

Electronic Theses and Dissertations, 2004-2019

2011

Incremental Lifecycle Validation Of Knowledge-based Systems Through Commonkads

Feras Batarseh
University of Central Florida

 Part of the [Electrical and Electronics Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Batarseh, Feras, "Incremental Lifecycle Validation Of Knowledge-based Systems Through Commonkads" (2011). *Electronic Theses and Dissertations, 2004-2019*. 2006.
<https://stars.library.ucf.edu/etd/2006>

**INCREMENTAL LIFECYCLE VALIDATION OF KNOWLEDGE-BASED
SYSTEMS THROUGH COMMONKADS**

by

FERAS BATARSEH

M.S. University of Central Florida, 2007

B.S. Princess Sumaya University for Technology, 2006

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida, USA

Spring Term
2011

Major Advisor: Avelino J. Gonzalez

© 2011 Feras Batarseh

ABSTRACT

This dissertation introduces a novel validation method for knowledge-based systems (KBS). Validation is an essential phase in the development lifecycle of knowledge-based systems. Validation ensures that the system is valid, reliable and that it reflects the knowledge of the expert and meets the specifications. Although many validation methods have been introduced for knowledge-based systems, there is still a need for an incremental validation method based on a lifecycle model. Lifecycle models provide a general framework for the developer and a mapping technique from the system into the validation process. They support reusability, modularity and offer guidelines for knowledge engineers to achieve high quality systems. CommonKADS is a set of models that helps to represent and analyze knowledge-based systems. It offers a *de facto* standard for building knowledge-based systems. Additionally, CommonKADS is a knowledge representation-independent model. It has powerful models that can represent many domains. Defining an incremental validation method based on a conceptual lifecycle model (such as CommonKADS) has a number of advantages such as reducing time and effort, ease of implementation when having a template to follow, well-structured design, and better tracking of errors when they occur. Moreover, the validation method introduced in this dissertation is based on case testing and selecting an appropriate set of test cases to validate the system. The validation method defined makes use of results of prior test cases in an incremental validation procedure. This facilitates defining a minimal set of test cases that provides complete and effective system coverage. CommonKADS doesn't define validation, verification or testing in any of its models. This research seeks to establish a direct relation between validation and lifecycle models, and introduces a validation method for KBS embedded into CommonKADS.

This dissertation is dedicated to who taught me the *lifelong valid* meaning of sacrifice,
Dad and Mom, this is for you.

ACKNOWLEDGMENTS

First and foremost, I want to show gratefulness to my parents for their endless support of all my endeavors. I would like to express gratitude to everyone in my family, especially my sisters (Suha, Reem, Rasha and Zain) for their sheer advice and unconditional love, also to my brothers-in-law (Amer and Wisam). Much appreciation goes out to my advisor, Dr. Avelino Gonzalez, for his continuous guidance throughout the years of my study. Many thanks to the respected committee members, Drs. Damla Turgut, Gita Sukthankar, Robert Franceschini and Rainer Knauf. I would like to acknowledge my Intelligent Systems Laboratory colleagues (Victor Hung, Ruben Ramirez-Padron, James Hollister, Steven Kobosko, Cynthia Johnson, Lisa Soros, Miguel Elvir and Brian Lichtman) for their companionship and help in the experiments. Additionally, I would like to thank Ruben Ramirez-Padron for his help in the statistical analysis. A show of thanks goes out to Dr. Issa Batarseh for his continuous moral support all through the course of my study. Moreover, I would like to express my thanks to Dr. Ronald DeMara for his aid and advice. A special thank you goes out to all my Jordanian friends at UCF and to all my dear friends (Toufic Jabbour, Luis and Maria Zea, Con and Mary Wilson, Aras Alkis and Engin Ayaz) for the utter support they provided during the years. I am thankful to everyone at the College of Engineering at UCF and the International Services Center for the true facilitation and professionalism they displayed. Last but not least, I am grateful to anyone else who motivated or inspired me.

“Pride only breeds quarrels, but wisdom is found in those who take advice”

Proverbs 13:10

“Education is what remains after one has forgotten what one has learned in school. I have no special talent. I am only passionately curious”

Albert Einstein

“The greater our knowledge increases the more our ignorance unfolds”

John F. Kennedy

“The more I see the less I know for sure”

John Lennon

“Every age has its massive blind spots. We might not see them, but our children will. We thought that we had the answers, it was the questions we had wrong”

Paul Hewson

“I have learned silence from the talkative, toleration from the intolerant, and kindness from the unkind; yet, strange, I am ungrateful to those teachers”

Jibran Khalil Jibran

“God help us from those who believe that they are the sole possessors of truth. How we manage at times to agree willingly to become prisoners within our own minds and souls of beliefs and ideas on which we can never be flexible”

Hussein bin Talal

“Software engineering today is a race between software engineers striving to build bigger and better idiot-proof software, and the universe trying to produce bigger and better idiots. So far, the universe is winning”

Unknown

TABLE OF CONTENTS

LIST OF FIGURES	xi
LIST OF TABLES.....	xiii
LIST OF ACRONYMS/ABBREVIATIONS.....	xvi
CHAPTER 1 : INTRODUCTION.....	1
1.1 The Significance of Correct Software	1
1.2 Lifecycle Models in Software Development.....	3
1.3 Introduction to Knowledge-Based Systems	5
1.4 Introduction to Validation and Verification	10
1.4.1 Importance of Validation and Verification - Roles and Differences.....	11
1.4.2 Definition of Validation.....	15
1.4.3 Definition of Verification	18
1.5 Validation and Verification for Knowledge-Based Systems.....	19
1.6 Verification Approaches for Knowledge-Based Systems	21
1.7 Validation Approaches for Knowledge-Based Systems.....	25
1.7.1 Validation through Analysis of Heuristics.....	26
1.7.2 Validation through Simulation.....	27
1.7.3 Face Validation.....	28
1.7.4 Predictive Validation	28
1.7.5 Subsystem Validation	28
1.7.6 Validation through Case Testing	29
1.7.7 Validation through Graphical Representations	30
1.7.8 Validation through formal Methods.....	31
1.7.9 Turing Testing	32
1.7.10 Sensitivity Analysis	33
1.7.11 Field Testing	33

1.8 Summary	34
CHAPTER 2 : STATE OF THE ART IN VALIDATION OF KNOWLEDGE-BASED	
SYSTEMS.....	36
2.1 Validation Methods for Knowledge-Based Systems	36
2.1.1 Knowledge Validation Methods	36
2.1.2 System Validation Methods	40
2.1.3 Multi-Purpose Validation Tools	48
2.2 Lifecycle Validation of Knowledge-Based Systems	51
2.3 Summary	55
CHAPTER 3 : PROBLEM DEFINITION AND CONTRIBUTION.....	
3.1 General Problem.....	56
3.2 Specific Problem	57
3.3 Contributions	57
3.4 Hypothesis.....	58
3.5 Evaluation Method	59
CHAPTER 4 : LIFECYCLE DEVELOPMENT MODELS FOR KNOWLEDGE-BASED	
SYSTEMS.....	60
4.1 Life Cycle Models for Knowledge-Based Systems.....	60
4.1.1 DESIRE	60
4.1.2 KBSDLC.....	61
4.1.3 Generic Tasks	61
4.1.4 KADS	61
4.1.5 CommonKADS.....	63
4.2 Selecting CommonKADS for Validation.....	65
4.3 The CommonKADS Models	69
4.3.1 The Context Models.....	69
4.3.2 The Concept Models.....	77

4.3.3 The Artifact Models.....	81
4.3.4 UML Diagrams in CommonKADS	84
4.3.5 System’s Specification and Implementation.....	85
4.4 Summary	86
CHAPTER 5 : THE MAVERICK VALIDATION METHOD	87
5.1 Validation through CommonKADS Case Testing	88
5.1.1 Test Case Format	89
5.2 Method for Automated Validation Embedded into the Reusable and Incremental CommonKADS (MAVERICK).....	90
5.3 Test case extraction in MAVERICK.....	94
5.3.1 The Extraction of Test Cases from CommonKADS Models.....	95
5.4 Inspection Validation	102
5.5 Context-Based Test Case Reduction (CBTCR).....	104
5.5.1 Local Importance	105
5.5.2 The number of test cases selected for each iteration (N)	106
5.5.3 Spiral Development and Validation.....	107
5.5.4 Test Case Reduction through CBTCR.....	110
5.6 Summary	113
CHAPTER 6 : PROTOTYPE KNOWLEDGE-BASED SYSTEM	114
6.1 The CBTCR Tool.....	114
6.2 The Housing KBS	117
6.3 CommonKADS Models for the Housing KBS.....	119
6.3.1 Context Models.....	120
6.3.2 Concept Models	127
6.3.3 Artifact Models.....	129
6.3.4 Diagrams for the Housing KBS	130
6.3.5 Iterative System Implementation and Validation.....	137
6.4 The Extracted Test Cases	140

6.5 Summary	149
CHAPTER 7 : EXPERIMENTAL EVALUATION OF MAVERICK	150
7.1 Introduction	150
7.2 Experiment #1: Detection of Errors seeded by the developer	151
7.2.1 Experimental Setup.....	152
7.2.2 Experimental Procedure.....	153
7.2.3 Experimental Results	155
7.2.4 Discussions	158
7.3 Experiment #2: Errors Seeded by the Human Test Subjects	160
7.3.1 Experimental Setup.....	160
7.3.2 Experimental Procedure.....	161
7.3.3 Experimental Results	163
7.3.4 Discussions (Statistical Analysis of Results).....	184
7.4 Experiment #3: Comparison of MAVERICK to Other Methods	186
7.4.1 Experimental Setup.....	186
7.4.2 Experimental Procedure.....	188
7.4.3 Experimental Results	189
7.4.4 Discussions	191
7.5 Summary of Experiments	192
CHAPTER 8 : CONCLUSIONS AND FUTURE WORK.....	193
8.1 Summary	193
8.2 Conclusion.....	195
8.3 Future Work	198
APPENDIX A: LIST OF TEST CASES FOR THE HOUSING KBS.....	201
APPENDIX B: JAVA CODE FOR THE HOUSING KNOWLEDGE-BASED SYSTEM.....	221
APPENDIX C: EXPERIMENTAL VALIDATION ITERATIONS.....	252
LIST OF REFERENCES	289

LIST OF FIGURES

Figure 1: The spiral model [5]	4
Figure 2: Different roles in building a knowledge-based system [9].....	10
Figure 3: Defects cost versus development time phases.....	12
Figure 4: Verification using KADS [29]	24
Figure 5: KVAT [54]	37
Figure 6: VKB validation method [51].....	41
Figure 7: VKB within the validation process [47].....	42
Figure 8: Bi-directional many sided explanation typed multi-step validation method [60].....	44
Figure 9: Validation of knowledge-based military systems [61].....	45
Figure 10: Validation process proposed by Abel et al. [65]	47
Figure 11: Spiral model for knowledge-based systems [73]	52
Figure 12: KADS models [83].....	62
Figure 13: CommonKADS set of models.....	64
Figure 14: The organizational model worksheets [9]	71
Figure 15: The communication model role [9].....	80
Figure 16: The design model [9].....	82
Figure 17: The validation method: MAVERICK	93
Figure 18: The effect of GI on sorting test cases.....	109
Figure 19: MAVERICK iterative development and validation	110
Figure 20: CBTCR flowchart.....	112
Figure 21: CBTCR Java tool interface.....	116

Figure 22: Structure and people in housing KBS [9].....	130
Figure 23: UML activity diagram for OM [9]	131
Figure 24: DFD for the main process (application-assessment task) [9].....	132
Figure 25: State diagram of a single flow of the application-assessment process [9]	132
Figure 26: Residence and application relation in knowledge representation [9].....	133
Figure 27: Inference structure for the KM [9]	134
Figure 28: Subtype hierarchy for knowledge about priorities in the system [9]	134
Figure 29: Tasks and their inferences (assessment task) [9].....	135
Figure 30: Domain schema for the housing KBS [9]	135
Figure 31: State diagram representing the communication model [9].....	136
Figure 32: Dialogue diagram of the communication model	136
Figure 33: Subsystems relationship for the design model	137
Figure 34: Assigner mode.....	138
Figure 35: Data typist mode.....	139
Figure 36: Magazine editor mode	139
Figure 37: Bar chart for errors' types inserted into the system.....	176
Figure 38: Bar chart for errors inserted by all human test subjects	182
Figure 39: Pie charts for MAVERICK errors results.....	182
Figure 40: Pie chart for the 22 undetected errors.....	183
Figure 41: Bar chart for the legitimate errors inserted by all the users.....	183

LIST OF TABLES

Table 1: Validation general approaches comparison	35
Table 2: Worksheet OM-1	71
Table 3: Worksheet OM-2	72
Table 4: Worksheet OM-3	72
Table 5: Worksheet OM-4	72
Table 6: Worksheet OM-5	73
Table 7: Worksheet TM-1	74
Table 8: Worksheet TM-2	75
Table 9: Worksheet AM-1	76
Table 10: Worksheet KM-1	79
Table 11: Worksheet CM-1	80
Table 12: Worksheet CM-2	81
Table 13: Worksheet DM-1	83
Table 14: Worksheet DM-2	83
Table 15: Worksheet DM-3	83
Table 16: Worksheet DM-4	83
Table 17: Example OM3 worksheet for test case extraction	96
Table 18: Worksheet OM-1 for housing KBS [9]	120
Table 19: Worksheet OM-2 for housing KBS [9]	120
Table 20: Worksheet OM-3 for housing KBS [9]	121
Table 21: Worksheet OM-4 for housing KBS [9]	121

Table 22: Worksheet OM-5 for housing KBS [9]	122
Table 23: Worksheet TM-1 for housing KBS (Task 1)	122
Table 24: Worksheet TM-2 for housing KBS (Task 1)	123
Table 25: Worksheet TM-1 for housing KBS (Task 2)	123
Table 26: Worksheet TM-2 for Housing KBS (Task 2)	124
Table 27: Worksheet TM-1 for housing KBS (Task 3) [9]	124
Table 28: Worksheet TM-2 for housing KBS (Task 3) [9]	125
Table 29: Worksheet TM-1 for housing KBS (Task 4)	125
Table 30: Worksheet TM-2 for housing KBS (Task 4)	126
Table 31: AM-1 for housing KBS (agent: assigner)	126
Table 32: AM-1 for housing KBS (agent: data typist)	127
Table 33: AM-1 for housing KBS (agent: magazine editor)	127
Table 34: KM-1 for housing KBS	128
Table 35: CM-1 for housing KBS (transaction 1) [9]	128
Table 36: CM-1 for housing KBS (transaction 2)	128
Table 37: CM-1 for housing KBS (transaction 3)	129
Table 38: DM-1 for housing KBS	129
Table 39: DM-2 for housing KBS	129
Table 40: DM-3 for housing KBS	129
Table 41: DM-4 for housing KBS	130
Table 42: Results for errors inserted into the system by the developer	156
Table 43: Errors inserted by test subject Red	164
Table 44: Errors inserted by test subject Blue	165

Table 45: Errors inserted by test subject Yellow	166
Table 46: Errors inserted by test subject White	167
Table 47: Errors inserted by test subject Orange	170
Table 48: Errors inserted by test subject Brown.....	172
Table 49: Errors inserted by test subject Grey	174
Table 50: Not legitimate vs. legitimate errors inserted by human subjects	184
Table 51: Error categories inserted by the human subjects	184
Table 52: General time consumed for MAVERICK	190
Table 53: General time consumed for EMBODY	190
Table 54: General time consumed for VIVA.....	191

LIST OF ACRONYMS/ABBREVIATIONS

AI: Artificial Intelligence

AM: CommonKADS Agent Model

BKB: Bayesian Knowledge Base

CBTCR: Context-Based Test Case Reduction

CLIPS: C Language Integrated Production System

CM: CommonKADS Communication Model

CML: CommonKADS Modeling Language

CommonKADS: Common Knowledge Acquisition and Design Support

DESIRE: Framework for Design and Specification of Interacting Reasoning Components

DFD: Data Flow Diagram

DKB: Decision Knowledge Base

DM: CommonKADS Design Model

DoD: US department of defense

eGanges: Electronic Glossed Adversarial Nested Graphical Expert System

EMBODY: Expert Modeling Based on Decision Modularity

ESPRIT: European Union Information Technology Program

GA: Genetic Algorithms

GenAID: Generator Artificial Intelligence Diagnostics

GI: Global Importance

HA: Highly Applicable

IEEE: Institute for Electronics and Electrical Engineers

IV&V: Independent Validation and Verification

KADS: Knowledge Acquisition and Design Structuring

KARL: Knowledge Acquisition and Representation Language

KBS: Knowledge-based Systems

KBSDLC: Knowledge-Based Systems Development Life Cycle

KM: CommonKADS Knowledge Model

KROL: Knowledge Representation Object Language

KVAT: Knowledge Validation Tool

LA: Low Applicability

LI: Local Importance

MAS-CommonKADS: Multi-Agent Systems CommonKADS

MAVERICK: called Method for Automated Validation Embedded into the Reusable and Incremental CommonKADS.

MIKE: Model-based and Incremental Knowledge Engineering

MW: Model Weight

N: Number of selected test cases

NA: Not Applicable

OM: CommonKADS Organization Model

QUEST: Quasi-Exhaustive Set of Test Cases

ReST: Reasonable Set of Test Cases

SHIVA: Spanish Language Acronym for Heuristic Integrated Validation System

SQA: Software Quality Assurance

STRIPS: Stanford Research Institute Problem Solver

TM: CommonKADS Task Model

UCF: University of Central Florida

V&V: Validation and Verification

VESA: Validation Expert System Agent

VKB: Validation Knowledge Base

VTB: Validation Techniques Base

VVR: Validation, Verification and Refinement

CHAPTER 1: INTRODUCTION

Software systems participate in many aspects of human life. They control many aspects of our world. Software systems are found at home, in the factory, on the road, in the office, indoors and outdoors. Software development is not an arbitrary process; it should be well defined, designed and planned. Validation and verification are performed to ensure validity, correctness and reliability of software. Knowledge-based systems are a special kind of software systems; they too require validation and verification. This dissertation introduces a robust validation method for knowledge-based systems.

1.1 The Significance of Correct Software

Software development is a challenging process. Different parameters affect this process, such as manpower, size of the software, project's budget, and many other factors. Building a robust software system is critical but can also be tricky. Software problems have postponed space shuttle launches, caused problems for airplanes and disrupted credit card and financial systems.

Software development is commonly referred to as a science as well as an art [1]. It is clearly an inexact process, as no two software developers will produce the same exact design, code or testing plans. The cost of failed software can be high indeed. For example, in 1996, a test flight of a European launch system, Ariane 5 no. 501, failed as a result of a software bug. Upon launch, the rocket veered off its path and was destroyed by its self-destruction system. This loss was later analyzed and linked to a 64-bit floating point number conversion to a 16-bit integer. This caused certain hardware components to fail. This rocket was destructed because of an arithmetic overflow in its software, resulting in a loss of more than US\$370 million [2].

A software bug can occur anytime, anywhere and with anyone. If a project fails, it can lead to financial loss and might jeopardize an organization's position in the market. Fox Meyer Drugs, a wholesale pharmaceutical distribution company in Texas developed a resources planning system that failed after implementation. The system was never tested thoroughly. When it was installed, many errors appeared and created many problems related to shipping. That put the organization into bankruptcy in 1996. However, the most dangerous software failures are those related to critical systems that affect or involve human lives. Three people died between 1985 and 1987 when a radiation therapy system called Therac erroneously subjected patients to lethal overdoses of radiation. The overdose also caused major and minor injuries to others [3]. The cause of this overdose was an arithmetic software bug in its control system.

Although many lessons have been learned from software systems failures, disasters and losses are still mounting. During 2005, Toyota recalled 160,000 Prius automobiles from the market because of a software bug in the hybrid car. These are just some of the many recent IT projects gone wrong in recent history. In 1995, the Standish Research Group reported that 51% of software projects fail [4]. They based this result on reviewing 40,000 software systems from different domains and countries. Software bugs will still happen, and coding errors increasingly affect our lives, directly or indirectly. The tragedy is that software disasters are avoidable, predictable and stoppable if the right processes are used while building the software and if enough resources are placed into testing the software for static and run-time bugs.

Although software disasters can be prevented, it is not an easy task to build fault-free software. Errors can happen because of a programming mistake, miscommunication among the code developers, a misunderstanding between the customer and the developer, a mistake in the requirements document, a politically biased managerial decision, a change in the domain market

standards and many other reasons. This dissertation describes a system that can enhance the ability of developers to ensure correct software by including validation in the lifecycle development process.

1.2 Lifecycle Models in Software Development

Building software is not a random process. It is a well defined and structured procedure. It is a challenging practice that needs to be planned and designed in a structured manner. Lifecycle models can help in building software systems by controlling the software development process. According to Sommerville [1], the traditional software engineering lifecycle phases are:

1. **Requirements definition:** The user's functional, non-functional and domain requirements are defined.
2. **System and software design:** The anticipated system is designed. It is usually represented in graphs, such as data flow diagrams and sequence diagrams.
3. **Implementation and coding:** This phase is important because here the system is coded.
4. **Integration and system testing:** The system modules are integrated in this step. Integration usually causes problems such as mismatches and miscommunication between the modules of the system. A full system testing is needed to overcome those difficulties, sometimes called integration testing.
5. **Operational Maintenance:** This phase takes place after the system is deployed and while the users are using the system.

This lifecycle is also called the Waterfall Model. Other example lifecycle methods include the *spiral model*, the *evolutionary development model*, the *reuse-oriented model*, the *incremental development model* and the *extreme programming model*. These are briefly described below [1].

1. **The Spiral Model:** In this model each process is represented as a spiral rather than a sequence of events, feedback, testing and all the other steps are performed multiple times. The spiral model is shown in Figure 1.
2. **Evolutionary Development:** In this model the software is delivered in versions, and specifications are defined and presented to the developer incrementally.
3. **Reuse-Oriented Development:** This model is based on assembling different pre-defined modules that form a system.
4. **Incremental Development:** The system is delivered in increments, and each increment includes a set of functionalities.
5. **Extreme Programming Development:** This model is based on very small deliverables, with user involvement and fast changes in the code.

The software life-cycle models presented here are not the only ones, but they are the best known models and have been widely used.

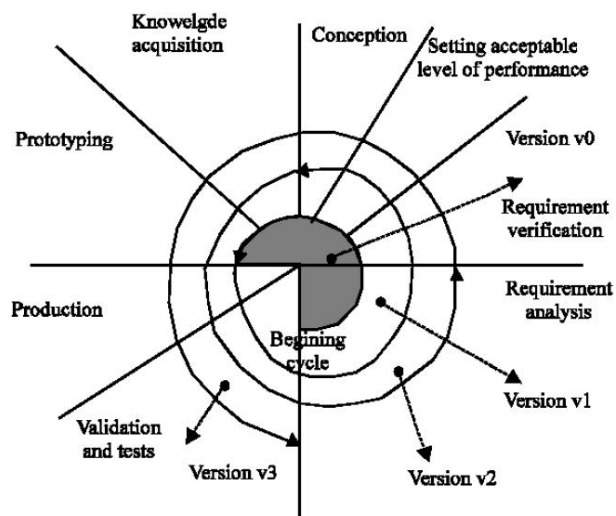


Figure 1: The spiral model [5]

There are many differences between these models. Using different models may lead to different results or might require a different set of resources and different budgets. Each of the models mentioned above is used for a different set of systems.

One thing that all software engineering lifecycle models have in common is the critical need for *validation* and *verification*. Likewise, any software system requires testing, validation and verification. The general and most common way of ensuring the robustness of systems is to perform extensive validation and verification. Before discussing validation and verification, it is appropriate to introduce knowledge-based systems; the type of software systems that this dissertation targets.

1.3 Introduction to Knowledge-Based Systems

According to the Webster's [6] definition, intelligent systems are "systems that perceive their environment and take actions which maximize its chances of success." In order to display such functionality, the system must be indeed *intelligent*.

Over the past 50 years there have been many efforts towards achieving artificial intelligence in machines. These efforts have resulted in significant advances in that field. Computer *agents* are a type of intelligent system that can interact with humans in a realistic manner. They have been known to beat the world's best chess player and locate hostages in a military operation [7]. A computer agent is an autonomous or semi-autonomous entity that can emulate a human. It can be either physical such as a robot, or virtual such as an avatar [7]. One known type of software intelligent systems is knowledge-based systems.

Artificial intelligence raises the question of whether machines can think and learn like humans. The ability to learn should be part of any system that has intelligence. Intelligent

systems must be able to adapt to changes in their environment. Knowledge-based systems belong to the field of artificial intelligence. Other disciplines in this field include:

1. **Machine Learning** happens when the agent learns by exploring its surrounding and figuring what actions are the most rewarding.
2. **Neural Networks** are a learning paradigm inspired by the human nervous system. In neural networks, information is processed by a set of interconnected nodes called neurons.
3. **Genetic Algorithms (GA)** is a method that finds a solution or an approximation to the solution for optimization and search problems. GA use biological techniques such as mutation, crossover and inheritance.
4. **Natural Language Processing (NLP)** is a discipline that deals with linguistic interactions between humans and computers. It is an approach dedicated to improving the human-computer interaction. This approach is usually used for audio recognition.
5. **Computer Vision** is when the computer captures and analyzes images of the 3D world. This includes making the computer recognize objects in real-time.

Knowledge-based systems (expert systems) are intelligent systems that reflect the knowledge of a proficient person. Knowledge-based systems are a specific kind of intelligent system that makes extensive use of knowledge. They are different from conventional software systems because they use heuristic rather than algorithmic approaches for decision making. Furthermore, knowledge-based systems separate the knowledge from how it is used [8]. The idea of a general problem solver (GPS) was introduced during the early 1960s. This idea used generic search techniques aided by heuristic knowledge to solve problems [8]. DENDRAL, developed during the early 1970s, realized the internal structure of unknown compounds. The GPS and the

DENDRAL experiences were instrumental in the development of MYCIN, a system that diagnosed blood disorders. MYCIN is a landmark medical rule-based system developed at Stanford University. More importantly, MYCIN influenced the creation of the field of knowledge-based systems. Using the MYCIN experience, a number of knowledge-based systems in other domains were developed during the 1970s and 1980s such as HEARSAY (for language understanding), PROSPECTOR (for geology), XCON (for systems configuration) and GenAID (for electrical equipment diagnosis) [8]. A great number of expert systems have been developed since that time.

The main component of the knowledge-based system is the knowledge base. Often, the knowledge reflected is that of an expert. Knowledge-based systems consist of an inference engine and a knowledge base. Inference engines act as the main controller of the system, where the knowledge is manipulated to address specific problems or answer specific questions. The inference engine contains the problem solving knowledge. The knowledge base is typically elicited from an expert or from an otherwise knowledgeable person in a certain domain. The knowledge is elicited, represented and built by the *knowledge engineer*, in a process known as *knowledge engineering*. Knowledge engineers can be said to be computer systems experts whom are responsible for representing knowledge in a computer system in order to solve problems that require human expertise. The first task of a knowledge engineer is to acquire this knowledge from its human source. This is known as knowledge acquisition and it is carried out using various methods such as:

1. **Interviews** - conducting interviews with the expert and discussing the domain.
2. **Questionnaire** - developing a questionnaire to be completed by the expert and other personnel related to the domain.

3. **Observation and protocol analysis** - observing the work environment of the expert whose knowledge is being elicited and analyzing the processes he/she uses for different tasks.
4. **Collecting cases** - collecting all the scenarios that might occur with the expert. This method is mostly used when a case-based knowledge is developed.
5. **Extracting cause-effect relationships** - this means identifying the reasoning patterns of the expert's tasks, processes and knowledge.

The second major task of a knowledge engineer is to represent the elicited knowledge.

Knowledge can be represented in many ways, but the most common ways are [8]:

1. **Rule-based systems** - where knowledge is represented in term of rules. *If some condition then some action.*
2. **Case-based systems** - where knowledge is stored as cases. This kind of system is used when solving new problems depends on the solutions of previous ones.
3. **Logic-based systems** - where knowledge is stored as logic. In this form of systems, knowledge is defined as a set of logical operators.
4. **Frame-based systems** - knowledge is represented in templates or frames. Templates have a number of slots to be filled. Related facts are clustered in groups in this representation.
5. **Object-based systems** – where data is represented in objects. In this representation an object is a collection of information that represents an entity from the real world and describes its functionalities.

Although expert systems are sometimes called knowledge-based systems, and vice-versa, expert systems are in fact a sub-set of knowledge-based systems. Not all knowledge is experts knowledge; it can be elicited from other resources too. Shells are development tools that help the developer in designing and implementing expert system. Example shells are: CLIPS (C Language Integrated Production System) and eGanges (Electronic Glossed Adversarial Nested Graphical Expert System).

Knowledge-based systems have many advantages. These include their efficient replacement of the human expert when appropriate, and they can act as a repository for human knowledge in case of loss of human expertise. Furthermore, design of the system and determination on what type of knowledge representation to adopt is a core set of decisions that the knowledge engineer needs to make. Finally, the knowledge engineer performs the task with which this dissertation is most concerned: validation and verification of the system. It takes more than one person to be able to build a knowledge-based system. Different people interact with the system and affect its quality. In knowledge-based systems, the knowledge engineer, the expert, the system developer and the end-user are the main people who relate to the system. Their roles are sometimes managed by a project manager or a knowledge manager. In some cases, the knowledge engineer is not directly responsible for implementing the system. A knowledge-based system developer develops the system while the knowledge engineer is responsible for the rest of the tasks. There can also be more than one knowledge engineer. Testing, validation and verification are performed by knowledge engineers and/or experts, and in some cases, by the knowledge system developer and/or the user. In most cases, a knowledge manager and a project manager become involved in the process when the project is sufficiently large. People who

interact with the knowledge-based system development and their different roles are illustrated in Figure 2.

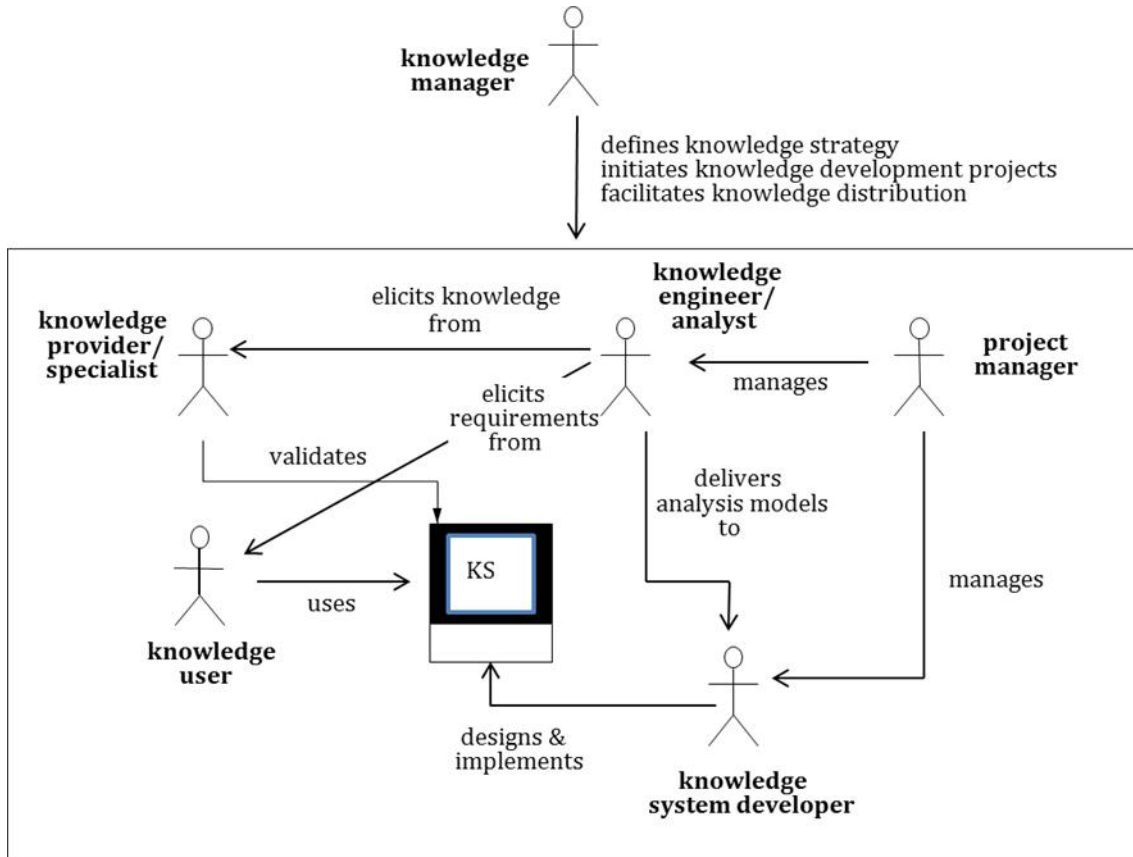


Figure 2: Different roles in building a knowledge-based system [9]

The next section introduces definitions for validation and verification, as well as a discussion about their importance, roles and differences.

