redundant software which: 1) provides a probabilistic framework for empirically evaluating the effectiveness of a general multiversion strategy when component versions are subject to coincident errors and 2) permits an analytical study of the effects of these errors. A condition under which a multiversion system is a better strategy than relying on a single version is given and some differences between the coincident errors model developed here and the model that assumes independent failures of component versions are studied.

KEY WORDS: Coincident errors, Fault-tolerant software, Multiversion software, Reliability of redundant software.

[EKAN79] Ekanadham, Bernstein, "Conditional Capabilities," IEEE Transactions on Software Engineering, pp. 458-464, September 1979.

Considers protection in capability-based operating systems. The concept of conditional capability which is a generalization

of conventional capability is proposed. The conditional capability can only be exercised when certain conditions relating to the context of its use are satisfied. It is shown that such capabilities form a basis upon which features domains of protection, revocation, and type extension can be built. The implementation of these features can be isolated into separate modules thus leaving the basic protection module uncluttered and simplifying the overall structure of the system.

KEY WORDS: Access control, Keys, Locks, Protection.

[ELLA81] Ellazy, "The Determination of Loop Invariants for Programs with Arrays," IEEE Transactions on Software Engineering, pp. 197-206, March 1981.

Describes a method for the generation of 'loop predicates' or 'invariant assertions' for programs operating on arrays. The technique described is an application of difference equations.

KEY WORDS: Invariant assertions, Loop predicates, Program validation.

[ENGE86] Engelhardt, Bain, "On the Mean Time Between Failures For Repairable Systems," IEEE Transactions on Reliability, pp. 419-422, October 1986.

Investigates the relationship between two of the most frequently considered alternatives in assessing the reliability of a repairable system: the reciprocal of the intensity function, and the mean waiting time from 't' until the next failure. The theorem states a necessary and sufficient condition for the mean time until the next failure to be asymptotically proportional to the reciprocal of the intensity function. A monotonicity property is also established between these two concepts which could be used to obtain conservative statistical confidence limits for the mean time until the next failure. The author suggests that until more is known about the mean time from 't' until the next failure, it would be advisable to use the reciprocal of the intensity function, which has been studied more extensively, as the basis of reliability assessment for a repairable system.

KEY WORDS: Mean time between failures, Nonhomogeneous Poisson process, Power-intensity process, Log-linear process.

[ESTR86] Estrin, Fenchel, Razouk, and Vernon, "SARA (Systems ARchitects Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent Systems," IEEE Transactions on Software Engineering, pp. 293-311, February 1986.

Describes an environment to support designers in the modeling, analysis, and simulation of concurrent systems. It is shown how a fully nested structure model supports multilevel design and focuses attention on the interfaces between the modules which serve to encapsulate behaviour. The effectiveness of the explicit environment model in SARA is discussed and the capability to analyze correctness and evaluate performance of a system model are demonstrated.

KEY WORDS: Concurrent systems, Hierarchical design, Interactive simulation, Performance models, Queueing models, Reachability analysis.

[FAGA86] Fagan, "Advances in Software Inspections," IEEE Transactions on Software Engineering, pp. 744-751, July 1986.

Presents new studies and experiences that enhance the use of the inspection process and improves its contribution to development of defect-free software at lower costs. Examples of benefits are cited, followed by a description of the process and some methods of obtaining the enhanced results.

KEY WORDS: Defect detection, Quality assurance, Software development, Software quality.

[FARO81] Faro, "Specifications and Validation of Protocols and Interfaces," INWG Workshop "Protocol Testing--Towards Proof?", National Physical Laboratory, Teddington, England, May 1981.

Discusses sets of rules about communications. Problems in developing protocols which involve establishing interactions between processes and interfacing the interaction are presented. Emphasis is placed on methods for formalizing and validating protocols for communication.

KEY WORDS: Networks, Communication protocols, Theory of colloquies, Automata, Petri nets.

[FARO83] Faro, Scollo, "SDL and CCS Based Description of Communicating Entities," Proceedings of the IFIP WG 6.1 Third International Workshop on Protocol Specification, Testing and Verification, North-Holland Publishing Company, June 1983.

Explores some aspects of the application of two formal techniques, namely the Specification Description Language (SDL) and the Calculus of Communicating Systems (CCS), to a model of architecture for communicating entities. The worths and limits of both are discussed and illustrated by examples.

KEY WORDS: Specification Description Language, Calculus of Communicating Systems, Computer communications.

[FERN85] Fernandez, Richier, and Voiron, "Verification of Protocol Specifications Using the Cesar System," IFIP Workshop on Protocols '85, pp. 1-28, June 13-15, 1985.

Presents a static analysis tool for the verification of specifications of communicating systems. The method consists of verifying, on the description of the behavior of the system, its specifications given by a set of properties expressed in a formalism based on logic. The interest of this approach is illustrated with an example of a real-life, time-dependent communication protocol.

KEY WORDS: Verification, Specifications, Communicating systems, Semi-automatic tools, Temporal logic.

[FICK85] Fickas, "Automating the Transformational Development of Software," IEEE Transactions on Software Engineering, pp. 1268-1277, November 1985.

Presents a report on implementing transformations in software development. The work has established goals, strategies, reasonable selection and user transformations which are presented. Emphasis is placed on problem solving and interactions between user and system.

KEY WORDS: Knowledge-based software development, Program transformation systems.

[FINK80] Finkel, Solomon, "Processor Interconnection Strategies," IEEE Transactions on Computers, pp. 360-370, May 1980.

Describes four families of topologies for interconnecting many identical processors into a computer network. Each family extends to arbitrarily many processors while keeping the number of neighbors of any one processor fixed. These families are investigated with respect to bus load, routing algorithms, and the relation between the average interprocessor distance and the size of the network.

KEY WORDS: Computer networks, Distributed computing, Message routing, Multiprocessor architectures.

[FLON77] Flon, et al, "Nondeterminism and the Correctness of Parallel Programs," IFIP Working Conference on the Formal Description of Programming Concepts, St Andrews, New Brunswick, Nova Scotia, August 1-5, 1977.

Presents weakest pre-conditions which describe weak correctness, blocking, deadlock, and starvation for nondeterministic programs. A procedure for converting parallel programs to nondeterministic programs is described, and the correctness of various example parallel programs is treated in this manner. A busy-wait mutual exclusion scheme, and the problem of the Five Dining Philosophers are included.

KEY WORDS: Parallel programs, Nondeterminism, Verification, Mutual exclusion.

[FLYN80] Flynn, Hennessy, "Parallelism and Representation Problems in Distributed Systems," IEEE Transactions on Computers, pp. 1080-1086, December 1980.

Uses a hierarchical view of program representation to explain the problems of matching various representations to underlying distributed architectures. Methods of detecting parallelism and their limitations are presented. The concept of an ideal machine leads to a representation employing a directly executed language. The issue of suitable initial representation for distributed hardware is approached employing a functional language basis.

KEY WORDS: Directly executed, Languages, Distributed systems, Parallelism.

[FOST80] Foster, "Error Sensitive Test Cases Analysis (ESTCA)," IEEE Transactions on Software Engineering, pp. 258-264, May 1980.

Presents a hardware failure analysis technique adapted to software which yielded three rules for generating test cases sensitive to code errors. These rules, and a procedure for generating these cases, are given with examples.

KEY WORDS: Program correctness, Program testing, Software errors, Software reliability.

[FRAN81] Franklin, "VLSI Performance Comparison of Banyan and Crossbar Communications Networks," IEEE Transactions on Computers, pp. 283-290, April 1981.

Compares performance characteristics of Banyan and crossbar communications networks in a VLSI Environment, where it is assumed that the entire network resides on a single VLSI chip and operates in a circuit switched mode. A high-level model of the space (area) and time (delay) requirements for these networks is developed and relative performance comparisons are made based on a space-time product measure.

KEY WORDS: Banyan networks, Crossbar networks, Space-time product, VLSI.

[FREE83] Freeman, Hirschman, McKay, Miller, and Sidhu, "Logic Programming Applied to Knowledge-Based Systems, Modeling, & Simulation," Conference on Artificial Intelligence, Oakland University, Rochester, MI, April 26-27, 1983.

Defends the application of logic programming. Specifications have the advantage of being a prototype of a system, being interactive and assurance of verification. Implementation of the rules of KNET has made a framework for specifications and helps form expressions for relations among models. The framework also has libraries of component types and interrelationships and it can also check itself for integrity. An approach to modeling performance characteristics of a system is the construction of model-based maintenance assistants is investigated.

KEY WORDS: Expert systems, Executable specifications, Logic programming systems.

[FUJI78] Fujiwara, "On the Computational Complexity of System Diagnosis," IEEE Transactions on Computers, pp. 881-885, October 1978.

Analyzes the computer complexity of system diagnosis. It is shown that several problems for instantaneous and sequential fault diagnosis of systems are polynomially complete and that for single-loop systems these problems are solvable in polynomial time.

KEY WORDS: Fault diagnosis, Polynomial time algorithm, Polynomially complete, Self-diagnosable systems, Turing machines.

[FUKU78] Fukunaga, Short, "Generalized Clustering for Problem Localization," IEEE Transactions on Computers, pp. 176-180, February 1978.

Describes a modification of conventional clustering for problem localization. The concept of clustering criteria which is used for partitioning a training set, and dependent on prior information in regards to the training set is introduced. A procedure is discussed for applications of piecewise linear classifier design and piecewise linear density estimation.

KEY WORDS: Clustering, Density estimation, Pattern recognition, Problem reduction or localization.

[GARC82] Garcia-Molina, "Elections in a Distributed Computing Systems," IEEE Transactions on Computers, pp. 48-59, January 1982.

Explains that after a failure occurs in a distributed computing system, it is often necessary to reorganize the active nodes so that they can continue to perform a useful task. The first step in such a reorganization of reconfiguration is to elect a coordinator node to manage the operation. Such elections, and reorganization are discussed. Two types of reasonable failure environments are studied. For each environment assertions which define the meaning of an election are presented. An election algorithm which satisfies the assertions is presented for each environment.

KEY WORDS: Crash recovery, Distributed computing systems, Failures, Mutual exclusion, Reorganization.

[GARD80] Gardarin, Chu, "A Distributed Control Algorithm for Reliably and Consistently Updating Replicated Databases," IEEE Transactions on Computers, pp. 1060-1067, December 1980.

Presents a deadlock-free distributed control algorithm for robustly and consistently updating replicated databases. This algorithm is based on local locking and time stamps on lock tables which permit detection of conflicts among transactions executed at different sites. Messages are exchanged in the network whenever a transaction commitment occurs, that is, at the end of every consistent step of local processing. Conflicts among remote transactions are resolved by a roll back procedure. Local restart is based on a journal of locks which provides backup facilities. Performance in terms of the number of messages and volume of control messages of the proposed algorithm is compared with that of the voting and centralized locking algorithms. These results reveal that the proposed distributed control algorithm performs, in most cases, comparably to the centralized locking algorithm and better than the voting algorithm.

KEY WORDS: Distributed control, Deadlock, Locking, Recovery, Replicated databases.

[GARM81] Garman, "The 'Bug' Heard 'Round the World," ACM SIGSOFT Software Engineering Notes, pp. 3-10, October 1981.

Discusses, in detail, the software problem that delayed the first shuttle orbital flight. The problem was the Backup Flight System was not in synch with the control computers. The development schedule which forces development to be iterative and incremental, not in an orderly life cycle, is blamed.

KEY WORDS: Software life cycle reliability, Software development, Software maintenance.

[GEHA85] Gehani, McGettrick, Software Specification Techniques, Addison-Wesley, Reading, MA, 1985.

Presents a collection of most of the important papers on formal specification published over the past 15 years. Several case studies are included that show the value of using formal techniques in requirements and design.

KEY WORDS: Verification, Specification, Concurrent.

[GILB72] Gilbert, "Interference Between Communicating Parallel Processes," Communications of the ACM, pp. 427-437, 1972.

Reports on interference in communication. Solutions and informal proofs are given for the mutual exclusion problem. System behavior is described by rules giving the transitions processes make from one state to another. The rules formulate the problem while the proofs verify or discredit a solution.

KEY WORDS: Concurrent programming, Cooperating processes, Mutual exclusion, Parallel processes.

[GLAS79] Glass, Software Reliability Guidebook, Prentice-Hall, 1979.

Surveys technological and management techniques that are intended to be useful for all application areas and sizes of software projects. Special emphasis is placed on the problems of large projects (eg. military/space applications, massive interrelated data bases).

KEY WORDS: Software reliability, Military/space applications.

[GLAS81] Glass, "Persistent Software Errors," IEEE Transactions on Software Engineering, pp. 162-168, March 1981.

Describes persistent software errors--those which are not discovered until late in development, such as when the software becomes operational--as by far the most expensive kind of error. Via analysis of software problem reports, it is discovered that the predominant number of persistent errors in large-scale software efforts are errors of omitted logic..., that is, the code is not as complex as required by the problem to be solved. Peer design and code review, desk checking, and ultra-rigorous testing may be the most helpful of the currently available technologies in attacking this problem. The author states that new and better methodologies are needed.

KEY WORDS: Complexity, Persistent software errors, Software problem report, Testing rigor.

[GLIG79a] Gligor, "Review and Revocation of Access Privileges Distributed Through Capabilities," IEEE Transactions on Software Engineering, pp. 575-585, November 1979.

Presents an approach to solving the problem of review and revocation of access privileges distributed through capabilities.

The approach requires that a capability propagation graph be maintained in memory spaces associated with subjects (e.g. domains, processes, etc.) that make copies of the respective capability; the graph remains inaccessible to those subjects, however. Advantages and disadvantages of this approach are examined.

KEY WORDS: Access privilege, Management policies, Capabilities, Shared objects, Selective revocation.

[GLIG79b] Gligor, Lindsay, "Object Migration and Authentication," IEEE Transactions on Software Engineering, pp. 607-611, November 1979.

Describes a mechanism to allow a type manager to authenticate and reinstate migrated objects. Problems stemming from the hierarchical structure of the system itself are solved. The mechanisn is based on combination of cryptographic techniques using (nondistributable) centralized, secret keys, and data redundancy.

KEY WORDS: Authentication, Capabilities, Encryption, Hierarchical systems, Object migration, Redundancy.

[GLIG80] Gligor, Shattuck, "On Deadlock Detection in Distributed Systems," IEEE Transactions on Software Engineering, pp. 435-439, September 1980.

Refers to a hierarchically organized and distributed protocol for deadlock detection in distributed databases in [MENA79]. It is shown that the distributed protocol it is incorrect and possible remedies are presented. The author maintains that the distributed protocol remains impractical because "condensations" of "transaction-wait-for" graphs make graph updates difficult to perform. Delayed graph updates cause the occurrence of false deadlocks in this as well as in some other deadlock detection protocols for distributed systems. The performance degradation that results from false deadlocks depends on the characteristics of each protocol.

KEY WORDS: Deadlock detection, Distributed systems, False deadlocks, Ostensibly blocked transactions.

[GOEL80] Goel, "A Summary of the Discussion on 'An Analysis of Competing Software Reliability Models'," IEEE Transactions on Software Engineering, pp. 501-502, September 1980.

Summarizes a technical dialogue on software reliability models. Strengths and weaknesses of several software reliability models are evaluated.

KEY WORDS: Software reliability models.

[GOEL85] Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability," IEEE Transactions on Software Engineering, pp. 1411-1424, December 1985.

Discusses a number of analytical models for assessing the reliability of a software system. An overview of the key modeling approaches is presented, and a critical analysis of the underlying assumptions is provided, and the limitations

and applicability of the models during the software development cycle are assessed. Also, a step-by-step procedure for fitting a model is proposed and illustrated via an analysis of failure data from a medium sized real-time command and control software system.

KEY WORDS: Failure count models, Fault seeding, Model fitting, NHPP, Software reliability, Times between failures.

[GOLD86] Goldberg, "Knowledge-Based Programming: A Survey of Program Design and Construction Techniques," IEEE Transactions on Software Engineering, pp. 752-768, July 1986.

Discusses an approach for improving the efficiency of software development which is the construction of a knowledge-based software assistant. This approach provides the medium of interaction for the development process and enforces the semantic consistency of the evolving program. The knowledge and formalization of how to represent problem-domain objects with the data structure facilities of the language, how to implement various search techniques and others, and the efficiency of various programming constructions, etc. are surveyed. This is presented from the point of view of algorithm design and high-level program optimization. Techniques for data structure selection, procedural representation of logic assertions store-versus-compute, finite differencing, loop fusion, and algorithm design methods are discussed.

KEY WORDS: Knowledge-based software development, Program optimization, Program synthesis, Program transformation.

[GOODM81] Goodman, Sequin, "Hypertree: A Multiprocessor Interconnection Topology," IEEE Transactions on Computers, pp. 923-933, December 1981.

Describes a new interconnection topology for incrementally expansible multicomputer systems which combines expansibility of tree structures with the compactness of n-dimensional hypercube. The derivations of a family of such Hypertree structures are outlined and the basic properties (average path length, uniformity of distribution of message traffic and routing algorithms) are analyzed.

KEY WORDS: Communication networks, Hypercube, Message traffic, Routing algorithms, Tree structures.

[GOODW81] Goodwin, "Why Programming Environments Need Dynamic Data Types," IEEE Transactions on Software Engineering, pp. 451-457, September 1981.

Discusses a programming environments (PE's) need to use data types dynamically since it is their function to support the programmer in all phases of work with a program. If the language in which the PE is written permits types to be defined at runtime, the PE can accurately model the supported program's types as its own. Language EL1 shows how to combine strong typing and an emphasis on efficient compilation with the necessary dynamic properties to support good PE's.

KEY WORDS: Computer oriented language, Data abstraction, Dynamic defining of types, LISP, Programming environments.

[GOPA81] Gopal, Wong, "Delay Analysis of Broadcast Routing Packet-Switching Networks," IEEE Transactions on Computers, pp. 915-922, December 1981.

Defines broadcast addressing as the capability to send a packet from a source node to all other nodes and suggests broadcasting has to be implemented by a routing algorithm in in store-and-forward and packet-switching networks. A source based forwarding algorithm is considered. With this algorithm, a spanning tree is defined for each node, and broadcast packets are sent along the branches of these trees. Approximation methods are presented to obtain a lower bound and estimates of the mean broadcast time. The accuracy of these methods is evaluated by comparison with simulation.

KEY WORDS: Broadcast routing, Packet switching networks, Queueing analysis, Source based forwarding.

[GOUD81a] Gouda, "How to Design a Communication Protocol and not Worry About Synchronization," INWG Workshop "Protocol Testing--Towards Proof?", National Physical Laboratory, Teddington, England, May 1981.

Outlines three approaches: 1) an abstract specification approach; 2) a partial design approach; and 3) a complete design/error detection approach to help a designer to design correct communication protocols. In particular, these approaches help the designer to deal with the problems and issues of synchronization in any protocol he designs so that the resulting communication is both deadlock-free and bounded.

KEY WORDS: Protocol correctness, Synchronization, Deadlock free.

