ELSEVIER

Decision Support Systems 21 (1997) 307–323

Decision Support Systems

A survey of tools for the validation and verification of knowledge-based systems: 1985–1995 [1]

Stephen Murrell [a], Robert T. Plant [b,*]

[a] *Department of Electrical and Computer Engineering, University of Miami, Coral Gables, FL 33124, USA*
[b] *Department of Computer Information Systems, University of Miami, Coral Gables, FL 33124, USA*

Abstract

This paper presents the findings of a survey of software tools built to assist in the verification and validation of knowledge-based systems. The tools were identified from literature sources from the period 1985–1995. The tool builders were contacted and asked to complete and return a survey that identified which testing and analysis techniques were utilised and covered by their tool. From these survey results it is possible to identify trends in tool development, technique coverage and areas for future research. © 1997 Elsevier Science B.V.

Keywords: Verification; Validation; Verification tools; Validation tools; Testing; Knowledge-based system testing

1. Introduction

A literature search was made of journal and conference publications for the period 1985–1995 for material on tools for the validation and verification of knowledge-based systems. Included in this survey were the workshop notes of the seven AAAI/IJCAI workshops on validation and verification, as well as EUROVAV and the ECAI workshops. This search identified 40 tools.

In order to evaluate the tools' characteristics, the tool creator was asked to assess their tool against a list of 145 testing techniques as identified in the literature. One hundred and twenty-nine of these were taken from the EPRI survey and assessment of conventional software verification and validation techniques [15]. EPRI categorised their 129 techniques into three major categories: Requirements/Design Methods, Static Testing, and Dynamic Testing. These in turn being broken down into their own subcategories. The classification scheme of EPRI was extended to cover testing techniques found in the validation and verification literature associated with testing knowledge-based systems, this included 16 additional techniques. A table

* Corresponding author: Tel.: +1-305-284-1963; fax: +1-305-284-5161; e-mail: rplant@umiami.miami.edu

[1] Research Sponsored in part by SBIR Contract Number DAHH01-96-C-R013.

PII S0167-9236(97)00047-X

Table 1
A summary of the testing techniques used in the survey

Description of major classes of techniques	
V&V classes/subclasses	Description
1.0 Requirements and design evaluation: Evaluation of the adequacy of the requirements and design	
1.1 Formal methods	Use of mathematical and logic formalisms for rigorous and unambiguous representation of initial system documents, including the requirements document, the requirements specification, and the design document. These representations may then be subjected to formal (sometimes automated) deductive reasoning to detect anomalies or defects such as 'correctness', 'contradiction', 'completeness', 'deadlock', and 'consistency'.
1.2 Semiformal methods	Techniques whose normal, forced, or prescribed method of use effectively constrain users in their specification of requirements or designs, such that various problems of expression and elaboration can be avoided or reduced. Such problems include aspects of ambiguity, incompleteness, inconsistency, contradiction, and 'ill-formedness'. These techniques, while often based on mathematical and logic formalisms, do not explicitly require the user to specify or use such formalisms. The techniques are typically embedded in function-rich, computer-based environments which provide sophisticated graphical representations of user input and often permit the user specifications to be simulated or animated to permit assessment of time and performance characteristics.
1.3 Formalized reviews	Reviews by various specified personnel of requirements or design products. The reviews follow a detailed checklist or set of procedures.
1.4 Traceability analyses	Determination of correspondence between individual requirements and design elements, between individual requirements and implemented system features, or between design elements and implemented system features. The two types of problems identified by these analyses are (1) unfulfilled requirements or design elements, and (2) unintended (unmotivated) design or implementation elements
2.0 Static testing: Examination of the program source code or some transformation or mapping to support various kinds of analyses (e.g., unused code, inconsistencies, anomalies)	
2.1 Algorithm analysis	Analysis of the overall algorithm(s) for achieving required function.
2.2 Control analysis	Analysis of the control characteristics of the program.
2.3 Data analysis	Analysis of the data specifications and flow of the program.
2.4 Fault/failure analysis	Analysis for particular or any kind of fault or failure, and/or an analysis to determine how particular faults and failures could occur.
2.5 Inspections	Examination of various aspects of the program by various personnel.
3.0 Dynamic testing: Actual execution of the program, generating output for sets of input conditions	
3.1 General testing	Generic and statistical methods for exercising program.
3.2 Special input testing	Special methods for generating test-cases to explore the domain of possible system inputs.
3.2.1 Random testing	Selecting test-cases according to some random statistical procedure.
3.2.2 Domain testing	Analysis of the boundaries and partitions of the input space and selection of interior, boundary, extreme, and external test-cases as a function of the orthganality, closeness, symmetry, linearity, and convexity of the boundaries.
3.3 Functional testing	Selecting test cases to assess required functionality of program.
3.4 Realistic testing	Choosing inputs/environments comparable to intended installation situation.
3.5 Stress testing	Choosing Inputs/environments which stress the design/implementation of the code.
3.6 Performance testing	Measuring various performance aspects for realistic input.
3.7 Execution testing	Actively following (and possibly interrupting) sequence of program execution steps.
3.8 Competency testing	Comparing the output 'effectiveness' against some preexisting standard
3.9 Active interface testing	Testing Various interfaces to the program
3.10 Structural testing	Testing selected aspects of the program structure
3.11 Error-introducing testing	Systematically introducing errors into the program to assess various effects
4.0 KBS error checks: Techniques that specifically cover the potential for errors in the structures associated with knowledge-based systems (e.g., rules, frames, etc. These may be logical, semantic, methodological, knowledge or data-induced	

Categories 1.0–3.11 are reproduced from table 5.2-2, pp. 5–6 of Ref. [15].

of concise definitions for these categories is given in Table 1, and the techniques are based on Refs. [1–12] [13,14,16–23] [24–33] [34–43] [44–53] [55–63].

The survey was initially sent by mail to the authors of the 40 tools identified in the literature search. Included in this mailing was a brief outline of each of the 145 testing techniques as well as references for each technique. Authors were also given instructions on how to complete the survey, which amounted to asking the tool builder to mark off each technique their tool covered. The survey respondent was free to annotate their comments if they deemed it necessary. A second round of survey requests was posted to the tool builders via electronic mail in order to ease and maximise survey returns.

2. Results

The survey resulted in 33 responses. The tools for which responses were received are outlined in Appendix A together with the major reference for that tool.

The seven tools that were identified in the literature but for which responses could not be obtained are given in Appendix B, with outlines and citations.

The 33 responses were taken and the data compiled into a matrix, shown in Appendix C. The matrix uses an '×' to identify a positive response confirming a tool performs a given technique, other entries in the table such as a number or star correspond to annotated comments from the tool builder. These are explained in Appendix D. The tools are ordered by year of the literature citation for that tool.

3. Data

The data as provided by the tool builders is as accurate an approximation of the tools capabilities as can be achieved through this form of standardised data collection format. An original intent was to develop the survey from the literature [54], however, this data source often does not detail all of a system's capabilities. This problem is inherent in the nature of the research literature that often only allows publication of new advances and does not permit authors to document aspects that may be found elsewhere in the literature.

4. Analysis

In considering the results of the survey, it can be seen that there are distinct areas in which work has been concentrated, primarily in the areas of control flows. The techniques that have had most coverage are listed in Table 2. The table indicates that many of the tools have significant overlap and that they vary only incrementally since VERITE in 1990 (it is also assumed that EVA also performed many of these tasks as early as 1987 but the EVA workers did not respond to the survey).