1.4 Introduction to Validation and Verification

Many definitions of validation and verification have been introduced in the literature. So many that there seems to be great confusion as to what each is, even to the point of direct contradiction.

Mainstream definitions introduce validation and verification as the two main tasks of evaluation. Other definitions have verification as part of validation, or testing as part of evaluation. The following sub-sections aim to insert some sense into the many definitions and arrive at a definition for the purposes of this dissertation.

1.4.1 Importance of Validation and Verification - Roles and Differences

Validation and verification have a high level of mutual interaction. Many interpretations were introduced in literature, and there are many publications about their definitions, roles, and differences [10] [11].

Although validation and verification (V&V) are mentioned together in most cases, they should not be confused with one another. Validation and Verification are different and have distinct goals and approaches. Both validation and verification are related to the concepts of software quality assurance. By themselves, validation and verification do not guarantee software quality; planning, traceability, configuration management and other aspects of software engineering are also required to accomplish that. The cost of fixing bugs increases exponentially if done late in the development phase. Software development managers prefer to create a validation and verification (V&V) group very early in the software development cycle [10]. Figure 3 illustrates the cost of defects versus time of development. In fact, most validation and verification groups are created at the same time as requirements specifications are being written.

The role of validation and verification (V&V) in performance assurance and continuous improvement is clear because validation and verification examine the software development process and provide an independent assessment of development and operational risk. V&V help to identify safety, reliability and performance concerns and have generally been demonstrated to save money by identifying errors early.

Validation and verification efforts ensure that requirements are complete, the proposed system architecture will meet requirements, and traceability is demonstrated among requirement, design and test cases. Furthermore, suggestions made by validation and verification after errors are found can help prevent similar errors in the system, thereby improving the overall quality. In some cases, a third party validation and verification contractor is responsible for performing validation and verification and is independent from the development group. The presence of a separate validation and verification contractor provides an incentive for developers to improve their internal practices.

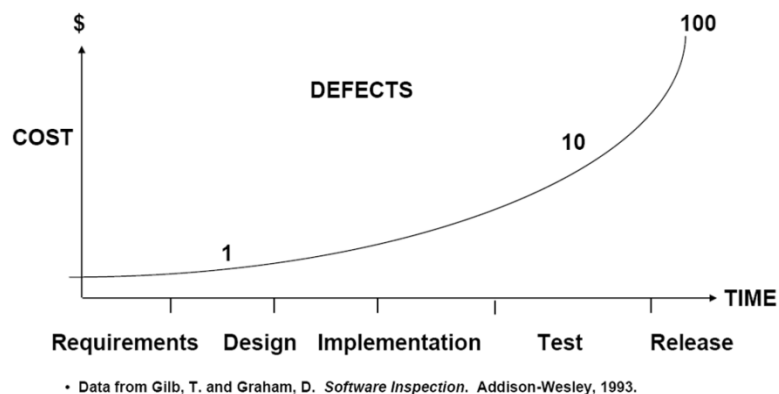


Figure 3: Defects cost versus development time phases

It is evident that a quantitative measure needs to be developed that can assess the effectiveness of validation and verification. When performed in parallel with software development, the validation and verification process yields several benefits [10]:

1. It reveals errors with high risk early. This gives the design team time to develop an ample solution, rather than forcing them into a quick fix to accommodate software deadlines.
2. It compares the product against the system requirements at an early stage. This ensures that the development team is meeting the specified requirements.

3. It provides management with continuous and comprehensive visibility into the quality and progress of the development effort, not just at major review milestones that may occur infrequently.
4. It gives the user an incremental preview of system with the chance to make early changes
5. It helps in the decision making process: whether to proceed to the next step in development or not.

Two independent studies were conducted [12] that focused on three projects of different sizes. Validation and verification were used in the product development from start to finish. The following results were reported by Anderson et al. [10]:

1. Rates of uncovering errors early in development were better
 2. Validation and verification found 2.3 to 5.5 errors per thousand lines of code
 3. Over 85% of errors affected reliability and maintainability
 4. Early errors detected saved 20-28% of validation and verification costs (for projects beginning at coding phase) and 92-100% (for projects beginning at requirement phase)
- [10]

On the other hand, some weaknesses of the validation and verification process include [11] [13]:

1. It can increase the short-term cost of the project.
2. The interface activities for documentation, data and software deliveries between developer and validation and verification groups may reduce the productivity.
3. Validation and verification efforts can take years if goals are not properly set. It can also demoralize the developers.

4. Validation and verification requires training and familiarity with the software and development techniques being used.
5. It can be hard to measure the effectiveness of validation and verification. Some people believe that validation and verification aren't needed, especially when a Software Quality Assurance (SQA) group exists.
6. There is an inherent conflict between developers and the validation and verification group because some developers might believe that the V&V team is trying to find errors in their work to embarrass them.
7. The biggest problem in validation and verification is the 'coordination problem' between different people interacting with the system during the development of the software. Example: Exchanging files and documentation while the size and the content of documents continue to evolve.

A comprehensive validation and verification effort should be administered by a specific group that may comprise a developer, someone from the end-user organization, contractors and subcontractors. It may also have members knowledgeable in software quality assurance, configuration management and data management teams.

Configuration management and data management groups supervise software versions and data during their development using techniques such as *formal audits, change control records, traceability of requirements, and sign-off records*. The user group provides a guarantee that the software product satisfies the users' requirements and specifications. It should be noted that configuration management and documentation collectively aid validation and verification efforts.

Generally, a validation and verification group works independently from the development group, and is sometimes called *independent validation and verification (IV&V)*. Validation and

verification tasks are oriented towards analysis and comprehensive testing. The purpose is to develop an independent evaluation of software quality and to decide whether the software satisfies all the critical specifications. Advantages of this approach are detailed analysis and test of software requirements, an independent determination of how well the software performs, and early detection of high-risk software and system errors. Disadvantages may include short-term higher cost to the project and additional development interfaces [10].

Several techniques are currently used for verification and validation. It is important to note that some of these techniques are used in several phases of product life-cycle. For example, algorithm analysis is a well-known technique that evaluates algorithms developed during the software development cycle. This analysis is done during the requirement specification phase to determine what kinds of algorithms will be needed, then in design phase to see how those algorithms will be developed, and in coding phase to determine whether the algorithms being coded conform to V&V requirements. The same is done during unit testing and maintenance of the software product.

1.4.2 Definition of Validation

As the name implies, validation can be considered to be a process through which a product is considered valid for fulfilling its intended purpose. Some researchers consider validation as part of other processes such as evaluation or testing. Other definitions introduce validation as a standalone process. Introducing a new definition for validation is not the goal of this dissertation. This section aims to discuss validation definitions in the literature and select one as the official definition for the purposes of this dissertation. The most popular definition of validation is introduced by O’Keefe et al. [14]. They define validation as “building the right system”. While this definition is catchy, straight forward and easy to remember, it doesn’t shed much light on

what validation is or how it should be carried out. Furthermore, it's hard to understand anything about validation from this definition. For validation to be understood, a more descriptive and comprehensive definition is necessary.

The US department of defense (DoD) defines validation as “The process of determining the degree to which a model is an accurate representation of the real world from the perspective of intended uses of the model” [15]. This definition is comprehensive and descriptive, yet in some cases it might not be true. This definition introduces validation as a comparison process, which is true, but not necessarily to the real world. Validation in some cases doesn't cover the non-functional aspects of the system, such as the influential personal moods and decisions that affect the work processes of an organization. Comparing the system to the real world is a challenging and often a nearly-impossible task. Instead, validation can be a comparison of the system to a useful model of the real world.

Adrion et al. [11] defined validation as “The determination of the correctness of the final program or software with respect to the user needs and requirements”. This definition emphasizes checking the correctness of a program, which is normally considered part of the verification process (discussed in the next section), and not validation. Another similar definition is presented by Zlatareva [13]. She defines validation as the intention to declare the functional correctness of the system's performance.

A recent definition of validation was introduced by Min et al. [16]. They claim that any validation process should go through the following steps:

1. Selecting a segment of the system.
2. Evaluating this segment, and reaching decisions about this segment.
3. Running the segment and evaluating its performance and output.

This definition is based on modular validation, which is one approach to validation. This definition, however, doesn't cover integration testing, which is commonly accepted to be different than component testing and not the simple sum of its parts. When software segments are validated individually, problems might rise when these segments are integrated. Therefore, this definition is not complete.

IEEE [17] defines validation as: "The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements." This definition describes validation as a major phase of the development lifecycle. It describes validation as evaluation, which is not correct. Evaluation is a vague term and it might include verification or any other form of testing that doesn't represent validation. Moreover, this definition is not dedicated to knowledge-based system, and hence doesn't meet the needs of this dissertation.

The most comprehensive and descriptive definition of validation in the context of knowledge-based systems was recently introduced by Gonzalez et al. [18]. They define validation as "the process of ensuring that the output of the intelligent system is equivalent to those of human experts when given the same input." This definition is general and it implies that validation compares the system to the real world. For knowledge-based systems (or expert systems), the real world knowledge is the human experts' knowledge. The system is to be compared to the expert's knowledge and not directly to the real world. Furthermore, this definition is specifically dedicated to knowledge-based systems. This definition, therefore, serves the purpose of this dissertation and will serve as the official definition of validation for this dissertation.

Therefore, it is asserted here that validation involves the execution of the completed system and its output is compared to benchmarks to ensure that the output is correct. Such benchmarks are often human expert opinion.

1.4.3 Definition of Verification

Verification typically doesn't involve executing the KBS, such as is done in validation. Nevertheless, it is sometimes considered a part of validation. It is the part that deals more with the code and the details of implementation. Again O'Keefe et al. [14] defined verification in an abstract way just as they did for validation. They defined verification as: "building the system right". This definition maybe easy to remember but it is unquestionably abstract and vague. Another very common definition of verification is that by the US Department of Defense: "process of determining that a model implementation accurately represents the developer's conceptual descriptions and specifications" [15]. This definition compares the system to the developer's descriptions, which is not correct. In most cases, the developer is assisted by the knowledge engineer and the expert, as well as sometimes the customer/user. The model should represent the expert's knowledge, the knowledge engineer's design, the developer descriptions and the customer's desires and needs. Therefore, verifying the developer's descriptions and specifications is not sufficient to reach a verified system.

Adrion et al. [11] defined verification as "The demonstration of the consistency, completeness and correctness of the software". This definition is correct, but it limits verification to three aspects: consistency, completeness and correctness. The IEEE [17] definition for verification is "the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase." This definition introduces verification as a process to be performed on a system as a whole or a

certain component. This definition doesn't limit verification to certain aspects such as consistency, but compares the system to the set of conditions imposed at every phase, which covers a wide set of expectations and required functionalities. Many other definitions for verification have been introduced. Gonzalez et al. [18] recently compared different definitions for verification. They also introduced a new, comprehensive and descriptive definition for verification: "Verification is the process of ensuring that the intelligent system conforms to specifications, and its knowledge base is consistent and complete within itself". This will serve as the official verification definition for this dissertation. The next section discusses validation and verification for knowledge-based systems and will introduce different V&V approaches that have been widely used in many disciplines.

Therefore, it is asserted here that verification involves: 1) Comparison of evaluation of specifications to ensure compliance and 2) examination of code, either manually or through theorem proving approaches, to ensure that the code is consistent and complete. It is further assumed that the system is not executed in the process of verification. When performing verification, one checks for syntactic errors in a rule base such as: redundant rules, conflicting rules, subsumed rules, circular rules, dead-end rules, missing rules, unreachable rules and unnecessary if conditions.

1.5 Validation and Verification for Knowledge-Based Systems

Knowledge-based systems generally have a different structure than conventional software systems. They separate the knowledge from its use with the knowledge base and inference engine. Each part of the system needs to be verified and validated separately. Inference engines are usually off-the-shelf software. Therefore, there is little need for their validation and

verification as presumably that has already been done by its developer/vendor. Knowledge-bases are critical parts that hold important data, and therefore need to be validated and verified.

Preece [19] provides an evaluation study on validation and verification for knowledge-based systems. In his paper, validation and verification processes are divided into types. The most common category is *inspection* but it is also the least reliable. It is a manual test of the knowledge base. The expert merely looks at the rules/cases and their relations in the system. Manual checks and face validation are examples that fall into the inspection category (these are discussed in later sections of this chapter). Another validation and verification type is formal methods. In formal methods, the system is represented in a logic-based format. The most common method of testing in conventional software engineering is empirical testing. It has two main methods: functional and structural testing. Functional testing concentrates on the functionality of the different modules of the system, while structural testing checks the output of the entire system and how it performs with the users.

There are many validation and verification approaches for knowledge-based systems, but how do we decide which one to use? Some of the V&V approaches are suitable to use with a certain type of systems in certain domains. For example, real-time systems require a specific approach to validation and verification because of their sensitive nature. Furthermore, different validation and verification approaches could be used at different phases of the knowledge-based life-cycle. For example, field testing might be used after the implementation phase is complete; while directed graphs might be used after knowledge elicitation (those approaches are introduced later in this chapter).

Another categorization of validation and verification approaches was introduced by Balci [20]. He describes the possible ways to do validation as:

1. **Informal:** relies more heavily on human reasoning than on automated processes and mathematical analysis. This category is the most common one, and it is used widely.
2. **Static:** this category is based on accurate analysis of the system models in the design phase.
3. **Dynamic:** requires multiple model and system executions then making changes to the system according to these executions' results and conclusions.
4. **Symbolic:** this category is similar to the dynamic methods, but symbols are used to represent the system modules and as inputs for the tests.
5. **Constraint:** this category is applied to check the system's boundaries and assertions.
6. **Formal:** Based on formal mathematical proof of correctness.

The major goal of this dissertation is to introduce a new validation method. Therefore, it is important to look at previous validation approaches and assess each. It is important to realize the difference between an approach and a method. An approach is a general strategy that doesn't give detailed steps on how to perform the process, it only provides general direction. On the other hand, a method is a well defined set of steps that aims to fully guide the process. While Chapter 2 introduces validation methods, the next two sections review the most important validation and verification approaches in the relevant literature. Verification is discussed first.

1.6 Verification Approaches for Knowledge-Based Systems

Many authors have introduced approaches to simplify and formalize the verification process for knowledge-based systems [21] [22] [23] [24]. Verification approaches based on conceptual models have been successful [25] [26]. At the same time, languages for the transformation between the model and the real system are important and should be defined. Therefore, many

contributions discuss what languages or logical representations are best for verification. Most recent studies concentrate on representing the knowledge in Petri nets [22] [27] to be able to extract anomalies.

A Petri net is a directed bipartite graph, in which the nodes represent transitions. Petri nets have three main parts: arcs that represent the direction of the data, usually input or output; nodes that represent discrete actions; and transitions - conditions that direct the flow of data [21]. A number of recent verification methods were introduced using the Petri nets approach as in [22] and [37]. Qingfeng et al. [23] represent the rule-based system in Petri nets and verify the rules using the graph. Other approaches represented the systems processes in Petri nets for verification. Representing the knowledge base in graphs or logical representation is desirable for verification. Other published approaches use meta-knowledge to indicate anomalies. [24] Meta knowledge is a higher abstraction of knowledge that also needs to be verified. In most cases, the anomalies that need to be detected are: [24].

1. Redundancy - having the same knowledge in two or more places.
2. Ambivalence - mixed knowledge or unclear representation of knowledge.
3. Circularity - closed loops in the knowledge; a rule leading to itself as a solution.
4. Deficiency - inefficient representation of knowledge.
5. Incompleteness - some expert knowledge not represented in the knowledge base.
6. Inconsistency - any untrue representation of the knowledge.

More recent verification methods tend to move towards formal approaches. Several papers introduce new languages that might help in formally modeling the knowledge base. Languages such as ALCNR [28] and KROL (knowledge representation object language) [29] were

introduced in 1998 and 2007 respectively. Other researchers dealt with uncertainty and discussed how verification could interact with testing and validation [30] using probabilistic models.

Embedding knowledge and the knowledge-based system in a conceptual model is an important step towards defining a comprehensive method. Many authors introduced verification methods based on structural and conceptual models. Marcos et al. [25] introduced a model-based verification method based on a planning model called STRIPS (Stanford Research Institute Problem Solver), which is an automated planner used in many artificial intelligence applications that consists of goals, operators and states. Other work used models of knowledge-based systems for verification. In [26], Al-Korany et al. used KADS (knowledge acquisition and design structuring) for verification of knowledge-based systems. Al-Korany et al. stated that KADS is a model that “lends itself to verification and validation”; KADS is discussed in Chapter 4 in great detail. In this method, every layer is verified separately and a verification report is produced. This method detects three kinds of anomalies: 1) circularity, 2) inconsistency and 3) incompleteness. This is illustrated in Figure 4. This method doesn't detect other anomalies that are discussed previously (ambivalence, redundancy...etc), which is a major drawback. KADS is an abstract representation of the system. To be able to perform successful verification, more details should be included and observed. The transformation between the design model and the real implementation is also important and if the anomalies could be recognized incrementally, this should be an effective way to minimize the work after the system is done. An important point to mention here is that models could introduce new types of defects; therefore, when any model is used, there should be clear definitions on what deficiencies does the model have within different domains.

The transformation between the design models and the real implementation is also important, and if the anomalies could be recognized incrementally, this should be an effective way to minimize the verification effort after the system is done. An important point to mention here is that models could introduce new types of defects; therefore, when any model is used, there should be clear definitions on what deficiencies does the model have within different domains.

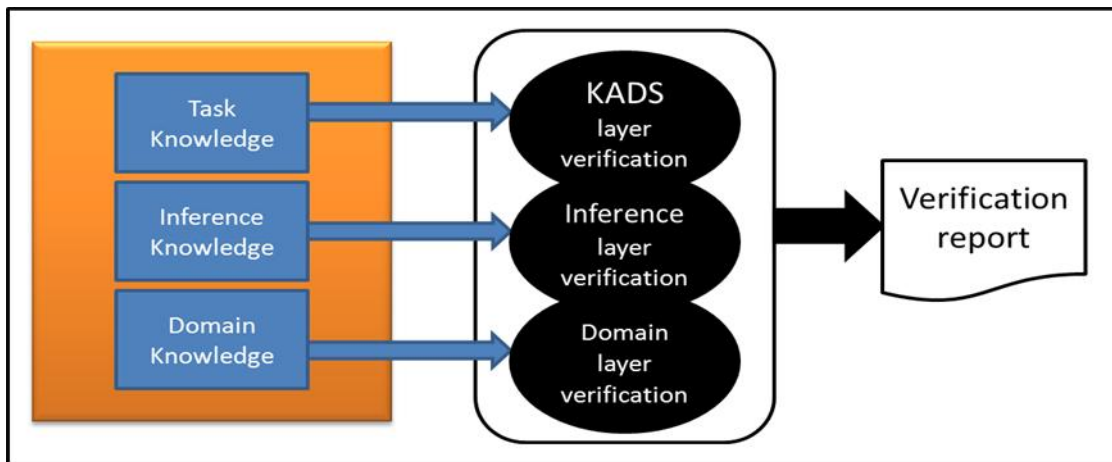


Figure 4: Verification using KADS [29]

In most cases, verification is more formal than validation. More formal software tools were built for verification than validation. Examples of the most common verification tools are [31]: CRSV-CLIPS (a tool that checks for inconsistency and incompleteness in the expert systems shell CLIPS) [32], KB-REDUCER (takes a rule-based system and uses reduction algorithms to reduce its size) [33], COVADIS (a tool similar to KB-REDUCER but mainly concerned with consistency checking) [34], ESC (expert systems checker - a tool dedicated for rule-based systems that represents the system in tables for verification) [35], IMVER (uses matrices to represent knowledge and look for anomalies) [36], PREPARE (rules are translated into transition nets and pattern matching techniques are applied on them to detect anomalies) [37], COVER

(transforms the rule base to the *cover* language and identifies conflicts between rules) [38], CHECKER (part of an expert system shell that uses uncertain reasoning and has two main algorithms to perform verification, check 1 and 2) [39] , EVA (is a toolset that detects unreachable, cyclic missing, redundant and dead end rules by firing them, the system also generated tests to help the developer point out anomalies in rules) [40] and COCO (an incremental checking system that checks for consistency and completeness of rule-based systems) [41].

Verification is an essential process that needs to be performed carefully. Researchers agree that verification becomes harder as the size of the system increases [11]. Smaller projects are easier to verify, but still, stating that a knowledge-based system is verified can be a very demanding conclusion. The main goal of verification is to detect anomalies in the knowledge base, especially incompleteness and inconsistency [42]. Most verification practices in the industry are based on manual verification approaches, such as reviews.

Containment checking is also used for verification of knowledge-based systems. Levy et al [43] introduced a verification method based on the containment checking approach that is derived from database querying. Containment checking indicates redundant and inconsistent rules or cases in a knowledge base by requiring knowledge as if the system is running and observing how many similar answers are returned. If too many similar answers are returned, this indicates a redundancy problem. Again, verification is an essential step but the industry still lacks a standard method that could be used at minimal expense.

1.7 Validation Approaches for Knowledge-Based Systems

Validation approaches can be categorized according to the level of abstraction of the testing process. Black box testing validates the system in a highly abstract manner, by testing inputs and

outputs only, without understanding or validating the underlying details. White box testing is the opposite of black box testing and it is a detailed evaluation of all parts of the system. Validation can be performed on every module of the system, by defining a set of expected outcomes from this module. Furthermore system-level validation is more challenging, as it deals with the system as a black box.

In this section, a list of general validation approaches for knowledge-based systems is presented, discussed and evaluated. The approaches described next are not mutually exclusive but most methods in practice combine several of these approaches. Different validation approaches can be used during different phases of software development. Validation can be performed towards the end of the development process or incrementally, where it is performed in parallel with the development process. The appropriate phase is stated for every general approach.

1.7.1 Validation through Analysis of Heuristics

This general approach is based on performing logical validation with heuristics with uncertainty. This validation occurs after the knowledge acquisition phase [44].

Validation through analysis of heuristics evaluates the outcome of an individual *situation* → *action* heuristic [45]. As Lenat [45] defines them, heuristics are: “pieces of knowledge capable of suggesting plausible actions to follow or implausible ones to avoid”. Langlotz et al. [16] also used the MYCIN expert system for their experiments on this validation method. They suggested that some rules might not be valid for a certain scenario, and that experts would have different views on the validity of those rules. In this validation approach, the knowledge engineer revises the rule base and checks the outcome of rules, based on a certain input using a decision

tree. The decision tree describes the rules structure in the knowledge base, and is used to validate the rules, and their order in the decision making process.

1.7.2 Validation through Simulation

Validation through simulation is an efficient approach used for critical, real time knowledge-based and conventional software systems. This approach is based on simulating the system under the same circumstances. It is important to note that simulation is different from prototyping. Simulation is a representation of the full system, with all parameters that affect its performance, including the human factors such as stress, fatigue and impatience if applicable. This validation approach is also used for validating hardware systems. Validation through simulation consumes much time, effort and money. In some cases, building a simulator could be more difficult than building the real system. This validation approach is common among military applications and other critical systems, but it is not used for socio-technical systems [46].

Validation through animation can be considered a category of validation through simulation. Validation through visual interaction is also called validation through animation. It is based on representing the system visually using different animation means. In this approach, the system is represented by visual entities such as nodes or colored shapes. Colors are important in this approach as they present different types of entities within the system. Example: Input data flow could be represented by a color and the output data flow by another color. After the animated presentation is designed, the system is validated by checking and reviewing all the modules and how they interact with each other. In some cases an animated video is designed to visualize the system modules in action and point out faults and mismatches with the initial requirements.

1.7.3 Face Validation

Face validation is a general preliminary validation approach, where a number of developers, knowledge engineers and some potential users run the system and compare its performance to a human expert knowledge, albeit not in a formal testing process. Face validation is an overly simple process, but it is still used in industry as one of the main validation approaches. Face validation is commonly used for all kinds of systems, including conventional software. It requires human effort and in most cases, it is hard to be automated. Usually, face validation is accompanied by other, more formal testing approaches.

1.7.4 Predictive Validation

This validation process compares previous validation results with corresponding and current results of the system. Predictive validation is performed iteratively. When predictive validation is used, the results are saved, and for the next iteration the same set of tests are performed and compared with the previous results. Predictive validation cannot be used in isolation; it is simply an approach to compare test results and evaluate the development process.

1.7.5 Subsystem Validation

Subsystem validation is based on evaluating the various parts of the system independently. When subsystem validation is performed, another validation approach is used to evaluate the entire system. This method defines the parts of the system that need to be validated but not how to do it. Integration testing is performed on the entire system after subsystem validation is completed to evaluate how different modules of the system will interact.

1.7.6 Validation through Case Testing

Validation through test cases is the most used approach for validation and testing. Knauf et al. [47] defined the main steps of this approach as:

1. **Test case generation:** The test cases are defined. This step is the most challenging, because it builds a basis for the next steps. If the design of the set of test cases was not done properly, the validation process will lose its value.
2. **Test case experimentation:** this is the step where the test cases inputs are executed on the system.
3. **Evaluation of the experiment:** The test case results are evaluated. In most situations, evaluation is done by comparing the test result to the experts' answer.
4. **Validity assessment:** actions toward fixing the errors and presenting the solutions are defined in this step.
5. **System refinement:** the knowledge engineer corrects the errors found in the system's modules, logic and code as a result of the validation process.

In some applications, use case diagrams are employed in designing test cases. Use case diagrams are behavioral diagrams used to represent the users' interaction with the system. They are created after performing use case analysis. Use case diagrams are defined in UML (Unified Modeling Language).

Test case design falls into three categories:

1. **Requirements-based testing:** test cases are defined based on the initial user requirements. A group of test cases are presented to cover every requirement.
2. **Partition testing:** Input data and the resulting output usually fall into categories where all members of a category are connected to each other. These categories are defined by the

knowledge engineer and/or the expert. As Sommerville states in his book [1], “Each of these classes is an equivalence partition or domain where the program behaves in an equivalent way for each class member”. Test cases should be chosen from each partition.

3. **Structural testing:** Also called white box testing, a system undergoes complete testing where all the statements are tested. It is important to note here that not all execution paths are tested but all the code, knowledge and the inference engine functionalities are.

The reliability of any validation process depends on the number of defects, errors and faults found before releasing the system to customers. This depends on the quality of test cases generated. Automated test case generation is an important topic in current research. It can reduce the time, effort and cost of validating the system and increases reliability [47]. Automated testing in validation by Knauf et al. [47] is discussed in detail in this dissertation.

1.7.7 Validation through Graphical Representations

Graphical representations can illustrate a knowledge base in an unambiguous manner. This helps knowledge engineers and/or the validation team in spotting problems in the system. In this subsection, decision trees and directed graphs are presented as ways to depict a knowledge base. They are the most common graphical representations for knowledge-based systems.

1.7.7.1 Decision trees

Decision trees are used for validation because of their clear representation of data. An important part of knowledge-based systems validation is validating the knowledge. Validating the knowledge could be a hectic process if the knowledge is not represented in a graph or any other easy form of visualization of the system. Decision graphs show different courses of action towards making a certain decision. In this structure, the knowledge engineer can explore the

options of a certain input. Furthermore, it can trace the course of decision making of the system to measure the similarity between the expert's knowledge and the system's decision process. The main challenge in using decision trees or any other graphical representation of a system is the size of the graph. Extensive validation is performed when systems are large, but it is hard to present the system in a decision tree when it is complicated and/or if it interacts with other system within or outside the organization.

1.7.7.2 Directed graphs

A directed graph represents the system in a sequential manner. This kind of graphs is used in most cases for data flow representation. It should be clear that validation using graphs doesn't assure the absence of errors, or a full match between the requirements and the knowledge-based system. However, it is helpful in pointing out problems in the system. Software tools are used to present the system in graphs, Murrel et al. [31] present a survey of tools for validation and verification of knowledge-based systems. Validation tools are discussed in later sections of this dissertation.

1.7.8 Validation through formal Methods

Mathematical and statistical representations are widely used for validation of software.

This category is also listed as formal methods. Formal methods are getting much attention in testing of software. In this section the most common approaches are introduced.

1.7.8.1 Simultaneous Confidence Intervals

This validation approach is based on evaluating the system's performance under different circumstances over a period of time. Different modules of the system are evaluated in a *before-after* situation. Confidence intervals are defined for sub parts of the system or for test cases and in certain situations for the entire system.

1.7.8.2 Paired T-Tests

The paired t-test is a testing approach used to evaluate whether the difference between two observations is equal to zero. In software, two scenarios are compared and the results are evaluated. If the difference between the two observations is zero, this means that the system fully reflects the knowledge or the requirements (depending on what is being tested). Paired T-Tests is used in many disciplines and applications. It is a commonly used statistical approach that delivers an informative comparison between two scenarios [48].

1.7.8.3 Consistency Measures

In many domains, experts can honestly disagree with each other on some of the domain's details. This validation approach ensures that the knowledge represented in the system meets the expert's knowledge. Performing this test requires many experts to perform testing on the system; their views and results are analyzed by the experts [48]. The main goal of this approach is to check consistency of the system.

1.7.8.4 Hotelling One-Sampling

Hotelling one sampling serves as a statistical approach to compare the human expert to the expert system. Tests are performed in pairs - expert result and system result. A set of pairs is generated (K). K pairs are compared and each corresponding set of cases is compared with different input values. Then one sample/test t is used to determine whether the means of the array of samples/tests are significantly different or not.

1.7.9 Turing Testing

The original Turing test was proposed by Alan Turing [49] [50] as a way to determine the intelligence level of a machine when compared to a human, where a machine and a human played a game anonymously. This validation version of the Turing test applies the concepts of

comparing human to machine and anonymity to the validation process for intelligent systems. The Turing test is a good candidate for validating systems, but it is a demanding approach that requires significant time and commitment from experts. This approach is based on validating the knowledge-based system by comparing its output to the expert's output. The system and the expert have the same set of inputs, and their outputs are compared [49]. In the Turing test with its input-output validation style, the knowledge engineer would not know the error that needs to be fixed, because of the black box view. They would just identify where the problem is.

1.7.10 Sensitivity Analysis

The sensitivity analysis validation approach is a helpful process for finding problems that the expert or the knowledge engineer couldn't locate or find. Sensitivity analysis is performed at the late stages of validation or testing. It is performed by using the initial experiments/test performed by the knowledge engineer or by the expert, and tweaking them by entering out of boundary values. Example: entering a human being's age = 1000 years old and evaluating the subsequent results. It can be a long and difficult process because of the many potential values that compose sensitivity. However, in special cases it can be quite useful.

1.7.11 Field Testing

Field testing is one of the most common black box validation approaches. In field testing, the system is put under use in the real environment and is used by the ultimate users. User feedback would be considered when changing the knowledge-based system according to the user needs and specifications. In field testing, the testers are the actual users. They have no knowledge about how the system functions.