[GOUD81b] Gouda, Yu, "A Methodology to Design Deadlock-Free and Bounded Communication Protocols," INWG Workshop "Protocol Testing--Towards Proof?", National Physical Laboratory, Teddington, England, May 1981.

Presents a methodology to design deadlock-free and bounded communication protocols. The methodology consists of a model to specify communicating processes and two algorithms, one for generating processes from a process and the other for generating channels between the processes.

KEY WORDS: Deadlock free, Communication protocols, Communicating processes, State deadlocks.

[GOUD85] Gouda, "On 'A Simple Protocol Whose Proof Isn't': The State Machine Approach," IEEE Transactions on Communications, pp. 380-381, April 1985.

Discusses how to model a synchronous protocol using communication finite state machines, and presents a proof for its safety and liveness properties. The proof is based on constructing a labeled finite reachability graph for the protocol. This reachability graph can be viewed as a sequential program whose safety and liveness properties can be stated and verified in a straightforward fashion.

KEY WORDS: Synchronous protocol, Communicating finite state machines, Reachability graph.

[GREI77] Greif, "A Language for Formal Problem Specification," Communications of the ACM, pp. 931-935, December 1977.

Introduces a language for specifying the intended behavior of communicating parallel processes. The language is used to write specifications of the readers/writers problem, and the writer priority of the second readers/writers problem.

KEY WORDS: Formal specifications, Program correctness, Parallel processing, Synchronization.

[GRIE77] Gries, "An Exercise in Proving Parallel Programs Correct," Communications of the ACM, pp. 921-930, December 1977.

Proves Dijkstra's on-the-fly garbage collector, a parallel program, correct using a proof method developed by Owicki. Difficulties with proving such parallel programs correct are described.

KEY WORDS: Garbage collection, Multiprocessing, Program correctness for multiprocessing tasks.

[GRIE81] Gries, The Science of Programming, Springer-Verlag, New York, 1981.

Provides major text on formal verification using Dijkstra's notation and predicate transformer formalism. Many interesting examples and exercises are included. Only sequential processes are covered, although non-determinism is handled.

KEY WORDS: Verification, Sequential.

[GRIF85] Griffeth, Miller, "Performance Modeling of Database Recovery Protocols," IEEE Transactions on Software Engineering, pp. 564-571, June 1985.

Describes performance modeling which compares several protocols that ensure a database can be recovered to a consistent state after a transaction failure or system crash. A collection of simple analytic models, based on Markov processes, for these protocols and some surprising results on the relative performance of these protocols are included. Results of systems obeying the assumptions are outlined, the policy of holding write locks to commit points is shown to be considerably less efficient than the policy which allows reading of uncommitted data, but risks cascading aborts. A multiversion policy is also studied.

KEY WORDS: Atomic actions, Concurrency control, Markov processes, Queueing models, Reliability, Transaction systems.

[GUTT78] Guttag, Horowitz, and Musser, "Abstract Data Types and Software Validation," Communications of the ACM, pp. 1048-1063, December 1978.

Suggests that a data abstraction can be naturally specified using algebraic axioms. These axioms permit a representation-independent formal specification of a data type. Algebraic axioms are employed at successive levels of implementation. Algebraic axiomatizations are used to simplify the process of

proving the correctness of an implementation of an abstract data type. Semi-automatic tools to automate these proofs of correctness and to derive an immediate implementation from the axioms are described.

KEY WORDS: Abstract data types, Software validation, Algebraic axioms.

[GUTT80] Guttag, "Notes on Type Abstraction," IEEE Transactions on Software Engineering, pp. 13-23, January 1980.

Discusses, in general, the role of type abstraction and the need for formal specifications of types abstractions. Two approaches to the construction of such specifications (Hoare approach and a version of algebraic specifications) are examined.

KEY WORDS: Abstract data type, Correctness proof, Specification, Software specification.

[HAAS81] Haase, "Real-Time Behavior of Programs," IEEE Transactions on Software Engineering, pp. 494-501, September 1981.

Develops a method of checking the fulfillment of real-time constraints based on the concepts of 'guarded commands' and PARC's, and using formal means of predicate transformers, a method of checking the fulfillment of real-time constraints is developed. The method allows the calculation of execution times of both sequential and parallel programs, both in single and multiprocessor systems. The key issue is the introduction of real time as a variable into the data space of the program. Manipulation of this time variable during the execution of a program is determined by the program structure, input data, and by the hardware properties of processors and memories.

KEY WORDS: Deadline scheduling, Guarded commands, Parallel processes, Real-time programming.

[HAC85] Hac, "A System Reliability Model with Classes of Failures," IEEE Transactions on Reliability, pp. 29-33, April 1985.

Presents an approach to system reliability involving s-dependence of the workload as well as the system configuration. Four classes of failures are described and then incorporated into the workload model. Model allows multiple classes of users and priority requests to be represented. The model is validated using measurement data collected in an IBM installation.

KEY WORDS: Workload modeling, System failure, System operating mode, Measurement data.

[HAIL85] Hailpern, "A Simple Protocol Whose Proof Isn't," IEEE Transactions on Communications, pp. 330-337, April 1985.

Presents proofs of one of the protocols proposed by Aho, Ullman, and Yannakakis that ensure reliable transmission of data across an error-prone channel. The finite-state-machine approach and the abstract-program approach are used in the proofs. It is shown that the abstract-program approach gives insight into the operation of the protocol.

KEY WORDS: Protocols for error-prone channels, Finite state machine, Abstract program.

[HALS78] Halstead, Multiple-Processor Implementations of Message-Passing Systems, pp. 1-172, MIT, Laboratory for Computer Science, Cambridge, MA, April 1978. Report # MIT/LCS/TR-198.

Develops a methodology for building networks of small computers capable of the same tasks now performed by single larger computers. The author states that such networks promise to be both easier to scale and more economical in many instances. The mu calculus, a simple syntactic formalism for representing message-passing computations, is presented and augmented to serve as the semantic basis for programs running on the network. The network implementation presented supports object references, keeping track them by using a new concept, the reference tree. A reference tree is a group of neighboring processors in the network that share knowledge of a common object. Also discussed are mechanisms for handling side effects on objects and strategy issues involved in allocating computations to processors.

KEY WORDS: Message passing, Distributed computing, Actor semantics, Networks.

[HANS78a] Hansen, "Distributed Processes: A Concurrent Programming Concept," Communications of the ACM, pp. 934-941, November 1978.

Introduces a language concept for concurrent processes without common variables. These processes communicate and synchronize by means of procedure calls and guarded regions. This concept is proposed for real-time applications controlled by microcomputer networks with distributed storage. Several examples of distributed processes are given and it is shown that they include procedures, coroutines, classes, monitors, processes, semaphores, buffers, path expressions, and input/output as special cases.

KEY WORDS: Concurrent programming, Distributed processes, Process communication and scheduling, Microprocessor networks.

[HANS78b] Hansen, Staunstrup, Specification and Implementation of Mutual Exclusion, Computer Science Department, USC, March 1978. Report 78-55.

Presents a constructive approach to the problem of specifying, implementing, and verifying operations that will give concurrent processes exclusive access to a resource. The need for auxiliary variable is eliminated and the correctness of a whole class of solutions to the same problem is established. Solutions are derived directly from the specifications using a language construct called guarded regions. Several new solutions to well-known exclusion problems are presented.

KEY WORDS: Concurrent programs, Mutual exclusion, Program specification, Program implementation, Program verification, Guarded regions.

[HASS86] Hassan, Agarwal, "A Fault-Tolerant Modular Architecture for Binary Trees," IEEE Transactions on Computers, pp. 356-361, April 1986.

Proposes a new modular, fault-tolerant scheme for the binary tree architecture. The approach uses redundant modular fault-tolerant building blocks to construct the complete binary tree. The scheme is shown to be more reliable and easier to implement than existing fault-tolerant schemes.

KEY WORDS: Binary tree, Fault tolerant, Reconfiguration, Redundant, Reliability.

[HAYE85] Hayes, "Applying Formal Specification to Software Development in Industry," IEEE Transactions on Software Engineering, pp. 169-178, February 1985.

Reports experience gained in applying formal specification techniques to an existing transaction processing system (CICS). The work has concentrated on specifying a number of modules of CICS application programmer's interface. Uses of formal specification techniques with particular reference to their application to an existing piece of software are outlined. Problems dealing with questions about the system design and documentation during the specification process are discussed. Also problems with the specification techniques themselves when applied to a commercial transaction processing system are discussed.

KEY WORDS: CICS, Formal specification, Large scale software.

[HAYE86] Hayes, "Specification Directed Module Testing," IEEE Transactions on Software Engineering, pp. 124-133, January 1986.

Describes testing techniques that apply to the testing of abstract data types. These techniques are illustrated by application to the implementation of a symbol table as an ordered list and as a height balanced tree.

KEY WORDS: Abstract data types, Data type invariant, Software reliability, Specification language.

[HECH86] Hecht, Hecht, "Software Reliability in the System Context," IEEE Transactions on Software Engineering, pp. 51-58, January 1986.

Reviews software reliability experience during the Operations and Maintenance (O&M) phase of the life cycle. A basic failure model that supports a unified approach to software and hardware reliability is presented and the effect of management activities on reliability is discussed. A combined hardware/software reliability model is outlined.

KEY WORDS: Computer failure models, Computer system reliability, Software management, Software reliability.

[HEND86] Henderson, "Functional Programming, Formal Specification, and Rapid Prototyping," IEEE Transactions on Software Engineering, pp. 241-250, February 1986.

Argues that functional programs combine the clarity required for the formal specification of software design with the ability to validate the design by execution. As such, they are ideal for rapidly prototyping a design as it is developed. An example is given which is larger than those traditionally used to explain functional programming. A method

of software design which efficiently and reliably turns an informal description of requirements into an executable formal specification is illustrated.

KEY WORDS: Functional programming, Software design, Specification, Validation.

[HENI80] Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and Their Applications," IEEE Transactions on Software Engineering, pp. 2-12, January 1980.

Concerns new techniques for making requirements specifications precise, concise, unambiguous, and easy to check for completeness and consistency. These new techniques are well-suited for complex real-time software systems; they were developed to document the requirements of existing flight software for the Navy's A-7 aircraft. The information that belongs in a requirements document is outlined and the objectives behind the techniques are discussed. Each technique is described and illustrated with examples. The purpose is to introduce the A-7 document as a model of a disciplined approach to requirements specification.

KEY WORDS: Documentation techniques, Functional specifications, Real-time software, Requirements, Specifications.

[HENR81] Henry, Kafura, "Software Structure Metric Based on Information Overflow," IEEE Transactions on Software Engineering, pp. 510-518, September 1981.

Defines and validates a set of software metrics which are appropriate for evaluating the structure of large-scale systems. The metrics are based on the measurement of information flow between system components. The validation, using the source code for the UNIX operating system, shows that the complexity measure are strongly correlated with the occurrence of changes. Further, the metrics for procedures and modules can be interpreted to reveal various types of structural flaws in the design and implementation.

KEY WORDS: Design methodologies, Software metrics, UNIX.

[HEWI77] Hewitt, Baker, Actors and Continuous Functionals, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1977.

Reports on 'laws' that govern communicating parallel processes. These laws are used to analyze mechanisms used by processes to communicate. Partial orders are used to express casualties involved in parallel computations and the laws for distributed computation.

KEY WORDS: Parallel processes, Computational analysis of systems.

[HOAR71] Hoare, "Procedures and Parameters: An Axiomatic Approach," Symposium on Semantics of Algorithmic Languages, Lecture Notes in Mathematics 188, pp. 102-116, Berlin-Heidelberg-New York: Springer, 1971.

Presents an example of the application of the axiomatic method to the definition of procedure and parameter passing features

of a high-level programming language. The ease of demonstrating program correctness and high-efficiency of implementation may be achieved simultaneously, provided that the programmer is willing to observe a certain familiar and natural discipline in his rise of parameters.

KEY WORDS: Natural deduction, Axiomatic method, Program correctness.

[HOAR85] Hoare, Communicating Sequential Processes, Prentice-Hall, Englewood Cliffs, NJ, 1985.

Provides major text on formal verification of concurrent processes. Event-based descriptions of process behavior are used. The synchronous message-passing model is assumed.

KEY WORDS: Verification, Concurrent.

[HOLL87] Holliday, Vernon, "Exact Performance Estimates for Multiprocessor Memory and Bus Interference," IEEE Transactions on Computers, pp. 76-85, January 1987.

Gives exact results for the processing power in a multibus multiprocessor with constant memory cycle times and geometric interrequest times. Both uniform and nonuniform memory accesses are considered and Generalized Timed Petri Nets are introduced .

KEY WORDS: Markov models, Multiprocessors, Performance comparison and evaluation, Petri nets.

[HOLO80] Holober, A Survey of Synchronization Problems, pp. 1-50, Yale University, Department of Computer Science, New Haven, CT, September 1980. Technical Report # 181.

States that decreased hardware costs will facilitate the use of parallel computation in the near future. Synchronization primitives will be needed to implement concurrent algorithms. Many such primitives have been proposed to date. Unfortunately, their power can only be accurately measured in terms of their ability to solve a particular set of synchronization problems. A correspondingly large number of such problems have been proposed along with the primitives. Some of these synchronization problems are presented and their parameters, variations, and histories are outlined.

KEY WORDS: Synchronization primitives, Synchronization problems, Synchronization parameters and variations.

[HOLT80] Holt, BCSP: A Small Language for Parallel Processing, pp. 1-42, Office of Naval Research, Arlington, VA, February 1980. Report N00014-79-G-0041.

Describes BCSP which is based on Hoare's Communicating Sequential Processes, but uses tail recursion for loops and linked ports for communication. An abstract parallel machine is described that can be used to implement BCSP; the machine description assumes an arbitrary number of processors.

KEY WORDS: Programming languages, Parallel programming, Abstract machines.

[HOLZ81] Holzmann, "An Algebra for Protocol Validation," INWG Workshop "Protocol Testing--Towards Proof?", National Physical Laboratory, Teddington, England, May 1981.

Uses an algebraic method for an extended type of regular expressions to analyze message passing protocols on various kinds of design errors. The steps involve modeling the behavior of processes sending and receiving messages. The operations used are division for analyzing output behaviors and multiplication for combining expressions representing interactions. By using these, an algebra for the validation of message passing systems is built. Much of the analysis method can be automated. The method has been applied successfully to a number of life-size protocols, including two Data Switch protocols for respectively five and seven interacting processes.

KEY WORDS: Message passing protocols, Algebraic method, Finite state machines.

[HORO81] Horowitz, Zorat, "The Binary Tree as an Interconnection Network: Applications to Multiprocessor Systems and VLSI," IEEE Transactions on Computers, pp. 247-253, April 1981.

Examines several aspects of the binary tree structure as it relates to both multiprocessor systems and to VLSI circuit design. An algorithm for mapping an arbitrary binary tree onto the plane is presented. The problem of routing messages within a binary tree under the assumption that certain nodes may be faulty is considered. The binary trees capacity to transfer information between nodes is analyzed and compared to the capacity of the linear array and the grid.

KEY WORDS: Binary trees, Parallelism, VLSI, Multiprocessing, Networks.

[HOWD80] Howden, "Functional Program Testing," IEEE Transactions on Software Engineering, pp. 162-169, March 1980.

Describes an approach to functional testing in which the design of the program is viewed as an integrated collection of functions. The selection of test data depends on the functions used in the design and on the value spaces over which the functions are defined. The basic ideas in the method were developed during the study of a collection of scientific programs containing errors. It was found to be significantly more reliable than structural testing. The techniques are compared and their relative advantages and limitations are discussed.

KEY WORDS: Effectiveness, Reliability, Testing.

[HOWD81] Howden, "Errors, Design Properties and Functional Program Tests," in Computer Program Testing, North-Holland Publishing Company, 1981.

Describes errors, design properties, and functional testing in detail. Both the traditional approach to functional testing as well as recent work which is aimed at making the method more methodical are discussed.

KEY WORDS: Errors, Design properties, Functional program tests.

[HUAN80] Huang, "A New Verification Rule and its Application," IEEE Transactions on Software Engineering, pp. 480-484, September 1980.

Describes a verification rule for loop programs and shows how it can be used in conjunction with the invariant relation theorem to facilitate verification of programs.

KEY WORDS: Consistency, Decomposition, Invariant-relation theorem, Program verification.

[HUNG85] Hung, "CCS Used As a Proof-Assistant Tool," Proceedings of the IFIP WG 6.1 Fifth International Workshop on Protocol Specification, Testing and Verification, North-Holland Publishing Company, May 1985.

Uses a non-trivial arbitration protocol (APK) as an example to demonstrate how the CCS (Calculus of Communicating Systems) can be used as a possible theoretical background for finding invariants in proving some types of concurrent programs.

KEY WORDS: Concurrent processing, Correctness proof, APK, Communication tree, Invariants generation.

[HUTC85] Hutchens, Basili, "System Structure Analysis: Clustering with Data Bindings," IEEE Transactions on Software Engineering, pp. 749-757, August 1985.

Examines the use of cluster analysis as a tool for system modularization. Several clustering techniques are discussed and used on two medium-size systems and a group of small projects. Data bindings between the routines of the system provide the basis for the bindings. It appears that the clustering of data bindings provides a meaningful view of system modularization.

KEY WORDS: Cluster, Coupling, Data binding, Measurement, System structure.

[IBAR81] Ibaraki, Kameda, and Toida, "On Minimal Test Sets for Locating Single Link Failures in Networks," IEEE Transactions on Computers, pp. 182-189, March 1981.

Considers a network which can be represented by an acyclic directed graph such that the links represented by the edges are subject to fail. Under the assumption that at most one link can fail at any time, the desire is to locate a failed link, if any, by means of certain tests. A test is performed by injecting a signal at a vertex and monitoring it at another vertex and can reveal if there is a failed link on any path between the two vertices. The objective is to find a minimal set of tests that can uniquely locate a single fault. Since this problem is, in general, NP-complete, a special case is investigated where the given network has a tree structure. An algorithm whose worst case running time can be bounded by a linear function of the input size is presented.

KEY WORDS: Complexity, Network diagnosis, NP-complete.

[IBAR82] Ibaraki, Kameda, "Deadlock-Free Systems for a Bounded Number of Processes," IEEE Transactions on Computers, pp. 188-193, March 1982.

Considers a computer system in which different types of serially reusable resources are shared by several classes of processes. It is assumed that each process in a process class has the same known maximum claim but actual sequence of requests is unknown. The resource manager uses the 'expediency policy' in granting requests for resources where at most 'k' processes can be in the system at any time. It is shown for such a system that: 1) whether this can deadlock with the given number of resource units can be tested in polynomial time; 2) the problem of designing a deadlock free system with minimum resource cost is NP-complete; and 3) the largest k for which it is deadlock-free for the given number of resource units can be found in polynomial time. It is also shown that problems 1) and 3) become NP-complete if a certain additional constraint is to be satisfied.

KEY WORDS: Algorithm complexity, Deadlock prevention.