In contrast to the multiple systems that cover the techniques listed in Table 2, many techniques have

Table 2
The most frequently utilised error search techniques

Technique	Number of tools employing the technique
Static testing	
Algorithm analysis	
Cause–effect analysis	11
Control analysis	
Control flow analysis	15
Data analysis	
Dependency analysis	13
Fault / failure analysis	
Anomaly testing	11
Dynamic testing	
General testing	
Unit/module testing	13
System testing	13
Structural testing	
Path testing	10
Knowledge-based checks	
Logic checker	22
Structure checker	17
Omission checker	16
Semantics checker	14
Rule statisfiability	13
Rule refiner	12
Test case generator	10

not been included in any tool, and these are listed in Table 3. These techniques are clearly an area richly deserving further development in terms of there applicability and use in knowledge-based systems validation and verification. However, in some cases, the absence of certain features is due to an absence of fundamental research into the underlying theory and the practical applicability of those techniques. This is particularly true of the categories covering formal and semiformal methods, and error induction.

Table 3
The least frequently utilised error search techniques

Technique	Number of tools employing the technique
REQ / Design methods	
Formal methods	
EHDM	0
VDM	0
Concurrent system calculus	0
Semiformal methods	
Ward–Mellor method	0
Hatley–Pirbhai method	0
Harel method	0
Extended systems modeling language	0
Systems engineering	0
Systems requiring engineering methodology	0
Critical timing/flow	0
Simulation language	0
PSL/PSA	0
Static testing	
Algorithm analysis	
L–D relation methods	0
Algebraic specification	0
Data analysis	
Concurrency analysis	0
Fault / failure analysis	
Fault-tree analysis	0
Inspections	
Standards audit	0
Dynamic testing	
General testing	
Software reliability	0
Special input testing	
Category partition method	0
Revealing subdomains	0
Uniform boundary testing	0
Gaussian whole program testing	0
Gaussian boundary testing	0
Realistic testing	
Benchmarking	0
Human factors experimentation	0
Stress testing	
Stress/accelerated L	0
Queue size, etc.	0
Active interface testing	
Organisational impact analysis/testing	0
Structural testing	
Linear code sequence	0
Error introduction	
Mutation testing	0

5. Comments and conclusions

It has been the aim of this paper to draw together the research and development in the area of tools for knowledge-based systems, covering the period 1985–1995. The research has shown that even though significant progress has been made by many researchers in the area in this decade, there has also been much duplication of effort. The paper provides an indication of the areas in which researcher can provide practitioners with valuable tools for validation and verification of knowledge-based systems where currently there are none, these primarily being formal techniques, useful in the early stages of the development life cycle.

	TOOL	ESC	CRSV-CLIPS	COVADIS	RITCaG	WRAPPING	KRUST	VERITE	KR-FOCL	CONKRET	VASIT	FAAK-MKAT	MELODIA	PROLOGA	CLINT	VVR	COVER	SACCO	SYCOJET	PJHUNTER	VITAL	KVAT	VALIDATOR	VALID	KB Reducer3	IMVER
	FAULT/FAILURE ANAL.																									
57	Failure Mode, Effects, Causality Analysis				X										X											
58	Criticality Analysis									X					X				X							
51	Hazards/Safety Analysis											X														
40	Anomaly Testing							X		X				X	X		X	X		X	12				X	
29	Fault-Tree Analysis																									
12	Failure Modelling									X											13					
51	Common-Cause Failure Analysis									X					X						14					
	INSPECTIONS																									
11	Informed Panel Inspections																							X		
15	Structured Walk-throughs										X															
42	Formal Customer Review									X											15			X		
35	Clean-Room Techniques																									
35	Peer Code Checking										X														X	
14	Desk Checking										X														X	
40	Data Interface Inspection										X															
41	User Interface Inspection									X																
14	Standards Audit																									
39	Requirements Tracing				X																					
	DYNAMIC TESTING																					X				
	GENERAL TESTING																									
40	Unit/Module Testing				X			X	X	X	X	X		X	X		X			X						
14	System Testing				X		X	X	X	X	X	X			X		X			X	16					
12	Compilation Testing		X					X		X							X									
5	Reliability Testing						X			X						X			X							
5	Statistical Record-Keeping								X			X									17					
8	Software Reliability Estimation																				17					
39	Regression Testing									X		X			X			X			17				2	
27	Metric-Based Testing								X		X										17					
14	Ad-hoc Testing						X								X						17					
	SPECIAL INPUT TESTING																									
3	Random Testing				X							X								X						
3	Uniform Whole Program Testing				X																					
3	Uniform Boundary Testing																									
3	Gaussian Whole Program Testing																									
3	Gaussian Boundary Testing																									
4	Domain Testing								X					X				X	X	X	18					
37	Equivalence partitioning				X										X											
37	Boundary Value Testing													X												
45	Category Partition Method																									
61	Revealing Subdomains Method																									
	FUNTIONAL TESTING																					X				
23	SpecificFunctional Requirement Testing				X			X									X			X		X				
48	Simulation Testing															X					18					
12	Model Based Testing														X	X										
42	Assertion Testing																									
33	Heuristic Testing				X																					
	REALISTIC TESTING																					X				
51	Field Testing				X							X									19					
51	Scenario Testing						X			X		X								X	19	X				
26	Qualification/Certification Testing						X																			
40	Simulator-based Testing																									
51	Benchmarking																									
9	Human Factors Experimentation																									
	STRESS TESTING																				17					
38	Stress/Accelerated Life Test																				17					
14	Stability Analysis																				17					
33	Robustness Testing				X															X	17					
40	Limit/Range Testing				X									X				X	X		17					
40	Bottleneck Testing		X																		17					
4	Queue Size, etc.																				17					
	EXECUTION TESTING																									
14	Activity Tracing				X		X			X											17					
40	Incremental Execution													X							17					
39	Results Monitoring						X														17					
26	Thread Testing										X	X									17					

	TOOL	ESC	CRSV-CLIPS	COVADIS	RITCaG	WRAPPING	KRUST	VERITE	KR-FOCL	CONKRFT	VASTT	FEAAK-MKAT	MELODIA	PROLOGA	CLINT	VVR	COVER	SACCO	SYCOJET	PHUNTER	VITAL	KVAT	VALID	KB Reducer3	IMVER
	COMPETENCY TESTING																					X			
51	Gold Standard																			X	17	X			
30	Effectiveness Procedures						X														17				
51	Workplace Averages									X															
	ACTIVE INTERFACE TEST																								
40	Data Interface Testing				X											X					17				
41	User Interface Testing				X					X											17				
41	Information System Analysis													X							17				
34	Operational Concept Testing									X											17				
6	Organisational Impact Analysis																				17				
4	Transaction-flow Testing									X											17				
	STRUCTURAL TESTING																								
17	Statement Testing									X	X				X				X	X					
33	Branch Testing				X					X	X			X	X					X					
56	Path Testing				X					X	X			X	X	X				X					
33	Call-Pair Testing									X															
33	Linear Code Sequence and Jump																								
17	Test Coverage Analysis Testing						X							X		X			X	X					
4	Conditional Testing				X									X						X					
4	Data-Flow Testing										X					X				X	19				
	ERROR INTRODUCTION																								
14	Error Seeding				X					X															
51	Fault Insertion						X																		
40	Mutation Testing																								
	KBS ERROR CHECKS																								
1	Elicitation Checker											X		X							17			X	
53	Structure Checker							X	X				X	X		X3	X	X		X	17		X	X	X	X
53	Logic Checker		X	X	X			X	X				X	X	X	X3	X	X		X	17		X	X	X	X
53	Extended Structure Checker							X					X	X		X3	X			X	17					
53	Extended Logic Checker							X						X	X	X3	X			X	17			X		
53	Semantics Checker		X	X				X						X		X	X	X		X	17			X	X	
43	Omission Checker							X	X				X	X	X	X	X				17		X	X	X	X
16	Rule Refiner						X		X				X	X	X	X					17			X	X	
60	Control Checker												X			X					17			X		
1	Behavior Verifier							X								X	X		X		17					
24	Test Case Generator				X								X	X	X	X			X		17			X		
1	Uncertainty Checker															4					17					
16	Rule Satisfiability		X				X	X						X		X	X			X	17		X	X	X	X
34	Model-based Verifier														X	X					17					
16	Case-Based Testing															X					17			X		
31	Cluster Analysis																				17					