Field testing can be only used with non-critical systems because the users could potentially rely on an un-validated system. Data collection can be used with field testing. Data collection is an overly simple procedure when compared to other validation approaches, but it can be very useful because it is done by the users who will actually be using the system. It is usually performed when prototypes are developed. Data collection is basically getting input from users by letting them fill questionnaires and/or interviews. Data collection is accompanied with other validation approaches; it is not comprehensive if performed solely.

1.8 Summary

This chapter discussed the problem of developing correct knowledge-based systems. In particular, validation and verification are introduced and discussed as a means of achieving correct knowledge-based systems. Many early software systems and knowledge-based systems have showed that it is difficult to evaluate and maintain them. As a result there has been an interest in developing methods to control the development of the expert system, to test, verify and validate it. The main focus of this dissertation is on the latter, validation of knowledge-based systems. In this chapter, intelligent systems were introduced. Knowledge-based systems - a special kind of intelligent systems is the major focus of this chapter. Validation and verification techniques are defined and discussed. All the validation approaches discussed in this section are summarized in Table 1.

Validation could be performed in every step of development, or could be performed after an initial prototype of the system is built, or even at the end of the real system development. Validation could be performed solely, or accompanied by evaluation. A set of validation approaches for knowledge-based systems was introduced in detail in this chapter. The aim of this dissertation is to introduce a new validation method for knowledge-based systems. The

validation method proposed here is not just another validation method. It is a structural and incremental validation method based on a conceptual model.

Table 1: Validation general approaches comparison

Validation Approach	Development Phase	Category/Description
Validation through Analysis of Heuristics	After knowledge Acquisition	Logical validation with uncertainty
Validation through Simulation	After prototyping	Result-oriented validation
Face Validation	After development or mature prototyping	Qualitative Validation/ Preliminary approach
Predictive Validation	After development or mature prototyping	Qualitative Validation
Subsystem Validation	All phases	Qualitative Validation
Validation through Case Testing	After development or mature prototyping	Result-oriented validation
Validation through Graphical Representations	After Knowledge Acquisition/Representation	Visual Validation
Decision trees	After Knowledge Acquisition	Visual Validation
Directed graphs	After Knowledge Acquisition	Visual Validation
Simultaneous Confidence Intervals	After Partial Development	Statistical/Quantitative Validation
Paired T-Tests	After Partial Development	Statistical/Quantitative Validation
Consistency Measures	After Partial Development	Statistical/Quantitative Validation
Hotelling One-Sampling	After Partial Development	Statistical/Quantitative Validation
Turing Testing	After development or mature prototyping	Result-oriented Validation
Sensitivity Analysis	After Experts Testing (if necessary)	Result-oriented Validation
Data Collection	After Partial Implementation	Usage-oriented Validation
Field Testing	After Full Implementation	Usage-oriented Validation
Visual Interaction Validation	After Knowledge Acquisition	Visual Validation

CHAPTER 2: STATE OF THE ART IN VALIDATION OF KNOWLEDGE-BASED SYSTEMS

This chapter describes the state of the art in methods to validate knowledge-based systems. Many validation methods have been recently introduced in the literature based on one or more of the validation approaches presented in the previous chapter. In the previous chapter of this dissertation, general approaches to validation were discussed. Those approaches cannot ensure that the system is valid, as they are not complete and they only provide a general description of what can be done. On the other hand, methods can ensure a valid system as they introduce step-by-step techniques towards complete system validation. Validation methods are reviewed and discussed in this chapter.

2.1 Validation Methods for Knowledge-Based Systems

This section discusses validation methods categorized in three sections: 1) validation of the knowledge base, 2) validation of the entire system, and 3) multi-purpose validation methods. The first section reviews validation methods strictly dedicated to validating the knowledge base. The second section discusses validation methods that aim to validate the entire system. The third section is dedicated to other types of validation methods and tools.

2.1.1 Knowledge Validation Methods

As the validation definition adopted in Chapter 1 declares, the output of the system needs to be equivalent to the human experts' output. The human experts' knowledge is represented in the knowledge base. Therefore, it is important to validate the knowledge base. Different methods [51] [52] [53] [54] have been introduced for this purpose. One of the ideas to achieve this goal is

to keep track of knowledge documentation as a reference for measuring the validity of the knowledge base.

A validation method based on this idea is KVAT (knowledge validation tool) [54]. KVAT performs knowledge elicitation, and validation for frame-based and logic-based systems.

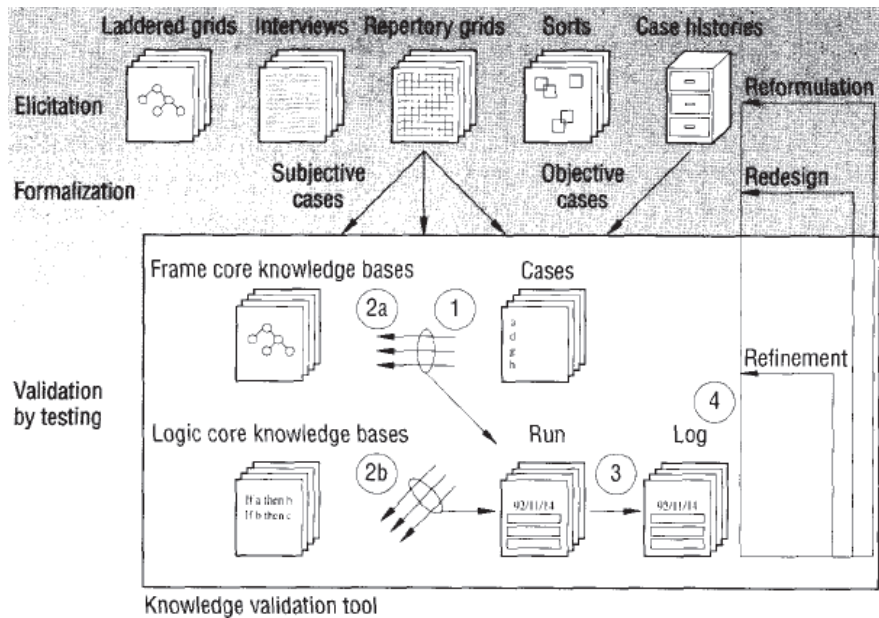


Figure 5: KVAT [54]

As illustrated in Figure 5, KVAT’s knowledge elicitation is based on labeled grids, interviews, sorts and case histories. The KVAT validation method starts by performing manual case testing on the system. If KVAT finds any error during case testing, it locates the source of inconsistency in the knowledge and compares it with the elicitation documents to perform refinement on the knowledge. This method was introduced to be used across all development phases, but it’s mainly useful for knowledge representation validation. KVAT automates the stage that compares the knowledge to the validation documents, but case testing is still manual and informal. Formal methods are more reliable as they provide a mathematical proof of the validity of the system.

Santos et al. [55] presented a formal validation method using case testing and the probabilistic Bayesian networks. To use the method presented in their paper, the knowledge base should be what the authors call, *Bayesian knowledge base* (BKB). BKB are represented in Bayesian networks in an *if-then* format. This is an advantage because knowledge engineers are familiar with such a format. For validation, a set of test cases is designed by deriving it from the Bayesian knowledge base and the initial user requirements [55]. Every test case is a pair of evidence and expected answer. The evidence represents the requirement location in the knowledge base and the expected answer is the right solution. These sets change depending on the incoming results and outputs of the testing process. Changing the set of test cases means that new test cases will be added to or deleted from the set. In this method, every requirement from the users is represented by a test case. The Bayesian representation in this method is based on nodes. Every evidence and expected answer is represented by a node. If there is a redundant or a cyclical representation, it indicates a problem that needs to be corrected.

A more generalized validation method for knowledge representation was introduced by Zlatareva [52]. Her method for validation, verification and refinement and is called VVR. For validation, VVR uses the concepts of overgeneralization and overspecialization to modify and refine the knowledge base. Overgeneralization of the knowledge base is when more rules need to be added to the knowledge to make it valid. Overspecialization of the knowledge base is when rules need to be removed from the knowledge base because they are redundant, wrong or not representative of the user specifications and requirements. In VVR, validation starts by manually generating test cases for the system. VVR uses two kinds of test cases: negative and positive cases. Positive test cases are used to show the validity of part of the knowledge base; they are represented in three variables: input, expected result and real result [52]. Negative test cases are

used to show that the negation of the inference outcome is wrong. It's harder to generate negative test cases, but they can be extracted from the positive cases. The idea of positive and negative test cases unveils different kinds of problems in the system being validated. In some cases, during validation and refinement, adding more rules to the knowledge base could be the solution. In such cases, the knowledge engineers would not know whether there is redundancy in the knowledge base or in the intelligent program's code. This can make the size of the program grow without a justified reason. Negative test cases and overspecialization can address such a problem. The idea of overspecialization reduces the size of the code and the knowledge base without affecting its performance.

Two other methods based on overgeneralizations and overspecializations that predate VVR are SEEK and SEEK2. They have been widely used for knowledge-based systems validation [56]. SEEK is considered one of the pioneers in validation of knowledge representation. SEEK was completed in 1982 while SEEK2 was completed in 1988. The main goal of the two methods was rule validation and refinement for the expert systems concerned with diagnosis of diseases. The main reason for the development of the tools was to ensure that the expert system conclusions were similar to known diagnosis results. SEEK and SEEK2 have applied more than 1500 times [56]. Another method for knowledge validation was recently presented by Barthelemy et al. [57] based on graphical representation using conceptual graphs. They introduced a method to detect anomalies in the knowledge base. They call the process validation although detecting anomalies is a verification practice according to many definitions [22] [23] [24] [25]. However, in this method, the knowledge is separated into two types, terminological (contains the main knowledge and the basic terms) and assertional knowledge (contains a number of conceptual graphs that are based on the terminological knowledge).

Assertional knowledge is similar to using meta-knowledge to represent the knowledge base. Terminological knowledge helps in forming the assertional knowledge that is validated manually by the experts. Graphical representation is helpful in envisioning the knowledge base, but without a tool that can transform the knowledge to graphical form, it is difficult and time consuming to use such methods.

2.1.2 System Validation Methods

The validation methods in this class perform validation on the entire system by executing it. In the previous section the methods were concerned with validation of the knowledge representation. In this section, methods compare the system's output to the expert's outcome. One approach to performing system validation is using validation software agents, or in other words, expert systems that can perform validation. A simulation validation method was introduced recently based on using three knowledge bases for validation [51], validation knowledge base (VKB), decision knowledge base (DKB) and validation techniques base (VTB). This validation method starts by analyzing the knowledge-based system. VKB represents the expected behavior of the system. A validation technique is selected from the VTB by the DKB according to the VKB system's representation [51]. Another module in this method is the evaluator. As VKB represents the expected final behavior, the evaluator compares the results of the validation technique selected from the VTB to VKB's representation. DKB selects a validation method based on the representation of VKB using a predefined set of rules from the expert. The evaluator creates a report as a result. The report consists of both the validation outcome and the VKB representations. Different parts of the validation system are shown in Figure 6. This validation method relies on the correctness of the decision making of DKB, and the representation logic of VKB. It is also based on the assumption that the three expert systems

(VKB, VTB and DKB) are valid and verified. Different validation methods could be incorporated within this validation system (specifically into VTB), which is an advantage. This system can be updated periodically to include new validation techniques.

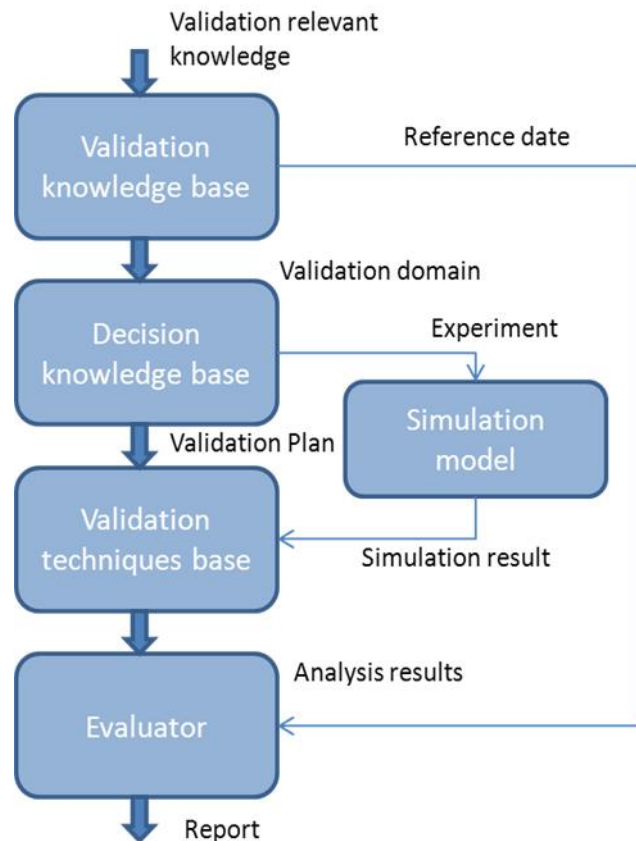


Figure 6: VKB validation method [51]

The authors didn't validate their software agent their paper. This is a problem that applies to any validation software tool, how do you validate the validation tool? The tool should be validated to ensure a robust performance. Building the tool on top of an already tested application/system is a straight forward way of solving this problem, or at least reducing the validation effort. Knauf et al. [47] introduced a validation method based on their previous work for introducing a reasonable set of test cases (ReST). They introduce the validation knowledge base (VKB), which is the

collection of many experts' expertise and the validation expert system agent (VESA), which is a software agent that represents one expert's knowledge. This method uses the case testing approach. The VESA and VKB modules can replace the experts after a certain amount of time of learning to validate knowledge-based systems. VKB is a group of test cases, and a set of results that represent the correct results of testing [47]. The use of VKB within the validation process is illustrated in Figure 7. VKB consists of knowledge dealing with: test case generation, test case reduction using ReST, test case rating, evaluation and refinement.

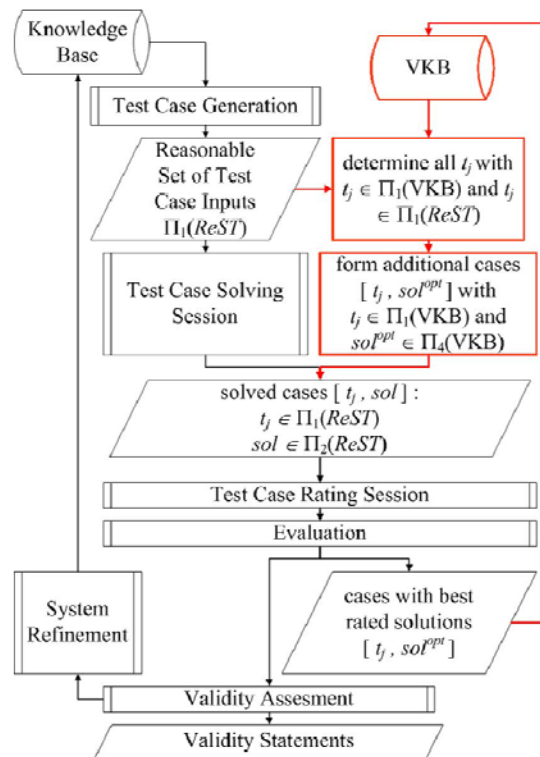


Figure 7: VKB within the validation process [47]

The main idea in this method is that when manual testing starts, VESA could replace any of experts performing testing. If the expert is not available during testing, the VESA should have gathered enough knowledge from the expert and VKB during earlier stages of development so it can replace him/her in the testing process. VESA uses VKB for decision making. Furthermore,

VESA extracts the experts' test cases ratings and certainties. This method reduces human involvement in the validation process. Experts and knowledge engineers could be replaced in this method. The only constraint is that this cannot happen before performing a certain number of tests on the system [47].

Reducing human involvement in validation is a desirable practice. The VESA method presented is successful in involving software agents and truly decreasing effort and time and increasing reliability. Test cases validate the user needs and specifications in a structured manner.

Another validation method based on test cases was introduced by Smith et al. [58]. They presented their method through a tool called the CASE VALIDATOR. It's an automatic record-keeping tool of the test cases used for validation and it provides analysis of these test cases and how they cover the knowledge base. The test cases are manually entered into the tool through a user-friendly interface. For every test case, the results, certainties and conclusions are saved after the execution. Part of the tool is called the record-keeping facility. It keeps track of any changes on the program and compares the results of the test cases associated with that change. The coverage analyzer is another part of this method/tool. It consistently checks for any part of the program not covered by the test cases in the record-keeping facility. CASE VALIDATOR is a manual approach that is impractical for large knowledge bases. The authors stated that their method runs test cases on the program in a straight forward manner [49] [58].

The Turing test was used to validate one of the most famous expert systems discussed previously in this dissertation: MYCIN. In MYCIN's validation, several physicians were performing an evaluation of the system [59]. There were many disagreements after performing Turing tests on how to evaluate MYCIN. Onoyama et al. [60] present a validation method and

compare it to Turing test. The authors declared that the Turing test is a good candidate for validating systems, but it is a high demanding method that requires significant time from the expert. Therefore, the authors presented a method called *Bi-directional many-sided explanation typed multi-step validation* method. It distributes the validation effort among the expert, knowledge engineer and the system. It is worth mentioning here that when experts are exposed to tests, their knowledge will be under test and all their knowledge will be in the system. Some experts might think that they will lose their superiority, which might affect their input into the test. This is considered a drawback for the Turing test. In this method the authors want to eliminate this factor. This method is a multiple step validation process. The tests that the expert performs are the ones that neither the computer nor the knowledge engineer can perform.

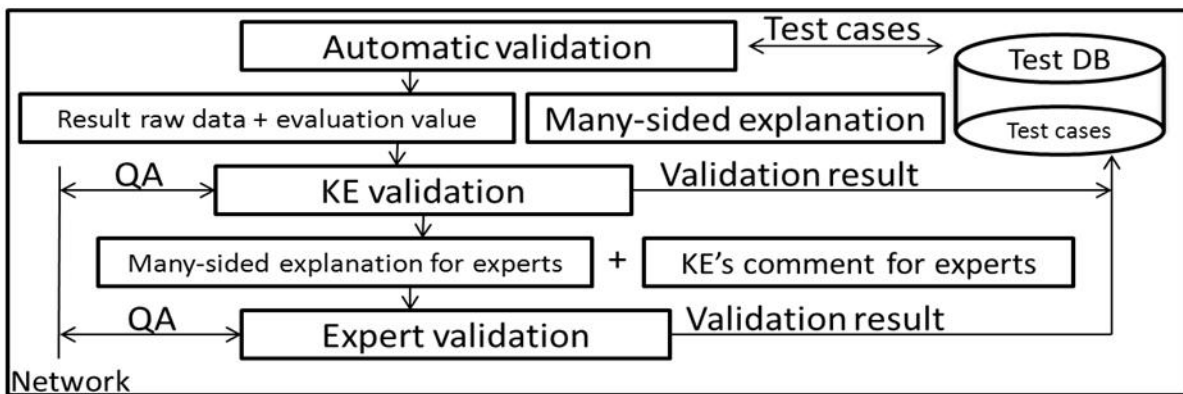


Figure 8: Bi-directional many sided explanation typed multi-step validation method [60]

The rest is left for the knowledge engineer and the automated computer process. The computer stores all the tests' answers. Comments are also added to the system by the knowledge engineer and the experts. The results of the validation are stored in the database, as a validation case base, which could be used in subsequent validation exercises. After the validation process, all the

results from the expert, KE and system are aggregated and documented. The Bi-directional many sided explanation typed multi-step validation method is illustrated in Figure 8 [60].

This method has two major drawbacks. The knowledge engineer and the expert perform validation separately. Additionally, this method requires effort in project management and communication between experts and knowledge engineers which can also be expensive. Min [61] introduces a method for validating military systems and introduces a tool based on this method. As illustrated in Figure 9, the knowledge base is parted into segments and an expected behavior for every segment is predefined.

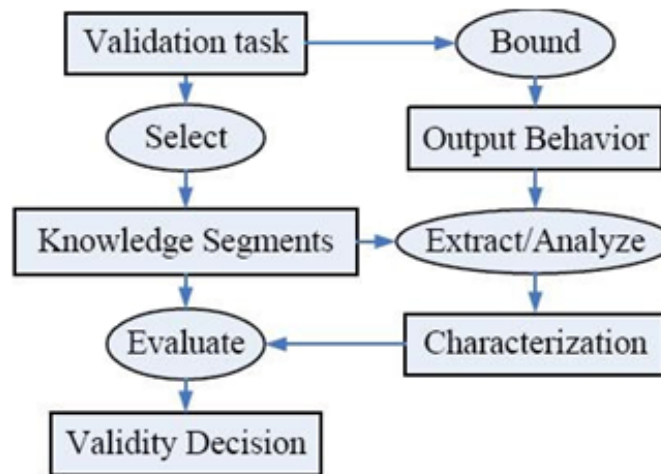


Figure 9: Validation of knowledge-based military systems [61]

In this method, a segment of the knowledge base is evaluated using simultaneous confidence intervals [61] (see Chapter 1). A formal validation method was introduced through a software tool KJ3 by Wu et al. [62]. KJ3 stands for knowledge judgment version 3. This tool uses the graphical representation general approach. KJ3 consists of four main modules: the user interface (facilitates the usage of the tool), the validation module (the main module responsible for validating the system), the axiom translator module (converts the validation tasks into

reachability first order logic) and the proof interpreter (provides messages to the user about the validation results). In this tool high-level Petri nets are used to represent the knowledge base. These high level Petri nets are a meta-representation scheme to present the knowledge. To validate the Petri net means validating the expert's knowledge. It is required that such "equivalence is in a one-to-one mapping relationship" [62] between the knowledge and the Petri net nodes. The selected procedure has to mirror the inference practice of the original knowledge base. KJ3 could be used to validate any knowledge representation.

A validation method that aims to reduce a large number of test cases was presented by Gonzalez et al. [63]. In their method, they based test cases selection on heuristic techniques. In their paper, they introduce and validate the Generator Artificial Intelligence Diagnostics (GenAID) system [63]. It was developed in the 1980's at Westinghouse Electric Corp. Because the cost of systems diagnosis can be high, customers look for a detailed and solid assurance that the GenAID system is reliable. GenAID was developed as semi-independent modules. The paper described two types of modules, self-contained modules and overlapping modules. Overlapping modules are harder to validate, as they cover more than one part of the system. The system is represented as a graph and different paths are tested. Once that is done, the test cases are generated using heuristics. One disadvantage of this method is that heuristics are fallible and they don't ensure a 100% valid system.

Abel et al. [64] presented a formal method to validate systems using structural knowledge. The idea of testing every input to the system is impossible to implement in most cases. Abel et al. [64] use formal approaches to reduce the number of test cases to present successful validation method. Their method builds a minimized test cases set. They try to eliminate the generation of a *functionally exhaustive set of test cases* and build a *quasi-*

exhaustive set of test cases (QUEST) instead. This set of test cases is meant to represent all the system cases in a reduced number of test cases. In the *quasi-exhaustive set of test cases* it is sufficient to assume that if a subset of the test cases (T) is valid then the whole set (S) is valid too. Test cases sets are defined and for each set a subset of test cases (T) will be executed. A set of mathematical classifiers is used to categorize the knowledge into sets where an object can belong to one or more sets. In this method the authors used statistical and formal means to achieve that. A year later, Abel et al. [65] criticized the *quasi-exhaustive set of test cases* discussing that it still might be unacceptably large and that it is impractical because of its large cardinality. They introduced another method to further reduce the number of test cases for expert systems. In their method, they proposed a criteria-driven reduction of test cases to generate a reasonable set of test case (ReST). This method includes four stages as illustrated in Figure 10.

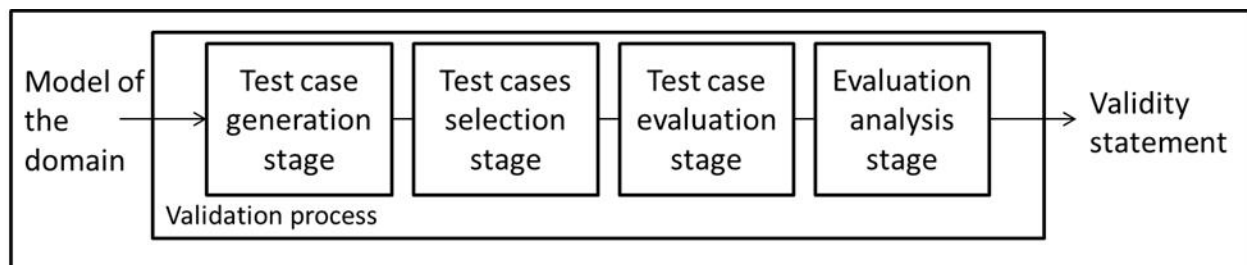


Figure 10: Validation process proposed by Abel et al. [65]

During the test case evaluation stage (refer to Figure 10), all test cases need to be evaluated by the validation group. These evaluations are analyzed in the fourth step. During the test case selection stage, a criterion is defined by answering the question: how well should the model perform for it to be considered valid? Answering this question requires looking at different criteria: domain related, user related, and expert related. This method ensures the reduction of the set of test cases, less than the *quasi-exhaustive set of test cases*. The reduction of test cases is still

a challenging problem. It's not feasible to run all the test cases on the system, especially when some of these may not be even physically possible in the real-world.

2.1.3 Multi-Purpose Validation Tools

This section describes generic validation methods and tools. One of the recent generic validation tools was introduced by Rey et al. [66]. They presented a tool called SHIVA (Spanish language acronym for Heuristic Integrated Validation System). SHIVA uses a black box general validation approach that contains three main steps: planning, application and implementation. In the first step, planning extracts the domain, user and system's characteristics. During the application step, SHIVA aims to decide on the quantitative method to be used for validation and applies it to the system. SHIVA selects one of the following quantitative methods for validation: pairs' method (two or more experts' interpretations are compared to the interpretations of the system), association measurement (models the outcome of the system and the expert then calculates the degree of linear association between them) and agreement measurement (for a given set of rules or any other knowledge representation. An agreement index is defined, and the knowledge is validated in comparison to this index).

The output of this process defines the distance between the index and the knowledge under test [66]. SHIVA is not incorporated into the development process, which is a drawback. The knowledge engineer needs to pick a validation method for the system. If the validation tool was already incorporated, this would save much time and effort and, most importantly, would increase reliability because of the lack of any compatibility issues. Based on unifying validation and development, a validation tool called DIVER is incorporated into the spiral development model [67]. During each validation cycle, DIVER finds potential logical and semantic inconsistencies defined in the knowledge base. Detected inconsistencies are sorted into levels.

This sorting is based on which inconsistency happened before; earlier ones need to be solved first because inconsistencies at earlier levels may have a negative effect on later ones. When the development is over with this method, there is no need for validation except for the last modifications on the system. Complete system testing is recommended here, however.

A generic validation tool called CORUS was introduced by Abdullah et al [68]. It is built on another system called the *resolver*. The resolver handles numerous decision criteria. CORUS was developed following the spiral model. For any iteration, the system returns five expected answers or less to the user for every test case. Additionally, it proposes one certain expected solution. CORUS has the following parts: a database, knowledge base, inference engine, a case acquisition tool, a user interface and an explanation module. The system validates the results based on assigning 'profiles'. For each of the returned solutions, the system associates a profile so that equal solutions have equal profiles. The expert and the knowledge engineer manually compare the outcome of the system to the recommended outcome from the tool.

A generic tool called KRFOCL was introduced by Murrel et al. [31] based on backward chaining. This tool automates the tracing of errors back to the rule-base and selects the rule or set of rules that are responsible for the errors. Traceability is used to link the outside view of the system to its inside. KRFOCL can detect missing, contradictory and unnecessary rules. This tool requires a large set of test cases to initiate the process of traceability and the test cases should run to indicate where the errors are happening. It would be useful to have traceability back to the requirements, so the real world would be represented in a more complete manner in the system.

Another tool was presented in the literature is called EITHER [69]. This system is a method as well as a tool. EITHER uses the logic of overgeneralization and overspecialization discussed earlier. Furthermore, EITHER uses a *theory tree*, which is a binary tree with AND/OR

representations. If the error is as a result of overspecialization, the rules associated with the problem will be removed after searching for replacements from the tree. EITHER only validates the knowledge base, it doesn't validate the entire system, which is a major drawback. Validation is intended to capture the experts' knowledge representation problems. Incomplete knowledge bases don't represent the knowledge of the expert fully, and incorrect knowledge bases represent knowledge in a wrong way. Taking that into consideration, Lockwood et al. [70] proposed that knowledge validation and knowledge elicitation should be performed in parallel. They introduced a tool called EMBODY (Expert Modeling Based on Decision Modularity) that performs both elicitation and validation. EMBODY can generate the rules of the knowledge base. It uses *knowledge charts* to generate the rules. Knowledge charts acts like a bridge between the expert and the rules. They are intended to capture the 'flow of thought' of the expert and illustrate it. For validation, EMBODY ensures that no redundancy, incompleteness, or any kind of error is occurring. EMBODY validates as it goes, and when the last step is reached, only the last step needs to be validated. This approach saves time and manpower, but it doesn't perform total system validation, which is essential for claiming that the system is valid. As claimed, EMBODY is embedded into the development process, but it doesn't validate against the real world, which is the main challenge of validation. EMBODY lacks a link to the user requirements and the domain. It captures the expert's knowledge in a valid way, but not the intended performance of the system.

Looking at all the validation methods in the literature, it should be noted that many efforts exist towards making validation a more robust process. Nevertheless, it is important to note here that validation in the conventional software world is more developed and more mature. Conventional software validation methods cannot be directly used for knowledge-based systems,

because of the differences in the nature of the two types of systems. Researchers in this field focused more on other aspects such as knowledge elicitation and knowledge representation but didn't give much attention to validation. Verification can be viewed as part of validation; a valid system is a verified system while a verified system is not necessarily valid. More work is done on verification of knowledge-based systems than on validation, for which general approaches may have been developed, but an overall incremental validation method that can be implemented across all phases and domains is still lacking.

2.2 Lifecycle Validation of Knowledge-Based Systems

As presented in previous sections, validation should be performed at any and all levels of the system development stages. There are different methods for validation that could be used for different purposes. Some of the validation methods were presented to perform validation on the knowledge base. Other validation methods perform validation on the entire system. Combining two methods to validate the knowledge base and the entire system can be a tricky practice. None of the methods proposed perform validation across all development phases. Furthermore, none of the mainstream methods presented is completely based on a life-cycle model for system development. Researchers have looked into incorporating validation into a conceptual software development model [71] but success there has been limited. A number of methods have been introduced in the literature but none is widely used or agreed upon. After working with different validation general approaches, O'keefe et al. [72] concluded that "We should build validation into the development cycle". A clear-cut paradigm to build software is still emerging. Knowledge-based systems are still behind in that aspect. This immaturity would lead to software failures, financial loss and in some cases loss of human life. Researchers have tried to embed knowledge-based systems development steps into some conventional software models. Lee et al

[71] incorporated knowledge-based systems development phases into the spiral model (shown in Figure 1). This incorporation is shown in Figure 11 [71]. It is important to note that in this model, the authors suggested using two types of validation approaches, case testing and field testing with data collection (all discussed previously in this dissertation). This indicates the need for more than one validation approach for the outcome of validation to be sufficient. This also indicates that validation is better when performed incrementally (spiral iterations) to avoid late risks and unfixable problems.