[IEEE85] Draft IEEE Standard P982 Software Reliability Measurement, IEEE, June 1, 1985.

Describes the major reliability metrics and the phases of the software life cycle in which they can be used. Both design metrics for source code complexity and statistical models for use with error data are included.

KEY WORDS: Metrics.

[IGAR75] Igarashi, "Automatic Program Verification I: A Logical Basis and its Implementation," Acta Informatica, pp. 145-182, Springer-Verlag, 1975.

Reports on proving that a program meets its specifications by using axioms and rules of inference. This deduction system is consistent with one developed by Hoare. The input is a subset of Pascal programs and the output gives verifications conditions to be proved.

KEY WORDS: Specifications, Verification conditions.

[IYER85] Iyer, Rossetti, "Effect of System Workload on Operating System Reliability: A Study on IBM 3081," IEEE Transactions on Software Engineering, pp. 1438-1448, December 1985.

Presents an analysis of operating system failures on the IBM 3081 running VM/SP. Three categories of software failures: error handling; program control or logic; hardware related are discussed. Possible reasons for the observed workload failure dependency based on detailed investigations of the failure data are given.

KEY WORDS: Failure analysis, Software reliability, System workload, VM/SP.

[IYER86] Iyer, Donatiello, and Heidelberger, "Analysis of Performability for Stochastic Models of Fault-Tolerant Systems," IEEE Transaction on Computers, pp. 902-906, October 1986.

Suggests performability, a composite measure for the performance and reliability, may be interpreted as the probability density function of the aggregate reward obtained from a system during its mission time. For large mission times it is shown that known limit theorems lead to an asymptotic normal distribution for the aggregate reward. For finite mission times and Markovian models the expressions for all moments of performability are obtained and recursions to compute coefficients involved in the expressions are given. The use of the results is illustrated through an example of a multiple processor computer system.

KEY WORDS: Analysis, Markov chains, Moments, Reliability.

[JACO86] Jacob, Survey and Examples of Specification Techniques for User-Computer Interfaces, Computer Science and System Branch, Naval Research Lab, Washington, D.C., April 1986. Report # NRL-8948.

Provides a survey of techniques suitable for specifying the user interface of a system and a detailed discussion of relevant literature. A collection of examples of the application of several representative specification techniques to a common set of examples in order to compare the relative merits of the technique is presented.

KEY WORDS: User interface, Specification techniques.

[JAHA86] Jahanian, Mok, "Safety Analysis of Timing Properties in Real-Time Systems," IEEE Transactions on Software Engineering, pp. 890-904, September 1986.

Formalizes the safety analysis of timing properties in real-time systems. Analysis is based on RTL (Real-Time Logic). Given the formal specification of a system and a safety assertion to be analyzed, the goal is to relate the safety assertion to the system's specification. There are three cases: 1) safety assertion is a theorem derivable from the system specification; 2) safety assertion is unsatisfiable with respect to the system's specification; 3) the negation of the safety assertion is satisfiable under certain conditions. A systematic method for performing safety analysis is presented.

KEY WORDS: Real-time logic, Safety analysis systems specification, Time-critical system, Verification.

[JALO86] Jalote, Campbell, "Atomic Actions for Fault-Tolerance Using CSP (Communicating Sequential Processes)," IEEE Transactions on Software Engineering, pp. 59-68, January 1986.

Proposes a mechanism for supporting an atomic action in a system of CSP. The atomic action is used as the basic unit for providing fault-tolerance. The atomic action is called an FT-Action and both forward and backward error recovery are performed in the context of an FT-Action. An implementation for the FT-action is proposed, which employs a distributed control, uses CSP primitives, and supports local compile and

run-time checking of the forward and backward error recovery schemes.

KEY WORDS: Atomic actions, Backward recovery, Forward recovery, Software fault-tolerance.

[JARD85] Jard, Monin, and Groz, "Experience in Implementing X.250 (a CCITT subset of Estelle) in Veda," Proceedings of the IFIP WG 6.1 Fifth International Workshop on Protocol Specification, Testing and Verification, North-Holland Publishing Company, May 1985.

Describes Veda as a tool for the verification of protocol specifications by simulation. Protocols described in a Formal Description Technique (FDT) based on extended State Transition Machines are compiled into a code oriented towards simulation and verification. After a brief summary of Veda and its environment, the concentration is on problems solved in implementing that kind of FDT.

KEY WORDS: Verification, Protocol specification, Formal description technique, Extended state transition machines.

[JEWE85] Jewell, "Bayesian Extensions to a Basic Model of Software Reliability," IEEE Transactions on Software Engineering, pp. 1465-1471, December 1985.

Gives a Bayesian Analysis of the software reliability model of Jelinski and Moranda, based on Meinhold and Singpurwalla. Important extensions are provided to the stopping rule and prior distribution of the number of defects, as well as permitting uncertainty in the failure rate. The author considers it easy to calculate the predictive distribution of unfound errors at the end of software testing, and to see the relative effects of uncertainty in the number of errors in the detection efficiency. The behavior of the predictive mode and mean over time are examined as possible point estimators, but are clearly inferior to calculating the full predictive distribution.

KEY WORDS: Bayesian Analysis, Program testing, Software reliability.

[JOHN86] Johnson, Aylor, "Reliability and Safety Analysis of a Fault-Tolerant Controller," IEEE Transactions on Reliability, pp. 355-362, October 1986.

Analyzes a fault-tolerant, microprocessor-based controller for an electric wheelchair. Two candidate architectures are considered, including reconfigurable duplication and standby sparing. The difference in the reliability and safety is determined through the use of Markov models. Safety is paramount in the wheelchair application because of the need to protect the physically disabled wheelchair user; reliability by itself is insufficient for selecting an appropriate architecture in this application. Results show that reconfigurable duplication is safer than standby sparing even though standby sparing is more reliable.

KEY WORDS: Safety, Reconfigurable duplication, Standby sparing.

[KANT81] Kant, Barstow, "The Refinement Paradigm: The Interaction of Coding and Efficiency Knowledge in Program Synthesis," IEEE Transactions on Software Engineering, pp. 458-471, September 1981.

Discusses a refinement paradigm for implementing a high-level specification in a low-level target language. Coding and analysis knowledge work together to produce an efficient program in the target language. A particular implementation of this program synthesis (PSI/SYN) that has automatically implemented a number of programs in the domain of symbolic processing is described.

KEY WORDS: Automatic programming, Program synthesis, Refinement, Stepwise refinement.

[KANT85] Kant, "Understanding and Automating Algorithm Design," IEEE Transactions on Software Engineering, pp. 1361-1374, November 1985.

Reports on human and automatic design forming a theory on algorithm design. The amount of space used, problem solving capabilities, and relationships between methods of algorithm design and between it and automatic programming are elaborated on.

KEY WORDS: Automatic programming, Automating algorithm design, Human problem solving, Protocol analysis.

[KART79] Kartashev, Kartashev, "A Multicomputer System with Dynamic Architecture," IEEE Transactions on Computers, pp. 704-720, October 1979.

Considers the organization of certain multicomputer systems with a particular type of dynamic architecture. The system allows one to reconfigure via software available hardware resources, forming computers with different word sizes. A multicomputer system is formed from identical dynamic computer groups. Basic principles of dynamic computer group implementation are discussed.

KEY WORDS: Dynamic architecture, Dynamic computer group, Multicomputer.

[KARU79] Karunanithi, "Analysis of Digital Systems Using a New Measure of System Diagnosis," IEEE Transactions on Computers, pp. 121-132, February 1979.

Uses a new measure of system diagnosis, t/s diagnosability, to study the diagnosability of digital systems. This new measure incorporates the concept of possible replacement of fault-free units in system repair, whereas the previous measures have only considered the replacement of faulty units. Two categories of the new measure, one-step t/s diagnosability and sequential t/s diagnosability, are investigated. Finally, all the diagnosis strategies are compared and the tradeoff between the number of units replaced and the number of test iterations performed is discussed.

KEY WORDS: Digital systems, Optimal systems design, Self-diagnosis, System-level diagnosis, Single-loop systems.

[KELL76] Keller, "Formal Verification of Parallel Programs," Communications of the ACM, p. 371, July 1976.

Presents two formal models for parallel computation: an abstract conceptual model and a parallel-program model. An induction principle is described which treats the control and data state sets on the same ground. Examples are presented in which the inductive principle is used to demonstrate proofs of mutual exclusion. An extension of the program model which allows each process to have its own local variables and permits shared global variables, and correctness of certain forms of implementation are discussed.

KEY WORDS: Parallel program, Correctness, Verification, Deadlock, Mutual exclusion, Petri net.

[KELL85] Keller, Wilkins, "On the Use of an Extended Relational Model to Handle Changing Incomplete Information," IEEE Transactions on Software Engineering, pp. 620-633, July 1985.

Considers approaches to updating databases containing null values and incomplete information and distinguishes between modeling incompletely known worlds and modeling changes in these worlds. The expanded closed world assumption is proposed and how to perform updates on databases containing set nulls, marked nulls, and simple conditional tuples are discussed. Some issues of refining incompletely specified information are addressed.

KEY WORDS: Relational databases, Incomplete information, Updates.

[KEMM85] Kemmerer, "Testing Formal Specifications to Detect Design Errors," IEEE Transactions on Software Engineering, pp. 32-42, January 1985.

Discusses why it is necessary to test specifications early in the software life cycle to guarantee a system that meets its critical requirements and also provides the desired functionality. Definitions to provide the framework for classifying the validity of a functional requirement with respect to a formal specification and the design of two tools for testing formal specifications are included.

KEY WORDS: Design and development, Formal verification, Reliable software, Requirements, Specifications.

[KENE86] Kenett, Pollak, "A Semi-Parametric Approach to Testing for Reliability Growth, with Application to Software Systems," IEEE Transactions on Reliability, pp. 304-311, August 1986.

Considers the following general model for reliability growth: the distribution of times between failures belongs to a known parametric family, and the parameter corresponding to the distribution of a particular time between failures is either an unknown constant or an unobservable random variable with a (possibly unknown) distribution which can depend on past observations. It is proposed that acceptable reliability can sometimes be formalized as a state in which the value of the parameters is lower than a level set before testing begins.

Sequential detection methodology is applied to the problem of ascertaining that an acceptable state of reliability has been attained and the approach is illustrated by applying it to testing for reliability growth of a software system, using actual data.

KEY WORDS: Failure rate, Sequential detection scheme, False detection rate.

[KESS77] Kessels, "An Alternative to Event Queues for Synchronization in Monitors," Communications of the ACM, pp. 500-503, July 1977.

Describes a synchronizing primitive which is nearly as expressive as the conditional wait, but can be implemented more efficiently. An implementation of this primitive in terms of P and V operations is given together with a correctness proof. Two examples are presented: the readers and writers problem and the problem of information streams sharing a finite buffer pool.

KEY WORDS: Monitor, Mutual exclusion, Synchronization, Conditional critical region.

[KESS81] Kessels, "The Soma: A Programming Construct for Distributed Processing," IEEE Transaction on Software Engineering, pp. 502-509, September 1981.

Proposes for parallel programming a Soma (Software Machine) which is a sequential process that communicates with other somas by exchanging messages via mailboxes. It is well-suited for implementation on conventional as well as on distributed computer architectures. A rationale is given for its communication characteristics and comparison and examples are included.

KEY WORDS: Concurrency, Message passing, Synchronization.

[KIM84] Kim, "Highly Available Systems for Database Applications," ACM Computing Surveys, pp. 71-98, March 1984.

Presents an overview of fault tolerant database systems, including discussion of the major commercial f/t systems (Tandem, Synapse, and others). Emphasis is placed on architectural issues. A description of IBM's Highly Available Systems research project is included.

KEY WORDS: Fault tolerance, Reliability, Database concurrency control and recovery, Relational database.

[KING80] King, "Program Correctness: On Inductive Assertion Methods," IEEE Transactions on Software Engineering, pp. 465-479, September 1980.

Studies several of the proof of correctness methods. In particular, the form of induction used is explored in detail. A relational semantic model for programming languages is introduced and its relation to predicate transformers is explored. A rather elementary viewpoint is taken in order to expose, as simply as possible, the basic differences of the methods and the underlying principles involved. These results were obtained by attempting to thoroughly understand the "subgoal induction" method.

KEY WORDS: Correctness assertions, Predicate transformers, Program correctness, Relational semantics, Subgoal induction.

[KLIG86] Kligerman, Stoyenko, "Real-Time Euclid: A Language for Reliable Real-Time Systems," IEEE Transactions on Software Engineering, pp. 941-949, September 1986.

Introduces Real-Time Euclid, a language designed specifically to address reliability and guaranteed schedulability issues in real-time systems. Real-Time Euclid employs exception handlers and import/export lists to provide comprehensive error detection, isolation, and recovery. Philosophy of the language is that every exception detectable by the hardware or the software must have an exception handler clause with it. The language definition forces every construct in the language to be time- and space-bounded.

KEY WORDS: Exception handling, Guaranteed response time, Real-time systems, Software reliability.

[KLUG82] Kluge, Lautenbach, "The Orderly Resolution of Memory Access Conflicts Among Competing Channel Processes," IEEE Transactions on Computers, pp. 194-207, March 1982.

Presents an abstract regulation scheme for the orderly resolution of multiple conflicts in dynamic systems, in which the comparable 'claims' of the conflicting parties can precisely be formulated. The scheme is applied to resolve multiple memory access conflicts among several concurrently operating high speed channel processors.

KEY WORDS: Channel processors, Memory access conflicts, Petri nets, Synchronic distance.

[KNIG86] Knight, Leveson, "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming," IEEE Transactions on Software Engineering, pp. 96-109, January 1986.

Presents hypothesis that programs that have been developed independently will fail independently. The experiment to test this and the results of the test are included. The conclusion is that N-version programming must be used with care and analysis of its reliability must include the effect of dependent errors.

KEY WORDS: Design diversity, Fault-tolerant software, Multiversion programming, Software reliability.

[KOHL81] Kohler, "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," ACM Computing Surveys, pp. 149-184, June 1981.

Surveys two related and fundamental problems in designing decentralized systems which support an object model of computation and proposed solution techniques. The first problem is synchronizing access to shared objects while allowing a high degree of concurrency. The second problem is the recovery of objects in spite of user errors, application errors, or partial system failure. Requirements and techniques for

implementing atomic actions in a decentralized environment are discussed.

KEY WORDS: Decentralized system, Access synchronization, Concurrency control, Crash recovery, Atomic action.

[KORE79] Koren, Su, "Reliability Analysis of N-Modular Redundancy (NMR) Systems with Intermittent and Permanent Faults," IEEE Transactions on Computers, pp. 514-520, July 1979.

Presents a statistical model for intermittent faults and uses it to analyze the reliability of NMR systems in mixed intermittent and permanent fault environments.

KEY WORDS: Intermittent fault, Modular redundancy, Permanent fault, Reliability.

[KORN79] Kornfeld, Using Parallel Processing for Problem Solving, Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA, December 1979. A.I. Memo No. 561.

Develops parallel processing as a conceptual aid in the design of programs for problem solving applications. A pattern-directed invocation language known as Ether is introduced. Ether embodies two notions in language design: activities and view-points. Activities are used for different goals of the system. Viewpoints are used for representing multiple world models. A number of problem solving schemes are developed making use of viewpoints and activities. The use of Ether helps eliminate deadlock and race condition.

KEY WORDS: Parallel processing, Distributed computation, Pattern-directed invocation, Problem solving.

[KRAM78] Kramer, Cummingham, "Towards A Notation for the Functional Design of Distributed Systems," Proceedings of the 1978 International Conference on Parallel Processing , pp. 69-76, IEEE, August 22-25, 1978.

Suggests an approach to the design of distributed processing systems. The design process promotes structural decomposition of the system corresponding to a factorisation of its specification. Designs include consistency checks at the system and module levels. The approach is based on the informal use of verifying assertions in a programming notation. The assertions provide a useful mechanism for the description and analysis of system properties.

KEY WORDS: Distributed processing systems, Finite state machines, Resource sharing.

[KRAM85] Kramer, Magee, "Dynamic Configuration for Distributed Systems," IEEE Transactions on Software Engineering, pp. 424-436, April 1985.

Introduces a model of the configuration process which permits dynamic incremental modification and extension. This model is used to determine the properties required by languages and their execution environments to support dynamic configuration. CONIC, the distributed system with has been developed at Imperial College with the specific objective of supporting dynamic

configuration is described to illustrate the feasibility of the model.

KEY WORDS: Configuration process, Configuration specification, Distributed systems, Flexibility, Reusability, System evolution.

[KROL86] Krol, "(N,K) Concept Fault Tolerance," IEEE Transactions on Computers, pp. 339-349, April 1986.

Describes a new fault-tolerant computer architecture based on a 'distributed implementation' of a symbol-error correcting code. The (N,K) concept is described in detail for N = 4 and K = 2. In order to cope with unreliable input devices, the interactive consistence problem is defined and an algorithm that solves the problem is presented. Practical implementation of this algorithm is the (4,2) concept described.

KEY WORDS: Consistency, Error-correcting codes, Fault tolerance, Hardware redundancy.

[KUMA80] Kumar, Davidson, "Computer System Design Using Hierarchical Approach to Performance Evaluation," Communications of the ACM, pp. 511-521, September 1980.

Introduces a hierarchy of performance models. In order for hierarchy to be a cost-effective tool in the design of computer systems, it should consist of models spanning a wide range of accuracy and cost. Judicious use of the hierarchy can satisfy the conflicting needs of high accuracy and low cost of performance evaluation. A system design procedure that uses such a hierarchy is developed and illustrated by applying concepts to a case study of system design. Results of optimizations and a simple cost model are discussed. The optimization procedure converges to a region very close to a locally optimum system. The efficiency of this procedure is considerably greater than the brute force approach.

KEY WORDS: Hierarchical modeling, Performance evaluation, System design, Optimization algorithms.

[KUMA86] Kumar, Hariri, and Raghavendra, "Distributed Program Reliability Analysis," IEEE Transactions on Software Engineering, pp. 42-50, January 1986.

Uses two reliability measures: 1) distributed program reliability; and 2) distributed system reliability to accurately model the reliability of distributed systems. Graph theory techniques are applied to systematically generate file spanning trees that provide all required connections.

KEY WORDS: Distributed program, Distributed system, Reliability, Spanning tree.

[LADN79] Ladner, "The Complexities of Problems in Systems of Communicating Sequential Processes," Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, pp. 214-223, Atlanta, GA, 1979.

Investigates the belief that systems of communicating sequential processes are harder to analyze than purely sequential processes. The belief is largely based on the observation that the parallelism in such systems leads to a large number

of possible interleavings of the actions of the different processes. This report supports the idea that the properties of communicating processes are intrinsically complex. Specifically, the concepts of potential deadlock and lockout are explained.

KEY WORDS: Parallelism, Lockout, Finite state processes, Deadlock, Critical section.

[LAMP76] Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," Communications of the ACM, pp. 558-564, July 1976.