COCO 94	MVP CA 94	Reference	TECHNIQUE	TOOL Year	PREPARE 94	SOCRATES 94	IRS-CBR 94	DERIVER 94	TRUBAC 95
			REQ/DESIGN METHODS						
			FORMAL METHODS						
		12	General Reqs. Language Analysis						
		27	Mathematical Verification						
		52	EHDM						
		10	Z						
		27	VDM						
		40	Refine Specification Language					X	
		38	Concurrent System Calculus						
			SEMI-FORMAL METHODS						
		60	Ward-Mellor Method						
		21	Hatley-Pirbhai Method						
		19	Harel Method						
		7	Extended Systems Modelling Language						
		59	Systems Engineering Methodology						
		2	Systems Reqs. Engineering. Methodology						
		10	FAM						
		58	Critical Timing/Flow Analysis						
		20	Simulation-Language Analysis						
		29	Petri-Net Safety Analysis		X				
		55	PSL/PSA						
			FORMALISED REVIEWS						
		39	Formal Requirements Review						
		39	Formal Design Review						
			TRACEABILITY ANALYSIS						
		39	Requirements Tracing Analysis						
		58	Design Compliance Anal.						
			STATIC TESTING						
			ALGORITHM ANALYSIS						
	X	27	Analytic Modeling		X		X		X
	X	12	Cause-Effect Analysis		X				X
	X	28	Symbolic Execution						
	X	44	Decision Tables						
	X	47	Trace-Assertion Method						
	X	35	Functional Abstraction					X	
		47	L-D Relation Methods						
	X	35	Program Proving					X	
	X	25	Metric Analyses						
		18	Algebraic Specification						
	X	22	Induction-Assertion Method						
			CONTROL ANALYSIS						
	X	60	Control Flow Analysis		X	X			X
		39	State Transition Diagram Analysis			X			
	X	42	Program Control Analysis			X			
		49	Operational Concept Analysis						
	X	39	Call Structure Analysis						
		21	Worst-Case & Process Trigger Timing Analysis			X			
		58	Concurrent Process Analysis					X	
			DATA ANALYSIS						
		13	Data Flow Analysis		X				X
		54	Signed Directed Graphs		X	X			X
	X	14	Dependency Analysis		X	X			X
	X	46	Qualitative Causal Models		X				
	X	39	Look-up Table Generator		X				
	X	40	Data Dictionary Generator			X	X		
	X	39	Cross Reference List Generator		X		X		
		39	Aliasing Analysis						
		50	Concurrency Analysis						
	X	39	Database Analyzer			X			
	X	39	Database Interface Analyzer						
	X	12	Data-Model Evaluation						

COCO	MVP CA	Reference		TOOL	PREPARE	SOCRATES	IRS-CBR	DERIVER	TRUBAC
			FAULT/FAILURE ANALYSIS						
		57	Failure Mode, Effects, Causality Analysis				X		
	X	58	Criticality Analysis						
		51	Hazards/Safety Analysis						
	X	40	Anomaly Testing		X				X
		29	Fault-Tree Analysis						
		12	Failure Modelling						
	X	51	Common-Cause Failure Analysis						
			INSPECTIONS						
		11	Informed Panel Inspections				X		
	X	15	Structured Walk-throughs						
		42	Formal Customer Review						
		35	Clean-Room Techniques						
		35	Peer Code Checking						
		14	Desk Checking				X		
	X	40	Data Interface Inspection						
	X	41	User Interface Inspection				X		
		14	Standards Audit						
	X	39	Requirements Tracing						
			DYNAMIC TESTING						
			GENERAL TESTING						
	X	40	Unit/Module Testing		X		X		
		14	System Testing		X		X		X
		12	Compilation Testing				X		
	X	5	Reliability Testing						
		5	Statistical Record-Keeping						
		8	Software Reliability Estimation						
		39	Regression Testing				X		
		27	Metric-Based Testing						X
		14	Ad-hoc Testing				X		
			SPECIAL INPUT TESTING						
		3	Random Testing				X		X
		3	Uniform Whole Program Testing		X				
		3	Uniform Boundary Testing						
		3	Gaussian Whole Program Testing						
		3	Gaussian Boundary Testing						
	X	4	Domain Testing		X				X
		37	Equivalence partitioning						
		37	Boundary Value Testing						
		45	Category Partition Method						
		61	Revealing Subdomains Method						
			FUNTIONAL TESTING						
		23	Specific Functional Requirement Testing						X
		48	Simulation Testing						X
	X	12	Model Based Testing		X				X
	X	42	Assertion Testing						
		33	Heuristic Testing		X		X		
			REALISTIC TESTING						
		51	Field Testing						X
		51	Scenario Testing				X		
		26	Qualification/Certification Testing						
		40	Simulator-based Testing		X		X		X
		51	Benchmarking						
		9	Human Factors Experimentation						
			STRESS TESTING						
		38	Stress/Accelerated Life Test						
		14	Stability Analysis				X		
		33	Robustness Testing						
	X	40	Limit/Range Testing				X		
		40	Bottleneck Testing						
		4	Queue Size, etc						
			EXECUTION TESTING						
		14	Activity Tracing		X		X		X
		40	Incremental Executution		X				X
		39	Results Monitoring		X		X		X
		26	Thread Testing						X