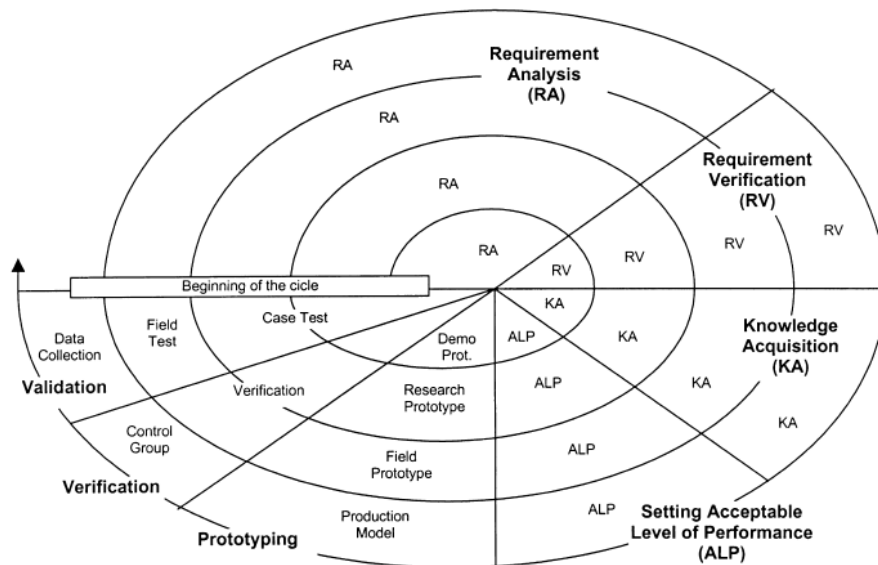


Figure 11: Spiral model for knowledge-based systems [73]

Researchers have looked into applying conventional software validation models into knowledge-based systems [71] [73]. Vermesan et al. [74] developed a system and a tool that evaluates software validation methods and checks their applicability to knowledge-based systems. They introduced a scale categories of classifications, including HA (highly applicable), A (applicable), LA (Low applicability), and NA (not applicable). What Vermesan et al. [74] presented helps in determining whether a method is applicable or not based on *mutation testing*, which is defined as

the process of modifying program's source code in a repetitive minor fashion. These so-called *mutations* are based on well-defined *mutation operators* that either imitates user errors (example: using the wrong operator) or oblige the formation of valuable test cases. The purpose is to help the tester develop effective tests or locate weaknesses in the test data used for the program or in sections of the code that are seldom or never accessed during execution. The approaches that were tested are control flow analysis and cause-effect graphing.

For most of the techniques developed for conventional software, applying these to KBS is not straight forward. The framework that the authors present accepts a validation method as an input and produces a method oriented to knowledge-based systems as an output; with some indication on how similar are they to each other. No significant results were reported from this system.

Other researchers looked at the problem from different points of view. Instead of deriving validation methods from other fields and defining a software development life-cycle model, they developed a validation method that is life-cycle independent. Wells [75] defined such a life-cycle independent method called VIVA. VIVA consists of two parts: a set of productions that describe the system, and a set of steps to show how the products can be used to validate the system. The VIVA method views the system as a group of entities and a group of links between them. The links have the following properties: link's existence (reflects whether the link between two objects exists or not), connection (the entities that the link connects), referencing (whether entities at each end reference themselves or each other in a correct manner), completeness (describes if the representation is complete), and correctness (describes if the link is connecting two right entities).

The definition of VIVA is ongoing and will continue by trying to align VIVA to all kinds of software systems in any domain and adding more method guidelines. As seen in this section, methods for the validation of the knowledge base were developed such as BKB, VKB, KVAT, SEEK and SEEK2. Furthermore, methods for system validation were developed, such as Bi-directional many-sided explanation typed multi-step validation, VESA, CORUS, Decision KB, CASE VALIDATOR, KJ3, VVR and quasi-exhaustive set validation. Additionally, other multi-purpose validation tools were developed such as SHIVA, DIVER, EITHER, CORUS and EMBODY. Some of the defined models were confirmed by the authors to be a success through experimentation (SHIVA, KVAT, KJ3, SEEK, SEEK2, VESA, CORUS), some others failed to introduce that proof (BKB, VKB, Decision KB, CASE VALIDATOR, VVR), while others were not tested or evaluated (EMBODY, DIVER, EITHER). One of the goals of this research is to show that none of these methods found complete success because none of them is fully incorporated into a life-cycle model. If validation is built within a life-cycle model, fewer errors and bugs could be found with less maintenance time. Bohr bugs (named after Bohr atom model) are bugs that could be easily regenerated. However, Heisen bugs (named after the Heisenberg uncertainly principle) are the ones that are difficult to regenerate or duplicate. Obviously, Bohr bugs are easier to trace. Thus, a well defined life-cycle model creates more Bohr bugs than Heisen bugs, which means less validation and maintenance time [76].

As it is known, developing knowledge-based systems is a multi-phase process. In every step major changes are made on the system. Furthermore, there are multiple modules that need to be validated. Selecting a validation method for every module or for every step is a troublesome process because many reliability and dependability problems might appear. Many researchers in this field [71] [72] [73] [74] agree that incorporating the validation process into a software

development life-cycle is vital. Some of the methods presented here failed because of their difficulty of implementing in real life because they need much effort from the knowledge engineer. Knowledge engineers would rather use a simpler validation method. Although many methods and approaches are presented to validate a knowledge-based system, measuring the validity of a system is still a challenging process. It is still difficult to know when to stop performing validation and when will the knowledge engineer know if the system is valid. Using a well defined life-cycle model for validation that is incorporated into the development process definitely helps reduce the ambiguity of when and where to stop, especially by performing validation in parallel with development in an incremental manner. Some researchers [68] [73] use the spiral model as the lifecycle model, but the spiral model is not a dedicated knowledge-based systems development model. Lifecycle models dedicated to knowledge-based systems are introduced in Chapter 4.

2.3 Summary

The state of the art in validation of knowledge-based is introduced in this chapter. Methods are presented in three categories: knowledge validation methods, system validation methods and multi-purpose validation methods. This chapter asserts that no formal validation method is fully built within a life-cycle development model. The next chapter states the problem definition and contributions.

CHAPTER 3: PROBLEM DEFINITION AND CONTRIBUTION

This chapter presents a concise statement of the problem that is the motivation behind this research. The first section introduces the problem definition. The first part states the problem of software validation in the field in general and the second part states the specific problem of performing validation as part of a life-cycle model. The second and the third sections respectively present the hypothesis and the contributions of this research to the knowledge engineering, software engineering and validation research communities. The final section introduces the test plan (evaluation criteria) for the proposed validation method.

3.1 General Problem

The general problem addressed in this dissertation is the improvement of validation of knowledge-based systems. For many years, researchers have been concerned with means of developing quality and reliable software. The best way to ensure a good system design and minimize the probability of failure is to rigorously perform validation and verification. It has been well established now that designing, implementing and validating knowledge-based systems are challenging processes. Different ways of approaching this problem have been established, both formal and informal.

Validation is a complex process, an important phase of software development and an essential part of the development life-cycle. Performing validation at the end of software development is unquestionably expensive, that's why it has to be planned for and guided within a well defined process.

3.2 Specific Problem

The specific problem addressed in this dissertation is how to make use of results of prior test cases in a progressive validation procedure to facilitate defining the minimal set of test cases that provides complete and effective system coverage of knowledge-based systems. In this dissertation, the case testing approach is used. Therefore, correctly executing all possible test cases for any system will surely result in a valid system. However, it is a highly impractical process. Therefore, creating a minimal set of test cases that provides coverage of the knowledge-based system is important. Minimizing the set of test cases could be done either randomly or based on a guided process. Generating random test cases is undesirable because it doesn't guarantee full coverage of the system. On the other hand, a guided process aids in ensuring that the right test cases are selected. When validation is performed, the system undergoes changes and refinements. Thus, in this research, test case reduction and selection are based on the feedback from the previous validation stages. Furthermore, a validation method for knowledge-based systems that is seamlessly integrated in an incremental lifecycle model facilitates extracting the test cases. This idea has never been done before and its absence creates a gap in the field. Thus, instead of introducing a new and untested lifecycle method with a validation step in it, this research seeks to incorporate validation into one of the most common knowledge-based systems development models.

3.3 Contributions

The research presented in this dissertation makes the following contributions to the software engineering, validation and knowledge engineering research communities:

1. Introduced a comprehensive and semi-automated validation method for knowledge-based systems.
2. Conceived a procedure that determines a minimal set of test cases that would cover the knowledge-based system. Test cases are reduced based on feedback from the previous validation stage.
3. Defined a relationship between lifecycle models and validation of knowledge-based systems.
4. Integrated an incremental validation method within CommonKADS. This would enhance the CommonKADS set of models by making it perform validation for knowledge-based systems throughout development. This is the first validation method integrated within CommonKADS.
5. Provided a knowledge-based system (the housing KBS) that could be used for experimentation.
6. Developed a user-friendly software validation tool that represents the validation method.
7. Performed rigorous evaluation of the proposed method and provision of test results.

3.4 Hypothesis

Creation of a minimal set of test cases based on previous validation results embedded within an incremental lifecycle model (CommonKADS) provides more effective and efficient means for validation of knowledge-based systems than current methods.

3.5 Evaluation Method

To ensure that our method is robust, it's necessary to establish a criterion against which the method is tested. First, a non-trivial knowledge-based system using CommonKADS was developed. The CommonKADS knowledge elicitation process was used and the knowledge was presented in all the suitable CommonKADS models. The validation method was then performed on this system. Test cases were extracted for the system and validation was performed incrementally. Test case reduction was carried out and evaluated. Time and manpower were recorded for this process.

Additionally, errors were intentionally seeded into the systems. The goal was to determine whether the set of test cases uncovers these seeded errors. Errors were seeded by two different users; the developer/author and multiple humans test subjects to avoid any kind of bias. Validity and results of the system after each stage were recorded and evaluated. Furthermore, the validation method presented was qualitatively compared against other validation methods in terms of resources consumption. This process evaluation process aims to suggest an effective and efficient method, as it is presented in the hypothesis. The error finding process evaluates effectiveness and comparing the method to other validation methods evaluates its efficiency.

CHAPTER 4: LIFECYCLE DEVELOPMENT MODELS FOR KNOWLEDGE-BASED SYSTEMS

This chapter is dedicated to introducing the lifecycle development models for knowledge-based systems with a special focus on the CommonKADS set of models. This chapter discusses the reasons why CommonKADS was selected as the lifecycle model for the validation method of this dissertation. Additionally, this chapter introduces the CommonKADS models in detail with all the worksheets, UML diagrams and other models' components. In order to understand the validation method within CommonKADS, one needs to understand CommonKADS with all its models, outcomes and processes.

4.1 Life Cycle Models for Knowledge-Based Systems

As it's mentioned previously, building knowledge-based systems is not a trivial task. As for conventional software systems, there are many life-cycle models to follow for knowledge-based systems development. Given the objective of this research to compose a validation method that is integrated within a KBS development model, such models are discussed next. DESIRE, KBSDL, Generic Tasks, KADS and CommonKADS are common examples.

4.1.1 DESIRE

DESIRE (framework for Design and Specification of Interacting Reasoning components) is a group of explicit models used to represent the knowledge, interaction, and coordination of complex tasks and reasoning capabilities in agent systems [77]. DESIRE was originally introduced as a model for specifying complex knowledge-based systems. It views the system as a series of interacting pieces (objects). DESIRE is good for building multi-agent systems or any

other type of system composed of multiple objects because it decomposes the tasks and/or the objects within the system. The main steps in DESIRE are: 1) task decomposition (this step classifies the different tasks that the targeted system should be performing), 2) information exchange (defines data exchange between tasks), 3) sequencing of subtasks (orders the subtasks for each task), 4) subtask delegation (allocates all the subtasks in the KBS), and 5) knowledge structures (defines the knowledge structure for the KBS).

4.1.2 KBSDLC

KBSDLC (Knowledge-Based Systems Development Life Cycle) is a prototyping model for knowledge-based systems [78]. KBSDLC is a sequential development model. It is one of the traditional knowledge-based systems development models that have been around for a long time. It is a helpful model but it over simplifies the development process.

4.1.3 Generic Tasks

Generic Tasks is another model for building expert systems [79]. Generic Tasks are high level building blocks for system development. Generic Tasks is similar to the object-oriented paradigm: it defines the system, the organization and the users interacting with the system as objects. Generic Tasks transforms everything to blocks/objects. These objects are assembled together to form the system. Generic Tasks support reusability, as the objects could be used in any other system. However, it is not highly dedicated for knowledge-based systems.

4.1.4 KADS

KADS (Knowledge Acquisition and Design Support) represents knowledge in three layers: 1) task, 2) inference and 3) domain layers, as shown in Figure 12. KADS was developed as a result of six years of work, and it was produced in the early months of 1990. Its power is in its usage of

models, where every stage is described in a model. KADS has four main models: domain (domain information), inference (description of all inferences), task (goals and tasks), strategic (control knowledge, Meta rules).

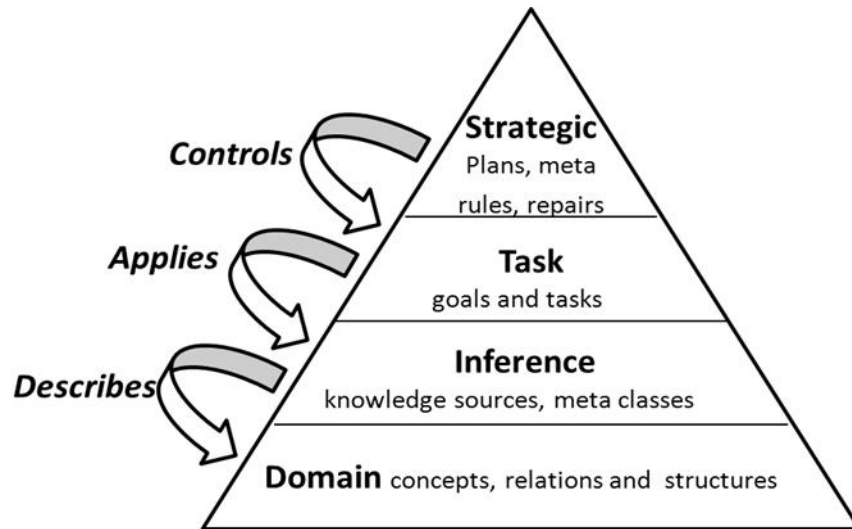


Figure 12: KADS models [83]

KADS presents elementary models, called *interpretation models* that describe the problem solving operations collected from the expert (examples: *identify, predict, repair, remedy and transformation*). The models are represented in a hierarchal way called the inference structures, to describe them in terms of knowledge sources. In KADS, goals, tasks and high level control structures are represented in the task structures.

Modality in KADS is the relation between the system and the user regarding the tasks between them. The interpretation models are a set of reusable items. They help in the case of knowledge absence. Those models help the developer in transforming the knowledge taken from the expert. The interpretation models are a set of predefined entities that could be applied and used in many domains. KADS has an inference library of useful predefined inferences that has

become enriched through the years. This library helped many developers follow KADS in a time saving manner.

4.1.5 CommonKADS

CommonKADS is a newer version of KADS that is far more detailed. It is based on KADS and it represents the organization, the users, their communication and many other necessary aspects.

CommonKADS concentrates on the conceptual structure of the knowledge and the system.

The six CommonKADS models are categorized in three groups [9]:

1. Context Models:

- a. Organization model: Supports the description and the analysis of the organization.
- b. Task model: Describes the tasks that might be performed by the system within the organization.
- c. Agent model: Supports the capabilities, constraints and roles of the agents performing the tasks.

2. Concept Models:

- a. Knowledge/Expertise model: Supports the description of the knowledge invoked in the tasks.
- b. Communication model: Describes the relation between the agents, their interaction and their communication.

3. Artifact Models:

- a. Design Model: Supports the design and the structure of the system.

The CommonKADS set of models is illustrated in Figure 13.

According to Schreiber et al. [9], CommonKADS models can be defined/ worked on in the order presented in Figure 13. However, they could be defined sequentially or even in parallel.

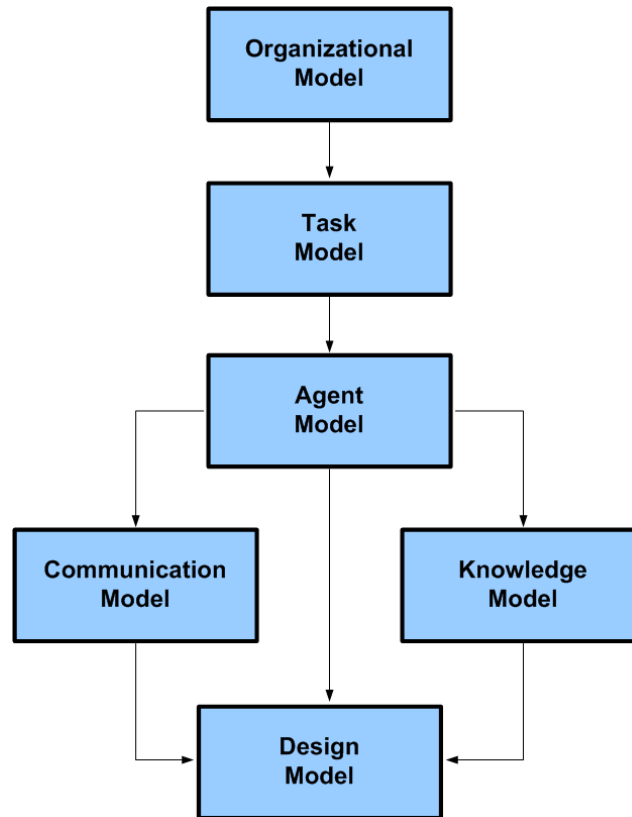


Figure 13: CommonKADS set of models

For instance, the communication model and the knowledge model could be defined in parallel because they do not rely on each other. On the contrary, the design model cannot be defined before all the other models are finished. In any case, the general products of CommonKADS are:

1. CommonKADS models documentation that represents the important aspects of the environment and the delivered knowledge-based system.
2. Additional documentation: information not represented in the filled model templates (e.g. project management information and UML diagrams)

3. The Software: knowledge system software [9].

When building a system using CommonKADS, not all the models need to be constructed; it depends on the project and its goals. In every model, a number of steps need to be executed. All the CommonKADS models are introduced in the next section.

CommonKADS is chosen as the KBS lifecycle model into which the validation method of this dissertation is integrated. CommonKADS is used as the lifecycle model for the validation method of this dissertation. Next, light is shed on why CommonKADS was chosen for the validation method presented in this dissertation.

4.2 Selecting CommonKADS for Validation

In this section CommonKADS is compared against all the other lifecycle models. DESIRE is discussed first. DESIRE was originally introduced as a model for specifying complex knowledge-based systems. It views the system as a series of interacting pieces (objects). This contrasts with general purpose specification languages such as Z [77]. CommonKADS has the ability to interact with general purpose languages such as KARL (Knowledge Acquisition and Representation language) [81] and CML (CommonKADS Modeling Language) [82]. DESIRE pays little attention to knowledge representation and domain requirements, while CommonKADS has a model for each. It is considered essential that domain requirements be represented in the system. As stated previously, validation compares the system to the real world, and domain requirements are the closest thing to the real world because they represent the organization in which the system is being developed. Therefore, validation of knowledge based systems cannot be easily built into DESIRE.

Validation is already included in KBSDLC. However, KBSDLC has not shown any significant success in validating knowledge-based systems. It has not been solely used for validation. Other validation approaches need to be used with it. Hence, building another validation stage into KBSDLC would both be confusing and inconsistent with the KBSDLC multi-step model. KBSDLC is effective for small knowledge-based systems and prototypes only, but not for large systems [78]. CommonKADS is useful for both large and small systems [15]. As Weitzel et al. [63] stated, validation in KBSDLC requires using a large number of test cases which is not efficient. Large number of test cases is a resource and time consuming problem. KBSDLC validation isn't effective or sufficient and it is not performed incrementally. Generic Tasks doesn't deal with knowledge elicitation or representation. Furthermore, it doesn't offer any support for organizational issues. It doesn't represent the tasks of the system while CommonKADS does all of these.

Because of its building block structure, Generic Tasks implies the use of subsystem validation, which limits the knowledge engineer's choice of a validation method. Integration testing is a must when using Generic Tasks. Knowledge bases require a clear presentation and a high level of modularity. Generic Tasks doesn't deal with knowledge representation and knowledge representation is important when performing validation. Because of its structure, Generic Tasks cannot perform incremental validation. Generic Tasks is not flexible for a validation method to be built within it.

Furthermore, to demonstrate CommonKADS soundness, many specialized development models have been introduced based on it. MIKE, MAS-CommonKADS and PROFORMA models fall under the family of CommonKADS. They are based on the CommonKADS concepts.

MIKE (Model-based and Incremental Knowledge Engineering) is a general model that can be used for any domain [84]. MIKE was introduced during the mid 1990s. As stated in [84], “MIKE proposes the integration of semiformal and formal specification techniques, prototyping, and life-cycle models into an engineering framework”. It defines a smooth transition between the semi-formal implementation of the knowledge towards a formal implementation and later towards development. MIKE uses KARL as its formal descriptive language and a tool was built for MIKE that uses KARL. MIKE doesn't have validation or verification process embedded in it.

PROFORMA was developed as a model for building clinical expert systems [85]. It is dedicated only for a special kind of medical systems, so building a validation method to it is not as helpful as building a generic validation method into CommonKADS. Vollebregt et al. [85] state that PROFORMA is new and untested while CommonKADS is widely accepted and more general. Incorporating a generic validation method into PROFORMA is therefore not practical because of its specialization. MAS-CommonKADS (Multi-Agent Systems-CommonKADS) also uses the CommonKADS models as part of its development cycle [86]. It is dedicated for AI systems that include multiple agents.

CommonKADS had been used in many domains, such as industry, medical systems, management systems, academia and many other disciplines. It was criticized for the heavy overhead it places on small projects. Therefore, a model called PragmaticKADS was introduced. It can be used for small knowledge-based systems. PragmaticKADS extracts the most important steps from CommonKADS and disregards all the steps that are oriented towards large systems.

Tools have been developed to help in implementing CommonKADS. Examples: Model-K and OMOS [87]. The development of these and other tools reflects the general acceptance of CommonKADS. Conceptual model languages had been introduced to support CommonKADS

representation formally such as ML², VITAL and FORKADS. VITAL is a workbench to support development of knowledge-based systems using KADS and CommonKADS. VITAL is a toolset that aims to commercialize the use of CommonKADS.

CommonKADS is a flexible model that successfully fits into many approaches. It has been used for large and small systems [83]. CommonKADS is a knowledge representation dependent model. CommonKADS was not created independently from other software models. Rather, other software models influenced the definition of CommonKADS.

According to Schreiber et al. [9] CommonKADS was influenced by structured design, object-oriented models and quality management. CommonKADS has powerful organizational sub-models that can represent many domains. CommonKADS offers a *de facto* standard for building systems and ensures a modular approach.

CommonKADS is the most comprehensive (as well as the most commonly used) model for knowledge-based systems development. Nevertheless, CommonKADS doesn't employ a complete validation representation and it doesn't guide the developer on how to perform validation. CommonKADS, however, does have all the means and the flexibility to have incremental validation method based on case testing incorporated within it. None of the other development approaches has the advantages of CommonKADS. CommonKADS is the most used knowledge-based systems lifecycle model and is the most accepted [9] [29] [82] [83] [86] [87]. CommonKADS supports reusability and offers guidelines for the developer to achieve high quality systems [9]. Considering all the mentioned advantages of CommonKADS, it was selected as the knowledge-based system development model for the validation method presented in this dissertation. The rest of this chapter is dedicated to describing CommonKADS.

4.3 The CommonKADS Models

CommonKADS (Common Knowledge Acquisition and Design Support) is a structured development lifecycle and design approach for knowledge-based systems. CommonKADS was initially developed by the project ESPRIT (European Union Information Technologies Program) as KADS. This program ran from 1983 until 1998. CommonKADS originated at the University of Amsterdam. Although this model was developed in Europe, it is gaining popularity throughout the world. CommonKADS was also combined with other methods used in the current ‘state of the art’ knowledge systems research topics, such as object-oriented methods (OO) and ontological representations of knowledge. CommonKADS supports knowledge modeling and the design of the knowledge-based system (KBS) by constructing models that form the system. The models presented in CommonKADS provide the KBS engineer with a set of templates with which to work and follow as the blue print of the project. CommonKADS provides a way of representing knowledge, means for knowledge analysis and knowledge storage. It describes the knowledge based on the organization environment in its organization model. However, none of the models of CommonKADS discusses or supports validation. It is important to note here that when we apply CommonKADS to a knowledge-based system development, new inconsistency errors might be discovered because of the interaction among its models. The CommonKADS models are introduced next. The six CommonKADS models are introduced in detail in this section. First, the context models are presented.

4.3.1 The Context Models

Knowledge-based systems not only automate the processes of an organization, but they also improve them. CommonKADS aims to do that at the organizational level. The context modeling

in CommonKADS consists of two main studies, a feasibility study and the impact and improvement study.

For the feasibility study there are two main steps: analysis of economical and organizational perspective, and synthesis (selecting the focus areas and targeted solutions). For the impact and improvement study there are two main steps: analysis of the relations between the tasks, agents and the organization and synthesis (ensuring organizational integration of the system and ensuring user acceptance). All the context modeling in CommonKADS is performed through worksheets. Nine worksheets are needed, five for the organizational model, two for the task model, one for the agent model and one for summary.

4.3.1.1 The Organizational model

The organizational model is the first model in CommonKADS and in most cases, using CommonKADS starts by building this model. The issues that need to be represented in the organizational model are:

1. The structure of the organization
2. Processes
3. The people
4. Power and culture
5. Organization's resources

The first four worksheets needed for the organizational model are presented in Figure 14. Worksheets for the organizational model are indicated to as OM-1, OM-2, OM-3, OM-4 and OM-5 [9]. The worksheets of the organizational model are tables with a list of items that needs to be addressed. As figure 14 shows, OM-1 identifies knowledge-oriented problems and opportunities. OM-1 is shown in table 2 [9]. The second worksheet in the organizational model

describes the problems introduced in OM-1. In most cases, more than one OM-2 sheet would be introduced for one system because every sheet describes a certain problem or aspect of the problem. OM-2 is more specific than OM-1. OM-2 components are shown in table 3. [9]

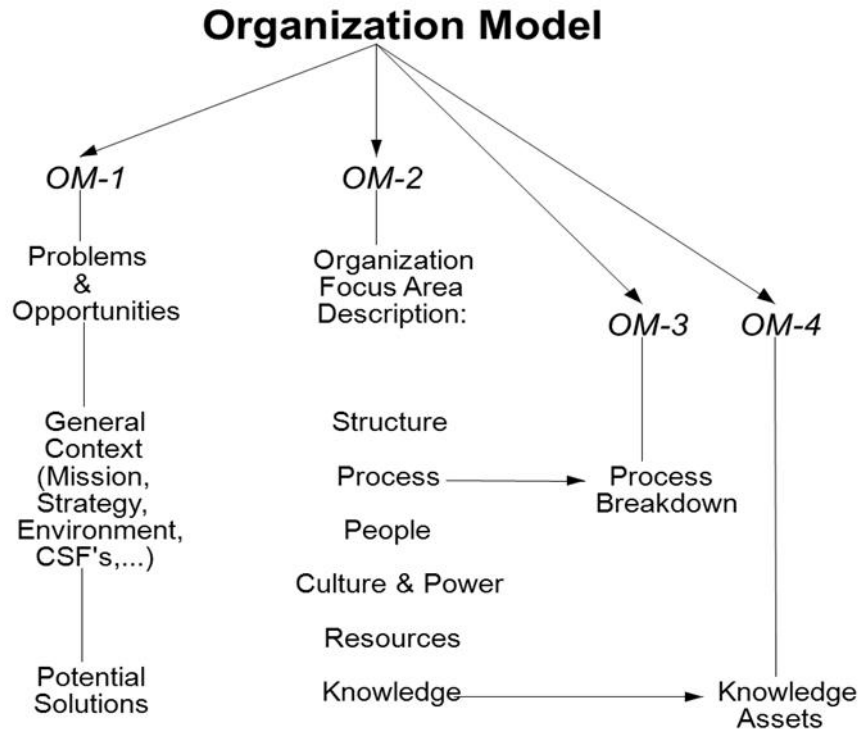


Figure 14: The organizational model worksheets [9]

Table 2: Worksheet OM-1

Problems and opportunities	This list is generated based on interviews, brainstorms and discussions.
Organizational context	Indicate important organization aspects into the following categories: <ol style="list-style-type: none"> 1. Mission, vision and goals 2. Important external factors 3. Strategy of the organization 4. Organizations value chain and major drivers
Solutions	Possible solutions for problems, as suggested by the interviews and the discussions.

Table 3: Worksheet OM-2

Structure	Structure chart of the organization’s part that is included in the problem represented by the sheet.
Process	Unified modeling language (UML) activity diagram of the processes related to the problem. The process is composed of tasks which are discussed in OM-3.
People	List all the people included.
Resources	Resources used for this problem, including: <ol style="list-style-type: none"> 1. Information systems 2. Equipment and materials 3. Technologies, patents and rights
Knowledge	Due to the importance of this part, it’s left out to its own worksheet OM-4.
Culture and power	List any informal issues and unwritten rules. This part is an essential one, because people in many cases don’t pay attention to it.

Table 4: Worksheet OM-3

No.	Task	Performed by	Where?	Knowledge Asset	Intensive?	Significance
Task identifier	Task name	The agent performing the process, either a human or a computer system.	Location in the organization	Knowledge resources used by this task	A Boolean value indicating whether this value is knowledge intensive.	Indication of the significance of the task, with a scale from 1-5.

Table 5: Worksheet OM-4

Knowledge Asset	Possessed by	Used in?	Right form?	Right place?	Right time?	Right quality?
Name of asset	Agent	Task (from OM-3)	Yes or No	Yes or No	Yes or No	Yes or No

OM-3 describes the breakdown of the business process. Each business process is broken down into more detail. OM-3 is the foundation for the task model. It consists of tasks and their information such as the location in the organization and the significance. OM-3 is shown in table 4. In OM-2 the knowledge section is discussed briefly. One column is not sufficient to discuss the knowledge needed.

Table 6: Worksheet OM-5

Business feasibility	The following points needs to be addressed to achieve this feasibility: Expected benefits for the organization. How long is the expected added value? What are the costs? Is the solution better than the alternatives? What are the required changes to the organization? What are the risks and the uncertainties?
Technical feasibility	The following questions needs to be answered to achieve this feasibility: How complex is the solution? What is the state of the art for the problem? What are the critical aspects? What are the measures for success? How to test for validity and quality? (Important point in the scope of this document!) How is the required interaction with the users and other systems?
Project feasibility	The following questions needs to be answered to achieve this feasibility: Is there any needed commitment from the users/stakeholders? What are the needed resources, knowledge, competences or changes to the organization?
Proposed actions	This part transforms the previous sections into actions, but needs to define the following points: Focus: what is the recommended focus to identify the solution? Target solution Results, costs and benefits Project actions: required actions. Risks: risks of this solution to the organization.

Therefore, OM-4 is dedicated to describing the knowledge assets in the organization. It is meant to be a first-cut analysis for the knowledge assets; more detail is included in the knowledge model (a different CommonKADS model). All the organization models and their relations are illustrated in Figure 14. After performing this analysis, all the information needed for the organizational model is ready. The last worksheet in the organizational model is OM-5 (shown in Table 6) [9]. The feasibility decision-making is in OM-5. It presents the feasibility from three perspectives, business, technical and project feasibility. The main topics which that OM-5 focuses are to help management in taking decisions. After the decision-making process and after

the new actions are proposed, zooming in on the tasks is the natural next step. The second model is the task model.

4.3.1.2 The Task model

Five worksheets are necessary to represent the organization in CommonKADS. More importantly however, is to present the tasks within the organization.

Table 7: Worksheet TM-1

Task	Task name and identifier
Organization	Indicates to the organization where the process is performed at
Goal and Value	The goal of the task and the value it adds to the system
Dependency and Flow	Represents what tasks are input to this task and any tasks that are outputted from this task. An activity diagram could be used here to represent the <i>input tasks</i> and the <i>output tasks</i> .
Objects Handled	Input objects (knowledge and information): objects inputted to this task. Output objects: objects outputted from this task. Internal objects: important objects used for this task that are not inputted or outputted. A class diagram is used here to illustrate the objects.
Timing and Control	Frequency and duration of the task. Describe the preconditions and the post conditions of this task. An activity or class diagram is illustrated here.
Agents	The staff members (or systems in some cases) who are responsible of performing this task.
Knowledge and Competence	TM-2 is responsible for the knowledge needed for a task. Here the needed skills and competences are listed.
Resources	Describe resources used in this task, such as time, equipment and money. This is a refinement of OM-2.
Quality and Performance	Defines the quality measures that the organization uses to conclude that the task was successful.