Examines the concept of one event happening before another in a distributed system and shows how to define a partial ordering of events. A distributing algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The algorithm is specialized for synchronizing physical clocks and a bound is derived on how far out of synchrony clocks can become.

KEY WORDS: Distributed systems, Computer networks, Clock synchronization, Multiprocess systems.

[LAMP77] Lamport, "Concurrent Reading and Writing," Communications of the ACM, pp. 806-811, November 1977.

Investigates sharing data among asynchronous processes. Emphasis is placed on theorems and algorithms including repeated reads and techniques for transmitting messages between processes.

KEY WORDS: Asynchronous multiprocessing, Multiprocess synchronization, Shared data.

[LEE82] Lee, Shin, "Rollback Propagation Detection and Performance Evaluation of FTMR2M--A Fault-Tolerant Multiprocessor," Proceedings on the 9th Annual Symposium on Computer Architecture, p. 171, April 1982.

Considers the rollback propagation and the performance of a fault-tolerant multiprocessor with a callback recovery mechanism (FTMR2M), which was designed to be tolerant of hardware failure with minimum time overhead. Approaches for evaluating the recovery overhead and analyzing the performances of FTMR2M are presented. Two methods for detecting rollback propagations and multi-step rollbacks between cooperating processes are also proposed.

KEY WORDS: Fault-tolerant, Rollback recovery, Cooperating processes, Reliability.

[LEMO80] Le Moli, "A Model for the Formal Description of Entities Which Perform Protocols," Compunet, CREI-Politecnico di Mialano, Piazza Leonardo Da Vinci, 7 Milano, Italy, 1980.

Presents a model of entities that performs protocols. This report is part of a series of papers on the same subject. The steps used in developing the model are standardizing, formalizing and implementing.

KEY WORDS: Standardization, Formalization methods.

[LESS80] Lesser, Erman, "Problems of Designing Supersystems with Dynamic Architectures," IEEE Transactions on Computers, pp. 1114-1126, December 1980.

Considers basic concepts of designing a Supersystem with dynamic architecture. The system is provided with two new sources of throughput increase: 1) by redistributing hardware resources it may maximize the number of parallel information streams handled by the available sources; 2) by reconfiguring resources into different types of architecture it may minimize the idle time and complexities of resources. System is assembled from dynamic computer groups with memory-processor bus that introduces minimal delay in communication between any two modules. The system reconfiguration from one architecture to another is studied and how such reconfiguration may be controlled by a system monitor is described.

KEY WORDS: Dynamic architecture, Powerful parallel system, Reconfigurable hardware resource, Reconfigurable memory processor.

[LEUN80] Leung, Ramamoorthy, "An Approach to Formal Specification of Control Modules," IEEE Transactions on Software Engineering, pp. 485-488, September 1980.

Discusses formal specification of program modules which control access to resources shared among concurrent processes. The concept of state space is defined for such program modules and the formal specification is given in terms of a program module invariant and input-output assertions defined on the state space. Examples are provided to illustrate the construction of specifications with this approach.

KEY WORDS: Concurrent processes, Invariant assertion, Control modules, Program specification.

[LICH86] Lichtman, "Generation and Consistency Checking of Design and Program Structures," IEEE Transactions on Software Engineering, pp. 172-181, January 1986.

Describes a mini methodology for generation and representation of design and program structures, and structural consistency checking between two successive designs or between a design and a program.

KEY WORDS: Consistency checking, Program Design Language, Software development, Software quality assurance.

[LIES86] Liestman, Campbell, "A Fault-Tolerant Scheduling Problem," IEEE Transactions on Software Engineering, pp. 1089-1095, November 1986.

Discusses a deadline mechanism which has been proposed to provide fault tolerance in real-time software systems. The mechanism trades the accuracy of the results of a service for timing precision. Two independent algorithms are provided for each service subject to a deadline. An algorithm to generate an optimal schedule for the deadline mechanism is introduced and a simple and efficient implementation is discussed. The schedule ensures the timely completion of the alternate algorithm despite a failure to complete the primary algorithm within real time.

KEY WORDS: Real-time systems, Scheduling, Software fault tolerance, Software reliability.

[LIN83] Lin, Liu, and Graff, "Verification of a Methodology for Designing Reliable Communication Protocols," Proceedings of the Eighth Data Communications Symposium, pp. 141-149, IEEE Computer Society Press, Silver Spring, MD, October 3-6, 1983.

Presents a new methodology for designing reliable communication protocols. This methodology enhances communicating processes with a synchronization mechanism so that they can detect and resolve the errors caused by collisions automatically. Also the application of a program verification technique to this methodology is discussed.

KEY WORDS: Communication protocols, Completeness, Synchronization.

[LIN86] Lin, Wu, "Reconfiguration Procedures for a Polymorphic and Partitionable Multiprocessor," IEEE Transactions on Computers, pp. 910-916, October 1986.

Presents a collection of reconfiguration procedures for a multiprocessor which employs multistage interconnection networks. These procedures are used to dynamically partition the multiprocessor into many subsystems, and reconfigure them to form a variety of commonly used topologies to match task graphs. With these procedures, a subsystem can be reconfigured in the form of the desired topologies without interfering with other subsystems. In addition, the reconfiguration of a subsystem can be accomplished in constant time, independently of subsystem size.

KEY WORDS: Parallel processing, Reconfigurable multiprocessors, Interconnection networks, Circuit switching, Connection conflicts.

[LING79] Linger, Mills, and Witt, Structured Programming, Theory and Practice, Addison-Wesley, 1979.

Describes functional semantics approach to formal verification of sequential deterministic processes. Many examples are included. The approach described in this text has been used as the basis of IBM's Software Engineering Institute training program.

KEY WORDS: Verification, Sequential.

[LIPT73] Lipton, On Synchronization Primitive Systems, Carnegie-Mellon University, 1973. Doctoral Thesis.

Presents a formal model of a process concept. The model is used to implement four synchronization primitives and the results are compared. The model includes conditional branches and a scheduler in order to better see the differences of the primitives.

KEY WORDS: Synchronization, Primitive process concept, PV, Interprocess communication.

[LIPT75] Lipton, "Reduction: A Method of Proving Properties of Parallel Programs," Communications of the ACM pp. 717-721, ACM, December 1975.

Indicates that when proving a parallel program has a given property, it is often convenient to assume that a statement is indivisible, i.e., the statement cannot be interleaved with the rest of the program. Sufficient conditions are obtained to show that the assumption that a statement is indivisible can be relaxed and still preserve properties such as halting.

KEY WORDS: Deadlock free, Reduction, Interruptible parallel program, Verification method.

[LISK81] Liskov, Guardians and Actions: Linguistic Support for Robust, Distributed Programs, pp. 1-30, Computation Structures Group, MIT, Cambridge, MA, November 1981. Memo 210-1.

Presents an overview of an integrated programming language and system designed to support the construction and maintenance of distributed programs--programs in which modules reside and execute at communicating, but geographically distinct nodes. The language addresses the writing of robust programs that survive hardware failures without loss of distributed information and that provide highly concurrent access to that information while preserving consistency. Several new linguistic constructs are provided; among them are atomic actions, and modules called guardians that survive node failures.

KEY WORDS: Integrated programming language, Distributed programs, Robust programs.

[LITT80] Littlewood, "Theories of Software Reliability: How Good Are They and How Can They Be Improved," IEEE Transactions on Software Engineering, pp. 489-500, September 1980.

Suggests ways to improve modeling assumptions with examples of mathematical implementations. Model verification via real-life data is discussed and minimum requirements are presented. An example shows how these requirements may be satisfied in practice.

KEY WORDS: Program error, Reliability growth, Software failure, Software life-cycle cost, Software reliability measurement.

[LOCK85] Locks, "Recent Developments in Computing of System-Reliability," IEEE Transactions on Reliability, pp. 425-436, December 1985.

Presents the following: 1) The inclusion-exclusion (IE), sum of disjoint products (SDP), and topological reliability (TR) algorithms all have the exponential-time property. 2) IE or SDP results in a system reliability formula which is a sum of products of the probabilities of the components. 3) In an m-out-of-n system, by ordering the paths so that each path differs from its predecessor by exactly one component, the SDP probability formula has the same number of terms as there are paths. 4) Certain proofs are provided for SDP. 5) Direct relationship between the IE and TR formulas is described.

KEY WORDS: System reliability, Inclusion-exclusion, Topological reliability, m-out-of-n system, Source-to-multiple terminal reliability.

[LU78] Lu, "Error-Correcting Tree Automata for Syntactic Pattern Recognition," IEEE Transactions on Computers, pp. 1040-1053, November 1978.

Defines the syntax error on trees in terms of five types of error transformations: substitution, stretch, split, branch and deletion. The distance between two trees is the least cost sequence of error transformations needed to transform one to the other. Based on this, a class of error-correcting tree automata (ECTA) is proposed. The proposal is illustrated by a character recognition example.

KEY WORDS: Character recognition, Error transformation, Pattern recognition, Syntactic pattern recognition.

[LUCE79] Lucena, Pequeno, "Program Derivation Using Data Types: A Case Study," IEEE Transactions on Software Engineering, pp. 586-592, November 1979.

Discusses some issues in program synthesis by relating the idea of systematic program derivation with the concepts of data type and correctness of data representation. The notion of an incomplete definition of a data type at a high level of abstraction is introduced. The ideas are illustrated through an example.

KEY WORDS: Correctness of data representation, Program derivation, Program schema, Program specification, Program synthesis.

[MA82] Ma, Lee, and Tsuchiya, "A Task Allocation Model for Distributed Computing Systems," IEEE Transactions on Computers, pp. 41-47, January 1982.

Presents a task allocation model that allocates application task among processors in distributed computing systems satisfying: 1) minimum interprocessor communication cost; 2) balanced utilization; 3) all engineering application requirements. A cost function is formulated to measure the interprocessor communication and processing costs. Model was applied to an Air Defense case study. Results indicate the allocation model is applicable to large practical problems.

KEY WORDS: Branch and bound, Distributed processing, Interprocess communication, Task allocation.

[MALL78] Mallela, "Diagnosable Systems for Intermittent Faults," IEEE Transactions on Computers, pp. 560-566, June 1978.

Studies the intermittent fault diagnosis capabilities of systems composed of interconnected units which are capable of testing each other. Necessary and sufficient conditions are derived and bounds are established for some well-known systems. In contrast to permanent fault diagnosable systems, there exists only a single type on intermittent fault diagnosable system. A procedure is given to determine the intermittent fault diagnosability of any given system.

KEY WORDS: Incomplete diagnosis, Incorrect diagnosis, Intermittent faults, Self-diagnosable system, Syndrome.

[MANC86] Mancini, "Modular Redundancy in a Message Passing System," IEEE Transactions on Software Engineering, pp. 79-86, January 1986.

Presents modular redundancy in the form of replicated computations in a concurrent programming model consisting of communicating sequential processes. Conditions which must always be verified to ensure correctness in the presence of nondeterminism are presented. Implementations which satisfy the given conditions are discussed.

KEY WORDS: Communicating sequential processes, Fault tolerance, Guarded commands, Nondeterminism, Replicated processing.

[MAO80] Mao, Yeh, "Communication Port: A Language Concept for Concurrent Programming," IEEE Transactions on Software Engineering, pp. 194-204, March 1980.

Introduces a new language concept--Communication Port (CP), for programming on distributed processor networks. These networks can contain an arbitrary number of processors each with its own private storage but with no memory sharing. Processors communicate via explicit message passing. CP is an encapsulation of two language properties: "communication nondeterminism" and "communication disconnect time." CP provides a tool to write well-structured modular, and efficient concurrent programs. A number of examples are given to demonstrate the power of the new concepts.

KEY WORDS: Communicating sequential processes, Communication ports, Concurrent programming, Distributed networks, Nondeterminism.

[MART86] Martin, De Millo, "Operational Survivability in Gracefully Degrading Distributed Processing Systems," IEEE Transactions on Software Engineering, pp. 693-704, June 1986.

Discusses the use of experimental methods and statistical analysis techniques to study factors influencing operational survivability in gracefully degrading systems. Survivability data generated using a statistically designed experiment in conjunction with a simulation model of network survivability are presented. Models that are acceptable from both an estimation and prediction viewpoint are also presented. Possible commercial and military applications are suggested.

KEY WORDS: Distributed processing system, Graceful degradation, Operational survivability.

[MARX81] Marxen, Muller-Zimmerman, and Schindler, "The OSA Project: The RSPL-Z Compiler," International Computer Conference '81, Denver, CO, 1981.

Explains that RSPL stands for "Reliable Software Production Language" and should support producing reliable software, at least in the area of communications software. There are two

main reasons for this: 1) RSPL is a formal specification language for which the pragmatic, semantic and syntactic elements are designed such as to make RSPL protocol/service specifications concise and clear; and 2) RSPL bridges the gap between formal specification and implementation: As far as protocol specifications are concerned, it is automatically compilable into executable code and therefore it is an implementation language, as well.

KEY WORDS: Reliable software, Specification language.

[MCKE85] McKendry, "Ordering Actions for Visibility," IEEE Transactions on Software Engineering, pp. 509-519, June 1985.

Explains that when concerned with synchronization to control ordering, a function is often associated with objects. An example to illustrate requirements for the ordering mechanism is presented. A model of nest actions is used as a basis for categorizing visibility requirements. Several expediencies that result from ordering requirements are established and the difficulty and potential cost of providing generalized mechanisms are illustrated. In many situations, a single synchronization variable can be used to control blocking, and recovery for nested actions can be implemented with a single backup copy of each item. These savings appear to be fundamental to making the object-action approach viable for OS construction.

KEY WORDS: Abstract data types, Concurrency control, Distributed systems, Synchronization.

[MCMI82] McMillen, Siegel, "Performance and Fault Tolerance Improvements in the Inverse Augmented Data Manipulator Network," Proceedings on the 9th Annual Symposium on Computer Architecture, p. 63, April 1982.

Discusses two aspects of the Inverse Augmented Data Manipulator (IADM) network design: performance and fault tolerance. A single stage look-ahead scheme for predicting blockage is presented to enhance performance. One method of adding some links to the network to enable it to tolerate one link failure is described. A different method of adding links is shown that both improves performance and allows the network to tolerate two switching element or two link failures. Also, a new routing tag scheme that accommodates the new links is discussed.

KEY WORDS: Large scale parallel distributed processing systems, Redundancy, Fault tolerant.

[MEDI81] Medina-Mora, Feiler, "An Incremental Programming Environment," IEEE Transactions on Software Engineering, pp. 472-481, September 1981.

Describes an incremental programming environment (IPE) based on compilation technology, but providing facilities traditionally found only in interpretive systems. In IPE the programmer has a uniform view of the program in terms of the programming language. The program is manipulated through a syntax-directed editor and execution is controlled by a debugging facility. Translator, linker, loader are automatically applied and are not visible to the programmer.

KEY WORDS: Ada environments, Incremental compilation, Interactive debugging, Syntax-directed editing.

[MEKL80] Mekly, Yau, "Software Design Representation Using Abstract Process Networks," IEEE Transactions on Software Engineering, pp. 420-434, September 1980.

Presents an approach to software design representation which is consistent with the concept of engineering blueprints. The main criteria for software engineering blueprints are defined and a network scheme of graphical representation is considered through an overview of Petri net techniques. The concept of an abstract process (AP)-basic element is introduced as the basic element of system representation. Methods of AP-net construction are presented and illustrated by examples.

KEY WORDS: Abstract process, AP-net, Petri net, Process expression, Software design representation.

[MENA79] Menasce, Muntz, "Locking and Deadlock Detection in Distributed Data Bases," IEEE Transactions on Software Engineering, pp. 195-202, May 1979.

Describes two protocols for the detection of deadlocks in distributed data bases--a hierarchically organized one and a distributed one. A graph model which depicts the state of execution of all transactions in the system is used by both protocols. A cycle in this graph is a necessary and sufficient condition for deadlock to exist. Nevertheless, neither protocol requires that the global graph be built and maintained in order for deadlocks to be detected. In the case of the hierarchical protocol, the communications cost can be optimized if the topology of the hierarchy is appropriately chosen.

KEY WORDS: Data bases, Deadlock detection, Distributed data bases, Graph theory.

[MERK78] Merkle, "Secure Communications Over Insecure Channels," Communications of the ACM, pp. 294-299, April 1978.

States that, according to traditional conceptions of cryptographic security, it is necessary to transmit a key, by secret means, before encrypted messages can be sent securely. It is shown that it is possible to select a key over open communication channels so communications security is maintained. A method is described which forces any enemy to expend an amount of work which increases as the square of the work required of the two communicants to select a key.

KEY WORDS: Security, Cryptography, Cryptology, Computer network security, Passive eavesdropping.

[MEYE78] Meyer, "An Efficient Fault Diagnosis Algorithm for Symmetric Multiple Processor Architectures," IEEE Transactions on Computers, pp. 1059-1063, November 1978.

Describes a new diagnosis algorithm for determining the existing fault situation in a symmetric multiple processor architecture. The algorithm assumes that there are n processors (each of which is tested by at least 't' other processors),

and at most 't' of which are faulty. The existing fault situation is always diagnosed if n >= 2t+1 and, in some cases, can still be diagnosed in n < 2t+1. Implementation is straightforward and suitable for microprocessor applications.

KEY WORDS: Diagnosis algorithm, Fault syndromes, Modular networks.

[MEYE80a] Meyer, "On Evaluating the Performability of Degradable Computing Systems," IEEE Transactions on Computers, pp. 720-731, August 1980.

Introduces a unified measure, called 'performability', and establishes the foundations of performability modeling and evaluation. A critical step in the modeling process is the introduction of a user-oriented performance levels. A hierarchical modeling scheme is used to formulate the capability function and capability is used to evaluate performability. These techniques are then illustrated for a specific application: the performability of an aircraft computer in the environment of an air transport mission.

KEY WORDS: Degradable computing systems, Fault-tolerant computing, Hierarchical modeling, Performability evaluation, Performance evaluation, Reliability evaluation.

[MEYE80b] Meyer, Furchtgott, and Wu, "Performability Evaluation of the SIFT Computer," IEEE Transactions on Computers, pp. 501-510, June 1980.

Discusses performability modeling and evaluation methods applied the SIFT Computer in an air transport mission environment. User-visible performance of the 'total system' is modeled as a random variable taking values in a set of 'accomplishment levels'. Base-model is a stochastic process whose states describe the internal structure of SIFT as well as relevant conditions of its environment.

KEY WORDS: Fault-tolerant computing, Performance evaluation, Reliability evaluation.

[MEYE81] Meyer, "A Fault Diagnosis Algorithm for Asymmetric Modular Architectures," IEEE Transactions on Computers, pp. 81-82, January 1981.

Emphasizes the analysis of an algorithm for the automatic fault diagnosis of asymmetric modular networks. The network model used is the one proposed by Preparata, the faults are assumed to be solid, and the Hakimi-Amin t-diagnosability hypothesis is supposed to be satisfied.

KEY WORDS: Connection assignment, Diagnosis algorithm, Modular architecture, Permanent fault.