COCO	MVP CA			TOOL PREPARE	SOCRATES	IRS-CBR	DERIVER	TRUBAC
			COMPETENCY TESTING					
		51	Gold Standard					
		30	Effectiveness Procedures					
		51	Workplace Averages					
			ACTIVE INTERFACE TEST					
	X	40	Data Interface Testing					
		41	User Interface Testing					
		41	Information System Analysis					
		34	Operational Concept Testing					
		6	Organisational Impact Analysis					
		4	Transaction-flow Testing					
			STRUCTURAL TESTING					
	X	17	Statement Testing	X				
	X	33	Branch Testing	X				
	X	56	Path Testing	X				X
	X	33	Call-Pair Testing					
		33	Linear Code Sequence and Jump					
	X	17	Test Coverage Analysis Testing	X				X
	X	4	Conditional Testing					
	X	4	Data-Flow Testing	X				X
			ERROR INTRODUCTION					
		14	Error Seeding	X				
		51	Fault Insertion	X				
		40	Mutation Testing					
			KBS ERROR CHECKS					
	X	1	Elicitation Checker					
	X	53	Structure Checker	X	X		X	X
X	X	53	Logic Checker	X	X		X	X
	X	53	Extended Structure Checker	X			X	
X	X	53	Extended Logic Checker				X	
X	X	53	Semantics Checker				X	
	X	43	Omission Checker	X			X	X
X	X	16	Rule Refiner				X	
	X	60	Control Checker		X		X	
	X	1	Behavior Verifier				X	X
	X	24	Test Case Generator			X		X
	X	1	Uncertainty Checker					
	X	16	Rule Satisfiability					X
	X	34	Model-based Verifier	X				
		16	Case-Based Testing			X		
	X	31	Cluster Analysis					

Acknowledgements

The authors would like to thank Dr. Lance Miller of SAIC for his insights on the issues surrounding the validation and verification of both knowledge-based systems and conventional systems, as well as Mr. Peter Kiss of SENTAR and ARPA for their support of an earlier stage in this research.

Appendix A. Annotated tool references

A.1. ESC

The Expert System Checker (ESC) is a decision table-based checker for rule-based systems. The system accepts rules expressed in an almost natural language, translating them into a more suitable internal representation; it checks for ambiguity, redundancy, and completeness of the rule base. The system also works with rule bases partitioned into multiple hierarchical levels [Brian Cragun, Harold Steudel, A Decision Table-Based Processor for Checking Completeness and Consistency, Int. J. Man Machine Stud. (26) (1987) 633–648].

A.2. CRSV-CLIPS

CRSV was originally a separate verification tool to assist in the development of expert systems in CLIPS (C Language Integrated Production System); it has since been incorporated into CLIPS. CRSV performs constraint and semantic checking (using metaknowledge), and checks efficiency, completeness, and syntactic correctness, as well as the more common logic tests [C. Culbert, R. Savely, Expert system verifications and validation, Proc. of First AAAI Workshop on V, V and Testing, Palo Alto, CA, August 1988].

A.3. COVADIS

COVADIS is very similar to KB-REDUCER, the main differences are that it accepts rules with a slightly more expressive syntax, and is mainly concerned only with consistency checking [M.C. Rousset, On the consistency of knowledge-bases: The COVADIS system, Proceedings of the ECA188 Conference, Mancher, Germany, 1988, pp. 79–84].

A.4. RITCaG

This tool is a test-case generator aimed at smaller rule-based systems. The system generates inputs designed to test different parts of the knowledge-base, together with their expected outputs, and checks that the system under analysis behaves as expected [U.C. Gupta, J. Biegel, RITCaG: A rule-based intelligent test case generator, Working Notes, AAAI Workshop on KBS, Verification, Validation and Testing, Boston, MA, July 1990].

A.5. WRAPPINGS

Wrappings is not a verification and validation tool in itself, but provides a framework that can contain

any or all of the standard V and V methods. Wrappings provides operational descriptions of all of the components of a system [K.L. Bellman, C. Landauer, The modeling issues inherent in testing and evaluating K-Based systems, Expert Systems Applications J. 1 (1990) 199–215].

A.6. KRUST

KRUST, like CLINT, is essentially a refinement system, which when presented with a test case that produces incorrect results, attempts to identify the cause of failure, and then produces refinements to the rule base (modifications to the rules) which would produce a correct solution, with minimal disruption [S. Craw, Automating the refinement of knowledge-based systems, PhD Dissertation, University of Aberdeen, Scotland, 1991].

A.7. COVER / VERITE

COVER requires that the knowledge base be converted into the COVER Rule Language, a canonical representation based on first order logic. COVER detects duplicate, subsumed, and redundant rules; it also identifies conflicts between rules, and uses a heuristic approach to assist developers in identifying deficiencies in the knowledge base. VERITE is an extension to COVER that uses graphics to assist the knowledge engineer in visualising faults in the system being examined [Alun Preece, Rajjan Shinghal, Aida Batarekh, Principles and practice in verifying rule-based systems, Knowledge Eng. Rev. 7 (2) (1992) 115–141].

A.8. KRFOCL

KRFOCL is restricted to backward-chaining classification expert systems. Given a large set of test cases together with their desired results, it partially automates the tracing of an error back to the incorrect rule that caused it. KRFOCL is generally capable of detecting missing, contradictory, unnecessary, and redundant rules, and superfluous or missing preconditions to rules [Michael J. Pazzani, A set covering approach to testing rule-based expert systems, Workshop Notes, AAAI, Boston, MA, 1990].

A.9. CONKRET

CONKRET is a tool for refining control knowledge, represented by meta-rules (expressed in the VETA language) that control the run-time creation of goals and strategies, rather than the rules themselves. CONKRET attempts to discover control deficiencies, such as following unnecessary goals or failing to follow necessary goals, and make corrections to the meta-rules [Beatriz Lopez, CONKRET: A control knowledge refinement tool, in: Marc Ayel and Jean-Pierre Laurent (Eds.), Validation, Verification and Test of Knowledge-Based Systems, Wiley, pp. 191–206].

A.10. VASTT

The Vitro Automated Structured Testing Tool introduces a methodology for using conventional VV&T for knowledge-based systems The system has two approaches to validation and verification embedded in their development methodology. The Functional testing of a formalised data flow diagram model of the specifications, and structural testing of an intermediate language representation of the source code. The structural testing focuses on control flows through complexity metrics, and improved standards of documentation [M. Steib, R. Small, C. Castells-Schofield, Tailoring VASTT for expert system verification, validation and testing, Workshop Notes: AAAI Workshop on V and V, 1991].

A.11. FFAAK

FFAAK (Feedback Facilitated Automatic Acquisition of Knowledge) is based on constructing a knowledge base in a way that guarantees consistency and certain other aspects of correctness, rather than the more traditional post-construction V and V techniques. FFAAK relies on the generation of test cases and prevents unreachable goals or values, unattainable rules, illegal input combinations, and illegal outputs [T.V. Cuda, C.P. Dolan, Tool-aided non-formal knowledge verification, Workshop Notes: AAAI Workshop on V and V, Anaheim, 1991].

A.12. MELODIA

Works on simple knowledge bases containing rules and constraints built from logical and relational operators, working variables and constraints. Its main purpose is to detect inconsistencies (i.e., the ability

to infer self-contradictory facts, or facts incompatible with the constraints), but is also capable of detecting redundant or subsumed rules [E. Charles, O. Dubois, MELODIA: Logical methods for checking K-bases, in: M. Ayel, J.P. Laurant (Eds), Validation, Verification and Test of Knowledge-Based Systems, Wiley, Chichester, 1991, Chap. 7, pp. 95–105].

A.13. PROLOGA

The PROcedural LOGic Analyzer is a decision table engineering workbench. The workbench is an interactive rule-based design tool for computer-supported construction and manipulation of decision tables [J. Vanthienen, Knowledge acquisition and validation using a decision table engineering workbench, World Congress of Expert Systems, 1991, pp. 1861–1868].