Tasks consume resources; they require knowledge and they are carried out by agents working at the organization. To represent the tasks two worksheets are needed per task, TM1 and TM2. Table 7 illustrates TM1 and Table 8 illustrates TM2.

For each task, a number of values need to be defined, such as: the goal of the task, the timing of the task, the knowledge required and the agent that performs the task. TM1 could be looked at as a refinement of the data in OM3 within the target process. TM2 represents the specification of the knowledge engaged for a task and acts as a refinement for OM-4.

Table 8: Worksheet TM-2

Nature of Knowledge	True/False	Bottlenecks
Formal, rigorous		
Empirical, quantitative		
Heuristic, rule of thumb		
Highly specialized, domain specific		
Experience based		
Action based		
Incomplete		
Uncertain, maybe incorrect		
Quickly changing		
Hard to verify		
Tacit, hard to transfer		
Form of Knowledge		
Mind		
Paper		
Electronic		
Action skills		
Other		
Availability of knowledge		
Limited in time?		
Limited in space?		
Limited in access?		
Limited in quality?		
Limited in form?		

TM2 is a ‘true or false’ worksheet. The knowledge engineer illustrates his/her understanding of the tasks in this worksheet. Values about the knowledge are defined, such as: uncertainty, forms of knowledge, access to knowledge, knowledge’s quality and availability. All the empty fields in TM-2 are filled by the knowledge engineer [9].

4.3.1.3 The Agent model

The Agent Model is presented in one worksheet AM-1. The main goal of the agent model is to understand the roles of all the agents involved with the system. The agent model provides an input to the communication model that presents the means of communication between the agents. Additionally, the agent model is represented in use-case diagrams [9]. Every agent needs a worksheet, AM-1. In this worksheet, the agent's tasks, knowledge, responsibilities and constraints are defined.

Table 9: Worksheet AM-1

Name	Name of the agent
Organization	Agents position in the organization
Involved In	List of tasks (reflects sheet TM-1)
Communicates with	List of agent names
Knowledge	Knowledge Items possessed by the agent (reflects TM-2)
Other competences	List of other competences of the agent
Responsibilities and constraints	Responsibilities within a certain task and constraints. Constraints might be authority, legal or professional issues.

The knowledge engineer should have a clear understanding of the organization, the tasks performed and who is performing each task after filling all the mentioned worksheets.

Additionally, after defining the Context models, a summary worksheet is used for managerial decision-making but is not part of the system development. This worksheet is called OTA-1 (Organization, Task and Agent Models-1) and is a checklist of items and actions that the managers look at, such as proposed actions, impacts on the organization and commitments. OTA-1 summarizes all the context worksheets. The context analysis is considered ready after this stage. Steps towards extracting test cases from the context models worksheets for the validation method are discussed in Chapter 5.

4.3.2 The Concept Models

Analyzing the organization, task and agent models is a knowledge engineering task. However, those three models are related to the managerial aspects of the organization and the system under development. The concept models deal with knowledge in a more detailed manner. Its representation, communication, features and usage are described in the two models discussed next.

4.3.2.1 The Knowledge model

Knowledge representation is a complicated task. This is partly because in many domains, the knowledge is not explicit, but it's tacit. Knowledge is more than just what you formally know. Knowledge from experts is what they do, what their experience dictates and what their actions are. That's why knowledge cannot be simply represented in worksheets as in the first three models discussed. Knowledge in CommonKADS has a life cycle that consists of three main steps: conceptualize, reflect and act.

First, knowledge is identified by the knowledge engineer, it is analyzed and its weaknesses and strengths are identified. Second, changes and improvements on the knowledge are planned. Finally, the changes are implemented on the knowledge, and the planning is reflected to improve knowledge. These steps are performed iteratively until the knowledge engineer and the expert are satisfied by the knowledge. Knowledge management agents, processes and assets are defined in CommonKADS based on the context models. OM-2 and AM-1 are used for the agents; OM-2, OM-3 and TM-1 are used for the processes and OM-4 and TM-2 for the knowledge assets. Knowledge in CommonKADS is presented by three categories: task, inference and domain knowledge.

Domain knowledge is represented using two kinds of diagrams: the UML class diagram that represents knowledge pieces. Classes in CommonKADS are called concepts because they don't include operations. The second diagram is a high level representation of all the classes and how they are linked together, their relations and types. Concepts in class diagrams can be linked by inheritance, association, composition or aggregation [9].

Task inference knowledge is the reasoning knowledge in CommonKADS. It uses the static domain knowledge used to reach conclusions in the system. Inference knowledge consists of two main types, inferences and transfer functions. Transfer functions interact with the external world, such as external agents, users and other systems. Inferences are mainly internal well-defined objects that can be re-used. CommonKADS has a set of predefined inferences that could be useful in many systems. Any inference developed or defined based on the CommonKADS standards, can be used in any other system. This is an example of CommonKADS knowledge reusability features. Inferences in CommonKADS are presented in a structure similar to Data Flow Diagrams (DFD). A DFD is an illustration of information flow within a process. They include functions, flow between functions and input/output data. The Task knowledge is the set of tasks that needs to be performed to achieve a certain goal in the system. Every task is broken into inferences, transfer methods and task methods. Transfer methods and inferences are extracted from the inference knowledge. Knowledge model construction steps are defined in CommonKADS and their documentation is in the form of worksheets. The knowledge model worksheet KM-1 is presented in Table 10. General knowledge templates (including domain, task and inference knowledge representations) are predefined in CommonKADS. Defined templates represent activities that can be performed in a wide range of knowledge-based systems, such as planning, diagnosis, monitoring and assessment [9].

Table 10: Worksheet KM-1

Document Entry	Description
Knowledge model	Full knowledge model specification in text and figures.
Information source used	List of resources used.
Glossary	List of application-domain terms with their definitions.
Components considered	Components used in identification stage.
Scenarios	A list of scenarios used for solving application problems.
Validation results	Description of the results of the validation study. Note: CommonKADS doesn't have a defined process for validation and verification of the system but paper-based manual validation and looking at the worksheets could be performed.
Elicitation material	Material gathered during elicitation activities.

Task methods are functions defined in the task model; they are broken into sub methods and presented in activity diagrams.

4.3.2.2 The Communication model

The communication model represents the relation between the agents, their shared tasks and the interaction with knowledge. Communication model consists of three main parts: plan, transaction and information exchange specification. These parts are illustrated in UML diagrams. The communication model main parts and its interaction with the other CommonKADS models are illustrated in Figure 15.

The communication plan governs all the relations between the agents. It is constructed based on dialogue diagrams and is illustrated using state diagrams. Dialogue diagrams present the detailed transactions and tasks for two communicating agents. The information exchange specification is performed using pseudo code. Constructs such as: SEND, RECEIVE, PROCESS, REPEAT, & and IF-Then are the main keywords used in the communication model. Communication between agents in CommonKADS is presented in two worksheets, CM-1 and CM-2. CM-1 is Table 11 and CM-2 is Table 12 [9].

Table 11: Worksheet CM-1

Transaction identifier name	Indicates to the name of the transaction
Information object	Indicates the core information object
Agents involved	Agents sending and receiving information
Communication plan	Indicate to the communication plan
Constraints	Any preconditions for the transaction to be carried out
Information exchange specification	Any further notes about the transaction. This is detailed in CM-2

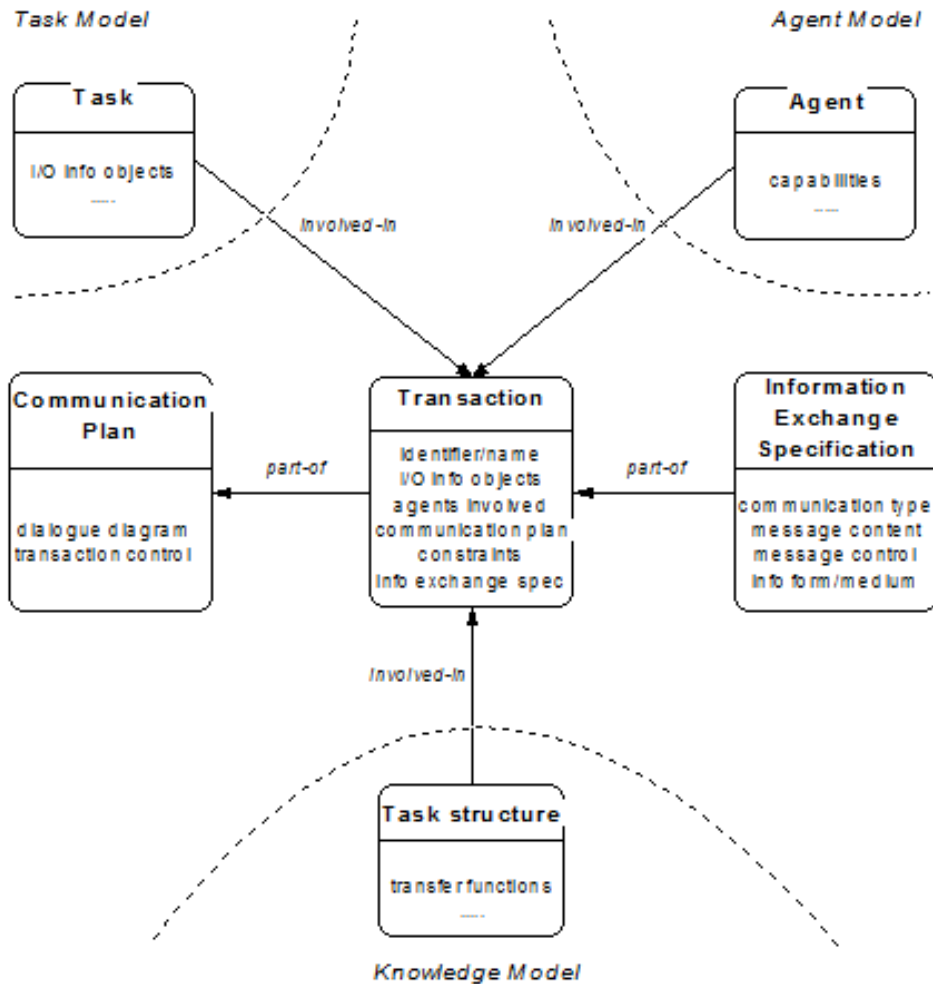


Figure 15: The communication model role [9]

Table 12: Worksheet CM-2

Transaction	Transaction identifier
Agents involved	Sender and receiver
Information items	All information items that are transmitted in this transaction
Message specification	All the information related to the messages exchanged, content and references
Control over messages	Control specification over the message, if necessary. This is done using the pseudo code notation.

The communication model acts as a link between all the previously defined models. It is the last model before defining the design of the system in the design model. CM defines the relation between the agents within the organization, defines how the tasks are communicated between the agents, and what knowledge is needed for each task.

4.3.3 The Artifact Models

This category of models consists of only one model, the design model. The design model is responsible for turning the analysis done previously into a software system. The functional requirements are presented in the communication and knowledge models, while the non-functional and domain requirements are presented in the organization, task and agent models.

4.3.3.1 The Design model

The design model constructs a structure for the software system. The first five models are concerned with the experts, protocols, reasoning strategies, problems and opportunities. The design model is concerned with different aspects, such as the algorithm design, hardware platform, implementation languages and software architecture. The design lifecycle consists of four main steps each of which is presented in a worksheet.

1. Design the system's architecture: specify the general design of the system.

2. Identify the target implementation platform: choosing the software and hardware platforms to be used for building the system. In most cases this is defined by the customer, but if it wasn't then this step is essential and will affect steps three and four immensely.
3. Specify the architectural components: in this step, a detailed design is presented, subsystems are defined and the interfaces are specified.
4. Specify the application within the architecture: in this step, all the ingredients from the analysis models are taken and transferred into the architecture. The four steps of design are illustrated in Figure 16.

Every step is presented in a worksheet. DM-1, DM-2, DM-3 and DM-4 are presented in Tables 13, 14, 15 and 16 respectively. DM-1 is the main design model.

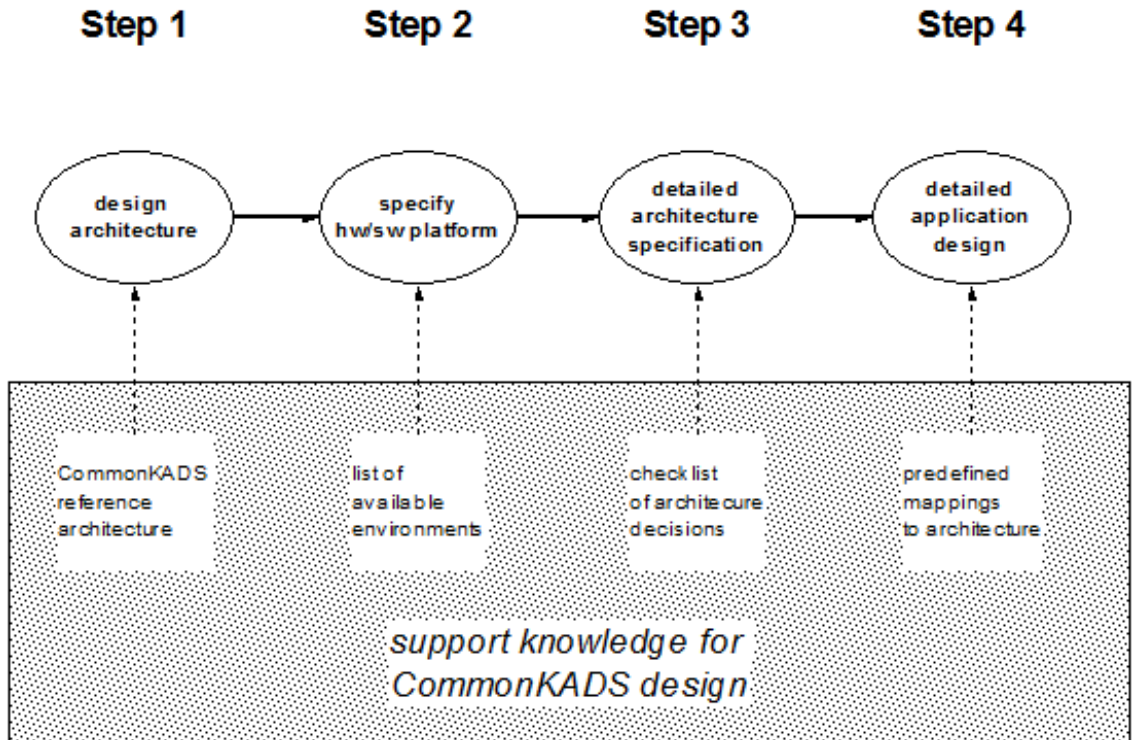


Figure 16: The design model [9]

In DM-1, the KBS is designed with the sub-systems. A diagram is drawn to illustrate all the subsystems. UML Diagrams in CommonKADS are discussed in a following section.

Table 13: Worksheet DM-1

Architecture decision	Format
Sub-system structure	Draw a diagram with all the subsystems
Control model	Characterization of the overall system controls
Sub-system decomposition	Refer to diagrams in which every subsystem is decomposed.

Table 14: Worksheet DM-2

Software package	Name of the software package
Potential hardware	Hardware platforms to run on
Visualization library	Libraries available for implementation
Language typing	Strong or weak typing, full object oriented?
Knowledge representation	Declarative or procedural
Interaction protocol	Protocols to interact with the outside world
Control flow	Message passing protocols
CommonKADS support	Does the software provide support tools for CommonKADS

Table 15: Worksheet DM-3

Architecture component	Typical decision points
Controller	Mechanism for internal/ external events handling
Task	Can a task fail?
Task method	Language for control structures, declarative or procedural
Inference	Define internal state variable
Inference method	Many-to-many mapping from inference to inference method
Dynamic role	Data types for roles
Static role	Define access operations
Knowledge base	Define the rule-instance representation
Views	Define interfaces. Standard graphical direct-manipulation interface? Special facilities required?

Table 16: Worksheet DM-4

Element	Design decision
Controller	Translate communication plan control into event handlers
Tasks methods	Formalize control structure
Dynamic roles	Choose data type for each role
Inferences	Specification of the invocation of the inference methods
Inference methods	Select inference methods
Knowledge bases	Translate knowledge base instances to representations formats
View objects	Select appropriate views for the application-model

DM-2 is a software/hardware decision-making worksheet. The decisions regarding which software platform to employ, what hardware to build upon and what programming language to use are defined in DM-2. The last two sheets in the knowledge model are used as a transition between the CommonKADS models to the targeted implementation. These models define the core of the system, including the methods, the knowledge base and inferences. Tables 13-16 introduce the design worksheets with all their elements. An example of a complete system with all its CommonKADS models is presented in Chapter 6.

4.3.4 UML Diagrams in CommonKADS

This section introduces the UML models in CommonKADS. The UML diagrams used in CommonKADS are:

Activity Diagrams are used to illustrate the information flow and model control. Activity diagrams are used in the organization model and could be used to represent control flow within a task. The diagram consists of states that represent the different stages of an activity. Activity diagrams are used in worksheets OM3 and TM1.

State Diagrams are used to illustrate dynamic behavior. Every state consists of a name, variables and actions. The states are linked by events and messages. In CommonKADS, state diagrams are used in the communication model and for business processes representation. State diagrams are used in worksheet CM1.

Class Diagrams capture static information structure, where a class is a well-defined entity in the system. Classes can inherit each other too. In CommonKADS, class diagrams can be used for task modeling. Each class has a number of attributes and procedures. Class diagrams are used in worksheets KM, DM1 and TM1.

Use-Case Diagrams illustrate the services that need to be in the system. They also illustrate the agents and their uses of the system. Every use-case diagram includes a set of actors and their use cases. In CommonKADS, use-case diagrams are used in the design model and to show solutions to the customer. Use-case diagrams are used in worksheet AM1. Chapter 6 will show a full example for a system defined using CommonKADS with all the UML diagrams.

4.3.5 System's Specification and Implementation

After performing all the steps of CommonKADS, the system's implementation should be performed with no major obstacles, because everything is well defined, documented and illustrated, and all the critical decisions have been made [9]. The last task for the knowledge engineer is to 'translate' the models into a programming language (This process is discussed in Chapter 6, where a complete KBS is built through CommonKADS).

All the decisions regarding the types of the system's components, the structure of the knowledge base and the design of the system are already defined in the six models. Moreover, CommonKADS controls project management decisions. This is done through two separate worksheets. PM-1 and PM-2 (PM stands for project management) are referred to as the *seventh model* but they are not related to system development. Project management activities through CommonKADS are defined by four actions: review, risk, plan and monitor. These actions are always performed after every major step of CommonKADS. The four predefined actions represent the activities carried out by managers of the project. PM-1 includes the risk identification and assessment, PM-2 includes the objectives of every model in CommonKADS for the system.

4.4 Summary

This chapter introduced the lifecycle development models for knowledge-based systems. It discussed a number of models and compared them to CommonKADS. CommonKADS is the most appropriate lifecycle model for validation. Therefore, it was introduced in this chapter in great detail. The context models (the organizational, task and agent models) analyze the organization and define the success factors for the system. They describe the tasks that need to be done and the main agents that interact with the system. The concept models (knowledge and communication models) are used to describe the problem solving functions and knowledge within the system [9]. Furthermore, the artifact models (design model) convert the other representations to the technical specification that are the basis for the system. All the CommonKADS models are discussed in detail. The relation between the models is illustrated to construct better understanding of the models. UML diagrams are an important part in CommonKADS; four types of diagrams are presented in this chapter. Additionally, the entire knowledge-based systems implementation process using CommonKADS is discussed. The mentioned models, diagrams and worksheets are used in the validation method of this dissertation. In the next chapter, MAVERICK, the validation method built within CommonKADS and the main contribution of this dissertation is introduced in a detailed manner.

CHAPTER 5: THE MAVERICK VALIDATION METHOD

In software engineering, the development lifecycle has always been an important material for discussion and improvement. Additionally, it has been established that incremental testing, validation and verification should be performed instead of the one-phase approach. The main contribution of this research is discussed in this chapter. The contributed validation method is called: **Method for Automated Validation EMBEDDED into the Reusable and Incremental CommonKADS (MAVERICK). MAVERICK uses the test cases validation approach. Extracting test cases from the CommonKADS models, embedding validation into its models and selecting an appropriate set of test cases are discussed in this chapter. MAVERICK consists of three main phases, which are performed in cycles:**

- 1. Test Case Extraction**
- 2. Inspection Validation**
- 3. Context-Based Test Case Reduction (CBTCR)**

The first two parts are based on CommonKADS; test cases are composed by the knowledge engineer based on the CommonKADS models and on the guidelines provided by MAVERICK. Inspection validation is embedded into CommonKADS; it is a manual process that is performed by the expert and the knowledge engineer. Inspection validation consists of two steps, *analysis validation* and *design validation*. CBTCR is an automated process that is performed iteratively using the CBTCR tool.

The MAVERICK validation method provides a multi-tier solution for validation of knowledge-based systems. Most importantly, MAVERICK reduces time, cost and effort needed

to validate a system by using the context-based test case reduction (CBTCR) process, which is introduced in detail later in this chapter.

Because CommonKADS is a model-based lifecycle, results of this MAVERICK validation method are measured based on models. This is performed by a model weight variable. Each model will have a validity level. On the other hand, test case reduction has typically been a process performed just once, usually, before validation starts. In MAVERICK, a dynamic test case reduction process is followed that provides coverage for the knowledge-based systems while reducing the number of test cases. Test case reduction is based on previous validation results. The importance of each test case either increases or decreases based on the outcome of executing them. This is presented in two variables, the *local* and *global importance*. Dealing with such a problem is complex; the solution presented is broad and deep because of the many sided solution that deals with test cases, an incremental approach and a lifecycle method. The steps of MAVERICK are introduced in detail in this chapter.

5.1 Validation through CommonKADS Case Testing

Validation through CommonKADS is performed by extracting test cases from the six models and their components such as the UML diagrams and worksheets.

Test cases extraction is different from one model to another. In the communication model, for example, pseudo-code is used to extract test cases. Furthermore, test cases are extracted from the knowledge model using its inference structures, transfer models, worksheets and data flow diagrams. It is worth mentioning here that not everything in the CommonKADS models is used for validation and that different objects in each model are used for extracting test cases. Some of the information in the worksheets or the UML diagrams, for example, is not useful for validation. For instance, some organizational model sheets would not be completely

filled because of the absence of a certain aspect such as the managerial pyramid in an organization, thus, no material is present for that section of the worksheet. Test case extraction is introduced in detail in section 5.3. The test case format for MAVERICK is introduced next.

5.1.1 Test Case Format

As presented previously, CommonKADS provides a general step-by-step process for knowledge engineers to build any type of knowledge-based system in any domain. Therefore, a generic validation method that could be used for any kind of knowledge-based system developed through CommonKADS should be developed. Several variables are defined for every test case in this method. MAVERICK's test case format has the following ten variables:

1. **Test case ID:** An incremental ID number that starts from zero and increments by one for every test case.
2. **CommonKADS model:** This indicates from which of the CommonKADS models the test case was extracted.
3. **Input variables:** Inputs relevant to this test case.
4. **Test setup values:** The values of the input variables of the test case.
5. **Test execution steps:** Defines what needs to be done to run this test case.
6. **Expected solution:** The expected output for the test case, defined by the expert.
7. **System's solution:** The actual output of the test case after execution of the test case.
8. **Local Importance:** Each test case is assigned a local importance. Local Importance is a function of: dependency, domain importance, criticality and occurrence. The word local comes from the idea that the importance of the test case is assigned in comparison to the

test cases of the model to which it belongs, not the global set of test cases. Detailed description of this variable is presented in its dedicated sub-section in this chapter.

9. **Number of execution times:** This parameter is an incremental number, each time the test case is executed; this parameter is incremented by one. This factor helps the knowledge engineer to realize what problems are not being solved throughout the testing process.
10. **Informal description:** Textual description of the test case, what are its goals, what does it test and any further comments.

Test cases for every model are extracted mainly from the model's worksheets and UML diagrams. An important issue to point out here is the test coverage level of our approach. Test coverage metrics are used to define the percentage of code covered by the test case sets. Test coverage metrics can indicate what code/knowledge is not tested, but they cannot accurately tell what part of the system is validated. Therefore, it presents a challenging issue in most cases. In CommonKADS, all the knowledge, the information needed to build the system and everything used in the system is defined within the six models and their components. This means that the test coverage issue in MAVERICK is addressed directly through CommonKADS. This also supports CommonKADS usability for validation. The phases of the MAVERICK validation method are presented next.

5.2 Method for Automated Validation Embedded into the Reusable and Incremental CommonKADS (MAVERICK)

This section formally introduces MAVERICK, the validation process.

Incremental validation is based on the old adage that prevention is better than cure. Incremental validation locates the problem in its early stages when they are less expensive to

correct. For example, if an error is introduced during knowledge elicitation as a result of miscommunication between the expert and the knowledge engineer, incremental validation can help in identifying the error before it's absorbed into the design and then implemented. The longer this error remains absorbed within design and development, the harder it becomes to identify it. Therefore, based on the CommonKADS structure, the test cases extraction and the life-cycle validation introduced here are performed in seven steps, in the following order:

1. **Context Test Cases Extraction:** This step defines the test cases that are extracted from the first three models (the context models: organization, task and agent).
2. **Analysis Test Cases Extraction:** In this step, the test cases are extracted from the communication and knowledge models. In CommonKADS, the analysis phase is done after building five models: organization, task, agent, communication and knowledge. These five models represent all the requirements of the system.
3. **Analysis validation:** This is the first step of inspection validation. After the first five models are defined and before moving into the design model, this validation process occurs. It checks for conflicting requirements, missing aspects in the analysis and any ambiguities. This process is performed by the experts and the knowledge engineer manually on all the documents and diagrams defined so far.
4. **Design Test Cases Extraction:** This is the last step for test case extraction, where test cases are extracted from the design model.
5. **Design Validation:** This is the second and last step of *inspection validation*. It is performed before the implementation of the knowledge-based system starts. Design validation inspects the class diagrams for DM1 to check the initial design. DM1

represents the structure of the entire system. Steps 3 and 5 are indicated to as inspection validation.

6. **Spiral System Implementation:** Implementation of the system being developed is performed iteratively. While iterating, system development proceeds and validation is performed by executing test cases. Test cases are selected in every iteration by the CBTCR algorithm described later in this chapter.
7. **Spiral System Validation:** Validation is performed iteratively too, test case selection is performed for each iteration and test cases are executed on the system. The validation approach is discussed and introduced in greater detail next. Steps 6 and 7 are indicated to as CBTCR.

Figure 17 illustrates the general approach towards performing incremental validation within the CommonKADS steps. Different validation steps are performed during the building of the CommonKADS models and the system. Nevertheless, the main goal of validation is to attain a valid system. By looking at the previous chapter, many test cases could be extracted from the various CommonKADS models because they have all the content that describes the system from many points of view, the agent, the task, the knowledge and the organization. Because CommonKADS is an effective model to be used for developing large systems, it is important to consider that large systems require more effort for validation, and in the scope of MAVERICK, more test cases. Executing many test cases is not desirable and many times it can be difficult or impossible to run all the tests. A method to reduce the number of test cases generated from the CommonKADS models is needed. In MAVERICK, test case reduction is addressed with CBTCR.

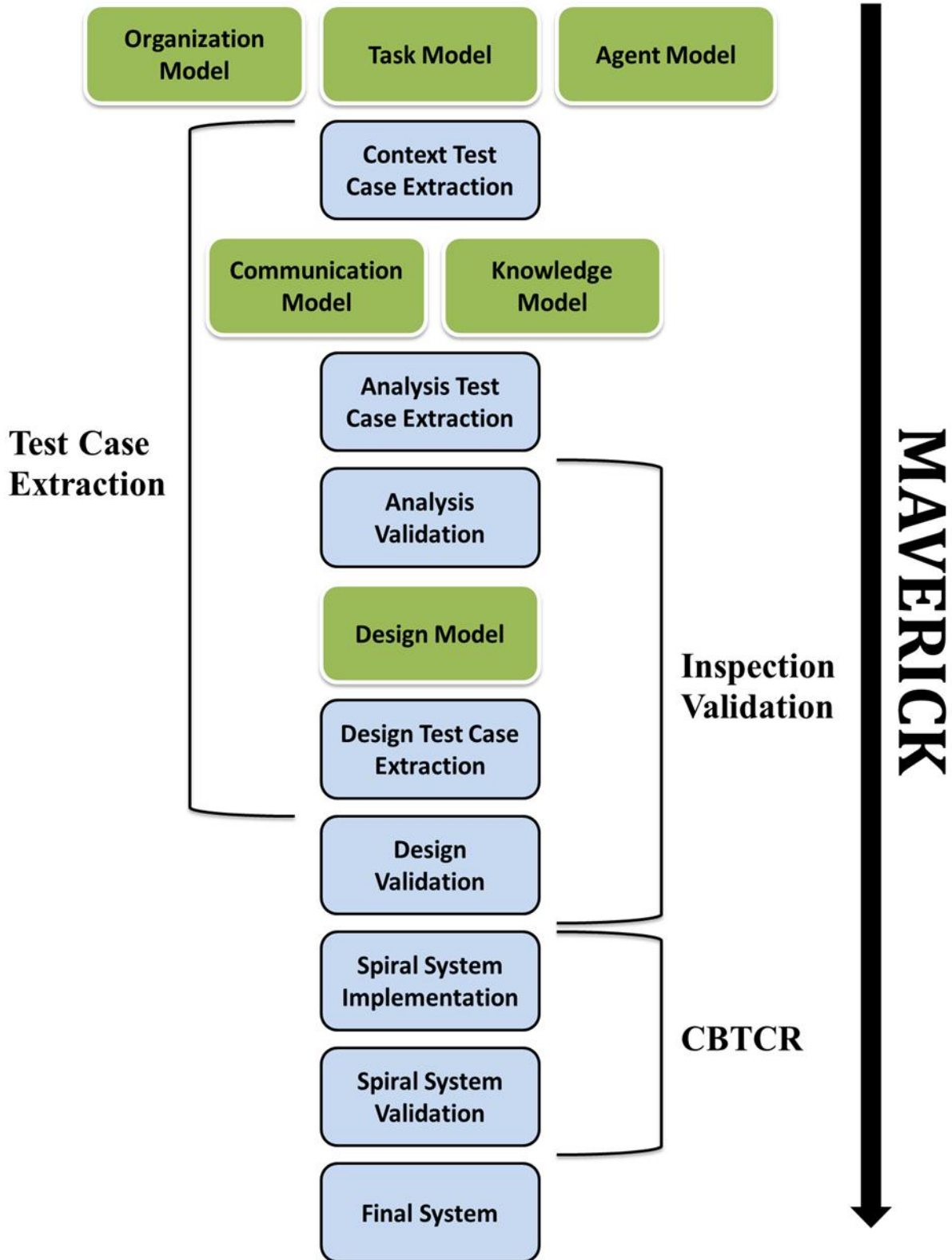


Figure 17: The validation method: MAVERICK

As it is illustrated in Figure 17, MAVERICK includes three main phases, the green boxes are the CommonKADS models and the blue boxes are the MAVERICK phases. The next section introduces test case extraction, section 5.4 introduces inspection validation and section 5.5 introduces CBTCR.

5.3 Test case extraction in MAVERICK

In this section, the first phase of MAVERICK is presented. This is the process of test case extraction from the CommonKADS models. Starting with the organizational model, an example is introduced of a test case that follows the format presented earlier in this chapter. As an example, this section uses a simple knowledge-based system developed using CommonKADS for a bank or a financial institution to demonstrate the test cases extraction.