[MILI85] Mili, "Towards a Theory of Forward Error Recovery," IEEE Transactions on Software Engineering, pp. 735-748, August 1985.

Defines forward recovery as that which consists of generating a sufficiently correct state from the current (not too) contaminated state. A tentative framework is presented for the

study of forward error recovery and then some preliminary results and some future research within the proposed framework are discussed.

KEY WORDS: Error recovery, Exception handling, Forward error recovery, Program fault-tolerance.

[MILL86] Miller, "Exponential Order Statistic Models of Software Reliability Growth," IEEE Transactions on Software Engineering, pp. 12-24, January 1986.

Explains that failure times of a software reliability growth process are modeled as order statistics of independent nonidentically distributed exponential random variables. Various characterizations, properties, and examples of this class of models: Jelinski-Moranda, Goel-Okumoto, Littlewood, Musa-Okumo to logarithmic and more are discussed.

KEY WORDS: Complete monotonicity, Nonhomogeneous Poisson processes, Probability models, Software reliability.

[MULA85] Mulazzani, "Reliability Versus Safety," Proceedings SAFCOMP '85, pp. 141-146, Pergamon Press, 1985.

Indicates reliability and safety are not identical and must be traded off against each other in computer-based, real time systems. A quantitative model for effect of computer control on reliability and safety of a system is derived. Reliability vs safety tradeoff under various fault tolerant techniques ("2 out of 2", "2 out of 3" (or N-version programming), and recovery block) are considered.

KEY WORDS: Real-time systems, Reliability, Safety, Fault tolerant.

[MURA80] Murata, "Synthesis of Decision-Free Concurrent Systems for Prescribed Resources and Performance," IEEE Transactions on Software Engineering, pp. 525-530, November 1980.

Presents a method for synthesizing or growing live and safe marked graph models of decision-free concurrent computations. The approach is modular in the sense that subsystems represented by arcs (and nodes) are added one by one without the need of redesigning the entire system. The following properties of marked graph models can be prescribed in the synthesis: 1) liveness (absence of deadlocks); 2) safeness (absence of overflows); 3) the number of reachability classes; 4) the maximum resource (temporary storage) requirement; 5) computation rate (performance); and 6) numbers of arcs and states.

KEY WORDS: Deadlock-freeness, Decision-free concurrent systems, Modular synthesis, Parallel computation model.

[MUSS80] Musser, "Abstract Data Type Specification in the AFFIRM System," IEEE Transactions on Software Engineering, pp. 24-31, January 1980.

Describes the data type definition facilities of the AFFIRM system for specification and verification. The rewrite rule concepts that form the theoretical basis for its data type

facilities are reviewed. Methods of ensuring convergence (finite and unique termination) of sets of rewrite rules and on the relation of this property to the equational and inductive proof theories of data types are emphasized.

KEY WORDS: Abstract data types, Algebraic specifications, Equational theories, Program verification.

[NARA86] Narasimhan, Nakajima, "An Algorithm for Determining the Fault Diagnosability of a System," IEEE Transactions on Computers, pp. 1004-1008, November 1986.

Discusses the fault diagnosability problem which is the problem of computing the maximum number of faulty units which a system can tolerate without losing its capability of identifying all such faulty units. The problem for the model introduced by Barsi, Grandoni, and Maestrini is studied. A new characterization of the model is presented, and an efficient diagnosability algorithm for a system in this model is developed.

KEY WORDS: Connection assignment, Diagnosable systems, Fault diagnosis, Self-diagnosis.

[NASA85] Annotated Bibliography of Software Engineering Laboratory Literature, NASA Goddard, Greenbelt, MD, November 1985.

Describes all SEL literature published up to November 1985.

KEY WORDS: Metrics.

[NATA85] Natarajan, "Communication and Synchronization Primitives for Distributed Programs," IEEE Transactions on Software Engineering, pp. 396-416, April 1985.

Presents a design of communication and synchronization primitives for distributed programs. Different kinds of communications failures are identified, and distinct mechanisms for handling them are provided. To enable the construction of atomic actions two new program components, atomic agent and manager are introduced. Notion of 'conflicts relation' is introduced in which a designer can construct either an 'optimistic' or 'pessimistic' concurrency control scheme. The design also incorporates primitives for constructing 'nested' atomic actions.

KEY WORDS: Atomic action, Communication failure, Computing agent, Distributed operating system, Distributed system.

[NECH85] Neches, Swartout, and Moore, "Enhanced Maintenance and Explanation of Expert Systems Through Explicit Models of Their Development," IEEE Transactions on Software Engineering, pp. 1337-1350, November 1985.

Describes a paradigm for constructing expert systems which attempts to identify that tacit knowledge, provides means for capturing it in the knowledge bases of expert systems, and applies it towards more perspicuous machine-generated explainations and more consistent and maintainable system organization.

KEY WORDS: Expert systems, Natural language generation, Software development, Software maintenance.

[NEGR84] Negrini, Sami, and Scarabottolo, "Policies for System-level Diagnosis in a Non-Hierarchic Distributed System," IEEE Transactions on Reliability, pp. 333-342, October 1984.

Presents a policy for system-level diagnosis and error confinement in distributed systems. The class of architectures to which the policy applies is defined, and the physical supports needed are described. It is proven that mechanisms presented permit reaching deterministic results of diagnosis in finite time and that system-level error confinement is effectively obtained. Mechanisms are implemented by means of software.

KEY WORDS: Fault tolerance, Distributed processing, System-level diagnosis, Error-confinement, Non-hierarchic system.

[NEUM86] Neumann, "On Hierarchical Design of Computer Systems for Critical Applications," IEEE Transactions on Software Engineering, pp. 905-920, September 1986.

Considers the design of computer systems that must be trusted to satisfy simultaneously a variety of critical requirements such as human safety, fault tolerance, high availability, security, privacy, integrity, and timely responsiveness--and to continue to do so throughout maintenance and long-term evolution. Hierarchical abstraction provides the basis for successive layers of trust with respect to the full set of critical requirements, explicitly reflecting differing degrees of criticality.

KEY WORDS: Abstraction, Critical requirements, Hierarchical design, Reliability, Safety, Security, Trusted subsystems.

[OKUM85] Okumoto, "A Statistical Method for Software Quality Control," IEEE Transactions on Software Engineering, pp. 1424-1430, December 1985.

Proposes a statistical method that can be used to monitor, control, and predict the quality (measured in terms of failure intensity) of a software system being tested. The proposed estimation method is validated through a simulation study. A method for predicting the additional execution time required to achieve a failure intensity objective is discussed. A set of failure data collected from a real-time command and control system is used to demonstrate the proposed method.

KEY WORDS: Additional software test time, Logarithmic Poisson model, Software reliability model, Software quality control.

[OSSF80] Ossfeldt, Jonsson, "Recovery and Diagnostics in the Central Control of the AXE Switching System," IEEE Transactions on Computers, pp. 482-491, June 1980.

Investigates AXE which is a stored program controlled (SPC) telephone exchange system. AXE consists of a duplicated central processor and a number of regional processors. It has secure fault detection and recovery, and efficient diagnosis of permanent and temporary faults (verified by physical fault simulations and field experience).

KEY WORDS: Concurrent diagnosis, Diagnostic programs, Fault-tolerant computing, Maintainability, Intermittent fault, Self-checking.

[OWIC79] Owicki, Gries, "Verifying Properties of Parallel Programs: An Axiomatic Approach," Communications of the ACM, pp. 279-285, May 1979.

Presents a method of proving properties of parallel programs by the use of axioms. This deductive system is more powerful than the one developed by Hoare because of the use of auxiliary variables. Properties proved include mutual exclusion, freedom from deadlock and program termination.

KEY WORDS: Structured multiprogramming, Correctness proofs, Program verification, Concurrent processes, Mutual exclusion, Deadlock.

[OWIC76] Owicki, "A Consistent and Complete Deductive System for the Verification of Parallel Programs," Proceedings of the 8th Annual ACM Symposium on Theory of Computing, pp. 73-86, Hershey, PA, 1976.

Presents the semantics of a simple parallel programming language in two ways: 1) deductively, by a set of Hoare-like axioms and inference rules; and 2) operationally, by means of an interpreter. It is shown that the deductive system is consistent with the interpreter. It is proved that the deductive system is complete relative to a complete proof system for the natural numbers; this result is similiar to Cook's relative completeness for sequential programs.

KEY WORDS: Parallel programming.

[PAPA81] Papakonstantinou, "An Interpreter of Attribute Grammars and its Application to Waveform Analysis," IEEE Transactions on Software Engineering, pp. 279-283, May 1981.

Presents a simple portable interpreter for testing the specifications of problems. These specifications are supposed to be expressed in the formalism of attribute grammars. Parsing and semantics are done simultaneously. Increased power of the grammars and context sensitive characteristics of a language can be described.

KEY WORDS: Attribute grammar evaluator, Attribute grammars, Formal specifications, Semantics.

[PARE85] Parent, Spaccapietra, "An Algebra for a General Entity-Relationship Model," IEEE Transactions on Software Engineering, pp. 634-643, July 1985.

Proposes a definition of a set of algebraic operators to be applied on a general entity-relationship (ER) database. The algebra is said to be complete through equivalence with the usual definition of completeness for relational data manipulation languages. The work is intended to provide a sound basis for the definition of complete entity-relationship data manipulation languages (DML's), an essential feature to make the ER model fully operational.

KEY WORDS: Algebraic operators, Completeness, Data manipulation language, Entity-relationship model.

[PARN79] Parnas, "Designing Software for Ease of Extension and Contraction," IEEE Transactions on Software Engineering, pp. 128-137, March 1979.

Discusses designing software to be extensible and easily contracted. A number of ways that extension and contraction problems manifest themselves in current software is explained. Steps in making software more flexible are discussed. It is shown that the identification of minimal subsets and minimal extensions can lead to software that can be tailored to the needs of a broad variety of users.

KEY WORDS: Contractibility, Extensibility, Modularity, Software engineering.

[PARN85] Parnas, Clements, and Weiss, "The Modular Structure of Complex Systems," IEEE Transactions on Software Engineering, pp. 259-266, March 1985.

Discusses the organization of software that is inherently complex because of many arbitrary details that must be precisely right for the software to be correct. It is shown how the software design technique known as information hiding or abstraction, can be supplemented by a hierarchically structured document, which is called a module guide. The guide is intended to allow both designers and maintainers to identify easily the parts of the software that they must understand, without reading irrelevant details about other parts of the software. An extract from a software module guide is included to illustrate the proposals.

KEY WORDS: Abstract interfaces, Information hiding, Modular structure of software.

[PEAC85] Peachey, Bunt, and Colbourn, "Some Empirical Observation on Program Behavior with Applications to Program Restructuring," IEEE Transactions on Software Engineering, pp. 188-193, February 1985.

Discusses program restructuring attempts to improve the behavior of programs by reorganizing their object codes to account for the characteristics of the VM environment. Part of the restructuring process involves a restructuring graph. An analysis of restructuring graphs of typical programs found edge weights to be distributed in a Bradford-Zipf fashion, implying that a large fraction of total edge weight is concentrated in relatively few edges. This observation can be used to improve the clustering phase of program restructuring.

KEY WORDS: Bradford-Zipf distribution, Clustering, Program behavior, Program restructuring.

[PERL81] Perlis, Sayward, and Shaw, Software Metrics: An Analysis and Evaluation, MIT Press, Cambridge, MA, 1981.

Includes papers presented at a workshop on metrics, as well as some discussion among participants. Metrics for resource and manpower estimation, complexity and quality, reliability, and performance are also indicated.

KEY WORDS: Metrics.

[PERL83] Perlman, "Fault-Tolerant Broadcast of Routing Information," Proceedings of IEEE Infocom '83, pp. 93-102, IEEE, April 18-21, 1983.

Presents an algorithm for the reliable broadcast of routing information throughout a network. The algorithm anticipates the possibility of long-delayed packets, line and node outages, network partitions, hardware failures, and a history of arbitrarily corrupted databases throughout the network. A comparison to ARPANET is included.

KEY WORDS: Network, ARPANET, Routing broadcast scheme, Self-stabilization.

[PERR86] Perry, Toueg, "Distributed Agreement in the Presence of Processor and Communication Faults," IEEE Transactions on Software Engineering, pp. 477-482, March 1986.

Proposes a model of distributed computation in which processes may fail by not sending or receiving the messages specified by a protocol. Solution to the Byzantine Generals Problem for this model is presented. The algorithm exhibits early stopping under conditions of less than maximum failure and is as efficient as the algorithms developed for the more restrictive crash-fault model in terms of time, message and bit complexity. Extant models to underestimate resiliency when faults in the communication medium are considered.

KEY WORDS: Byzantine agreement, Distributed computing, Early stopping, Fault tolerance.

[PETE79] Peterson, "Time-Space Trade-Offs for Asynchronous Parallel Models", Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, pp. 224-230, Atlanta, 1979.

Studies the question of relative efficiencies in the context of a simple model of communicating asynchronous processes. The fundamental problem is whether a simple distributed system, with arbitrary size variables, is any more powerful than a system where only binary valued variables are permitted. The answer was found to be negative, with an intuitive definition of the power of systems. The development of these notions required formalization of concepts such as equivalence of models, and the reduction of systems between models. It was discovered that requiring a strong definition of equivalence which decreased time apparently results in an increase in space.

KEY WORDS: Communicating asynchronous processes, Synchronization.

[PIAT80] Piatkowski, "An Engineering Discipline for Distributed Protocol Systems," Proceedings of the NATO Advanced Study Institute: New Concepts in Multi-user Communication, Norwich, 1980.

Provides an in-depth report on engineering a protocol. The total design process, general features of an engineering discipline and the role of the computer aids are explained. Also included are features a specification method should have, aspects of state-architecture and an example of such an architecture.

KEY WORDS: Distributed data processing, Specification, Validation, Data communications, Computer networking.

[POLA79] Polak, "An Exercise in Automatic Program Verification," IEEE Transactions on Software Engineering, pp. 453-457, September 1979.

Describes the computer-aided proof of a permutation program obtained using the Stanford Pascal verifier. The systematic way in which a proof can be developed from an intuitive understanding of the program is emphasized. The current state of the art in automatic program verification is illustrated.

KEY WORDS: Assertion language, Inductive assertions, Permutation, Theorem proving.

[PRAD80] Pradhan, "A New Class of Error-Correcting/Detecting Codes for Fault-Tolerant Computer Applications," IEEE Transactions on Computers, pp. 471-481, June 1980.

Develops separable error-correcting/detecting codes that provide protection against combinations of both unidirectional and random errors. Codes are presented which can both correct (detect) some 't' random errors and detect any number of unidirectional errors which may also contain 't' or fewer errors. Necessary and sufficient conditions for the existence of these codes are also developed. Decoding algorithms, and implementation of these algorithms are discussed. These codes are effective against both transient and solid faults. These codes are specifically suited for fault tolerant logic built out of memory, read-only memories, certain mass memories, etc.

KEY WORDS: Decoder logic, Error correction and detection, Multiple errors, Multiple faults, Self-checking, Transient faults, Unidirectional errors.

[PROV86] Provan, "Bounds on the Reliability of Networks," IEEE Transactions on Reliability, pp. 260-268, August 1986.

Describes criteria for acceptable schemes to approximate system reliability and investigates such schemes for a special class of network reliability problems. In this framework, we are able to use powerful combinatorial theory to obtain strong bounds for network reliability which can be computed by efficient algorithms. These bounds are demonstrated on a small example, and some computational experience is given.

KEY WORDS: Network reliability.

[RAGH86] Raghavendra, Varma, "Fault-Tolerant Multiprocessors with Redundant-Path Interconnection Networks," IEEE Transactions on Computers, p. 307, April 1986.

Studies fault-tolerant multiprocessor systems employing redundant-path multistage interconnection networks. The graph-theoretic techniques are used to study the problem of routing permutations in extra-stage delta networks when faults are present in the network. The problem of performing an arbitrary permutation on the fault-free network is formulated as a vertex-coloring problem and is extended later to networks with noncritical faults. Although the general problem of realizing a permutation in the minimum number of passes is intractable, classes of permutation with some regularity can be routed optimally. The class of BPC (bit permute-complement) permutation is considered: algorithms for performing arbitrary permutations in this class on the extra-stage delta network are given, both for the fault-free network and for a network with noncritical faults.

KEY WORDS: BPC Permutations, Fault-tolerant routing, Multiprocessor systems.

[RAMA80] Ramamoorthy, Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," IEEE Transactions on Software Engineering, pp. 440-449, September 1980.

Presents some analysis techniques for real-time asynchronous concurrent systems. A system is classified, based on the Petri Net model, as either consistent or inconsistent. Procedures for predicting and verifying the system performance are presented.

KEY WORDS: Asynchronous, Concurrent, Petri net, Real-time.

[RAMA81] Ramamoorthy, Mok, Bastini, Chin, and Suzuki, "Application of a Methodology for the Development and Validation of Reliable Process Control Software," IEEE Transactions on Software Engineering, pp. 537-555, November 1981.

Discusses the necessity of a good methodology for the development of reliable software, especially with respect to final software validation and testing activities. A formal specification development and validation methodology is proposed. The methodology has been applied to the development and validation of a pilot software, incorporating features of critical software for nuclear power plant safety. This experience and the impact on quality of software are discussed.

KEY WORDS: DCDS, Assertion, Dual-programming, Path analysis, Process control.

[RAMA85] Ramamoorthy, et al, "Metrics Guided Methodology," Proceedings COMPSAC '85, pp. 111-120, IEEE, 1985.

Proposes requirement level metrics for the specification language RSL and discusses techniques to use metrics at the requirement phase for the waterfall development model. It is expected that the techniques could be modified for use in other development approaches. The techniques are applied to a moderately complex example.

KEY WORDS: Software complexity, Metrics, Specification language, Waterfall development model.

[RAMA86] Ramamoorthy, Garg, and Prakash, "Programming in the Large," IEEE Transactions on Software Engineering, pp. 769-783, July 1986.

Investigates the problems faced in developing large software which include starting from fuzzy and incomplete requirements, enforcing a methodology on the developers, coordinating multiple programmers and managers, achieving desired reliability and performance in a system, managing a multitude of resources in a meaningful way, and completing the system within a limited time frame. Some of the trends in requirement specification, life cycle modeling, programming environments, design tools, and other software engineering areas for tackling the above problems are discussed. Several phase-independent and phase-dependent techniques for programming in the large are suggested. It is shown how research in automatic programming, knowledge-based systems, metrics, and programming environments can make a significant difference in developing large systems.

KEY WORDS: Information abstraction, Knowledge-based systems, Metrics, Reusability, Software life cycle.

[RAND78] Randell, Lee, and Treleaven, "Reliability Issues in Computing System Design," ACM Computing Surveys, pp. 123-166, June 1978.