A.14. CLINT

CLINT systematically generates sets of test cases, on which the system under consideration is run, and the expert is asked to confirm the results. When a test fails, or an inconsistency is discovered, CLINT locates the cause of the failure, and automatically corrects the rule-base, synthesizing new rules from the old ones and the test case results [L. de Raedt, G. Sablon, Bruynooghe, Using Interactive Concept Learning for Knowledge-Base Validation and Verification and Test of KBS, Wiley, Chichester, 1991, Chap. 12, pp. 177–190].

A.15. VVR

VVR works on a rule-base expressed as simple if–then rules, together with semantic constraints expressed as logical formulae. It detects cyclic, redundant, and contradictory rules, and rules that may lead to violations of the semantic constraints [N. Zlatarova, VVR: A uniform framework for expert system knowledge bases verification, validation and refinement, Workshop Notes: AAAI Workshop on V and V, Anaheim, 1991].

A.16. SACCO / SYCOJET

SACCO and SYCOJET are complementary tools designed to perform testing and coherence checking of a knowledge base. SACCO is used in the analysis of the whole knowledge base in terms of its facts and rules. SACCO uses a conceptual model against which the knowledge base is checked for coherence. SYCOJET automatically builds test cases, assesses the results of the test cases against a quality threshold required for the system [Marc Ayel, Jean Pierre Laurent, SACCO–SYCOJET: Two different ways of verifying knowledge-based systems, in: Marc Ayel, Jean-Pierre Laurent (Eds.), Validation, Verification and Test of Knowledge-Based Systems, Wiley, pp. 62–76].

A.17. PATH-HUNTER and PATH-TRACER

Path Hunter applies structural testing to OPS5 style rule-based systems, it constructs a set of test inputs that should cause all rules to fire in all possible sequences. Path Tracer checks the sequences of rules that actually fire during a run, analysing the degree to which the set of all possible sequences are tested [A.D. Precce, C. Grossner, P.G. Chander, T. Radhakrishnan, Structural validation of expert systems using a formal model, Working Notes: Workshop on V and V of KBS at AAAI '93, Washington DC, August 1993, pp. 19–26].

A.18. VITAL

The VITAL workbench is aimed at supporting a V and V extension to the KADS methodology for domain and data knowledge. VITAL addresses the issues of integration and openness. VITAL supports the knowledge engineer in the task of integrating systems that have been created under different methodologies and through different tools and languages [Alain Rouge, Jean Yves Lapicque, Florent Brossier, Yves Lozinguez, Validation and verification of KADS data and domain knowledge, EUROVAV '93, Univ. Polit. de Madrid (UPM), Palma De Mallorca, Spain, 24–26 March 1993, pp. 69–83].

A.19. KVAT

The Knowledge Validation Tool is part of the Knowledge Engineering Workbench (KEW); a support environment for knowledge acquisition developed as an ESPRIT project. KVAT automates the process of running test cases (inputs and desired outputs, provided by an expert in the form of frames) for frame-based reasoning systems. It creates tables showing where test cases failed or succeeded, and is

suitable for incremental testing after additional knowledge acquisitions [O.J. Mengshoel, A tool for incremental K-validation in a K-engineering workbench, Proc. of the European Workshop on the Verification and Validation of KBS, EUROVAN '91, Cambridge, England, July 1991, pp. 133–146].

A.20. VALIDATOR

VALIDATOR is an interactive tool that performs validation in nine areas: Illegal use of reserved words; Rules that could never fire; Unused facts; Unused questions; Unused legal values; Repeated questions; Multiple Methods; Rules with illegal values; Inconsistent instantiations [Y. Kang, T. Bahill, A tool for detecting expert system errors, AI Expert, February 1990, pp. 42–51].

A.21. VALID

The VALID project provides a Common Conceptual Representation (CCR), which provides an (almost) universal system independent description language for knowledge-based systems, and a meta-validation language (VETA) providing system-independent primitives for accessing the objects, external components (such as file systems and utility programs), and a library of higher level functions. This creates an abstract, high level environment in which V and V tools may be designed; existing KBSs may be converted to the CCR, on which validation tools implemented in VETA operate [J. Cardevosa, N. Juisto, General overview of the VALID project, in: J. Cardenosa, P. Meseguer (Eds.), EVROVAV '93, Proceedings of the European Symposium on the Validation and Verification of Knowledge-Based Systems, pp. 53–67].

A.22. DERIVATION TOOL

This paper describes a formal approach to developing concurrent rule-based programs. The program derivation strategy starts with a formal specification of the problem. Specification refinement is used to generate an initial version of the program. Program refinement is then applied to produce a highly concurrent and efficient version of the same program. Techniques for deriving concurrent programs through either specification or program refinement have been described in previous literature. The main contribution of this paper consists of extending the applicability of these techniques to rule-based programs. The derivation process is supported by a powerful proof logic; a logic that recently has been extended to cover rule-based programs. The presentation centers around a rigorous and systematic derivation of a concurrent rule-based solution to a classic problem [G.-C. Roman, R.F. Gamble, W.E. Ball, Formal derivation of rule-based programs, IEEE Trans. Software Eng. 19 (3) (March 1993) 277–296].

A.23. KB-REDUCER3

Takes a rule-based system specified in terms of its set of input, intermediate and output variables and their domains, a constraint (a formula over those variables, that must always be true), and a set of rules. Rules have the format:

Formula → variable = expression

Formula → variable [i.e., variable = true]

Expressions involve arithmetic operations, the variables, and constraints; formulae involve the logical operator $\wedge$, $\vee$, $\neg$ and the relational operators $>$, $=$, $<$ applied to expressions). The rules must be stratified according to data dependencies. Using an algorithm based on an extension of the knowledge-base reduction technique, to detect inconsistencies, redundancies and incompleteness in the rule-base, together with some simpler faults, such as unattainable rules and assignments to input variables [K. Williamson, M. Dahl, Knowledge-based reduction for verifying rule bases containing equations, Working Notes, Vol. 6, Workshop on V and V of KBS at AAAI, Washington, DC, pp. 66–71], [A. Ginsberg, L. Rose, KB-reducer: A system that checks for inconsistency and redundancy in knowledge-bases, Technical Report, AT&T Labs, 1987].

A.24. IMVER

IMVER uses a compressed form (essentially using bit-vectors instead of unpacked arrays) of incidence matrix to represent simple propositional rules and detects subsumption, identity, and inconsistency anomalies, and examines the connectivity of rules, to detect circularity, dead ends, etc. [F. Coener, T. Berch-Capon, A. Kent, A binary-encoded incidence

matrix representation to support KBS verification, Working Notes, Vol. 7, Workshop on V and V of KBS at AAAI '94, pp. 84–93].

A.25. COCO

COCO is an incremental checking system that computes the consistency of a rule-base. An inconsistency is detected through the use of a conceptual model of the rule-base provided by an expert in the form of integrity constraints. The knowledge-base is consistent when the rules satisfy the constraints specified by the conceptual model. The second part of the tool 'X', assists the expert to identify why a rule base is inconsistent. This forms the COCO-X tool, an interactive refinement system [Stephane Loiseau, A method for checking and restoring the consistency of rule bases, Int. J. Human–Computer Stud. 40 (1994) 425–442].

A.26. MVP-CA

Multi-View Point Clustering Analysis is based upon the idea that it is generally not possible to group the rules in a rule-based system into any single meaningful structure; instead, different structurings of the rules are required, depending upon the viewpoint under consideration [M. Mehrotra, C. Wild, Multi-viewpoint clustering analysis, Working Notes, Vol. 6, Workshop on Verification and Validation of KBS at AAAI, Washington, DC, pp. 52–63].