An example system requirement is that the initial deposit to open an account be at least \$200. Three test cases should be sufficient to cover this requirement: One with a deposit value less than \$200, one with exactly \$200 and one with more than \$200, the test case presented next is the first case:

1. Test case ID: *1*.
2. CommonKADS model: *organizational model (worksheet: OM3 (organization tasks))*.
3. Input variables: *deposit = \$100*.
4. Test setup values: *initialize all account numbers; this is a new account test case*.
5. Test execution steps:
 - *Open a new account using the registration screen*
 - *Fill in the personal information*
 - *Deposit \$100*.

6. Expected solution: *a message indicating that the account couldn't be opened.*
7. System's solution: *no message was displayed, the system accepted the deposit.*
8. Local Importance: *3.75* (this is assigned by the knowledge engineer and the expert)
(formula for calculating LI is introduced in later sections of this chapter).
9. Number of execution times: *1*.
10. Informal description: *this bank requires \$200 as a minimum opening deposit.*

It's worth noting that by comparing the system output and the expected outcome, this test case has failed after execution. While this indicates an invalid aspect in the system that requires refinement, this test case should be executed again after corrections are made to the system to ensure it has a positive outcome. Extracting test cases per CommonKADS model is discussed using the example financial institution system next.

5.3.1 The Extraction of Test Cases from CommonKADS Models

In this sub-section, the guidelines on how to extract test cases from each model are discussed. The test case extraction process starts early, while defining the organization model. The first worksheet from which to extract cases is OM3. It is the process breakdown sheet, thus it is the most important worksheet for test case extraction in the organization model. All the processes in OM3 break down into the task model for more details. In this sheet, a relation between the task, the agent and the knowledge is established. Each task is defined by who is performing it and what part of knowledge is needed for it. Each of these relations is transformed into a test case. Every instance where a task requires access to the knowledge is represented in one test case and every instance where the agent requires access to a certain task is represented in another test case. Such test cases reflect where knowledge should be allocated in the system and how an

agent can access them. This worksheet doesn't deal with the core of the task or the knowledge. That's the responsibility of the task and the knowledge models. The following example demonstrates how a test case extracted from OM3 (Table 17) looks like.

Table 17: Example OM3 worksheet for test case extraction

No.	Task	Performed by	Where?	Knowledge Asset	Intensive?	Significance
1	Print documents	Paul Hewson	Bank Teller Station	Documents 1 and 2 are needed for this task	False	3

Example: Task1 is performed by Paul Hewson and documents 1 and 2 are needed for this task. When the system is built, a test case would be necessary to check the availability of the needed documents when this task is performed by the mentioned employee. The test case for this specification is the following:

1. Test case ID: 2.
2. CommonKADS model: *organizational model (worksheet: OM3 (organization tasks))*.
3. Input variables: *Paul Hewson's user name and password*.
4. Test setup values: *Logout from all accounts and close all documents*.
5. Test execution steps:
 - *Run task 1 by clicking on the "start task" button*
 - *Log in as Paul Hewson*
 - *Click on "get documents 1 and 2"*
6. Expected solution: *2 PDF files opening on your computer with documents 1 and 2*.
7. System's solution: *Document 1 opened but document 2 didn't*.

8. Local Importance: 2.5. (formula for calculating LI is introduced in later sections of this chapter)
9. Number of execution times: 1.
10. Informal description: *Paul Hewson needs access to documents 1 and 2 with task 1.*

As the example shows, test cases from OM3 strictly test the availability of the knowledge and the tasks to the agents. On the other hand, OM2 has a ‘culture and power’ part in the worksheet that deals with social issues, political constraints and rules of thumbs at the organization. This part doesn’t apply to many organizations, but in cases where it’s necessary, then there should be also test cases to cover every point in this part of the worksheet. The ‘culture and power’ section is a bulleted list, each bullet should be considered for a test case. For example, a certain document needs to go through an approval of a certain person because of his/her position in the organization; this would require a test case to check whether this is reflected in the system.

OM1 and OM4 are used to introduce the knowledge engineer to organizations, its assets, departments and structure. They are purely descriptive in nature and nothing from OM1 and OM4 is used as a part of the system. Therefore, no test cases are extracted from them.

An important part where test cases are to be extracted is the worksheet TM1. In this worksheet, each task needs a number of test cases. In the task model, each task has a number of elements defined in the worksheet. These elements are used as parts in the test case based on the following guidelines: 1) the inputs of the test case are from the *dependency* and *flow* section. In this section, the *input objects* and the *output objects* are defined, which are transformed into the input variables and the test setup values of the test case. 2) In the expected output part of the test case format, the *quality* and *performance* part of the worksheet are copied into the test case. The quality and performance part in the worksheet deals with expected outcome of the task; this

would be the criteria for the test case failure or success. 3) Furthermore, in TM1, one part contains the *preconditions* and the *post conditions* of the task. For each condition, a test case should be defined as in the banking example (test case 1) introduced previously in this chapter (condition: to open account a there should be a deposit of \$200).

TM2 deals with making the knowledge engineer familiar with assigning tasks to knowledge; it won't be used for test case extraction. Knowledge test cases are extracted from the knowledge model.

Worksheet AM1 defines the agents' usage of the system. Test cases extracted from this worksheet are related to *security*, *roles* and *accesses*. As previously introduced in example test case 2 above, Paul Hewson needed access to task 1. Similar test cases are extracted from AM1. The knowledge model is a critical model in CommonKADS as it is transformed to represent the knowledge-base. In MAVERICK, the inference structure and the domain schemas provide the set of test cases to validate the knowledge. The inferences and the transfer functions are parts of the inference structure; each instance of them is transformed into a test case. KM1 is a central worksheet for test case extraction as it defines important parts of the knowledge. In the knowledge model, the knowledge engineer represents the domain requirements in the domain schemas. Every object in the domain schema is presented by a test case. In KM1, an important part in the 'scenarios' section, all scenarios related to a certain part of the knowledge are introduced. Other parts in this worksheet include a *glossary of terms*, the *elicitation material* and other sections that will not be incorporated into the knowledge-based system being developed. The following example shows what a test case extracted from KM looks like. An example of a *scenario* and a test case: The employee Dave Evans needs knowledge about credit cards overdraft fees to answer a bank's client. A test case for this scenario extracted from KM1 is:

1. Test case ID: 3.
2. CommonKADS model: *Knowledge model (worksheet: KM1)*.
3. Input variables: *Dave Evans user name and password*.
4. Test setup values: *Run the credit card sub-system*.
5. Test execution steps:
 - *Log in as Dave Evans*
 - *Enter a clients name and account number*
 - *Click on "Display credit cards fees rules"*
6. Expected solution: *Correct overdraft fees list of rules should display to employee Dave Evans*.
7. System's solution: *Correct overdraft fees list of rules displayed to employee Dave Evans*.
8. Local Importance: *1.75*.
9. Number of execution times: *1*.
10. Informal description: *Overdraft fees rules display when required by the employee*.

After the knowledge model is defined, the communication model defines the interaction between the tasks, the agents and the system. CM1 and CM2 are used for test case extraction, because both of these worksheets components are built into the targeted knowledge-based system. In CM1, each constraint in the *constraints* section is copied as a test case (refer for test case 1 for an example) and the agents involved in this test case (refer to test case 2 for an example). CM2 defines the contents of the communication messages and the control over the messages. Each transaction needs to be tested using at least one test case. In the communication model, all the *information exchange*, *message sending* and *processes* between agents are represented in a pseudo-code syntax defined specifically for CommonKADS. Pseudo-code constructs examples

are introduced in Chapter 4 of this dissertation. A test case should be defined for each pseudo construct. If the construct presents a receive operation, the test case should reproduce that operation and check if it is performed correctly. The following example demonstrates how a test case extracted from CM looks like.

Example: a message for a new loan is to be sent from the teller Adam Clayton to the management department employee Larry Mullen, indicating that a new loan is granted to a client has the following construct: *SEND transaction1 (loan granted) from teller to RECEIVE management*. The test case for this communication construct is:

1. Test case ID: 4.
2. CommonKADS model: *Communication model (worksheet: CM2)*.
3. Input variables: *Loan number, client name, client age...etc*
4. Test setup values: *Add a new loan with all the required information.*
5. Test execution steps:
 - *Submit new loan.*
 - *Click on “notify management”*
 - *Enter the loan information*
6. Expected solution: *A message should be received by Larry Mullen about the loan and its information from teller Adam Clayton.*
7. System’s solution: *A message was received by Larry Mullen about the loan and its information from teller Adam Clayton.*
8. Local Importance: 4.
9. Number of execution times: 1.

Another important part of the communication model is the dialogue diagram. It is used to test the sequence of the tasks performed by the system and the agents. Additionally, the design model in CommonKADS represents the initial design of the targeted system. DM2, DM3 and DM4 are worksheets that help the knowledge engineer select the hardware platform, software platform and all technical issues related with building the system, but the real system design is in DM1. DM1 defines all the subsystems. Test case extraction from this worksheet targets the issue of the integration of those subsystems. Relations among the subsystems are reflected by communication between the subsystems and the tasks sequencing among subsystems. In all the subsystems, the domain specifications are introduced in the organizational, task and agent models. The functional specifications are presented in the knowledge and communication models. Worksheets DM2, DM3 and DM4 are not used for test case extraction.

Different systems are obviously defined with different specifications and within different domains. The extracted test cases introduced here strictly fit the CommonKADS models and their contents. CommonKADS is a generic set of models; therefore, the guidelines presented in this section need to be general but somehow detailed at the same time. The knowledge engineer is encouraged to define more test cases if they are deemed necessary by the expert in any phase. Chapter 6 introduces a complete system built using CommonKADS with all the test cases extracted for the system.

Using the test case extraction guidelines defined in this section, all the aspects of the models are covered and test cases are generated from all the entities included in the targeted system. The next section introduces the Inspection validation phase.

5.4 Inspection Validation

This phase is mainly a “*sanity check*” for the design and the outcome of the models and their test cases. It is performed to ensure that there are no major problems and that system implementation can commence. Inspection validation consists of two parts:

- 1- **Analysis validation:** performed after the first five models are defined
- 2- **Design validation:** performed after the design model is defined

Design validation and analysis validation in steps 3 and 5 of MAVERICK are performed by inspecting the worksheets and diagrams defined in CommonKADS. The reason why no validation phase was defined after the context models is because the context models are strictly used for managerial decisions. Mostly, the two concept models (knowledge and communication models) contribute to the final KBS more than the first three context models. Although the abstract nature of the content of the context models doesn't support the idea of validation, analysis validation validates the context models too, along with the concept models.

Because the worksheets are developed by the knowledge engineer, in inspection validation, the expert reviews the models for any mistakes. The knowledge engineer might have understood some aspect of the organization or the tasks incorrectly and presented it with mistakes in the models' worksheets or diagrams. The expert's role here is to manually check the worksheets' contents and modify the worksheets with the knowledge engineer if any mistake is found.

Inspecting diagrams is an efficient technique to see the ‘big picture’ of a CommonKADS model. Therefore, a number of UML diagrams are manually inspected by the knowledge engineer and the expert. For analysis validation the UML diagrams inspected are: activity diagrams of OM3 and TM1, the class diagrams of TM1 and KM, the state diagram of CM and

the use case diagram of AM. For design validation, one class diagram is inspected: DM-1, which checks all the design decisions before the real implementation. The guidelines for performing inspection validation are:

1. Inspecting the organization, task and agent worksheets for correctness
2. Inspecting the knowledge and communication worksheets to check their consistency/similarity with the first three models
3. Inspecting the knowledge inferences and diagrams for correctness
4. Inspecting the UML diagrams to ensure that they are consistent with the worksheets
5. Inspecting the communication model constructs to ensure their consistency with the communication model worksheets
6. Inspecting the design of the system in the design model for correctness
7. Inspecting the test cases to check for any problems, gaps or redundancies

The incremental aspect of MAVERICK is that test cases are not all defined at the same time and inspection validation is not performed in one phase. Test cases are defined after every set of models is developed and incremental inspection validation is performed in two phases. Doing so improves the quality of test cases and the inspection validation process because of the following reasons:

- 1- The ideas about the model would be still fairly new in the mind of the knowledge engineer. Therefore, the test cases are designed with more knowledge about the models.
- 2- This incremental approach minimizes the possibility of forgetting some aspects of the model and so would decrease the chances of forgetting to define the test cases completely.

- 3- This incremental fashion would let the knowledge engineer define specific test cases for this model without looking at the global picture and the expert to inspect the models in great detail. If this is not the case, this would sometimes make the knowledge engineer or the expert overlook local aspects of a model.

Next section presents an important part of MAVERICK, the CBTCR process.

5.5 Context-Based Test Case Reduction (CBTCR)

MAVERICK is designed to be used throughout the development process rather than at the end as in other validation methods. To use MAVERICK, the knowledge-based system development and validation are performed in iterations. During any iteration in development, the values of variables are modified while the system undergoes refinement. This work reduces the number of test cases based on the context of validation. This is where from the concept of context-based test case reduction came. In problem solving, the *context* would inherently contain much knowledge about the situation in which the problem is to be solved or the environment of the problem [88]. In CBTCR, testing is intensified for the model that failed the most in the previous testing cycle (the model with the higher number of failing test cases). In other words, test case selection is affected by the current context.

CBTCR is performed iteratively and is partly automated. The iterative model of this method is illustrated in Figure 19. CBTCR test case reduction is controlled by a number of variables. The most important ones are *local importance* and *N*. They are discussed in the following two subsections.

5.5.1 Local Importance

Local Importance is a variable associated with every test case. It is factor of *dependency*, *domain importance*, *criticality* and *occurrence*. The values of these four variables are set by the expert and the knowledge engineer for each test case. The numerical ranges and calculations of these four variables are introduced in section 5.5.3, but first, the reasoning behind selecting these variables is discussed next.

Dependency: CommonKADS models are dependent upon each other, thus, test cases extracted from these models inherit this relation. For instance, the items in the task model depend on the items in the organizational model, making their test cases dependent on one another. Therefore, dependency is defined for each test case by the knowledge engineer as part of the test case importance.

Domain importance: A test case represents a certain function in the system. Some test cases have high importance because of their high representation of certain important functionality within the domain. Other test cases with less importance represent functions that are not highly related to the domain.

Criticality: In any organization, some tasks are more important than others. Any test case is defined to partially or fully evaluate certain functionality. Tasks (and thus, test cases) with more criticality to the overall process have higher importance.

Occurrence: In a process, some procedures occur more frequently than others. This variable reflects the level of a task occurrence in the system.

These four variables are influenced by the structure of CommonKADS. The dependency is in the core philosophy of CommonKADS and the criticality is defined in the worksheets as '*significance*'. The organization model reflects the domain and its most important parts and tasks

have different occurrences. For every test case, the average of these four variables is the value of the local importance.

5.5.2 The number of test cases selected for each iteration (N)

The value for N is chosen by the knowledge engineer. Nevertheless, MAVERICK provides a recommendation for N through the CBTCR tool (presented in the next chapter) for each iteration. In most cases, the number of required test cases (N) increases with the size of the KBS. The value of N is based on three factors.

- 1- The number of rules in the system (to reflect the size of the knowledge base): The number of test cases is generally greater than the number of rules in any project, because in most cases any rule needs one or more test case to validate it. One exception is in the case of chain of rules that fire each other; several rules can be covered by one test case. Nevertheless, the number of rules might change at each refinement iteration; the number of rules might increase or decrease.
- 2- The number of test cases generated by MAVERICK through CommonKADS
- 3- Project size (inherited from the CommonKADS design model): The size of the project could be measured in many ways. Commonplace methods include counting the number of lines of code in the system or the number of cases in a use case diagram. In MAVERICK, the project size is measured using the design model of CommonKADS. The design model presents the overall design of the KBS. Thus, the size is measured by the number of entities/modules in the DM model.

The previous two subsections introduce the local importance and N. Other variables are used in MAVERICK and they are introduced in the next section, along with the formulas for calculating N and LI.

5.5.3 Spiral Development and Validation

This section introduces the main variables used for test case selection.

Before the knowledge engineer starts with system implementation, it is necessary to define a number of control variables that are used to choose the test cases to be used in every iteration.

These variables are:

1. **Local Importance (LI):** Each test case is assigned a local importance value that is a real number between one and five.

Local importance = Average of (dependency, domain importance, criticality, occurrence).

Local importance is a function of dependency (value assigned from 1-5), domain importance (value assigned from 1-5), criticality (value assigned from 1-5) and occurrence (value assigned from 1-5). As discussed earlier, these values are defined by the knowledge engineer and the expert. The frequency of each task in the system is indicated in TM2 and this is used as the basis for defining the occurrence factor. Dependency is in the nature of CommonKADS: the design model depends on the knowledge and communication models, which depend on defining the task and the agent models, which are, both based on the organization model which is defined based on the knowledge elicitation. The organization model has the lowest dependency rate (1), because it is the first model. The design model has the highest dependency rate (5), because

it is the last model. For each task, “*significance*” needs to be assigned as part of worksheet OM3 (refer to Chapter 4). This can be used as a benchmark for assigning local importance.

2. **Model Weight (MW):** Model weight reflects the assurance level of the CommonKADS model - how valid is the model representation in the system. Every CommonKADS model is assigned a weight after each iteration of development. Initially, all the models have the same weight (MW is set to 5), and the same significance. However, when the development starts, model weights will constantly change based on the outcomes of the test case execution. The model weight values range between 1 and 10. MW could be set to any value before the first iteration, 5 is the midpoint from 0 to 10 and therefore, it was selected as the initial value. After the first validation iteration the knowledge engineer has no control over the MW; it is controlled by the previous validation results. When the assurance of all models reaches 10 (100%) and system development is done, validation stops.
3. **N:** Represents the reduced number of test cases to be selected in any iteration. N obviously needs to be less than the total of the extracted test cases; the number of rules is typically less than the number of test cases. The difference between the number of test cases and rules represents their relation. If this difference is divided by the number of modules of the system from DM (project size), the outcome would present a sensible value that is more than the number of rules, more than the number of modules of the system and less than the number of test cases. This way, N is always proportional with those three values and it presents a practical number of test cases that correlate well with the effort of the system’s development. The formula for N is:

(Number of test cases-Number of rules of current iteration)/Project size.

4. **Global Importance (GI):** This variable is used to define the importance of any test case within the universal set of test cases. This variable reflects the importance of the test case within all the models, not only the model that the test case belongs to.

$$\text{Global Importance} = \text{Local Importance} * \text{Model Weight}.$$

Figure 18 shows the GI effect on selecting test cases. Although Test case 10 (T10) has higher LI than Test case 50 (T50), T50 is selected for execution before T10. T10 is in TM which has a model weight of 3, while T50 is in DM which has a model weight of 8. This would result in T10 with GI: $5*3=15$ and T50 with GI: $2*8=16$. Therefore, by comparing the GI for T50 and T10, T50 is greater and thus selected before T10.

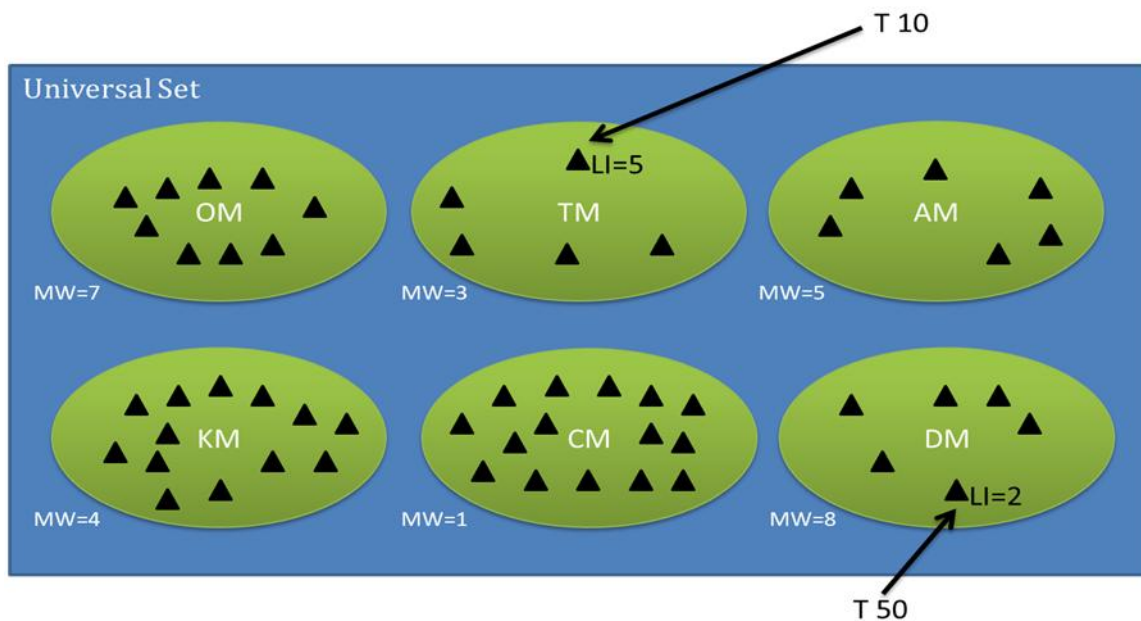


Figure 18: The effect of GI on sorting test cases

These four variables are used for test case reduction. The steps for CBTCR are presented in the next section.

5.5.4 Test Case Reduction through CBTCR

As discussed in Chapter 2, several researchers have tried to reduce the number of test cases [52] [64], but it is not easy to determine the value of a sufficient number of test cases for a given system. Approaches to reduce test cases have varied between random [58], formal [65] and informal [55]. Knauf et al. [47] introduced an exhaustive set of test cases as the absolute benchmark for validity (discussed previously). Then Abel et al. [65] continued on that work to introduce a smaller set of test cases that has an equivalent effect. While they obtained encouraging results, reducing test cases was static, in CBTCR, test case reduction is performed dynamically and it aims to reduce the selected set of test cases even more.

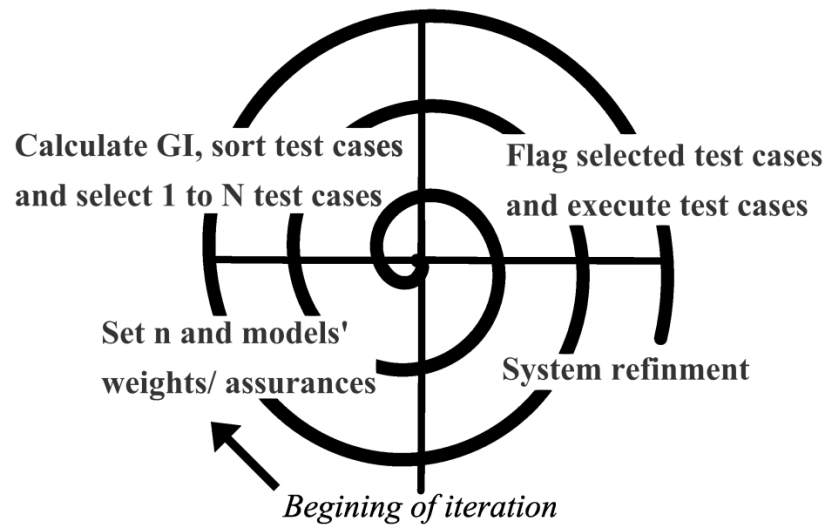


Figure 19: MAVERICK iterative development and validation

Because CommonKADS covers all aspects of the system, it provides solid ground for test cases. It helps in claiming that test cases extracted using this method cover the system in a complete manner. The steps of CBTCR that compose the validation of a system built on CommonKADS are presented next as well as in Figure 20. This algorithm is built in the CBTCR tool (presented

in the next chapter). For every CBTCR step, the degree of automation is indicated in the following list, whether it's manual or automated within the CBTCR tool. The steps are as follows:

1. Assign *local importance* for each test case. (Manual)
2. Set the size of test case subset: *N* based on the criteria discussed previously. (Automated)
3. Set all models' *weights/assurance* to 5 (Automated)
4. Calculate *global importance* for all test cases (Automated)
5. Order all test cases according to global importance (Automated)
6. Start iterative KBS implementation, building the KBS (Manual)
7. At the end of the development iteration, select *N* number of test cases from the ordered list select test cases 1 to *N* (Automated)
8. Execute the test cases on the system (Manual)
9. Compare results of executed test cases to its expected result (Manual)
10. Based on results for each CommonKADS model test cases, re-assign assurance for each model. Example: if 30% of the test cases in a certain model are incorrect, that model's assurance will be 7 using the following formula: $100 - (\text{percentage of successful test case})/10$ (Automated)
11. Recalculate global importance for all test cases and reorder (Automated)
12. **Flag** test cases with a positive outcome (not to be picked again unless a change to their status was made). **Flag** test cases with unexpected outcomes (this is used to make sure that the test case is reselected before end of validation). **Flag** test cases that are affected by the refinements (to be selected again). Select different test cases and go to the next iteration (Automated)

13. Refine the system. This might lead to adding new rules, deleting rules and adding new test cases. This step is performed by the knowledge engineer in a manual fashion.

(Manual)

14. Go back to step 6

15. Stop when assurance of all models is equal to 10 (Automated)

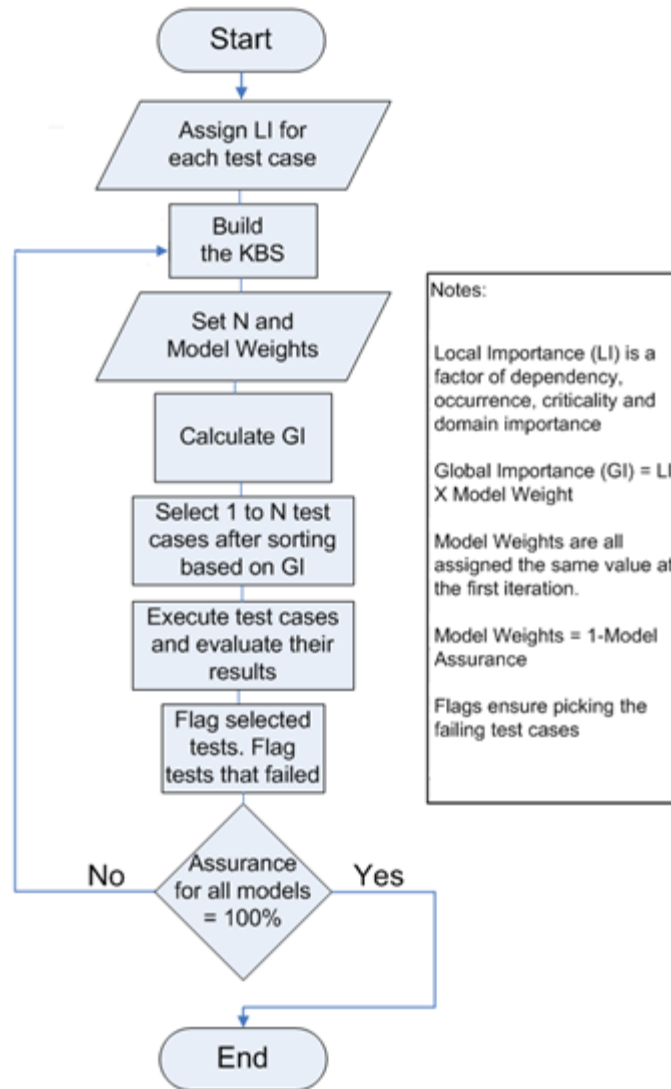


Figure 20: CBTCR flowchart

This approach requires some non-trivial manual work by the knowledge engineer, expert or any other person performing validation, but it has many advantages:

1. **Flexibility:** the initial values of the weights and the models could be modified by the knowledge engineer. This gives the knowledge engineer full control.
2. **User-oriented:** this approach is based on the user needs and a real time testing feedback based on context. It is not a static function, rather a resilient one.
3. **Based on a comprehensive, well-defined and well-structured model:** This function is based on CommonKADS, which as discussed previously, has many advantages on its own.
4. **Semi-automated:** this method is partly automated by the CBTCR tool (presented in the next chapter).
5. **Effort and time reduction:** reducing the number of test cases reduces effort and time.

Experiments and analysis are performed to test the MAVERICK approach. Experimental evaluation of MAVERICK is introduced in Chapter 7.

5.6 Summary

This chapter introduced the validation method MAVERICK in detail. MAVERICK consists of three main steps including test case extraction, the inspection validation phase and context based test case reduction to reduce the number of test cases. In the next chapter, a prototype KBS is introduced. A system is built using CommonKADS and validated using MAVERICK. The KBS developed is a housing KBS.

CHAPTER 6: PROTOTYPE KNOWLEDGE-BASED SYSTEM

This chapter describes the KBS built to evaluate the MAVERICK validation method presented in this dissertation. This KBS is called the housing KBS, and it is built through CommonKADS and it is validated using MAVERICK. Additionally, a Java software tool is introduced that implements the automated process of sorting, selecting and reducing test cases. This process is called: context based test case reduction (CBTCR). The first section introduces the CBTCR tool. Sections 2 and 3 introduce the CommonKADS models and the details of the housing KBS. Section 4 lists the extracted test cases for the housing KBS.

6.1 The CBTCR Tool

The housing KBS defined in the CommonKADS book [9] and the information introduced in the book for the application is a benchmark and a solid foundation for the experiments. The housing KBS is validated using MAVERICK. The first step in MAVERICK is to extract the test cases from the system as discussed in Chapter 5.

The test cases are entered into the tool's spread sheet with the following fields: 1) Test case ID (An incremental integer that starts from zero and is incremented by one for every test case), 2) CommonKADS Model (a number between one and six, one for the organizational model, two for the task model, three for the agent model, four for the knowledge model, five for the communication model and six for the design model), 3) Local Importance (an integer number from 1-5), 4) Number of Runs (is initially set to zero, this number is incremented by one every time the test case is executed), 5) CommonKADS Weight (initially is set to 5 for all models then it is modified every iteration), 6) Global Importance (the multiplication outcome of local importance and model weight), 7) Result (all test cases are initially set to two because none of

the test cases is executed; when a test case is executed with success, the GI value will change to one, if it failed, value will change to zero), 8) Input Variables, 9) Test Setup Values, 10) Execution Steps, 11) Expected Solution, 12) System Solution and 13) Informal Description. When the spiral development starts, the tool will recommend test cases for execution on the system. The CBTCR tool's interface has six panels as illustrated in Figure 21. The six panels are indicated on the screenshot, and they are discussed next:

Panel 1 displays the selected test cases in a list where the knowledge engineer can indicate the outcome of the test case, whether it is success or failure. This panel also displays the iteration number and the value of N. This is the main panel for the knowledge engineer, where the test cases could be monitored and the results of the test cases after every iteration could be modified.

Panel 2 displays all the test cases, each with its importance, execution results, CommonKADS model and the number of test cases in the database. In this panel, all the test cases changing statuses can be observed real time, after every test case execution.

Panel 3 has two functionalities; it displays the algorithm's steps and explains how things work in the tool. Additionally, it is the panel the recommended N value is calculated. The knowledge engineer enters the number of test cases, the number of rules in the project and the project size to get N. Project size is calculated using the CommonKADS design model diagram. The number of entities in the diagram represents the project's size. N is calculated based on the three variables and the formula presented in the previous chapter.

Panel 4 shows the validation percentage/assurance for all CommonKADS models. This is calculated by averaging of all the models assurances. A progress bar displays this percentage.

Panel 5 displays the console showing all the steps and all the actions performed. The console serves as a good documentation tool; it keeps all the test cases as well as all the models and their changing status. Everything is saved and displayed here, then saved to a file on the hard disk.

Panel 6 is where the knowledge engineer can input the system and tester information. This includes the name, organization, date, department, role and KBS name. This information is displayed on the console when it is saved.