Surveys the various problems involved in achieving very high reliability from complex computing systems, and discusses the relationship between system structuring techniques of fault tolerance. Topics covered include: 1) protective redundancy in hardware and software; 2) the use of atomic actions to structure the activity of a system to limit information flow; 3) error detection techniques; 4) strategies for locating and dealing with faults and for accessing the damage they have caused; and 5) forward and backward error recovery techniques, based on the concepts of recovery line, commitment, exception, and compensation. Three systems are discussed: JPL-STAR, the Bell Laboratories ESS No. 1A processor, and the PLURIBUS.

KEY WORDS: Fault tolerance, Fault avoidance, Hardware reliability, Software reliability, System structure.

[RAO79] Rao, "Assignment of Tasks in a Distributed Processor System with Limited Memory," IEEE Transactions on Computers, pp. 291-299, April 1979.

Shows how to assign modules to a two-processor computer system (one processor has limited memory capacity) with the distributed execution so as to minimize execution costs and interprocessor communication costs. Although this problem is NP-complete, techniques based on the Gomory-Hu tree from network flow theory can be applied to instances of the problem to obtain a reduction in complexity. A new technique based on the graph called the inclusive cut graph is shown to be an even more powerful tool. These two techniques can solve some instances of the problem completely while others are reduced sufficiently to be susceptible to enumerative techniques. In the worst case, the techniques yield no reduction in problem size.

KEY WORDS: Computer networks, Cutsets, Distributed computers, Load balancing, NP-complete problems.

[RAPP85] Rapps, Weyuker, "Selecting Software Test Data Using Data Flow Information," IEEE Transactions on Software Engineering, pp. 367-374, April 1985.

Defines a family of program test data selection criteria derived from data flow analysis techniques. This procedure associates with each point in a program at which a variable is defined, those points at which the value is used. Several test data selection criteria, differing in type and number of these associations are defined and compared.

KEY WORDS: Data flow, Program testing, Test data selection.

[RAU79] Rau, "Interleaved Memory Bandwidth in a Model of a Multiprocessor Computer System," IEEE Transactions on Computers, pp. 678-681, September 1979.

Performs an approximate analysis of an often studied model of an interleaved memory, multiprocessor system consisting of 'M' memory modules and 'N' processors. A closed-form solution is obtained and the one approximation used is found to result in negligible error. This solution is shown to be about an order of magnitude more accurate than the best previous result.

KEY WORDS: Analytical models, Interleaved memories, Memory bandwidth, Memory interference, Multiprocessors, Performance evaluation.

[REDD78] Reddi, Fenstel, "A Restructurable Computer System," IEEE Transactions on Computers, pp. 1-20, January 1978.

Presents an architecture for a reconstructurable computer system which reconfigures its resources according to the problem environment for efficient performance. The architecture converts the user's program into an intermediate level language, 'Realist', capable of specifying arbitrary resource structures and the computation to be performed upon these structures. An architectural design for the system is presented with special attention to bus units. It is shown how APL, a vector processing language, can be implemented on the system. Storage schemes for organizing vectors and matrices to facilitate efficient retrieval and manipulation are given. A comparison of the proposed system to existing high speed architectures is included.

KEY WORDS: APL implementation, Architectural design, Parallel computation, Parallel languages, Reconstructible computers.

[REED78] Reed, Naming and Synchronization in a Decentralized Computer System, pp. 1-182, MIT, Cambridge, MA, September 15, 1978. Ph.D. Thesis, Report # MIT/LCS/TR-205.

Develops a new approach to the synchronization of accesses to shared data objects. An object that is modifiable is regarded as a sequence of immutable versions, each version is the state of the object after an update is made to the object. Synchronization can then be treated as a mechanism for naming versions to be read and for defining where in the sequence of versions the version resulting from some update should be placed.

KEY WORDS: Distributed computer systems, Reliability, Synchronization.

[RICA80] Ricart, Agrawala, An Algorithm For Mutual Exclusion In Computer Networks, pp. 1-29, Air Force Office of Scientific Research, February 1980. Technical Report TR-869.

Proposes an algorithm which creates mutual exclusion in a computer network whose nodes can communicate only by messages and do not share memory. The "readers and writers" problem is solved by a simple modification of the algorithm. The modifications necessary to make the algorithm robust are described.

KEY WORDS: Mutual exclusion, Message passing, Deadlock, Starvation.

[RICH85] Richardson, Clarke, "Partition Analysis: A Method Combining Testing and Verification," IEEE Transactions on Software Engineering, pp. 1477-1490, December 1985.

Presents a partition analysis method which compares a procedures implementation to its specification both to verify consistency between the two and to derive test data. It is applicable to a number of different types of specification languages both procedural and nonprocedural and to high level descriptions and low level design. The partition analysis method is described and the results obtained from an evaluation of its effectiveness is reported.

KEY WORDS: Software testing, Software verification, Symbolic evaluation.

[ROBI77] Robinson, Levitt, "Proof Techniques for Hierarchically Structured Programs," Communications of the ACM, pp. 271-283, April 1977.

Proposes a method for describing and structuring programs that simplifies proofs of their correctness. The method formally represents a program in terms of levels of abstraction, each level of which can be described by a self-contained nonprocedural specification. This method is applicable to semiautomatic and automatic proofs as well as manual.

KEY WORDS: Hierarchical structure, Program verification, Formal specification, Abstraction.

[ROSE85] Rosenthal, Guidance on Planning and Implementing Computer System Reliability, pp. 1-42, NBS, Gaithersburg, MD, January 1985. NBS Special Publication 500-121.

Presents an overview of the fundamental concepts and concerns associated with system reliability, and identifies elements and activities involved in planning and implementing a reliability program. The underlying theme is that a knowledge of reliability is important in the development of new system specifications as well as in the continual assessment of existing computer systems.

KEY WORDS: Reliability, Recovery.

[ROSEN81] Rosen, Vulnerabilities of Network Control Protocols: An Example, Bolt Beranck and Newman Inc, 1981.

Discusses ways in which unusual circumstances can bring out vulnerabilities in network control protocols. The network-wide disturbance on the ARPANET are described.

KEY WORDS: Network disturbances, ARPANET, Protocols.

[ROSS85a] Ross, "Statistical Estimation of Software Reliability," IEEE on Software Engineering, pp. 479-483, May 1985.

Discusses when a new computer software package is developed, a testing procedure is often put into effect to eliminate the faults in the package. One common procedure is to try the package on a set of well-known problems to see if errors result, and once stopped, the package is checked to determine the bugs and then altered to remove the bugs. This procedure is modeled, the error rate for the revised package is determined, and how to estimate this quantity under a variety of assumptions as to what is learned when the debugging occurs is shown.

KEY WORDS: Estimation, Reliability, Poisson process.

[ROSS85b] Ross, "Software Reliability: The Stopping Rule Problem," IEEE Transactions on Software Engineering, pp. 1472-1476, December 1985.

Discusses two problems of interest: 1) how to estimate the error rate of the software at a given time 't'; and 2) how to develop a stopping rule for determining when to discontinue the testing and declare the software ready to use. A model for the above is proposed as an estimation and stopping rule procedure.

KEY WORDS: Failure rate, Software reliability, Stopping times.

[RUBI82] Rubin, West, "An Improved Protocol Validation Technique," Computer Networks 6, pp. 65-73, North-Holland Publishing Company, 1982.

Discusses the nature of message exchanges between components of a communication system. The primary concern is with systems in which the transmission delay is not negligible and cannot be ignored. It is shown for a system of two processes that sequences of exchanged messages can be expressed in canonical form by decomposing them into interactions that may, in their simplest form, contain several concurrent message exchanges. Expressing the communication between processes in this way permits a clearer understanding of the limits of applicability of the validation technique and of the role of the more complex types of interactions in communication systems. The nature of interactions in systems containing more than two processes is also discussed.

KEY WORDS: Protocol validation, Communication systems, Interactions.

[RUDI81] Rudin, West, Validation of Protocols Using State Enumeration: A Summary of Some Experience, IBM Zurich Research Laboratory, 8803 Rueschlikon, Switzerland, March 1981.

Records experience of an effort with the goal of developing automated techniques for validating protocols. The validation method used consists first, of defining an intial global system state, including states of all communication system components. All system states accessible from this initial state are then iteratively generated, each being checked for the occurrence of a number of well-defined syntactical error conditions.

KEY WORDS: Reception errors, Static deadlocks, Dynamic deadlocks.

[RUSS80] Russell, "State Restoration in Systems of Communicating Processes," IEEE Transactions on Software Engineering, pp. 183-193, March 1980.

Indicates that in systems of asynchronous processes using message lists with SEND-RECEIVE primitives for interprocess communication, 'recovery primitives' are defined to perform state restoration: 1) MARK saves a particular point in the execution of the program; 2) RESTORE resets the system state to an earlier point (saved by mark); 3) PURGE discards redundant information. Errors may be propagated through the system, requiring state restoration also to be propagated. Different types of propagation of state restoration are identified. Data structures and procedures are sketched that implement the recovery primitives. In ill-structured systems the 'domino effect' can occur, resulting in a catastrophic avalanche of backup activity and causing many messagelist operations to be undone. Sufficient conditions are developed for a system to be domino-free. Explicit bounds on the amount of unnecessary restoration are determined for certain classes of systems.

KEY WORDS: Domino effect, Error recovery, Parallel backtracking, Process communication, State restoration.

[RYPK79] Rypka, Lucido, "Deadlock Detection and Avoidance for Shared Logical Resources," IEEE Transactions on Software Engineering, pp. 465-471, September 1979.

Defines logical resources as shared passive entities that can be concurrently accessed by multiple processes. Concurrency restrictions depend upon the manner in which a process may manipulate a resource. Models incorporating these single unit resources can be used to analyze information locking for consistency and integrity purposes. Mode compatibility is used to derive deadlock detection and avoidance methods. These methods permit greater concurrency while guaranteeing data consistency. This model is applicable to the standard shared and exclusive access modes.

KEY WORDS: Allocation modes, Deadlock avoidance, Deadlock detection, Logical resource, Resource allocation, Resource sharing.

[SAXE86] Saxena, Robinson, "Accumulator Compression Testing," IEEE Transactions on Computers, pp. 317-321, April 1986.

Proposes a new test data reduction technique call accumulator compression testing (ACT). It is shown that the enumeration of errors missed by ACT for a unit under test is equivalent to the number of restricted partitions of a number. A comparison

is made between signature analysis (SA) and ACT. Theoretical and experimental results indicate that with ACT a better control over fault coverage can be obtained than with SA. Built-in self tests for processor environments may be feasible with ACT. For general VLSI circuits the complexity of ACT may be a problem as an adder is necessary.

KEY WORDS: Built-in test, Fault coverage, Testing.

[SCHA78] Schaffner, "Processing by Data and Program Blocks," IEEE Transactions on Computers, pp. 1015-1027, November 1978.

Presents a processing system that implements simultaneously the efficiency of the special-purpose and the total applicability of the general-purpose computer. This is achieved through specializing the machine by programming the hardware structure, rather than by adding software systems to it. Data are organized in circulating pages which form a multiplicity of local dynamic memories for each process. Programs are made up of modules, each describing a transient special-purpose machine. A characteristic of this approach is that processes are data-driven rather than program-driven. The language presents flexibility and efficiency in modeling certain classes of problems, and it may be of interest as an implementation model in a broader context. Applications of real-time processing of radar signals are reported. The relevance of characteristics of this system to problems in multiprogramming and multiprocessing systems is discussed.

KEY WORDS: Computer architecture, Data-driven processing, Microprogramming, Multiprogramming, Radar signal processing, Real-time processing.

[SCHL85] Schlichting, "A Technique for Estimating Performance of Fault-Tolerant Programs," IEEE Transactions on Software Engineering, pp. 555-563, June 1985.

Presents a technique for estimating the performance of a program written for execution on fail-stop processors. This technique is based on modeling the program as a discrete-time Markov chain and then using z-transforms to derive a probability distribution for time to completion.

KEY WORDS: Fail-stop processors, Fault-tolerant computing, Markov chains, Performance evaluations.

[SCHO86] Scholz, "Software Reliability Modeling and Analysis," IEEE Transactions on Software Engineering, pp. 25-31, January 1986.

Examines a discrete and a continuous model for the software reliability growth process. The discrete model is based on independent multinomial trials and concerns itself with joint distribution of the first occurrence of its underlying events (bugs). The continuous model is based on the order statistics of N independent nonidentically distributed exponential random variables. Estimated upper bounds and confidence bounds for the residual program error content are given based on the spacings of the first 'k' bugs removed.

KEY WORDS: Conditional inference, Exponential order statistics, Identifiability, Multinomial trials, Order restricted maximum likelihood estimates.

[SCHU79] Schutz, "On the Design of a Language for Programming Real-Time Concurrent Processes," IEEE Transactions on Software Engineering, pp. 248-255, May 1979.

Describes ILIAD--a high-level language for programming real-time applications which involve concurrent processing. It was designed to help write reliable programs that can be read and maintained. An ILIAD program consists of a group of concurrent tasks that are autonomous, and must share memory and devices in the execution environment. A programming example illustrates how ILIAD might be applied.

KEY WORDS: Concurrent programming, ILIAD, Multiprogramming, Multiprocessing, Real-time languages, Real-time programming.

[SEAQ80] Seaquist, A Semantics of Synchronization, pp. 1-111, MIT, Cambridge, MA, August 1980. M.S. Thesis, Report # MIT/LCS/TM-176.

Presents a rigorous framework in which to discuss the synchronization necessary to coordinate accesses to a resource. The framework provides a method for specifying concurrency and forms the semantic basis of a synchronization mechanism which avoids certain unfortunate characteristics of monitors and serializers. Many examples of uses of the mechanism are given and its advantages are discussed.

KEY WORDS: Synchronization mechanism, Concurrency, Resource guardians, Event sequences.

[SEDM80] Sedmak, Liebergot, "Fault-Tolerance of a General Purpose Computer Implemented by Very Large Scale Integration (VLSI)," IEEE Transactions on Computers, pp. 492-500, June 1980.

Describes preliminary results of a research effort to design a general purpose computer with VLSI components which will achieve a level of fault detection, recovery, and failure isolation far exceeding non-VLSI implementations. The fundamental approach is to design and partition the logical elements in such a way that effective fault detection can be done by a novel method which replaces the detection responsibility inside the VLSI chip. There is a high level of fault tolerance and tolerance of most transient and many solid features. The rationale for the design tradeoffs which must be made in the development of a general purpose computer is also explored.

KEY WORDS: Availability, Fault detection, Fault recovery, Fault tolerance, Maintainability, Reliability, Self-checking.

[SHAN82] Shankar, "A Functional Approach to Module Verification," IEEE Transactions on Software Engineering, pp. 147-159, March 1982.

Develops a method for designing and verifying data abstractions using the fundamental approach. These techniques are

then modified and extended to verify data abstractions. By using the concept of a mathematical function, one can model the behavior of a procedure abstraction and give a more uniform and clearer meaning to the stepwise refinement and verification of procedure abstractions. Using state machine specification, a technique for expressing the design of a data abstraction is given. A method is developed to verify the design of a data abstraction with respect to its specifications.

KEY WORDS: Abstract model specification, Data abstraction, Procedure abstraction, State machine, Verification.

[SHEN85] Shen, Yu, Thebaut, and Paulsen, "Identifying Error-Prone Software--An Empirical Study," IEEE Transactions on Software Engineering, pp. 317-323, April 1985.

Discusses a study undertaken to assess the potential usefulness of various product- and process- related measures in identifying error-prone software. An empirical basis is established for the efficient utilization of limited testing resources using objective, measurable criteria.

KEY WORDS: Defect density, Error-prone modules, Probability of errors, Software errors, Software metrics.

[SHIN86] Shin, Lee, "Measurement and Application of Fault Latency," IEEE Transactions on Computers, pp. 370-375, April 1986.

Presents a new methodology for indirectly measuring fault latency, derives the distribution of fault latency from the methodology and applies the knowledge of fault latency to the analysis of two important examples. The new methodology has been implemented for measuring fault latency in the Fault-Tolerant Multiprocessor (FTMP) at the NASA Airlab.

KEY WORDS: Detection time, Fault and error latency, Fault injection, Maximum likelihood estimator.

[SHIN87] Shin, Ramanathan, "Clock Synchronization of a Large Multiprocessor System in the Presence of Malicious Faults," IEEE Transactions on Computers, pp. 2-12, January 1987.

Presents a new method to remedy the problem of clock synchronization in the presence of malicious faults that: 1) requires little time overhead by using phase-locked clock synchronization; 2) needs a clock network very similiar to the processor network; and 3) uses only 20-30 percent of the total number of interconnections required by a fully connected network for almost no loss in the synchronizing capabilities. An example hardware implementation is given to show the feasibility of this method.

KEY WORDS: Clock synchronization, Fault-tolerant real-time multiprocessors, Malicious faults.

[SHOO84] Shooman, "Software Reliability: A Historical Perspective," IEEE Transactions on Reliability, pp. 48-55, April 1984.

Discusses progress since 1970 in software reliability functions and mean time between software error metrics. The author states that future progress requires a database of software reliability information.

KEY WORDS: Software reliability, History.

[SIDH81] Sidhu, "Toward Constructing Verifiable Communication Protocols," INWG/NPL Workshop, National Physical Laboratory, Teddington, England, May 27-29, 1981.

Presents an in-depth review of progress made in producing protocols. Emphasis is placed on specification and verification and a formal description is given. Formal specification languages are discussed in term of syntax, style and automated tools. There are also discussions on verification techniques and two approaches for syntactically correct protocols.

KEY WORDS: Communication protocols, Specification, Verification, Specification techniques, Protocol design.

[SIDH82a] Sidhu, "Protocol Design Rules," Protocol Specification, Testing and Verification, pp. 283-298, North-Holland Publishing Company, 1982. IFIP.

Suggests that the design rules approach to protocol synthesis is a promising method for constructing reliable protocols. Three approaches to communication protocols synthesis are compared. Extensions of synthesis approaches to formal systems based on temporal logic are also discussed for proving semantic properties of protocols.

KEY WORDS: Protocol synthesis, Design rules, Communication protocols, Completeness, Deadlock-freeness, Temporal logic.

[SIDH82b] Sidhu, "Synthesis of Communication Protocols," International Computer Conference '82, Philadelphia, PA, June 13-17, 1982.

Presents an approach to communication protocols synthesis which permits the development of general (FIFO and non-FIFO channels), N-party (N >= 2) protocols with the following properties: completeness, deadlock freeness, termination or cyclic behavior, liveness, boundedness and absence of non-executable interactions.

KEY WORDS: Communication protocols, Protocol properties, Design rules, Protocol synthesis.

[SIDH83] Sidhu, "Protocol Verification via Executable Logic Specifications," Proceedings of the 3rd International Workshop on Protocol Specification, Testing, and Verification, Zurich, Switzerland, May 31 - June 2, 1983.

Discusses the use of logic programming techniques in the specification and verification of communication protocols. Protocol specifications discussed are formal and directly executable. Horn clause logic is discussed, which has a procedural interpretation, and the predicate logic programming language, PROLOG, is used to specify and verify the functional correctness of protocols.