A.27. SOCRATES

This tool (System for Optimizing and Checking Rules for Analysis and Translation to Expert Shells) is a prototype providing basic functionality of displaying the structure of variable declarations, facts and rules to the user in the form of templates and push buttons. Basic structure, logic and control checking are also provided [B. Traylor, U. Schwuttke, A. Quan, A tool for American verification of real-time expert systems, Working Notes, Vol. 7, Workshop on V and V of KBS at AAAI '94, Seattle, WA, August 1994, pp. 79–83].

A.28. IRS-CBR

IRS-CBR (Intelligent Information Retriever with a Case-Based Reasoner) is used as a front–end system on a personal computer to communicate with a main frame computer on which financial statistical databases are stored. IRS-CBR has two components: a knowledge-based reasoner for general problem solving and a case-based reasoner for using past cases [Terano, personal communication, 1995] [T. Terano, K. Kobayshi, Changing the traces: Refining a rule-base by genetic algorithms, IJCAI Workshop on V and V of Knowledge-Based Systems Notes 1995, Montreal, Canada, 19 August 1995].

A.29. TRUBAC

TRUBAC uses a dataflow approach to the validation, verification and testing of rule-based systems through an AND/OR representation of the knowledge-base. A series of rule base coverage measures are used to guide the selection of test data and determine the extent to which a test suite has covered the rule base. The coverage measures are based on an execution path within the DAG representation, corresponding to all the rules fired along a 'reasoning chain' executed by a given input [V. Barr, Rule-based coverage measures applied to testing rule-bases with uncertainty, IJCAI Workshop on V and V of Knowledge-based Systems Notes 1995, Montreal, Canada, 19 August 1995].

A.30. PREPARE

PREPARE takes a novel approach to the traditional problem of detecting inconsistent, redundant, subsumed, circular, and incomplete rules in a knowledge base. The rules are translated into a 'Predicate/Transition Net' to which simple pattern recognition techniques may be applied to detect the anomalies [D. Zhang, D. Nguyen, PREPARE: A tool for knowledge-base verification, IEEE Trans. Knowledge Data Eng. 6 (6) (December 1994) 983–989].

Appendix B. Tool systems for which no survey response was obtained

B.1. EVA

EVA, (the Expert systems Validation Associate) provides a consistent tool set for KBSs implemented in expert system shells. EVA detects dead end rules, unreachable rules, redundant rules, cyclic rules, in-

consistencies (possible firings of rules leading to incomplete deductions), failures to satisfy semantic constraints, missing rules (possible input combinations not covered). The system also generates test cases, through which the developer verifies the behaviour of the system, both in terms of single rules, and possible sequences of rules. It also provides some checking of the consistency of certainty factors [R.A. Stachowitz, C.L. Chang, T.S. Stock, J.B. Coombs, Building validation tools for knowledge-based systems, First Annual Workshop on Space Operations Automation and Robotics (SOAR '87), Houston, TX, 5–7 August 1987, pp. 209–216].

B.2. SPT

SPT (Schematic Programming Tool) is designed to increase the understandability of knowledge-based systems through a graphical representation (similar to a flow chart) of the knowledge embedded in them. A program (or schema) is built by connecting simple operations via arcs representing data and control flow. This construction is an interactive, graphics-based process. SPT allows a program to run step-by-step under the direct control of the developer, so that its behaviour may be accurately observed [R. Phelps, W. Aerts, Improving validation and verification of knowledge-based systems through naturally comprehensible flow representations, in: J. Cardenosa, P. Meseguer (Eds.), EUROVAV '93, Proceedings of the European Symposium on the Validation and Verification of Knowledge-Based Systems, pp. 295–309].

B.3. FEAT

FEAT takes a manually generated representation of a system as a directed graph. Nodes represent events or states of the system, arcs represent a causal relationship. FEAT simulates the system, detecting failures and determining what previous events led to them [S.W. French, C. Culbert, D. Hamilton, Experiences in improving the state of practice in verification and validation of knowledge-based systems, Workshop Notes, Sixth Annual Workshop on V and V: AAAI 1994, Washington DC, pp. 86–93].

B.4. EFG translator

This tool translates horn-clause rule-bases into an evidence flow graph representation. The performance of the graphs are then analysed through a simulator. Static analysis is performed on the graph for consistency checking, and dynamic sensitivity analysis and test case analysis is also performed through the simulator [L.A. Becker, P.G. Green, J. Bhutinager, Evidence flow graphs for V and V of expert systems, NASA Contractor Report 181810].

B.5. SYSIFE

This system supports both dynamic and static validation. The system attempts to detect flaws in the knowledge base by examining the contexts in which they occur. This is achieved through three functional agents (a student, an examiner and a jury) who interact to determine the systems constraints [P. Mazas, Designing knowledge validation through experimentation: The SYSIFE system, in: M. Ayel, J.P. Laurent (Eds.), Validation, Verification and Test of Knowledge-Based Systems, Wiley, 1991, pp. 119–145].

B.6. SAVES

SAVES examines coverage of rules within a knowledge base through the use of validation test cases. The system analyses the knowledge base to determine all possible values for the rule variables. These are then examined against test cases held in a repository. The results of the validation being statistically measured and presented to the developer [S. Smith, A. Kandel, Verification and Validation of Rule-based Expert Systems, CRC Press].

B.7. CHECKER

Checker, is part of MIDST (Mixed Inferencing Dempster Schafer Tool) an expert system shell that utilises mixed-initiative reasoning and uncertain reasoning. Checker has two components: Check-1 and Check-2. Check-1 is a rule comparison algorithm that detects redundancy, conflict, subsumption and unnecessary conditions. Check-2 detects possible semantic errors where two or more rules that have identical conditions but differ in their conclusions, as well as detecting circular rules, unsolvable conditions and unreachable conclusions [X. Yu, Biswas, CHECKER: An efficient algorithm for knowledge-based system verification, Proc. of the Third International Conference on Engineering Applications of A.I., IEA/AIE-90, 1990].