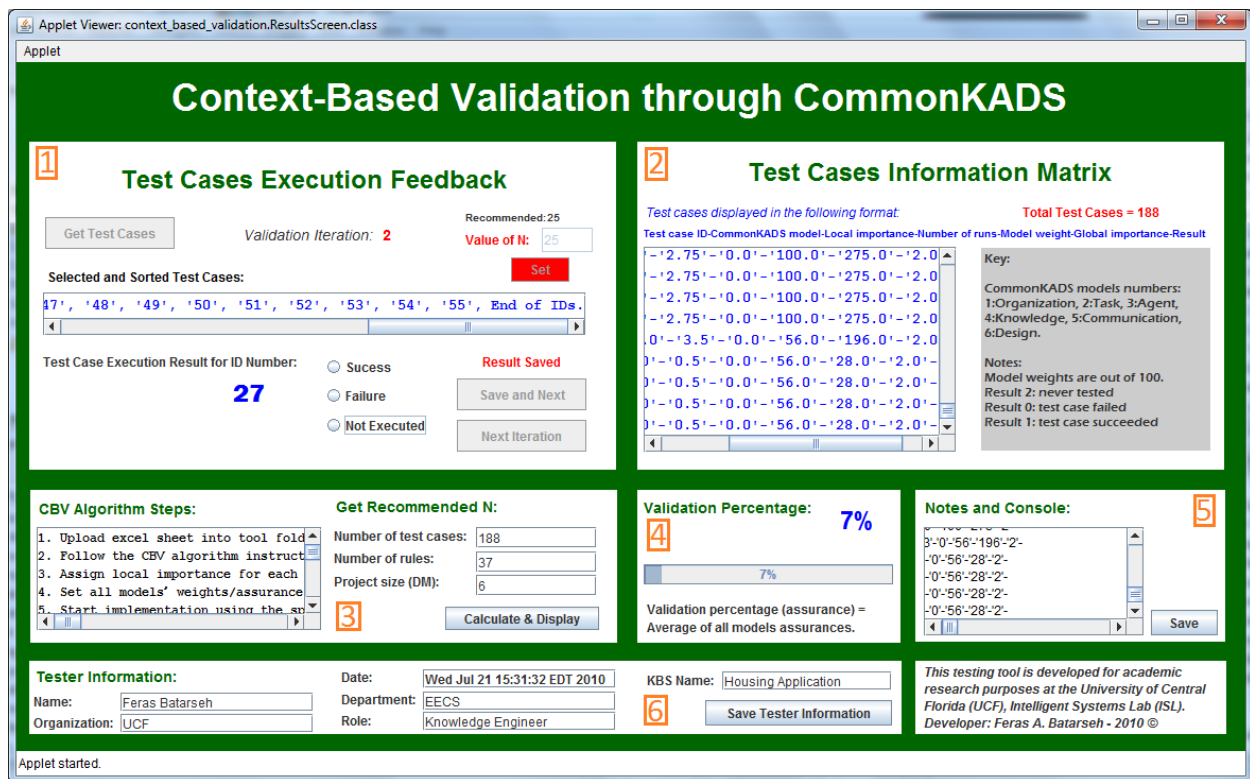


Figure 21: CBTCR Java tool interface

The CBTCR tool is available for research purposes upon request to the author.

The CommonKADS development steps, models, diagrams and details about developing the housing KBS are introduced in the next section.

6.2 The Housing KBS

The rule-based housing KBS is introduced in chapter 10 of Schreiber et al.'s CommonKADS book [9] as a case study. This housing KBS is used by the Dutch government to assign apartments to citizens who apply for one. This process is called *residence distribution* and citizens who want an apartment must apply to be a *potential applicant*. Every two weeks a magazine is published and distributed in the Netherlands with a list of residences for which people can apply. Another published proceeding has the names of the winning applicants. This system's only user is the Dutch government.

To be eligible for a residence, applicants must meet a number of criteria. If an applicant doesn't fit the conditions of any of the residences or all the residences that fit the applicant are taken, a residence that best fits the criteria out of the remaining available residences is assigned to the applicant. Residents must consider the number of family members in their household. Residence category is defined by the area where the residence is located. A fair relation between the applicant's income and the rent of the apartment should be established. Other parameters may apply for every specific case/applicant. The knowledge-based system under development is to automate the applicants' assessment process. Inputs to this system include data about the applicant and the residences. The output is a set of residence assignments for the applicant. The knowledge-based system communicates with two other systems, a database that contains information about the applicants, and a program that computes a list that prioritizes the applicants for every residence.

The knowledge base contains the rules of assignments. The following information is needed for each applicant: a) Name, b) age, c) date of submitting the application, d) number of family members and e) annual income. For all the residences, three main parameters will be

included in the criteria of assigning applicants: a) Area, b) rent and c) location. The system considers every applicant for a resident. The first filter is the date of submitting the application; each applicant is assigned a residence that suits his/her information. If no apartment was found to meet applicant's information, the apartment that best suits the applicant is assigned to him/her. The applicants' information is categorized into groups, age is categorized as: 0-20 years, 20-40 years, 40-60 years and 60 years and up. The number of family members is defined into five categories: 1, 2-3, 4-5, 6-7, 8 and more. Annual income is categorized into four categories: €0-€40,000, € 40,000- €100,000, €100,000- €500,000 and more than €500,000. For each residence, the area of the apartment is categorized into five categories: 0-600 m², 600-800 m², 800 -1000 m², 1000 -1200 m² and more than 1200 m². Moreover, the apartments are divided into five categories for the location: A, B, C, D and E. Finally, different apartments have different rents: €545, €750, €1047 and more than €1047.

The living area of the apartment should fit the number of family members. If the number of family members is equal to one, then the area should be the smallest category: 0-600. The Rent of the apartment should fit the annual income: Category #1 in rent will be assigned Category #1 in income, and so on. Age is used to consider more options for the applicant, if the applicant is of age 20-40 years and has an income more than €40,000, he/she would have more apartment options. This means that if the applicant is young and has a high salary or old with a steady job, then the probability of him/her not paying rent is fairly low. Thus, a better apartment is assigned to him/her. All assignments are approved by the applicants, so if an applicant is not comfortable with their new apartment location, area or rent, they can decline the assignment and apply again in the next cycle.

The system was built iteratively and validated using the MAVERICK method, first step was to build the CommonKADS models and extract the test cases. All the steps of building the systems are introduced next.

6.3 CommonKADS Models for the Housing KBS

In this section, the CommonKADS models for the housing KBS are described. Furthermore, all the diagrams and graphs included in the system documentation are introduced too. Additionally, test cases are extracted based on the extraction method introduced in Chapter 5. Test cases are entered to the CBTCR tool and executed. The results are recorded and introduced in Chapter 7. This section is structured based on the MAVERICK validation method presented in Chapter 5. The main five steps for the validation process are:

1. Context Test Cases Extraction: Introduced in section 6.3.1 and 6.4
2. Analysis Test Cases Extraction: Introduced in section 6.3.2 and 6.4
3. Design Test Cases Extraction: Introduced in section 6.3.3. Section 6.4 includes all the test cases extracted from the diagrams and the worksheets in CommonKADS.
4. Spiral System Implementation: Introduced in section 6.3.5
5. Spiral System Validation: : Introduced in section 6.3.5

The worksheets and diagrams copied from the CommonKADS book [9] have the reference number 13 next to their title. All the other components were influenced by the book, but developed by the author. Next section introduces the CommonKADS context models for the housing KBS.

6.3.1 Context Models

CommonKADS context models are shown in Tables 18 – 33.

Table 18: Worksheet OM-1 for housing KBS [9]

Problems and opportunities	Assessment of individual applications takes too much time. There is no sufficient staff to handle urgent cases.
Organizational context	Mission: Enable people to take as much as possible themselves responsibility for finding a suitable home. Enable insight into the dynamics of the rental housing market. External factors: Local council National regulations Applicants/public opinion Rental agencies Strategies: Provide high quality for a reasonable price. Move to semi-private service company. Broaden scope (include lower segment of privately owned residences).
Solutions	Solution 1: Develop and automated system for application assessment. Set up a training program for a group of assigners to specialize in urgency handling.

Table 19: Worksheet OM-2 for housing KBS [9]

Structure	See Figure 22
Process	See Figure 23
People	See Figure 22
Resources	Database of data about applicants and residence. Priority calculator: a program that computes the applicants' priorities for each residence.
Knowledge	Assessment criteria: knowledge for judging correctness of individual applications. Assignment rules: knowledge used for selecting an applicant for a particular house. Urgency rules: special rules for urgent cases (e.g. handicapped applicants)
Culture and power	Hierarchical organization. Employees view the future with some trepidation. Management style is still based on history as civil servant department.

These models are based on the system description from the CommonKADS book [9]. The housing KBS has three agents (the assigner, the data typist and the editor) and four main tasks (data entry, magazine production, application's assessment and residence assignment) introduced in the work sheets. The CommonKADS worksheets are self-descriptive. They include all the details of the KBS.

Table 20: Worksheet OM-3 for housing KBS [9]

No.	Task	Performed by	Where?	Knowledge Asset	Intensive?	Significance
1	Magazine production	Magazine producer	Public service	N/A	No	3
2	Data entry of application	Data typist/automated telephone number	Residence assignment	N/A	No	2
3	Application assessment	Assigner	Residence assignment	Assessment criteria	Yes	5
4	Residence assignment	Assigner	Residence assignment	Assignment rules and urgency rules	Yes	5

Table 21: Worksheet OM-4 for housing KBS [9]

Knowledge Asset	Possessed by	Used in?	Right form?	Right place?	Right time?	Right quality?
Assessment criteria	Assigner	Application assessment (3)	No: paper form → electronic	Yes	Yes	Yes
Assignment rules	Priority calculator	Residence assignment (4)	Yes	Yes	Yes	Yes
Urgency rules	Assigner	Residence assignment (4)	Yes	Yes	Yes	No: often incomplete, ambiguous and inconsistent.

Table 22: Worksheet OM-5 for housing KBS [9]

Business feasibility	The KBS will cost \$150,000 for development and \$10,000 for yearly maintenance. This investment is cost-effective if we assume that fewer than three people will be needed to do the application-assessment work. Training costs need to be considered. The system will provide higher quality and fewer errors which is important for the public image of the organization.
Technical feasibility	Assessment tasks are well understood. Many existing system tackle this task type. the knowledge needed is available.
Project feasibility	There is no real expertise in this domain. This minimizes the well-known risk of lack of experts. Skills needed in this project: experience in building an assessment application, knowledge about the database and the priority calculator.
Proposed actions	Set a team and schedule for system development. Start with the required organizational changes: training assigners as urgency handlers, but first, coordinate with the residence-assignment and computer departments for support of the new proposed KBS.

Table 23: Worksheet TM-1 for housing KBS (Task 1)

Task	Magazine production (1)
Organization	Magazine production
Goal and Value	Task should ensure the inclusion of all the residences and publish the magazine on time
Dependency and Flow	<i>Input tasks:</i> - <i>Output tasks:</i> -
Objects Handled	<i>Input tasks:</i> data about residences <i>Output tasks:</i> published magazine
Timing and Control	Carried out once every two weeks
Agents	Magazine editor and producer
Knowledge and Competence	-
Resources	-
Quality and Performance	Not time critical, every 2 weeks.

Table 24: Worksheet TM-2 for housing KBS (Task 1)

Nature of Knowledge	True/False	Bottlenecks
Formal, rigorous		
Empirical, quantitative		
Heuristic, rule of thumb		
Highly specialized, domain specific		
Experience based		
Action based	X	
Incomplete		
Uncertain, maybe incorrect		
Quickly changing		
Hard to verify		
Tacit, hard to transfer		
Form of Knowledge		
Mind		
Paper	X	
Electronic		
Action skills		
Other	X	
Availability of knowledge		
Limited in time?		X
Limited in space?		
Limited in access?		
Limited in quality?		
Limited in form?	X	

Table 25: Worksheet TM-1 for housing KBS (Task 2)

Task	Data entry of applications (2)
Organization	Residence assignment department by the data typist
Goal and Value	Task should ensure that all applicants' applications are entered into the system.
Dependency and Flow	<i>Input tasks:</i> applications <i>Output tasks:</i> application assessment
Objects Handled	<i>Input tasks:</i> - <i>Output tasks:</i> -
Timing and Control	Carried out for every application.
Agents	Knowledge system
Knowledge and Competence	-
Resources	-
Quality and Performance	-

Table 26: Worksheet TM-2 for Housing KBS (Task 2)

Nature of Knowledge	True/False	Bottlenecks
Formal, rigorous		
Empirical, quantitative		
Heuristic, rule of thumb		
Highly specialized, domain specific		
Experience based		
Action based		X
Incomplete		X
Uncertain, maybe incorrect		X
Quickly changing		
Hard to verify	X	
Tacit, hard to transfer		
Form of Knowledge		
Mind		
Paper		
Electronic	X	X
Action skills		
Other		
Availability of knowledge		
Limited in time?		
Limited in space?		
Limited in access?		
Limited in quality?		
Limited in form?		

Table 27: Worksheet TM-1 for housing KBS (Task 3) [9]

Task	Application assessment (3)
Organization	Residence assignment department by the assigner
Goal and Value	Task should ensure that applicants are treated fairly. Essential to deliver assignment service.
Dependency and Flow	<i>Input tasks:</i> magazine production and data entry <i>Output tasks:</i> residence assignment
Objects Handled	<i>Input tasks:</i> application and data about residence. <i>Output tasks:</i> validated application
Timing and Control	Carried out for every application. Save a log for all the processes with a summary for each one.
Agents	Knowledge system
Knowledge and Competence	Assessment criteria
Resources	-
Quality and Performance	Not time critical, but requires seconds to get result, availability should be high

Table 28: Worksheet TM-2 for housing KBS (Task 3) [9]

Nature of Knowledge	True/False	Bottlenecks
Formal, rigorous	X	
Empirical, quantitative		
Heuristic, rule of thumb		
Highly specialized, domain specific	X	
Experience based		
Action based		
Incomplete		
Uncertain, maybe incorrect		
Quickly changing	X	X
Hard to verify		
Tacit, hard to transfer		
Form of Knowledge		
Mind	X	
Paper		
Electronic		
Action skills		
Other		
Availability of knowledge		
Limited in time?		
Limited in space?		
Limited in access?		
Limited in quality?		
Limited in form?	X	X

Table 29: Worksheet TM-1 for housing KBS (Task 4)

Task	Residence assignment (4)
Organization	Residence assignment department by the assigner
Goal and Value	Task should ensure that applicants are treated fairly. Essential to deliver assignment service.
Dependency and Flow	<i>Input tasks:</i> magazine production, application assessment and data entry <i>Output tasks:</i> -
Objects Handled	<i>Input tasks:</i> - <i>Output tasks:</i> -
Timing and Control	Carried out for every applicant. Save a log for all the processes with a summary for each one.
Agents	Knowledge system
Knowledge and Competence	Assignment rules and urgency rules.
Resources	-
Quality and Performance	Time critical

Table 30: Worksheet TM-2 for housing KBS (Task 4)

Nature of Knowledge	True/False	Bottlenecks
Formal, rigorous	X	
Empirical, quantitative		
Heuristic, rule of thumb		
Highly specialized, domain specific	X	
Experience based		
Action based		
Incomplete		
Uncertain, maybe incorrect		
Quickly changing	X	X
Hard to verify	X	
Tacit, hard to transfer		
Form of Knowledge		
Mind	X	
Paper		
Electronic	X	
Action skills		
Other		
Availability of knowledge		
Limited in time?		
Limited in space?		
Limited in access?		
Limited in quality?	X	
Limited in form?	X	X

Table 31: AM-1 for housing KBS (agent: assigner)

Name	Assigner
Organization	Residence assignment department
Involved In	Tasks 3, 4
Communicates with	Database, priority calculators and rental agencies.
Knowledge	Assessment criteria, assignment rules and urgency rules.
Other competences	Ability to handle problematic cases.
Responsibilities and constraints	Make sure people are treated equally.

Table 32: AM-1 for housing KBS (agent: data typist)

Name	Data typist
Organization	Residence assignment department
Involved In	Task 2
Communicates with	Magazine and applications
Knowledge	-
Other competences	Speed of data entry
Responsibilities and constraints	Reduce amount of mistakes

Table 33: AM-1 for housing KBS (agent: magazine editor)

Name	Magazine editor
Organization	Magazine production department
Involved In	Residence assignment
Communicates with	-
Knowledge	-
Other competences	No mistakes in magazine
Responsibilities and constraints	-

The agent models presented here are all the agents involved with the system tasks. These tasks will be transferred to create the KBS. Other agents presented in Figure 22 are not included in the systems' tasks. This concludes all the worksheets for the context models for the housing KBS. In worksheet OTA-1, the decision is to start developing the KBS by creating a new human role: urgency handler. The KBS will only handle the regular cases; urgent cases will be processed by humans who will undergo specialized training.

6.3.2 Concept Models

For this system, one knowledge model worksheet (Table 34) is needed and three for the communication model.

This is because three communication transactions can occur in the system:

1. Order application assessment (Table 35)
2. Obtain application data (Table 36)
3. Report decision (Table 37)

CM2 is not needed for this system because of the lack of any special control over messages or any specific constraints. The knowledge is not only represented in the worksheet but is also illustrated in diagrams, Figures 26-31.

Table 34: KM-1 for housing KBS

Document Entry	Description
Knowledge model	See Figures 26-31
Information source used	Experts
Glossary	-
Components considered	Components used in identification stage.
Scenarios	Four scenarios, one for each task. Other scenarios are considered urgent cases which will be handled by human employees.
Validation results	Not valid yet
Elicitation material	Interview material and documentation form the context model.

Table 35: CM-1 for housing KBS (transaction 1) [9]

Transaction identifier name	Order application assessment
Information object	A residence application
Agents involved	Data entry + knowledge system + assigner
Communication plan	See Figures 31 and 32
Constraints	System must interact with the application data entry
Information exchange specification	<i>Order</i>

Table 36: CM-1 for housing KBS (transaction 2)

Transaction identifier name	Obtain application data
Information object	Attribute value pairs of an applicant and residence
Agents involved	Data base and KB
Communication plan	See Figures 31 and 32
Constraints	Correct mapping of data requests
Information exchange specification	<i>Ask-reply</i>

Table 37: CM-1 for housing KBS (transaction 3)

Transaction identifier name	Report decision
Information object	Decision document
Agents involved	Assigner + KB
Communication plan	See Figures 31 and 32
Information exchange specification	<i>Ask-reply</i>

6.3.3 Artifact Models

Table 38: DM-1 for housing KBS

Architecture decision	Format
Sub-system structure	See Figure 33
Control model	-
Sub-system decomposition	See Figure 33

Table 39: DM-2 for housing KBS

Software package	Housing KBS
Potential hardware	Intel Core 2 Duo Computer
Visualization library	-
Language typing	Object oriented
Knowledge representation	Rule based
Interaction protocol	Class will be developed to interact with outside objects
Control flow	-
CommonKADS support	CBTCR tool

Table 40: DM-3 for housing KBS

Architecture component	Typical decision points
Controller	Mechanism for internal/ external events handling
Task	No
Task method	Java
Inference	-
Inference method	Presented in system
Dynamic role	-
Static role	-
Knowledge base	Housing KB
Views	One interface for each agent presented in the agent model. Agents need to log in to their interface. All interfaces are Java Swing interface objects.

Table 41: DM-4 for housing KBS

Element	Design decision
Controller	System on one machine, no communication between machines
Tasks methods	Formalize control structure
Dynamic roles	-
Inferences	Inferences defined in knowledge model
Inference methods	Inferences defined in knowledge model
Knowledge bases	-
View objects	System interface

The four design models with their decisions are introduced in tables 38, 39, 40 and 41.

6.3.4 Diagrams for the Housing KBS

This section includes all the diagrams needed for each set of models.

6.3.4.1 Diagrams for the Context Models

This subsection introduces all the diagrams needed to present the context models. Figure 22 shows the agents of the system and their relations.

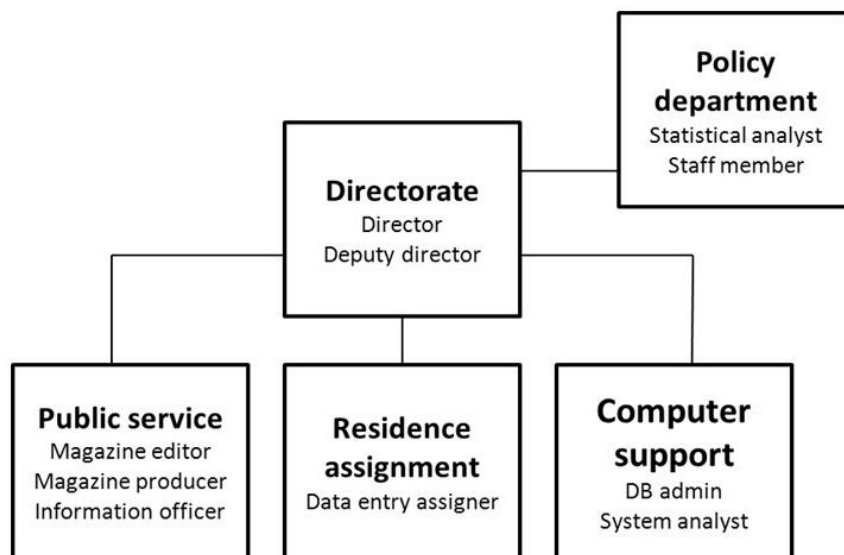


Figure 22: Structure and people in housing KBS [9]

Figure 23 illustrates the main flow of tasks within the organization (this is initially introduced in worksheets TM and OM).

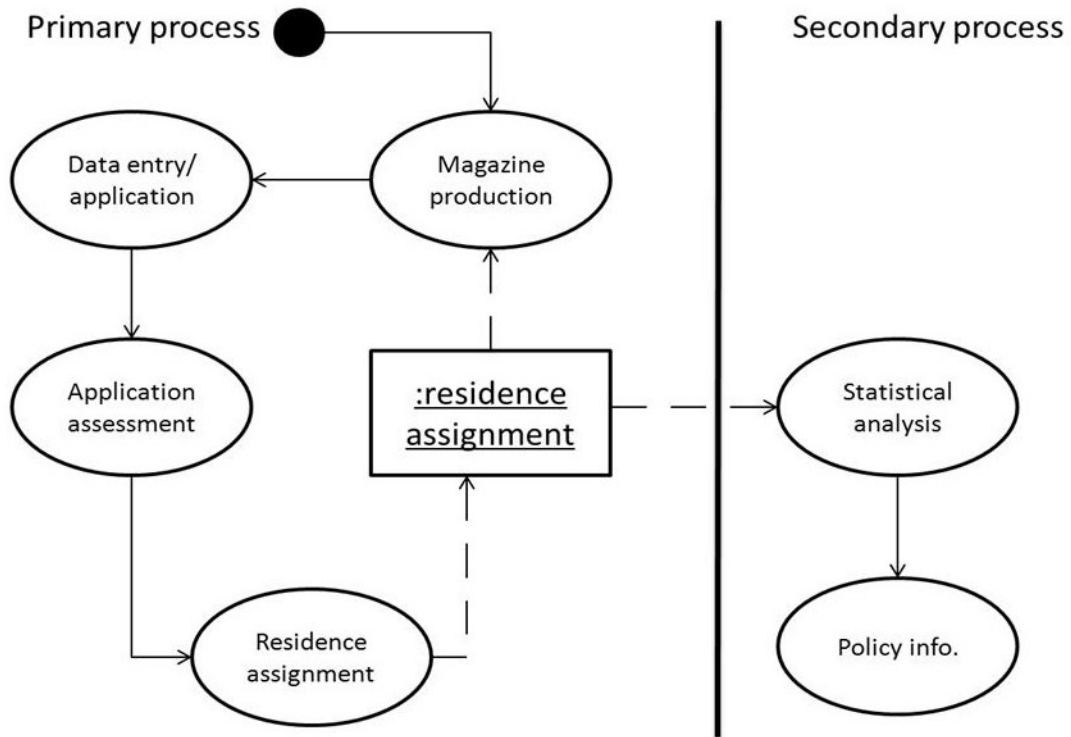


Figure 23: UML activity diagram for OM [9]

Figure 24 shows the flow of events of the assignment task in a DFD. The assignment task is the main task in the KBS. Figure 24 includes the actors, processes, data store and data flow. The last figure for the context models includes the flow of an application within the organization, displays the flow of tasks and the resulting outcomes.

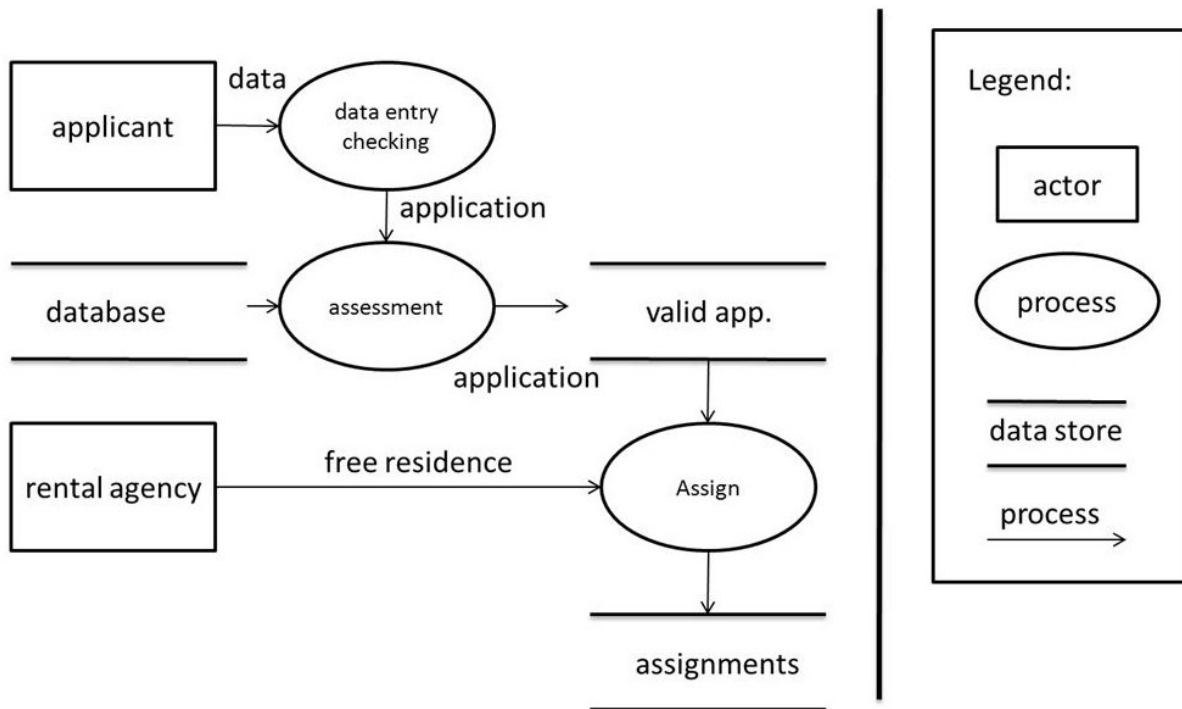


Figure 24: DFD for the main process (application-assessment task) [9]

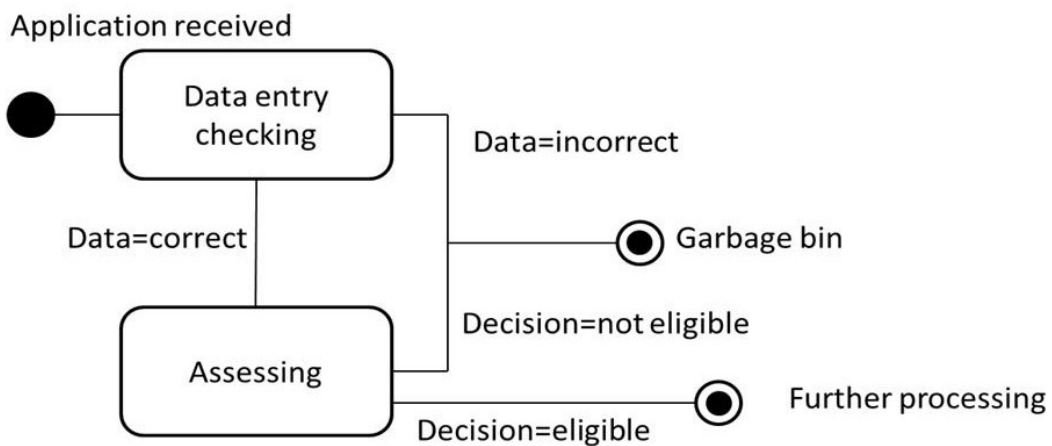


Figure 25: State diagram of a single flow of the application-assessment process [9]

6.3.4.2 Diagrams for the Concept Models

This subsection introduces all the diagrams needed to illustrate the knowledge and the communication tasks between the agents. Figure 26 presents the parameters associated with the residences and the applicants.

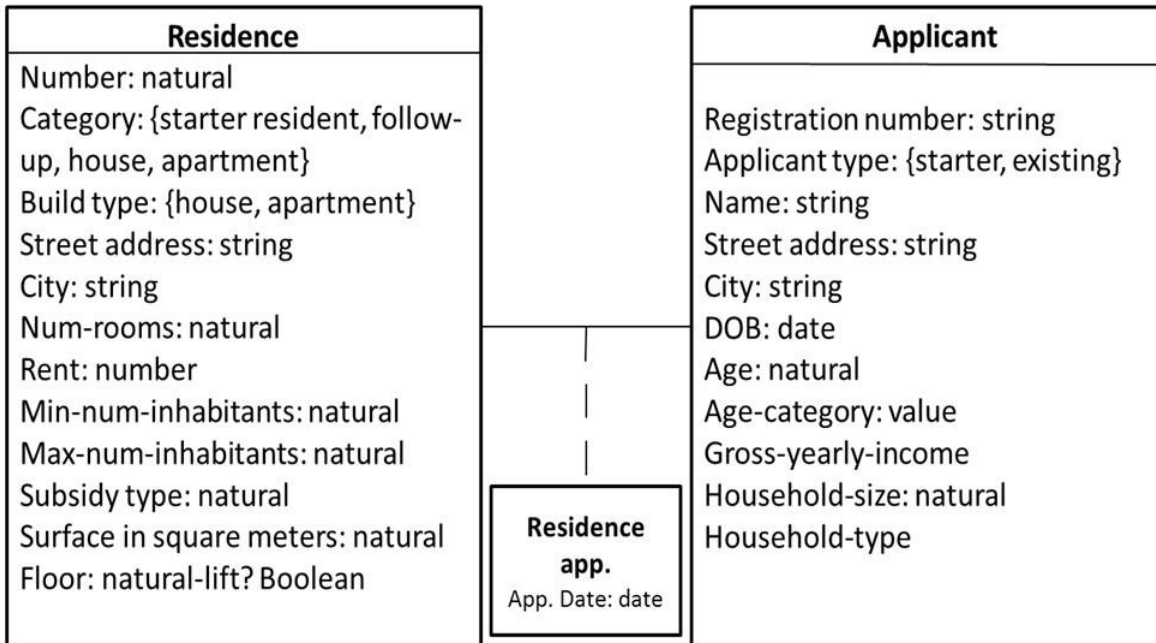


Figure 26: Residence and application relation in knowledge representation [9]

Figure 27 introduces the inference structures and constructs. Each inference flow starts with one applicant and one residence as its input raw data and outcomes a decision based on the rules of the inference. The decision is either applicant is ‘eligible’ or ‘not eligible’. Figure 28 introduces the assignment’s decision making priorities for residences.