KEY WORDS: Communication protocols, Executable specification, Verification, Horn clause logic, PROLOG.

[SIDH86] Sidhu, Blumer, "Verification of NBS Class 4 Transport Protocol," IEEE Transactions on Communications, pp. 781-789, August 1986.

Discusses the verification of the connection management aspects of a transport layer protocol available from National Bureau of Standards. An automated protocol development technique is used to verify a subset of the protocol with respect to the protocol properties of completeness, deadlock freeness, boundedness, and termination. An overview of the protocol development technique used in specification and verification of the protocol is given. The transport layer protocol is described and the results obtained by applying the automated verification technique to this protocol are presented.

KEY WORDS: Transport protocol, Formal description techniques, Automated development tools.

[SILB79] Silberschatz, "Communication and Synchronization in Distributed Systems," IEEE Transactions on Software Engineering, pp. 542-546, November 1979.

Examines Hoare's suggestion of using input and output constructs for handling of communication and synchronization in greater detail by concentrating on the following two issues: 1) allowing both input and output commands to appear in guards; and 2) simple abstract implementation of the I/O constructs. In the process of examining these two issues a framework is developed for the design of appropriate communication and synchronization facilities for distributed systems.

KEY WORDS: Distributed systems, Guarded commands, Input/output commands, Synchronization.

[SING85] Singpurwalla, Soyer, "Assessing (Software) Reliability Growth Using a Random Coefficient Autoregressive Process and its Ramifications," IEEE Transactions on Software Engineering, pp. 1456-1464, December 1985.

Motivates a random coefficient autoregressive process of order 1 for describing reliability growth or decay. Several ramifications of this process are introduced, some of which reduce it to a Kalman Filter model. The usefulness of the approach is illustrated by applying these processes to some real life data on software failures. Pairwise comparison is made of the models in terms of the ratio of likelihoods of their predictive distributions.

KEY WORDS: Dynamic linear and nonlinear models, Kalman Filtering, Likelihood ratios, Predictable distributions, Prequential analysis, Reliability growth, Software reliability.

[SING86] Singh, Asgarpoor, "Reliability Evaluation of Flow Networks Using Delta-Star Transformations," IEEE Transactions on Reliability, pp. 472-477, October 1986.

Presents a new approach based on delta-star and/or star-delta transformations for calculating the maximum network flow between a pair of nodes. Procedure consists of simplifying complex structures by converting them into equivalent series parallel configurations and successively reducing the network until a single arc is obtained. The system reliability can be determined.

KEY WORDS: Flow-network, Delta-star transformation, Maximum flow.

[SMIT85] Smith, Kotik, and Westfold, "Research on Knowledge-Based Software Environments at Kestrel Institute," IEEE Transactions on Software Engineering, pp. 1278-1295, November 1985.

Presents a summary of the CHI project highlighting prototypes that commenced the research on knowledge-based software environments. The objective of this project was to perform research on knowledge-based software environments. Two major results of the project are: 1) development of a wide-spectrum language that could be used to express all stages of the program development in the system; 2) the prototype compiler was used to synthesize itself from very-high-level description of itself. Overall nature of the work is described, highlights of implemented prototypes are given, and implications that this work suggests for the future of software engineering are described.

KEY WORDS: Automatic programming, Knowledge-based system, Program synthesis.

[SMOL81] Smoliar, "Operation Requirements Accommodation in Distributed System Design," IEEE Transactions on Software Engineering, pp. 531-536, November 1981.

Concentrates on three important operation requirements: reliability via fault tolerance, growth, and availability. Accommodation of these requirements is based on an approach to functional decomposition involving representation in terms of virtual machines. Functional requirement may be accommodated through hierarchal decomposition of virtual machines, while performance requirements may be associated with individual virtual machines. Virtual machines may then be mapped to a representation of a configuration of physical resources, so that performance requirements may be reconciled with available performance characteristics.

KEY WORDS: DCDS, Distributed processing, Fault tolerance, Real-time systems, Reliability.

[SMOT86] Smotherman, Geist, and Trivedi, "Provably Conservative Approximations to Complex Reliability Models," IEEE Transactions on Computers, pp. 333-338, April 1986.

Suggests provably conservative reliability models can be systematically derived from the complex model. These derived models incorporate a reduced state space and fewer transitions, and therefore, have solutions that are more cost-effective than those of the original complex models. The designer can extensively explore the design space without incurring the expense of solving multiple complex models. A conservative-optimistic pair derived models produces a band that includes the solution of the complex model. Sensitivity analysis can be performed on

this pair of models to determine those parameters of the original model that are most sensitive to change and hence warrant further expense in obtaining tighter specifications.

KEY WORDS: Fault coverage, Fault-tolerant computers, Reliability bounds, Sensitivity analysis.

[SNYD81] Snyder, "Formal Models of Capability-Based Protections Systems," IEEE Transactions on Computers, pp. 172-181, March 1981.

Explains the role of formal modeling in the study of capability-based protection systems. Historical background and a model of a computer science department computer system is presented. A survey of several important capability-based models is given including the Harrison, Ruzzo, Ullman model, the Take-Grant model, and grammatical models. The results of the models are compared, contrasted and interpreted in the context of numerous examples.

KEY WORDS: Capabilities, Grammatical protection systems, Security, Take/Grant models.

[SOI81] Soi, Aggarwal "Reliability Indices for Topological Design of Computer Communication Networks", IEEE Transactions on Reliabi pp. 438-443, December 1981.

Explains that incorporating network reliability parameters in the design of reliable computer communication networks makes the computations prohibitive. Interdependence among network topological parameters does not permit the design of maximally reliable networks using any one of the parameters and thus, there arises a real need for a composite reliability index which gives a more realistic assessment of network reliability. After experimental results regarding the effects of various topological parameters on network reliability are discussed, two heuristic reliability indices which give a fair indication of overall reliability are presented. A design procedure for reliable computer communication networks based on local search technique incorporating these reliability indices is suggested. Having only one composite reliability index, which is very simple to evaluate, saves computation while designing maximally reliable computer networks as compared to the existing techniques based on several reliability measures.

KEY WORDS: Computer communication network, Network topology, Reliability indices.

[SPEE79] Speelpenning, Nievergelt, "A Simple Model of Processor-Resource Utilization in Networks of Communicating Modules," IEEE Transactions on Computers, pp. 927-929, December 1979.

Presents a simple model of a system of processors competing for resources. The model permits the derivation of upper bounds for the utilization of the system components that are surprisingly close to those obtained for more complicated analyses or simulations.

KEY WORDS: Interleaved memories, Interaction model, Performance analysis, Resource utilization.

[SPIT75] Spitzen, Wegbreit, "The Verification and Synthesis of Data Structures," Acta Informatica 4, pp. 127-144, Springer-Verlag, 1975.

Presents the concept of machine extension used for implementing complex software. The technique is applied to automatic programming. Specifying data structures, verification by a programming language, and proving data structures in programs correct are all discussed.

KEY WORDS: Verification, Automatic programming, Correctness.

[SPIT78] Spitzen, Levitt, and Robinson, "An Example of Hierarchical Design and Proof," Communications of the ACM, pp. 1064-1075, December 1978.

Describes a formal method for hierarchical program specification, implementation and proof. This method is applied to a significant list processing problem and a number of extensions to current programming languages that ease hierarchical program design and proof are also discussed.

KEY WORDS: Program verification, Specification, Data abstraction, Hierarchical structures.

[SPRE85] Spreij, "Parameter Estimation for a Specific Software Reliability Model," IEEE Transactions on Reliability, pp. 323-328, October 1985.

Studies problems of maximum likelihood estimation in the Jelinski-Moranda software reliability model. Distribution of the stochastic variable that completely determines the maximum likelihood estimate is obtained. s-confidence intervals for the parameter of interest can be constructed by using the same stochastic variable. An example is given.

KEY WORDS: Software reliability, Parameter estimation, s-confidence interval, Jelinski-Moranda model.

[STAN85] Stankovic, "Stability and Distributed Scheduling Algorithms," IEEE Transactions on Software Engineering, pp. 1141-1152, October 1985.

Discusses stability issues for distributed scheduling algorithms in general terms. Two different distributed scheduling algorithms which contain explicit mechanisms for stability are presented and evaluated with respect to individual specific stability issues. One is based on stochastic learning automata and the other on bidding. Results indicate how very specific the treatment of stability is to the algorithm and environment under consideration.

KEY WORDS: Bidding, Distributed computing, Stability, Stochastic learning automata.

[STEI85] Steier, Kant, "The Roles of Execution and Analysis in Algorithm Design," IEEE Transactions on Software Engineering, pp. 1375-1385, November 1985.

Reports on research in artificial intelligence. Emphasis is placed on a language for practical design algorithms and a process to help design algorithms. A new system developed as a result of research is discussed, and a comparison between evaluations and execution techniques is given.

KEY WORDS: Algorithm design, Automatic programming, Developmental evaluation, Meta-evaluation.

[STEM86] Stemple, Vinter, and Ramamritham, "Functional Addressing in Gutenberg: Interprocess Communication without Process Identifiers," IEEE Transactions on Software Engineering, pp. 1056-1065, November 1986.

Presents an interprocess communication facility provided by the kernel of the Gutenberg experimental operating system. In Gutenberg all interprocess communication is via channels which are typed by the service which can be requested on them. Ports are created by reference to their service without using the identifier of the process providing the service, a technique referred to as functional addressing. By using functional addressing, interprocess transfer of port use privileges, and a new concept of cooperation class, arbitrary process interconnection topologies can be achieved without any explicit use of process identifiers by processes. Examples of object sharing with abstract data type managers and data-driven protocol of database query execution are presented to illustrate the methods of constructing systems of cooperating processes using the Gutenberg system.

KEY WORDS: Capabilities, Functional addressing, Interprocess communication, Port.

[STRO86] Strom, Yemini, "Typestate: A Programming Language Concept for Enhancing Software Reliability," IEEE Transactions on Software Engineering, pp. 157-171, January 1986.

Introduces 'Typestate'--a new programming language concept which is a refinement of 'type'. Whereas the type of a data object determines the set of operation ever permitted on the object, typestate determines the subset of those operations which is permitted in a particular context. A definition of typestate is given along with examples of its execution, and it is shown how typestate checking may be embedded into a compiler. Consequences of typestate checking for software reliability and software structure are discussed, and experience using a high level language incorporating typestate checking is reported.

KEY WORDS: Program analysis, Program verification, Security, Software reliability, Type checking.

[SUMI86a] Sumita, Masuda, "Analysis of Software Availability/Reliability Under the Influence of Hardware Failures," IEEE Transactions on Software Engineering, pp. 32-41, January 1986.

Presents a new hardware-software reliability model where lifetimes and repair times of the software subsystem have general system state-dependent distributions. A numerical

example is given which demonstrates speed, accuracy, and stability of the procedures.

KEY WORDS: Integrated hardware-software reliability model, Multiple error generation and removal, State-dependent general lifetimes and repair times, Time-dependent compound performance measures.

[SUMI86b] Sumita, Shanthikumar, "A Software Reliability Model with Multiple-Error Introduction and Removal," IEEE Transactions on Reliability, pp. 459-462, October 1986.

Considers a stochastic model for a software system where multiple errors can be introduced and removed from the software system during repairs. It assumes that the software failure rate is proportional to the number of software errors present in the system. A general Markov model is developed. Expressions for software reliability measures are derived and corresponding computation procedures are developed. A numerical example is given.

KEY WORDS: Software reliability, Multiple-error introduction, Markov models, Performance evaluation.

[SUZU86] Suzuki, "Experience with Specification and Validation of a Complex Computer Using Prolog," Logic Programming and Its Applications, Apex Publishing Co., Norwood, NJ, 1986.

Describes use of Prolog for validation of a VLSI design.

KEY WORDS: Verification, Concurrent, VLSI.

[TAI80] Tai, "Program Testing Complexity and Test Criteria," IEEE Transactions on Software Engineering, pp. 531-539, November 1980.

Explores 'Testing Complexity' of several classes of programs which is measured in terms of the number of test data required for demonstrating program correctness by testing. Two new test criteria are proposed, one for testing a path and the other for testing a program. These suggest how to select test data to obtain confidence in program correctness beyond the requirement of having each statement, branch, or path tested at least once.

KEY WORDS: Program testing, Testing complexity, Test criteria.

[TAMI80] Tamir, "ADI: Automatic Derivation of Invariants," IEEE Transactions on Software Engineering, pp. 40-48, January 1980.

Describes an interactive computer program (ADI) which automatically generates the needed inductive assertions for mechanical program verification. ADI is also able to extend partial loop assertions supplied by the user to form complete assertions. Implementation is in QLISP and INTERLISP. ADI is a small step toward interactive, practical program verification.

KEY WORDS: Invariants, Partial correctness, Program verification, QLISP, Synthesis of invariants.

[TANA78] Tanaka, "Error-Correcting Parsers for Formal Languages," IEEE Transactions on Computers, pp. 605-616, July 1978.

Describes error-correcting parsers for context-free and context-sensitive languages with substitution, insertion, and deletion errors. It is shown that the ability of the proposed parsers can be expressed in terms of the weighted Levenshtein metric.

KEY WORDS: Context-sensitive parser, Error-correcting parser, Formal languages, Maximum-likelihood parser.

[TAYL80a] Taylor, Morgan, and Black, "Redundancy in Data Structures: Improving Software Fault Tolerance," IEEE Transactions on Software Engineering, pp. 585-594, November 1980.

Discusses one way to improve software reliability--by adding redundant structural data to data structures. This is used to detect and correct (structural) errors in instances of a data structure. The intuitive approach of this paper, which makes heavy use of examples, is complemented by the more formal development of the companion paper "Redundancy in Data Structures: Some Theoretical Results".

KEY WORDS: Error correction, Error detection, Software fault tolerance, Software reliability.

[TAYL80b] Taylor, Morgan, and Black, "Redundancy in Data Structures: Some Theoretical Results," IEEE Transactions on Software Engineering, pp. 595-601, November 1980.

Presents the underlying theory for robust data structures and uses it to discuss the synthesis and cost effectiveness of robust data structures.

KEY WORDS: Compound data structures, Error correction and detection, Robust data structures, Software fault tolerance and reliability.

[TAYL80c] Taylor, Osterweil, "Anomaly Detection in Concurrent Software by Static Data Flow Analysis," IEEE Transactions on Software Engineering, pp. 265-277, May 1980.

Presents algorithms for detecting errors and anomalies in programs which use synchronization constructs to implement concurrency. Data flow analysis techniques are employed. Important classes of errors unique to concurrent process programs are described, and algorithms for their detection are presented.

KEY WORDS: Concurrent software, Error detection, HAL/S, Process synchronization errors.

[TAYL86a] Taylor, "Concurrency and Forward Recovery in Atomic Actions," IEEE Transactions on Software Engineering, pp. 69-78, January 1986.

Presents an analysis of the following problems: 1) some difficulties and complexities occur only when the concept of atomic actions is extended to allow concurrency within atomic actions and to allow a single atomic action to execute at a

number of different sites; 2) providing facilities for both forward and backward recovery. A general structure for a solution is proposed, syntax to specify this structure is given and illustrated with examples. The practicality of the scheme is justified by sketching one possible implementation.

KEY WORDS: Atomic actions, Backward recovery, Exception handling, Forward recovery, Software reliability.

[TAYL86b] Taylor, Seger, "Robust Storage Structures of Crash Recovery," IEEE Transactions on Computers, pp. 288-295, April 1986.

Discusses a robust storage structure which is intended to provide the ability to detect and possibly correct damage to the structure. Some of the general principles of using robust storage structures for crash recovery are examined. A particular class of linked list structures which can be made arbitrarily robust, and which are suitable for crash recovery are described.

KEY WORDS: Crash recovery, Software fault tolerance, Global and local error correction.

[THAY82] Thayse, "Synthesis and Optimization of Programs of Means of P-Functions," IEEE Transactions on Computers, pp. 34-40, January 1982.

Defines a program as an indexed sequence of instructions; each of these instructions is formed by an interconnection of branching instructions followed by an interconnection of execution instructions. A program is an efficient tool, allowing the digital system designer to describe the microprograms of discrete systems and to synthesize their control automation. This paper deals with a method of transformation and of optimization of programs. The presented algorithm obtains, for any given program, an equivalent one with a minimum number of conditional vertices.

KEY WORDS: Algorithmic state machine, Microprogrammed structures, Transformation of programs.

[TJAD76] Tjaden, "Hierarchical Properties of Concurrency," Proceedings of the 1976 International Conference on Parallel Processing, pp. 55-64, 1976.

States that for any given machine language program, a request hierarchy can be constructed which has interesting applications to the problem of the dynamic hardware detection and control of execution of concurrency. Starting with a binary vector-pair model of instructions and knowledge of the destinations and branch instructions, a hierarchy of tasks is constructed which allows a global dynamic analysis of large programs to be made by the hardware during the execution of the program. This approach can lead to detectable program execution speed-ups.

KEY WORDS: Concurrency, Hierarchy of tasks.

[TRAC85] Trachtenberg, "The Linear Software Reliability Model and Uniform Testing," IEEE Transactions on Reliability, pp. 8-16, April 1985.

Discusses the Jelinski-Moranda, Shooman, and Musa software reliability models which all predict that the software error detection rate is a linear function of the detected errors. Studies show that error rates during system testing correlate best with the Musa model, and progressively less with the Shooman, and Jelinski-Moranda models. Simulation shows: 1) testing the functions of a software system in a random order gives linearly decaying system-error rates; 2) testing each function exhaustively one at a time gives flat system-error rates; 3) testing different functions at widely different frequencies gives exponentially decaying system-error rates; and 4) testing strategies which result in linear decaying error rates tend to require the fewest tests to detect a given number of errors.

KEY WORDS: Software reliability model, Software testing, Jelinski-Moranda model, Shooman model, Musa model.

[TROY85] Troy, Moawad, "Assessment of Software Reliability Models," IEEE Transactions on Software Engineering, pp. 839-848, September 1985.

Proposes a method for assessing software reliability models and its application to the Musa and Littlewood-Verrall models. A taxonomy of the criteria to be considered for assessing a software reliability model and a method for application are presented. Results of an assessment of the Musa and Littlewood-Verrall models are given.

KEY WORDS: Model comparisons, Software reliability.

[TRST84] Trstensky, "An Alternative Index for the Reliability of Telecommunication Networks," IEEE Transactions on Reliability, pp. 343-346, October 1984.

Discusses a new index for evaluating the reliability of telecommunication networks which compares the effectiveness of the modelled actual system with that for an ideal system. The new index is not only general, but the procedure for its calculation allows more consistent choice of the special indexes of reliability. It is suggested that some previous indexes can be derived from this as special cases.

KEY WORDS: Telecommunication network, Network topology, System effectiveness.

[TSUB86] Tsubotani, Monden, Tanaka, and Ichikawa, "A High Level Language Based Computing Environment to Support Production and Execution of Reliable Programs," IEEE Transactions on Software Engineering, pp. 134-146, January 1986.