Appendix C. Tool / technique cross-reference matrix

Reference	TECHNIQUE	TOOL	ESC	CRSV-CLIPS	COVADIS	RTCaG	WRAPPING	KRUST	VFRITE	KR-FOCL	CONKRET	VASTT	FAAK-MKAT	MELODIA	PROLOGA	CLINT	VVR	COVER	SACCO	SYCOJET	PHUNTER	VITAL	KVAT	VALIDATOR	VALID	KB Reducer3	IMVER
		Year	87	88	88	90	90	90	90	90	91	91	91	91	91	91	91	91	91	91	93	93	93	93	93	94	94
	REQ/DESIGN METHODS																										
	FORMAL METHODS																										
12	General Reqs. Language Analysis															X			X								
27	Mathematical Verification																X		X						X		
52	EHDM																										
10	Z																		X						X		
27	VDM																										
40	Refine Specification Language																								1		
38	Concurrent System Calculus																										
	SEMI-FORMAL METHODS																										
60	Ward-Mellor Method																										
21	Hatley-Pirbhai Method																										
19	Harel Method																										
7	Extended Systems Modelling Language																										
59	Systems Engineering Methodology																										
2	Systems Reqs. Engineering. Methodology																										
10	FAM																								X		
58	Critical Timing/Flow Analysis																										
20	Simulation-Language Analysis																										
29	Petri-Net Safety Analysis																								X		
55	PSL/PSA																										
	FORMALISED REVIEWS																										
39	Formal Requirements Review																								X		
39	Formal Design Review																								X		
	TRACEABILITY ANALYSIS																										
39	Requirements Tracing Analysis				X																				X		
58	Design Compliance Anal.																								X		
	STATIC TESTING																										
	ALGORITHM ANALYSIS																										
27	Analytic Modeling				X						X	X				X				X						
12	Cause-Effect Analysis				X	X			X		X							X		X					X	X	
28	Symbolic Execution				X				X									X									
44	Decision Tables		20		X	X			X		X				X			X							X		X
47	Trace-Assertion Method								X	X	X							X									
35	Functional Abstraction																										
47	L-D Relation Methods																										
35	Program Proving								X		X							X									
25	Metric Analyses											X															
18	Algebraic Specification																										
22	Induction-Assertion Method									X									X								
	CONTROL ANALYSIS																										
60	Control Flow Analysis									X	X	X	X		X	X				X	X	X	5		X	X	
39	State Transition Diagram Analysis							X								X						X	6				
42	Program Control Analysis							X				X	X									X					
49	Operational Concept Analysis										X		X										7				
39	Call Structure Analysis											X	X			X							8				
21	Worst-Case & Process Trigger Timing Analysis																										
58	Concurrent Process Analysis																										
	DATA ANALYSIS																										
13	Data Flow Analysis					X					X	X									X		9			X	
54	Signed Directed Graphs																X				X				X		
14	Dependency Analysis				X				X			X	X				X	X	X		X	10			X	X	
46	Qualitative Causal Models															X											
39	Look-up Table Generator												X								X						
40	Data Dictionary Generator																							X			
39	Cross Reference List Generator								X			X						X			X			X		X	
39	Aliasing Analysis											X														X	
50	Concurrency Analysis																										
39	Database Analyzer									X		X							X						X		
39	Database Interface Analyzer																		X						X		
12	Data-Model Evaluation												X									11		X			

Appendix D. Key to Survey Responses

KB REDUCER

1 Implemented in REFINE
2 In progress of implementation

VVR

X3 Tests for: Circular rules, redundant rules,inconsistent rules, inconsistent data, irrelevant premises,irrelevant rules
4 Under development for non-monotonic KBtheories

VITAL

5 KADS Task Layer Analysis
6 KADS Behaviour Analysis
7 KADS Operational Model Analysis
8 Applied on KADS Behaviours Level
9 KADS Reference Layer Analysis
10 KADS Domain Layer Analysis
11 Validate KBS Domain Layer Correctness
12 To validate KADS domain layer data and knowledge
13 To represent experts knowledge on domain
14 To use experts knowledge in KADS models validation process
15 We have developed a language to represent expert knowledge on the domain and we validate the KADS model with this knowledge
16 Validation of KADS conceptual model
17 To be applied on KBS test
18 Applied on KADS operational model
19 Applied on KADS operational model

ESC

20 ESC was totally decision table based, methods were shown how to break large problems into smaller,checkable sized decision tables.

CRSV-CLIPS

21 'CRSV is no longer available with the later version of CLIPS because the features it contained have been integrated in most part into CLIPS. CRSV's popular feature was the strong type checking and that capability is available in CLIPS right now.' [Bebe Ly, NASA JSC]

WRAPPING

22 Wrappings provide a framework that can contain any or all of these methods. They do not do any of these. Because wrappings are descriptions of all components of a system, they also facilitate V&V of system architecture.

References

[1] E.P. Andert, Integrated knowledge-based system design and validation for solving problems in uncertain environments, Int. J. Man–Machine Stud. 36 (1992) 357–373.
[2] M. Alford, SREM at the age of eight: The distributed computing design system, Computer 18 (4) (1985) 36–46.
[3] M. Barnes, P. Bishop, B. Bjarland, G. Dahll, D. Esp, J. Lahti, H. Valisuo, P. Humphreys, Software testing and evaluation methods (The STEM Project), OECD Halden Reactor Project Report, No. HWR-210, May 1987.
[4] B. Beizer, Software Testing Techniques, 2nd edn., Van Nostrand-Reinhold, New York, 1990.
[5] Barry W. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[6] H. Booher (Ed.), MANPRINT: An Approach to Systems Integration, Van Nostrand-Reinhold, New York, 1990.
[7] W. Bruyn, R. Jensoon, D. Keskar, P. Ward, ESML: An extended systems modeling language, ACM Software Eng. Notes 13 (1) (1988) 58–67.
[8] BSI Standards, Guide to the Assessment of Reliability of Systems Containing Software.
[9] CHI (Computer–Human Interaction): Conference Proceedings on Human Factors in Computing Systems, sponsored by the ACM SIGCHI, Austin, 30 April–4 May 1989, pp. 265–268.
[10] G.H. Chisholm, B.T. Smith, A.S. Wojcik, Formal system specifications—a case study of three diverse representations, Argonne National Laboratory Report ANL-90/43, December 1990.
[11] C. Culbert, G. Riley, R.T. Savely, Approaches to the verification of rule-based expert systems, Proceedings of the First Annual Workshop on Space Operations Automation and Robotics Conference (SOAR '87), Houston, TX, 1987.