Presents an environment which involves a debugging tool to help detect logic errors and remove them efficiently. The debugging tool is supported by a special architecture named SPRING which was originally developed for reliable execution of Ada or Pascal programs. The details of SPRING architecture are described, and the implementation of high level debugging on the SPRING architecture is discussed.

KEY WORDS: Ada, High level language architecture, Software reliability.

[TYRR86] Tyrrell, Holding, "Design of Reliable Software in Distributed Systems Using the Conversation Scheme," IEEE Transactions on Software Engineering, pp. 921-928, September 1986.

Examines the problems of error detection and recovery in a number of concurrent processes expressed as a set of communicating sequential processes (CSP). A method is proposed which uses a Petri net model to identify formally both the state and the state reachability tree of a distributed system. These are used to define systematically the boundaries of a conversation including the recovery and test lines which are essential parts of the fault-tolerant mechanism. Techniques described are implemented using the occam programming language. Application of this method is shown by a control example.

KEY WORDS: Communicating sequential processes, Concurrent processes, Distributed systems, Fault-tolerant software, Occam, Petri-nets.

[UPAD86] Upadhyaya, Saluja, "A Watchdog Processor Based General Rollback Technique with Multiple Retries," IEEE Transactions on Software Engineering, pp. 87-95, January 1986.

Discusses a general rollback strategy with 'n' (n >= 2) retries which takes into consideration multiple transient failures as well as transients of long duration. Ways to derive practical values of 'n' for a given program are discussed. The use of a watchdog processor as an error detection tool to initiate recovery action through rollback--low error latency are proposed. Merging watchdog processor with rollback recovery technique are described for enhancing the overall system reliability.

KEY WORDS: Error detection, Error latency, Recovery time, Rollback recovery, Transient errors.

[VANC86] van Caneghem, Warren, Logic Programming and Its Applications, Apex Publishing Co., Norwood, NJ, 1986.

Presents a collection of papers on logic programming and Prolog. Some papers on the use of Prolog for verifying designs in VLSI or software are included.

KEY WORDS: Verification, Concurrent.

[VANE79] van Emden, "Programming with Verification Conditions," IEEE Transactions on Software Engineering, pp. 148-159, March 1979.

Contains an exposition of the method of programming with verification conditions. It is shown that the method has the advantage of simplicity and flexibility over one proposed by Dijkstra. The method is directly based on Floyd's inductive assertions. Although the method has no use for the sequencing primitives of 'structured programming', it is highly secure and systematic.

KEY WORDS: Control structure, Correctness-oriented programming, Invariant assertions, Verifications.

[VENK85] Venkatraman, Piatkowski, "A Formal Comparison of Formal Protocol Specification Techniques," Proceedings of the IFIP WG 6.1 Fifth International Workshop on Protocol Specification, Testing and Verification, pp. 1-20, North-Holland Publishing Company, May 1985.

Describes the principal attributes which will be used to characterize and compare the protocol specification techniques. State models, sequence models, temporal logic models are examined. A comparison of all previously detailed techniques is presented.

KEY WORDS: Estelle, Specification techniques, Network systems, Petri nets, Temporal logic.

[VISS85] Vissers, Logrippo, The Importance of the Service Concept in the Design of Data Communications Protocols, pp. 1-15, Proceedings of the IFIP WG 6.1 Fifth International Workshop on Protocol Specification, Testing and Verification, North-Holland Publishing Company, May 1985.

Analyzes the level of recognition that the service concept has acquired in the world of protocol designers. Opposition against the concept and some significant cases of misuse are expounded and refuted. The authors argue for an increased role of the service, and its underlying architectural concepts, as the proper bases for designing protocol systems as well as suitable specification, verification, and testing techniques.

KEY WORDS: Protocol designers, Architecture, Specification, Verification, Service concepts.

[VOGE80] Voges, Gmeiner, and von Mayrhauser, "SADAT--An Automated Testing Tool," IEEE Transactions on Software Engineering, pp. 286-290, May 1980.

Describes the automated testing tool SADAT, which supports the testing of single Fortran modules. The different functions which are integrated in this system are explained, the usage of the tool is demonstrated, and some output results discussed. Summary of the special benefits of the SADAT system are presented. The history and present status of the system are outlined.

KEY WORDS: Automated test systems, Dynamic analysis, Program testing, Static analysis.

[VOSS80] Voss, "Using Predicate/Transition - Nets to Model and Analyze Distributed Database Systems," IEEE Transactions on Software Engineering, pp. 539-544, November 1980.

Proposes a net model for decentralized control of user accesses to a distributed database. It is developed in detail for the case of updating distributed copies of a single database. Predicate/Transition - Nets provide suitable means for concise representation of complex decentralized systems and their rigorous formal analysis. It is demonstrated how net models can be constructed and interpreted in a natural manner. The modeled Distributed Database system is deadlock-free and guarantees a consistent database as well as a fair and effective service to the user.

KEY WORDS: Concurrency, Deadlock-free, Decentralized control, Distributed databases, Petri nets, Predicate/transition nets.

[WALT81] Walters, Gray, and Thompson, "Self-Diagnosing Cellular Implementations of Finite-State Machines," IEEE Transactions on Computers, pp. 953-959, December 1981.

Shows that cellular spaces possess properties favorable to reconfiguration. The implementation of arbitrary finite-state machine in self-diagnosing cellular spaces is demonstrated. The results cover single cell failures caused by erroneous state transitions or by erroneous outputs. Results demonstrate that the control structure of any computing device can be implemented as a self-diagnosing entity without hard core.

KEY WORDS: Decomposition for diagnosability, Design for diagnosability, Fault detection.

[WASS85] Wasserman, "Extending State Transition Diagrams for the Specification of Human-Computer Interaction," IEEE Transactions on Software Engineering, pp. 699-713, August 1985.

Shows the derivation of the USE (User Software Engineering) transition diagrams based on perceived shortcomings of the 'pure' state transitions diagram approach. In this way, the features of the USE specification notation are presented and illustrated. The automated tools that support direct execution of the specification are described. Specification is easily encoded in a machine-processable form to create an executable form of the computer-human interaction.

KEY WORDS: Executable specifications, Interactive information systems, Rapid prototyping, Software development methodology

[WASS86] Wasserman, Pircher, and Shewmake, "Building Reliable Interactive Information Systems," IEEE Transactions on Software Engineering, pp. 147-156, January 1986.

Presents User Software Engineering (USE)--a methodology with supporting tools, for the specification, design, and implementation of interactive information systems. The USE transition diagrams and the formal specification approach are described. It is shown how these tools and techniques aid in the creation of reliable interactive information systems.

KEY WORDS: Interactive information systems, Software development methodology, Software reliability.

[WEBE86] Weber, Ehrig, "Specification of Modular Systems," IEEE Transactions on Software Engineering, pp. 784-798, July 1986.

Presents a modularity concept for structuring large software systems. The concept enforces an extreme modularity discipline which is meant to be applied to tightly control side-effects in the execution of systems constructed of independently developed modules. A family of specification languages is introduced whose members are based on the modularity concept. The construction of large software systems as interconnections of modules is shown to lead to manageable system structures and to new degrees of freedom in the structuring of the software

development process. The suitability of the modularity concept has been evaluated in a large software project for the development of a database management system.

KEY WORDS: Data abstraction, Informal and procedural and algebraic specifications, Software development, Software structuring.

[WEIS80] Weischedel, Practical Suggestions for Writing Understandable, Correct Formal Specifications, pp. 1-57, Department of Computer and Information Sciences, University of Delaware, Newark, DE, August 1980.

Describes the three major classes of formal specification languages and argues that understandability is critical for formal specifications. Reasons why they are difficult to understand, and practical suggestions for making them more understandable are given. The suggestions are illustrated by the specification of a pattern-matching facility. Three practical suggestions for checking the correctness of formal specifications are presented.

KEY WORDS: Formal specification languages, Structured programming methodology.

[WEST86] West, Protocol Validation by Random State Exploration, pp. 1-26, IBM Zurich Research Laboratory, 8803 Rueschlikon, Switzerland, March 7, 1986.

States that the availability of the validated, executable Session Layer definition, together with the validation results has made it possible to evaluate an alternative methodology for automated protocol validation whereby a systematic state exploration is performed by executing a random sequence of interaction sequences between protocol machines rather than a systematic, exhaustive state exploration. The exhaustive state-exploration validation technique and the problems that have led to evaluation of random state exploration are discussed. The random exploration technique is outlined and the results of applying it to the validation of the OSI Session layer are presented. The paper concludes with a discussion of the effectiveness of the random state exploration, and of the implications of the results to protocol testing.

KEY WORDS: Protocol validation, State exploration, Protocol specification, Session layer, Specification, Random exploration.

[WEYU80] Weyuker, Ostrand, "Theories of Program Testing and the Application of Revealing Subdomains," IEEE Transactions on Software Engineering, pp. 236-246, May 1980.

Examines the theory of test data selection proposed by Goodenough and Gerhart. Concepts of a revealing test criterion and a revealing subdomain are proposed, which provide a basis for constructing program test. Three programs are discussed and tested using the notions discussed.

KEY WORDS: Program testing, Software error detection, Software reliability, Theory of testing.

[WHIT80] White, Cohen, "A Domain Strategy for Computer Program Testing," IEEE Transactions on Software Engineering, pp. 247-257, May 1980.

Presents a testing strategy designed to detect errors in the control flow of a computer program, and the conditions under which this strategy is reliable. The input space is partitioned into a set of domains. The testing strategy generates test points to examine the boundaries of a domain to detect whether a domain error has occurred.

KEY WORDS: Control structure, Domain errors, Software reliability, Software testing.

[WITT81] Wittie, "Communications Structures for Large Networks of Microcomputers," IEEE Transactions on Computers, pp. 264-272, April 1981.

Compares nine network interconnection schemes and introduces 'dual-bus hypercubes', a cost effective method of connecting thousands of dual-port single-chip microcomputers into a room-sized information processing system, a 'network computer'. Each network node is a chip containing memory and a pair of processors for tasks and input/output. Nodes are linked by shared communication buses.

KEY WORDS: Bus topologies, Cube-connected cycles, Dual-bus hypercubes, Hypercube spanning buses.

[WOOD80] Woodward, Hedley, and Hennell, "Experience with Path Analysis and Testing of Programs," IEEE Transactions on Software Engineering, pp. 278-285, May 1980.

Suggests a hierarchy of structural test metrics to direct the choice of a path to use as test cases, and to monitor the coverage of test paths. Experience with the use of 'allegations' to circumvent the problem that many of the chosen paths may be infeasible in the sense that no test data can ever execute them is reported.

KEY WORDS: Allegations, Infeasible paths, Path testing, Test metrics.

[WU81] Wu, Liu, "A Cluster Structure as an Interconnection Network for Large Multimicrocomputer Systems," IEEE Transactions on Computers, pp. 254-263, April 1981.

Presents a cluster structure, characterized by a set of structure parameters and a set of interconnection functions, used as a conceptual interconnection scheme for large multimicrocomputer systems. It is shown that three popular interconnection structures (hypercube, hierarchy, and tree structures) are examples of the cluster structure. Two communication problems (traffic congestion and message delay) are analyzed. The analysis provides a way to understand structural properties such as complexity, capacity, and limitation. Topological optimization is presented to show how interconnection limitation can be minimized.

KEY WORDS: Interconnection network, Topological optimization.

[WUU85] Wuu, Bernstein, "False Deadlock Detection in Distributed Systems," IEEE Transactions on Software Engineering, pp. 820-821, August 1985.

Refers to detecting a nonexistent deadlock in distributed systems as false deadlock detection. It is shown that false deadlock will never occur in a system of two-phase locking transactions. Also, an algorithm to avoid false deadlock detection when transactions are not two-phase locking is described.

KEY WORDS: Distributed system, False deadlock, Transaction-wait-for-graph, Two-phase locking.

[YAMA85] Yamada, Osaki, "Software Reliability Growth Modeling (SRGM): Models and Applications," IEEE Transactions on Software Engineering, pp. 1431-1437, December 1985.

Summarizes existing SRGM's described by the nonhomogeneous Poisson processes. The maximum-likelihood estimations based on SRGM's for software reliability data analysis and software reliability evaluation are examined. Application examples using actual software error data are presented.

KEY WORDS: Error detection rate, Maximum-likelihood estimation, Nonhomogeneous Poisson processes, Software reliability analysis.

[YANN86] Yanney, Hayes, "Distributed Recovery in Fault-Tolerant Multiprocessor Networks," IEEE Transactions on Computers, pp. 871-879, October 1986.

Develops a methodology for characterizing dynamic distributed recovery in fault-tolerant multiprocessor systems using graph theory. Distributed recovery, which is intended with no central supervisor, depends on the cooperation of a set of processors to execute the recovery function, since each processor is assumed to have only a limited amount of information about the system as a whole. Facility graphs, whose nodes denote the system components and whose edges denote interconnection between components, are used to represent multiprocessor systems and error conditions. A general distributed recovery strategy 'R', which allows global recovery to be achieved via a sequence of local actions, is given. 'R' recovers the system in several steps in which different nodes successively act as the local supervisor. 'R' is specialized for two classes of systems: loop networks and tree networks. For each of these cases, fault-tolerant designs and their associated distributed recovery strategies are presented.

KEY WORDS: Distributed recovery, Fault tolerance, Fault-tolerant multiprocessor systems, Reconfiguration.

[YAO79] Yao, "Some Complexity Questions Related to Distributive Computing," Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, pp. 209-213, Atlanta, GA, 1979.

Presents an evaluation of a problem of minimizing information transfer. Variations of Abelson's are used to help solve the problems by calculating the bounds for the minimal exchange of information.

KEY WORDS: Distributive computing, Complexity, Probabilistic models.

[YAU80] Yau, Chen, "An Approach to Concurrent Control Flow Checking," IEEE Transactions on Software Engineering, pp. 126-137, March 1980.

Proposes a control flow checking scheme capable of detecting control flow errors of programs resulting from software coding errors, hardware malfunctions, or memory mutilation during execution of program. Capabilities, limitations, implementations and over-head of this approach are presented.

KEY WORDS: Capabilities, Concurrency, Control errors, Control flow checking, Program design.

[YAU81] Yau, Grabow, "A Model for Representing Programs Using Hierarchical Graphs," IEEE Transactions on Software Engineering, pp. 556-573, November 1981.

Presents a hierarchical graph model for programs based on the concepts of recursive graphs (RG's) and Codd relations. Purpose is to clearly represent the structure of a program implemented in a structured language so that the program can be analyzed and modifications can be clearly specified. Model for an example Pascal program is given.

KEY WORDS: Codd relations, Graph grammar, Hierarchical graph, Recursive graph.

[YAU85] Yau, Collofello, "Design Stability Measures for Software Maintenance," IEEE Transactions on Software Engineering, pp. 849-856, September 1985.

Presents design stability measures which indicate the potential ripple effect characteristic due to modifications of the program at the design level. Measures that can be generated at any point in the design phase of the software life cycle are examined. Validation of these measures and future research efforts which incorporates the design stability measures as well as other measures are discussed.

KEY WORDS: Design stability measures, Program modifications, Software maintenance.

[YAU86] Yau, Tsai, "A Survey of Software Design Techniques," IEEE Transactions on Software Engineering, pp. 713-721, June 1986.

Surveys important techniques for software design, including architectural and detailed design stages. Recent advances in distributed software system design methodologies are investigated. Various design verification and validation techniques are examined. Current software metrics and error-resistant software design methodologies are discussed.

KEY WORDS: Design methodologies, Design representation, Design verification and validation, Distributed software system design, Error-resistant software design, Software metrics.

[YONE77] Yonezawa, Modelling Distributed Systems, Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA, 1977.

Presents ideas and techniques used in modelling distributed systems and its application to Artificial Intelligence. A model of distributed systems and its specification and verification techniques are discussed. An example of an airline reservation system is introduced to illustrate specification and verification techniques.

KEY WORDS: Multiprocessor information processing systems, Distributed systems, Artificial intelligence, Modelling systems.

[YUAS85] Yuasa, Nakajima, "IOTA: A Modular Programming System," IEEE Transactions on Software Engineering, pp. 179-187, February 1985.

Presents a highly interactive programming system which supports hierarchical and modular program development with abstraction mechanisms. By taking advantage of abstraction mechanisms, the system provides a "truly modular" environment, in which modules are constructed, debugged, verified, and compiled in a module-by-module fashion. Such an environment naturally requires system management of the information concerning ongoing program development, in the form of module databases. As a result, further problems arise as to how to modify the information in efficient and consistent ways. Design objectives for modular programming systems are described by focusing on such issues as information management, interactive construction and modification of modules, separate processing, specification and verification, and support for cooperative program development.

KEY WORDS: Data abstraction, Formal specification, Program verification.

[ZAFI80] Zafiropulo, West, Rudin, Cowan, and Brand, "Towards Analyzing and Synthesizing Protocols," IEEE Transactions on Communications, pp. 651-660, April 1980.

Presents techniques for both the detection of errors in protocols and for prevention of errors in their design. The methods have been used successfully to detect and correct errors in existing protocols. A technique based on a reachability analysis is described which detects errors in a design. The types of errors handled are state deadlocks, unspecified receptions, nonexecutable interactions, and state ambiguities. An interactive design technique is described that prevents design errors.

KEY WORDS: Reliable communications, Error-free protocols, Error detection and correction.

[ZAVE76] Zave, "On the Formal Definition of Processes," Proceedings of the 1976 International Conference on Parallel Processing, pp. 35-42, 1976.

Presents a comparison between two models of logically concurrent asynchronous interacting processes, one that communicates through shared variables which cannot be formalized and one based on message communication which can. Results show that the second is better in offering computational structures.

KEY WORDS: Shared variables, Concurrent processes, Formalization, Synchronization.

[ZAVE86] Zave, Schell, "Salient Features of an Executable Specification Language and Its Environment," IEEE Transactions on Software Engineering, pp. 312-325, February 1986.

Presents the executable specification language PAISLey and its environment as a case study in the design of computer languages. Conclusions are drawn concerning the differences between executable specification languages and programming languages, and potential uses for PAISLey are given.

KEY WORDS: Distributed systems, Executable specifications, Functional programming, Operational approach to software development, Parallelism, Real-time systems.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 87-3559	2. Performing Organ. Report No.	3. Publication Date MAY 1987

4. TITLE AND SUBTITLE

Annotated Bibliography on Reliable System Design

5. AUTHOR(S)

Wayne H. McCoy, Kathleen M. Roessing, Mary K. Ruhl

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

The difficulty in assuring some level of fault-tolerance, reliability, safety, availability or survivability in large, complex distributed system has long been recognized. Techniques are now emerging that try to address this issue in system design, including formal description, design tools, automatic implementation and system simulation. This bibliography contains brief summaries of 350 papers from various computer science and engineering journals, books, dissertations and technical report in the years 1972-1987, on these and related topics.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

automatic implementation; distributed computer systems; fault-tolerance; formal description; reliability, system design

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

115

15. Price

$18.95