[12] A. Davis, Software Requirements: Analysis and Specification, Prentice-Hall, New York, 1990.
[13] M. Deutsch, Software Verification and Validation: Realistic Project Approaches, Prentice-Hall, Englewood Cliffs, NJ, 1982.
[14] R.H. Dunn, Software Defect Removal, McGraw-Hill, New York, 1984.
[15] EPRI '93, Survey and assessment of conventional software verification and validation techniques, SPRI TR-102106, Project 3093-01, Final report, February 1993.
[16] M.E. Fagan, Advances in software inspection, IEEE Trans. Software Eng. SE-12 (7) (1986) 744–751.
[17] A. Ginsberg, A metalinguistic approach to the construction of knowledge-based refinement systems, Proc. IJCAI, Los Angeles, CA, August 1985, pp. 367–374.
[18] J.B. Goodenough, S.L. Gerhart, Towards a theory of test data selection, IEEE Trans. Software Eng. SE-1 (2) (1975) .
[19] J.V. Guttag, J.J. Horning, J.M. W ng, The larch family of specification languages, IEEE Software 2 (5) (1985) 24–36.
[20] D. Harel, Statecharts: A visual formalism for complex systems, Science of Computer Programming, Vol. 8, North-Holland Elsevier, New York, 1987, pp. 231–274.
[21] B. Hartway, J. Young, D. Thomas, Simulation characterization, Proceedings of the Third International Conference on Software for Strategic Systems, 27–28 February 1990, Huntsville, AL, pp. 64–85.
[22] D. Hatley, I. Pirbhai, Strategies for Real-Time System Specification, Dorset House, New York, 1987.
[23] C.A.R. Hoare, J.C. Sheperdson (Eds.), Mathematical Logic and Programming Languages, Prentice-Hall, New York, 1985.
[24] W.E. Howden, Functional program testing, IEEE Trans. Software Eng. SE-6 (2) (1980) 162–169.
[25] D.C. Ince, The Automatic Generation of Test Data, Computer J. 30 (1) 63–69.
[26] H. Jensen, K. Vairavan, An experimental study of software metrics for real-time software, IEEE Trans. Software Eng. SE-11 (2) (1985) 231–234.
[27] R. Jensen, C. Tonies, Software Engineering, Prentice-Hall, Englewood Cliffs, NJ, 1979.
[28] C. Jones, Systematic Software Development Using VDM, Prentice-Hall, New York, 1986.
[29] J.C. King, Symbolic execution and program testing, Commun. ACM 1 (7) (1976) 385–394.
[30] N.G. Leveson, J.L. Stolzy, Safety analysis using Petri nets, IEEE Trans. Software Eng. SE-13 (3) (1987) .
[31] J. Llinas, S. Riui, M. McCown, The test and evaluation process for knowledge-based systems, SAIC final contract report, Contract no. F30602-85-G-0313 C, Task 86-001-01, prepared for Rome Air Development Center, 1987.
[32] M. Mehrotra, C. Wild, Multi-viewpoint clustering analysis, Working Notes, Vol. 6, Workshop on Verification and Validation of KBS at AAAI, Washington, DC, pp. 52–63.
[33] E. Miller, Better Software Testing, Proceedings of the Third International Conference on Software for Strategic Systems, 27–28 February 1990, Huntsville, AL, pp. 1–7.
[34] L.A. Miller, Dynamic testing of knowledge bases using the heuristic testing approach, Expert Systems Applications Int. J. 1 (1990) 249–269.
[35] L.A. Miller, A realistic industrial-strength life-cycle model for knowledge-based system development and testing, Paper presented at the Third Annual Workshop on Verification and Validation of Knowledge-Based Systems. Held at the meeting of the American Association for Artificial Intelligence, 29 July 1990, Boston. Published in AAAI Proceedings. Winter 1990.
[36] H. Mills, V. Basili, J. Gannon, R. Hamlet, Principles of Computer Programming: A Mathematical Approach, Wm. C. Brown, New York, 1987.
[37] R. Milner, A calculus of communicating systems, Laboratory for the Foundations of Computer Science, Edinburgh University Report No. ECS-L FCS-86-7, 1986.
[38] G.J. Myers, The Art of Software Testing, Wiley, New York, 1979.
[39] NBS 500-75, Validation, verification, and testing of computer software, February 1981.
[40] NBS 500-93, Software validation, verification, and testing technique and tool reference guide, September 1982.
[41] P. Ng, R. Yeh (Eds.), Modern Software Engineering: Foundations and Current perspectives, Van Nostrand-Reinhold, New York, 1990.
[42] NREG/CR-4227, W. Gilmore, Human engineering guidelines for the evaluation and assessment of video display unit, July 1985.
[43] NUREG/CR-4640, PNL-5784, Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry, August 1987.
[44] T. Nuyen, W. Perkins, T. Laffey, D. Pecora, Knowledge base verification, AI Mag. 8 (1987) 65–69.
[45] A. Omar, F. Mohammed, A survey of software functional testing methods, ACM SIGSOFT Software Eng. Notes 16 (2) (1991) 75–82.
[46] T. Ostrand, M. Baker, The category-partition method for specifying and generating functional tests, Commun. ACM 31 (6) (1988) .
[47] O. Oyeleye, Qualitative modeling of continuous chemical processes and applications to fault diagnosis, PhD dissertation, Massachusetts Institute of Technology, February 1990.
[48] D. Parnas, D. Smith, T. Pearce, Making formal software documentation more practical, a progress report, Technical Report 88-236, ISSN 0836-0227, Queen's University at Kingston, 1988.
[49] A.A.B. Pritsker, Introduction to Simulation and SLAMII, Wiley, New York, 1986.
[50] J. Rasmussen, K. Vicente, Cognitive control of human activities and errors: Implications for ecological interface design, Paper presented at the Fourth International Conference on Event Perception and Action, Trieste, Italy, 24–28 August 1987.
[51] C. Rattray (Ed.), Specification and Verification of Concurrent Systems, Springer-Verlag, New York, 1990.
[52] J. Rushby, Quality measures and assurance for AI software, NASA Contractor Report No. 4187, prepared for Langley Research Center under Contract NASA-17067, October 1988.

[53] J. Rushby, An introduction to formal specification and verification using EHDM, SRI International CSL Technical Report SRI-CSL-91-02, February 1991.
[54] SENTAR '95, Distributed Hybrid Systems Validation and Verification Database Annex C, Sentar, 4910 Corporate Drive, Suite C, Huntsville, AL 35805.
[55] R.A. Stachowitz, C.L. Chang, T.S. Stock, J.B. Coombs, Building validation tools for knowledge-based systems, First Annual Workshop on Space Operations Automation and Robotics (SOAR '87), Houston, TX, 5–7 August 1987, pp. 209–216.
[56] A. Sudduth, Diagnostic reasoning using qualitative causal models, Paper presented at the Electric Power Research Institute Conference and Expert System Applications for the Electric Power Industry, Boston, 9–11 September 1991.
[57] D. Teichroew, E. Hershey III, PSL/PSA: A computer-aided technique for structure documentation and analysis of information-processing systems, IEEE Trans. Software Eng. SE(3)-1 (1977) 41–48.
[58] C. Tung, On control flow error detection and path testing, Proceedings of the Third International Conference on Software for Strategic Systems, Huntsville, AL, 27–28 February 1990, pp. 144–153.
[59] UK Ministry of Defense Draft Interim Defense Standard 00-55, Requirements for the Procurement of Safety Critical Software in Defense Equipment, 9 May 1989.
[60] R. Wallace, Dolores, R.U. Fujii, Software verification and validation: An overview, IEEE Software 6 (3) (1989).
[61] R. Wallace, J. Stockenberg, R. Charette, A Unified Methodology for Development Systems, McGraw-Hill, New York, 1987.
[62] P. Ward, The transformation schema: An extension of the data flow diagram to represent control and timing, IEEE Trans. Software Eng. 12 (2) (1986) 128–210.
[63] E. Weyuker, T. Ostrand, Theories of program testing and the application of revealing subdomains, IEEE Trans. Software Eng. SE-G (3) (1980).

Robert T. Plant, is an Associate Professor in the Department of Computer Information Systems at The University of Miami. Dr. Plant obtained his PhD in Computer Science at The University of Liverpool, England. Previously having studied at The Programming Research Group, Oxford University, England, and Wadham College, Oxford, Dr. Plant was Chairman of the Seventh National Workshop on Validation and Verification of Knowledge-based Systems at AAAI, 1994 and Co-Chair of the 1997 Workshop. Dr. Plant is a Chartered Engineer (UK), a European Engineer, a Fellow of the British Computer Society and holds a Visiting Professorship in Computer Science at The University of Wolverhampton in England.

Stephen Murrell is a Lecturer in the department of Electrical and Computer Engineering of the University of Miami. Dr Murrell obtained his PhD in Computation from the Programming Research Group in Oxford University, England. Previously obtaining a BSc in Mathematics and Computer Science form the University of Essex, England, his research interests are in Formal Specifications of Systems, Parallel Processing and Programming Language design. Dr. Murrell has published widely on these areas and is principal investigator in an NSF-funded project: Specialized, Fully Declarative Logic Programming Languages for Expert Systems and Databases.