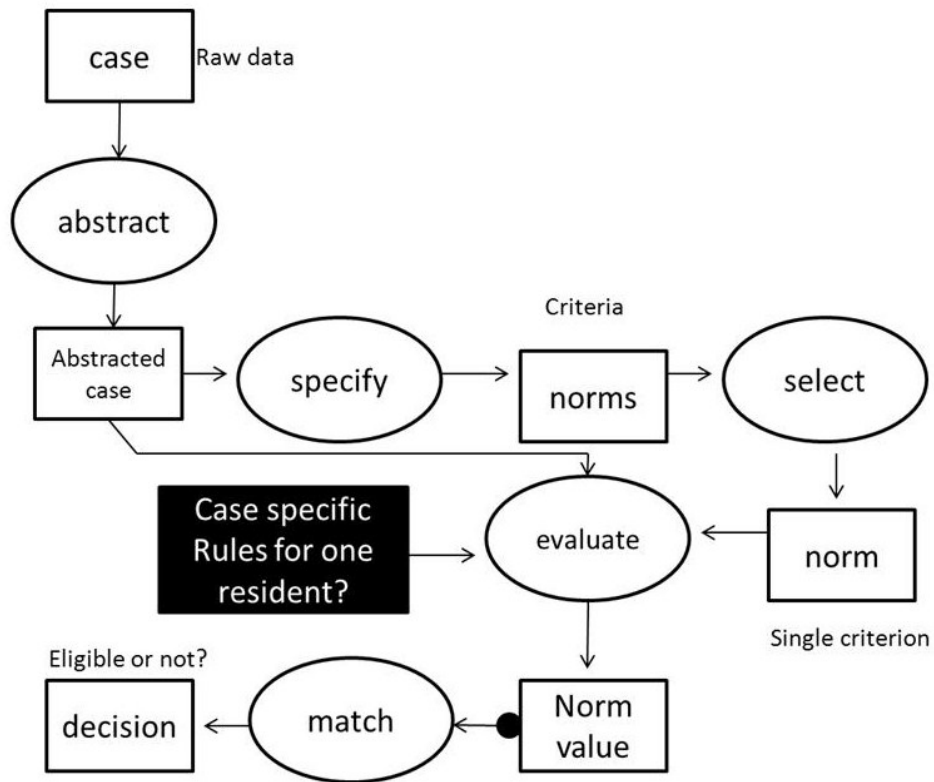


Figure 27: Inference structure for the KM [9]

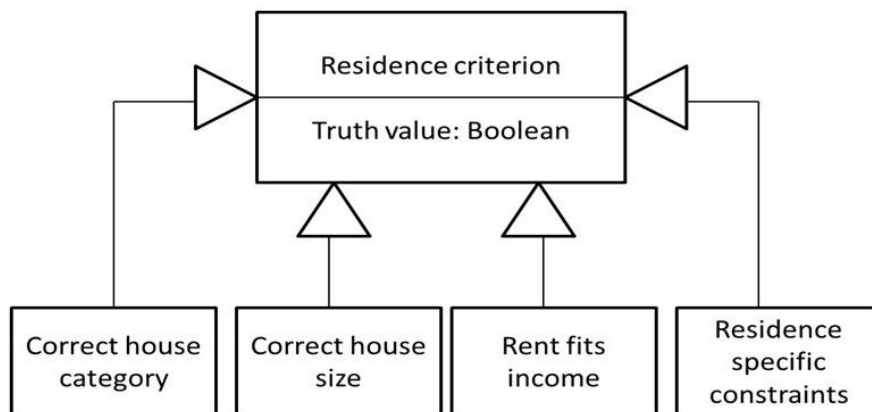


Figure 28: Subtype hierarchy for knowledge about priorities in the system [9]

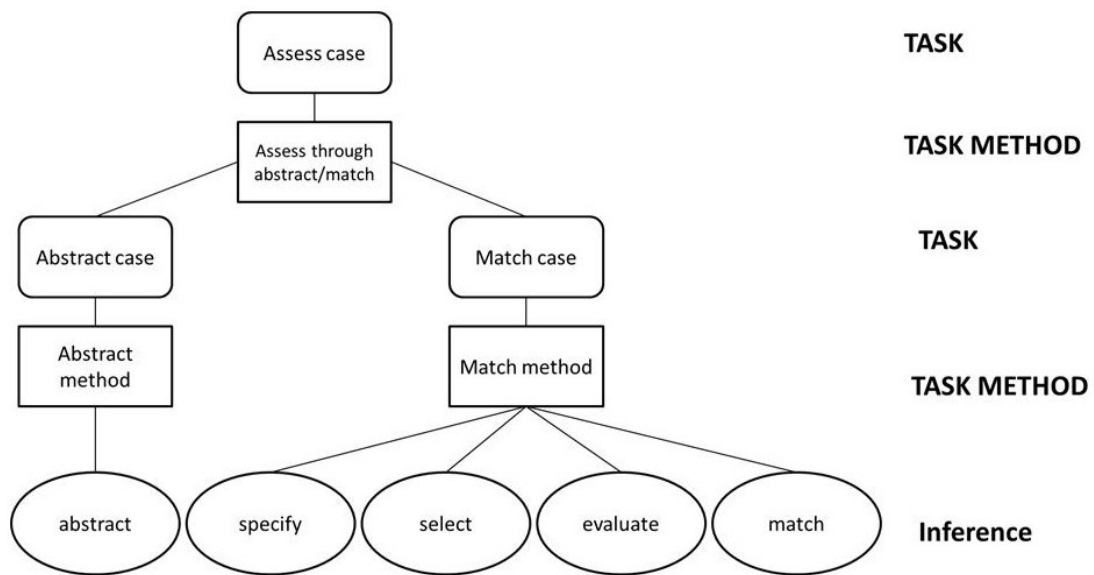


Figure 29: Tasks and their inferences (assessment task) [9]

Figures 29 illustrate the “*assess case*” task hierarchy. It establishes the relation between the applicant and the residence.

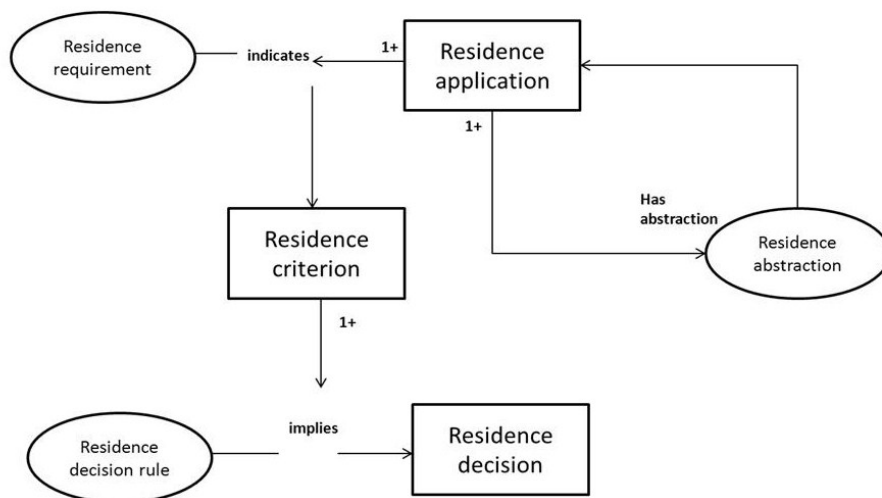


Figure 30: Domain schema for the housing KBS [9]

Figure 30 shows the big picture for the housing KBS. It shows all the tasks, their relations and flow directions. Figures 31 and 32 illustrate the communication modules between the tasks and the agents in the KBS.

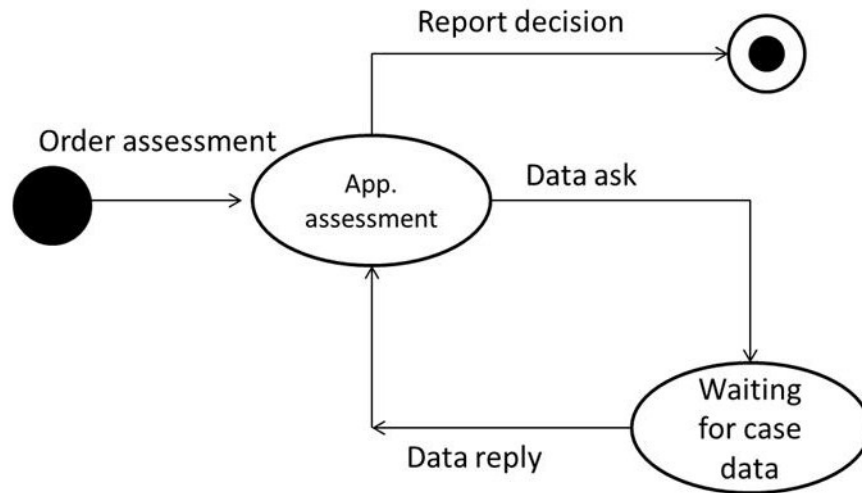


Figure 31: State diagram representing the communication model [9]

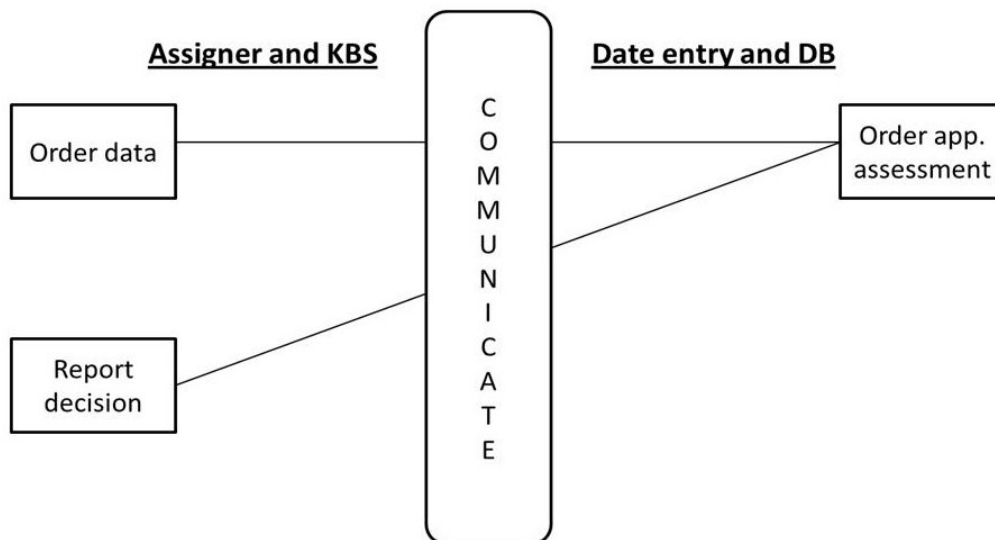


Figure 32: Dialogue diagram of the communication model

6.3.4.3 Diagrams for the Artifact Models:

One diagram is needed to present the design of the KBS. Figure 33 illustrates the design of the KBS, the three agents' interfaces and the relations between the KBS sub-components. This system includes six sub-components as it is illustrated in Figure 33. This is the final figure for the housing KBS.

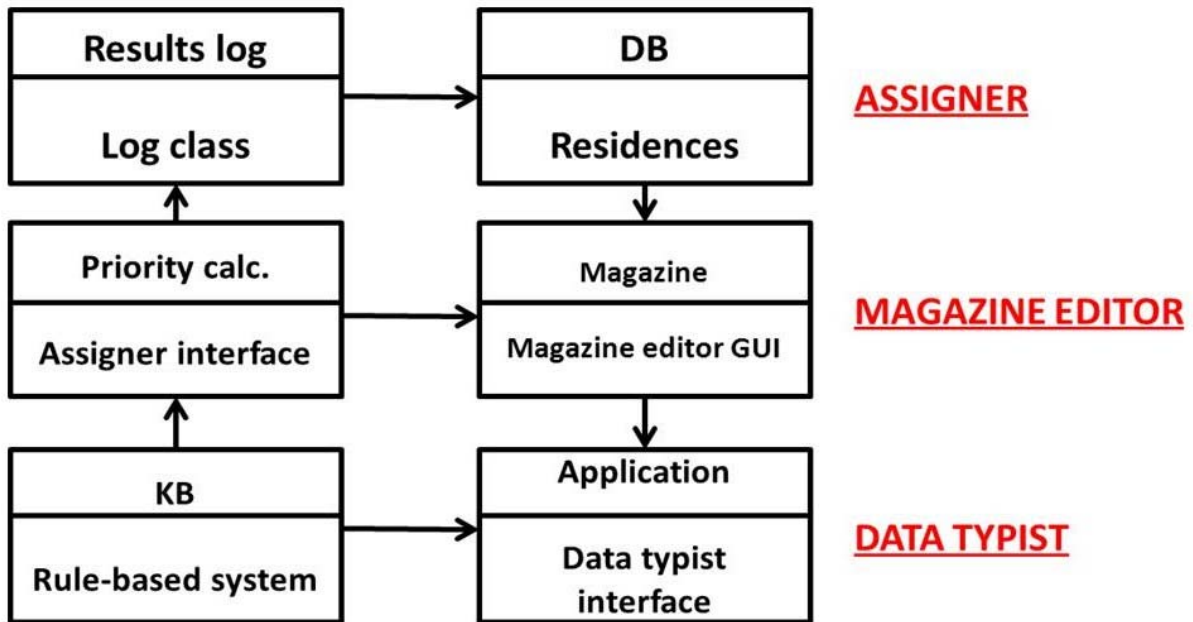


Figure 33: Subsystems relationship for the design model

6.3.5 Iterative System Implementation and Validation

At this stage, all the models have been well defined and the system design is complete. Mapping from the CommonKADS models to the knowledge base is straightforward. All the components are designed and all the crucial decisions are already made. The following steps are required to transform the CommonKADS models into the executable system:

1. Define the subsystems within the hardware/software platforms defined in the design model
2. Build the knowledge base based on the KM components
3. Code the tasks in the task model as the main program methods
4. Define the agents as the users of the system and build the communication between these agents from the communication model

After this phase, the system becomes developed iteratively and validation is performed using MAVERICK. The 188 test cases extracted are listed in section 6.4.

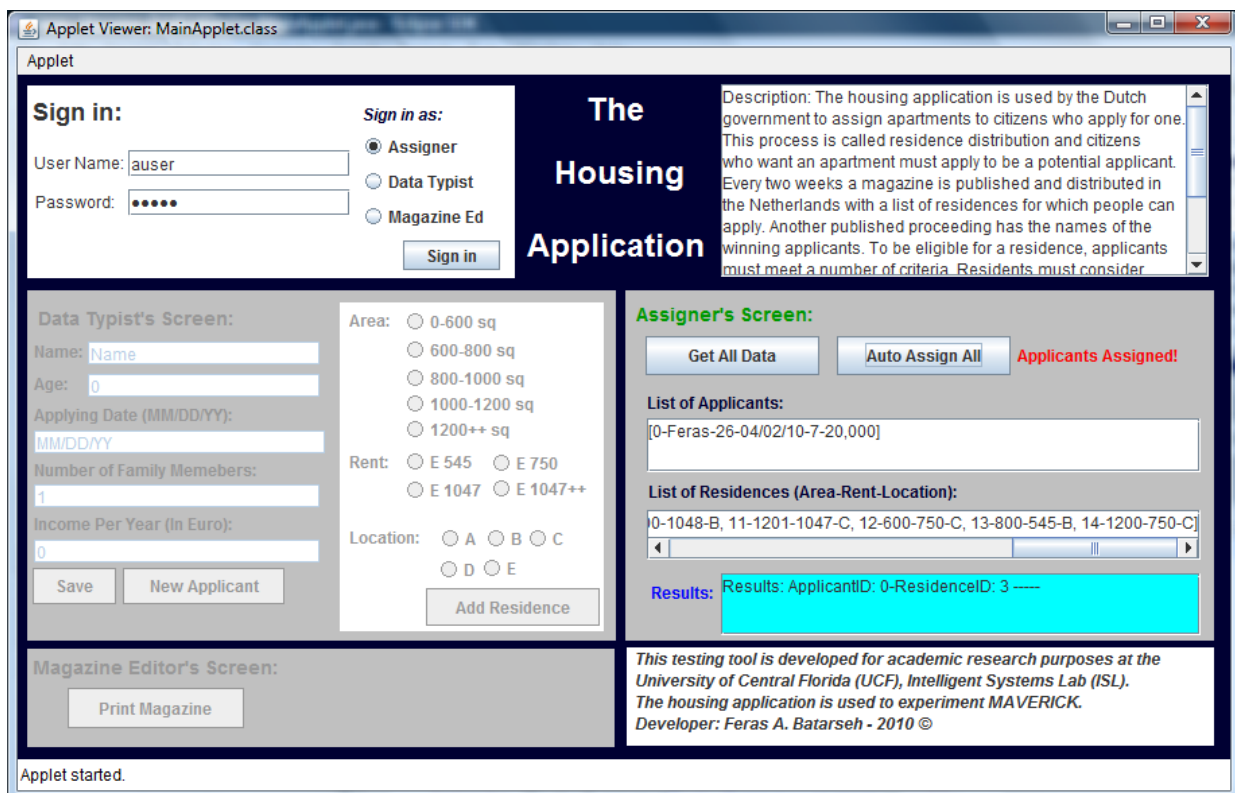


Figure 34: Assigner mode

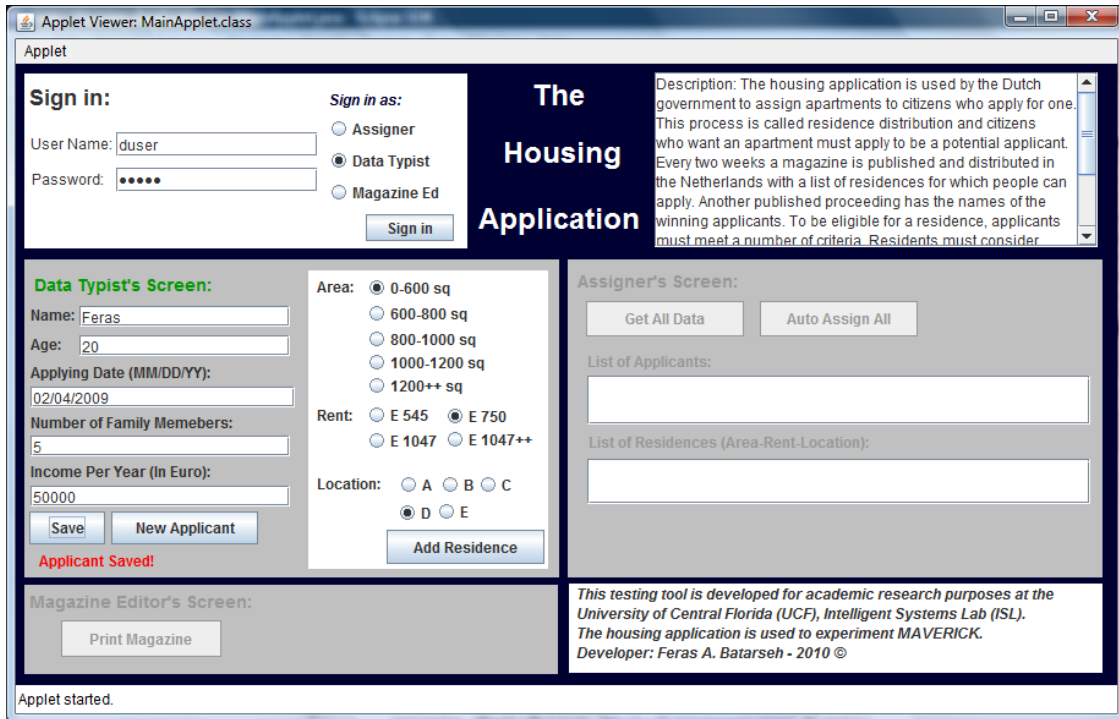


Figure 35: Data typist mode

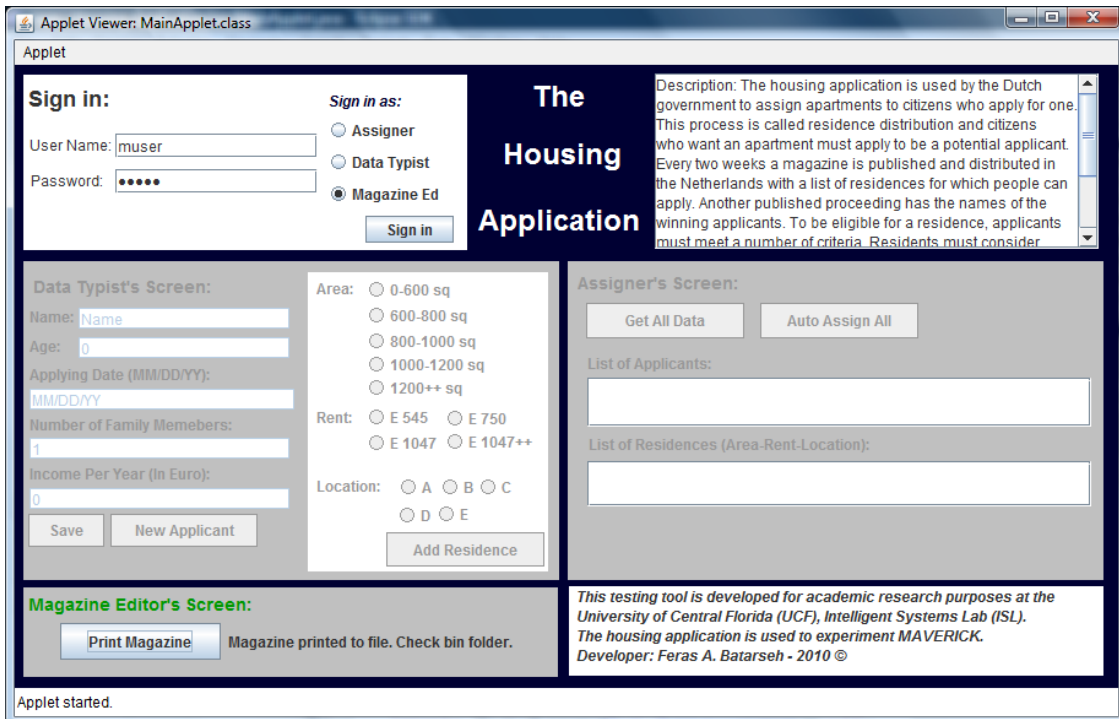


Figure 36: Magazine editor mode

The system built has two main parts: *Knowledge base* and *Main applet interface*. The *Knowledge base* Java file consists of the rules for the KBS. These rules reflect the assigning task, which is the main task of this system. They include the conditions for the applicants and the residences. The complete system code including the knowledge base is found in Appendix B.

The *Main applet interface* consists of the methods for controlling the interface and the access rules for the agents. The system interface is illustrated in three modes, the assigner mode, the magazine editor mode and the data typist mode. The modes are shown in figures 34, 35 and 36. The data typist enters all the applications (including all their personal information and residence needs) and the residences with the necessary information (area, location and rent). The assigner verifies all the assigned residences automatically and can review the outcome using his/her interface. Furthermore, the magazine editor uses all the information and publishes it automatically into a text file called: MAGAZINE. The evaluation of the time consumed to build the KBS and validate it using MAVERICK is included in Chapter 7. A brief description of the system is also included in the interface. The assigner, magazine editor and the data typist need to log in to activate their modes.

The CBTCR tool is used for selecting test cases and they are executed on the housing KBS manually. The information assigned for each test case is used to select the appropriate test cases for each iteration. Extracted test cases for the housing KBS are presented next.

6.4 The Extracted Test Cases

In this section, all the test cases for the housing KBS are listed with their parameters based on the test case format presented previously in Chapter 5. The system consists of three agents (magazine publisher, assigner and data typist) and three main interfaces. The extraction method introduced in Chapter 5 is used to create test cases from the worksheets and the diagrams.

The first set of test cases is extracted from worksheet OM3. In OM3 there are four tasks, each has an *agent* and a *knowledge asset* in the worksheet. Test cases extracted here are designed to test access of each agent to the needed knowledge asset and to its appropriate interface in the system (system screens/interfaces are illustrated in Figure 34, 35 and 36). Additionally, it is important to test whether agents have any access that they are not supposed to have. The first test case is:

1. Test case ID: *1*
2. CommonKADS model: *organizational model*
3. Input variables: *login information for the agent: assigner*
4. Test setup values: -
5. Test execution steps: *run system and login as assigner to magazine producing interface.*
6. Expected solution: *system does not allow access.*
7. System's solution: -
8. Local Importance: *2.75*
9. Number of execution times: *0*
10. Informal description: *this test case ensures that the assigner doesn't have access to the magazine production screen.*

The second test case for worksheet OM3 is:

1. Test case ID: *2*
2. CommonKADS model: *organizational model*
3. Input variables: *login information for the agent: assigner*
4. Test setup values: -

5. Test execution steps: *run system and login as assigner to residence assignment interface.*
6. Expected solution: *system allows access.*
7. System's solution: -
8. Local Importance: 2.75
9. Number of execution times: 0
10. Informal description: *this test case insures that the assigner has access to the residence assignment screen.*

The third test case is to ensure assigner has no access to the data typist screen:

1. Test case ID: 3
2. CommonKADS model: *organizational model*
3. Input variables: *login information for the agent: assigner*
4. Test setup values: -
5. Test execution steps: *run system and login as assigner to data entry interface.*
6. Expected solution: *system does not allow access.*
7. System's solution: -
8. Local Importance: 2.75
9. Number of execution times: 0
10. Informal description: *this test case insures that the assigner doesn't have access to the data entry screen.*

Each test case has an assigned local importance and it is calculated as described in Chapter 5. Local importance for the first three test cases is calculated as follows (this applies to the first three test cases):

$LI = \text{Average} (\text{dependency} + \text{domain importance} + \text{criticality} + \text{occurrence})$

$LI = 1(\text{OM is always 1}) + 0(\text{case not related to domain}) + 5(\text{significance in OM3 is 5}) + 5(\text{occurs every time assigner logs in}) = 11/4 = 2.75$

The first three test cases test the assigner access rights; test cases 4, 5 and 6 test the magazine producer access rights in a fashion similar to the first three cases. Likewise, test cases 7, 8 and 9 test the data typist access rights. These are not shown here but are included in Appendix A. The next three test cases test access to the necessary knowledge asset for all the agents.

1. Test case ID: *10*
2. CommonKADS model: *organizational model*
3. Input variables: *login information for the agent: assigner*
4. Test setup values: -
5. Test execution steps: *run system and login as assigner to residence assignment interface then click on the button that assigns residences to applicants.*
6. Expected solution: *residences are assigned to applicants. (the correctness of the assignment will be tested in the knowledge model)*
7. System's solution: -
8. Local Importance: $(1+5+5+4)/4 = 3.75$
9. Number of execution times: 0
10. Informal description: *this test case insures that the assigner can assign residences to applicants by having access to the priority calculator.*

Test cases 11 and 12 ensure access for the magazine producer and the data typist in the same fashion. Test cases extracted from the agent model worksheet AM1 are for the assigner access to

database (test case 13) and the priority calculator (test case 14). For the data typist access to the applications submitted by applicants (test case 15). See Appendix A for these test cases.

Test cases extracted from the task model's worksheet TM1 are in four groups (because we have four main tasks in the housing KBS). Test case number 16 ensures the order of actions when performing application assignment. As mentioned in Chapter 5, TM1 worksheet information is transformed into a number of test cases using the *input task*, *output task* and *timing* parts of TM1.

1. Test case ID: *16*
2. CommonKADS model: *Task model*
3. Input variables: *data about residence and an application*
4. Test setup values: -
5. Test execution steps: *run system and login as assigner to residence assignment interface then click on the button that assigns residences to applicants (check that data about applicant is displayed and residence is displayed, use calculator to manually check the correctness of the residence assignment).*
6. Expected solution: *residence is assigned using the application and the data about residence.*
7. System's solution: -
8. Local Importance: $(5+5+5+5)/4= 5$
9. Number of execution times: 0
10. Informal description: *this test case insures that the assignment is based on the right application and the right residence information.*

Test cases 17, 18, and 19 are testing the order of tasks execution for tasks 2, 3 and 4 from the task model. One of the system requirements is to save all the actions in a system log. Since this part deals with the main tasks, after each task, a test case is used to access the log and read it to see if the action was saved. Test cases 20, 21, 22 and 23 check the log for tasks 1, 2, 3 and 4 respectively. LI for them is $(0+0+0+2)/4=0.5$. See Appendix A for these test cases.

The knowledge model of this system consists of many diagrams but one worksheet with four scenarios. As discussed in Chapter 5, each scenario needs a test case. Test cases in the knowledge model are important ones as they test the knowledge. The first four scenarios are general ones; they test each task by running it in its general form. More detailed test cases are introduced from the diagrams for the rules and the essence of the knowledge base. Test case 24 is dedicated for task 1, it has the following parameters:

1. Test case ID: 24
2. CommonKADS model: *Knowledge model*
3. Input variables: *magazine produced reflects all the applications and the right information*
4. Test setup values: *all applications entered by the data typist*
5. Test execution steps: *run system and login as magazine producer to magazine interface then click on the button that publishes magazines (check that magazine reflects residences by checking their numbers).*
6. Expected solution: *magazine is published with the right number of residences and the right information.*
7. System's solution: -
8. Local Importance: $(5+5+5+5)/4= 5$
9. Number of execution times: 0

10. Informal description: *this test case insures that the magazine publishing is performed soundly.*

Test cases 25, 26 and 27 check for tasks 2, 3 and 4 respectively. Test cases 24-27 have the same local importance. As discussed previously in this chapter, there are three main transactions in the communication model. The *constraints* part of worksheet CM1 is used to generate a test case for each transaction. See Appendix A. Test case 28 is (for transaction 1):

1. Test case ID: 28
2. CommonKADS model: *Communication model*
3. Input variables: *attribute value pairs of an applicant and the residence*
4. Test setup values: -
5. Test execution steps: *run system and order application assessment.*
6. Expected solution: *assigner screen displays an order is being made. (this will ensure that the transaction went through).*
7. System's solution: -
8. Local Importance: $(5+4+3+2)/4= 3.5$
9. Number of execution times: 0
10. Informal description: *this test case insures that transaction 2 goes through smoothly.*

Test case 29 and 30 (Appendix A) test transactions 2 and 3.

Another test case is needed to check the correctness of the transformed data for each transaction. Test cases 31, 32 and 33 test that for transactions 1, 2 and 3 respectively. Any other transaction needed in the system would need more test cases such as the ones defined for transactions 1, 2 and 3.

Important test cases are introduced for the knowledge from the diagrams. As discussed previously, rules in the knowledge base should reflect the applicants' information to assign them the right residence. As mentioned previously, three parameters are involved: the number of family members, the area of the residence and the income of the applicant. Test case need to be defined to cover the three parameters and test the priority calculator's functionality (refer to Figures 29 and 32). Different numbers are generated for testing the knowledge base, e.g. number of family members = 3, area of residence: area C, income is: \$40,000/ year. Test cases generated here fall under the following categories: family members: 1, 2, 4, 6, 8 and more than 8. Annual income in Euros: 0-10,000, 10,000-40,000, 40,000-100,000, 100,000-250,000 and 250,000-500,000. Age: Less than 18, 18-35, 35-45, 45-60 and more than 60. These values are chosen by the expert not the knowledge engineer as they require knowledge in the domain.

The total of running all the combinations of all the cases is: 6 (income categories) *5 (age categories) *5 (number of family members categories) =150 test cases. For each test case, the knowledge engineer and the expert evaluate the results and decide if the test case failed or succeeded accordingly. Test case 34 for instance is:

1. Test case ID: *34*
2. CommonKADS model: *Knowledge model*
3. Input variables: *Family members=1, income= 1000, age = 18.*
4. Test setup values: -
5. Test execution steps: *run system and fill application with the input values and assign a residence (check for validity by the expert).*
6. Expected solution: *Applicant should be evaluated based on the expert knowledge, and get a residence.*

7. System's solution: -
8. Local Importance: $(4+5+5+5)/4= 4.75$
9. Number of execution times: 0
10. Informal description: *test the knowledge*.

Test cases 35 – 183 cover all the other variations of income, age and number of family members. The complete list of test cases is introduced in Appendix A of this dissertation.

Diagrams defined in the CommonKADS models are used for inspection and early stages of validation (analysis and context validation as defined in Chapter 5). Additionally, the diagrams are used to extract test cases. For each system a different number of diagrams are defined and thus different diagrams are used to extract test cases. Every entity in each diagram is used to extract test cases. An entity in a diagram is a process, data, agent, transaction, inferences...etc. In the housing KBS, 12 diagrams were drawn. All are presented in section 6.3 of this chapter. For this small system, only five test cases are extracted from the diagrams and the design model because most of the cases are covered by the worksheets. An example test case is given from this diagram (entity is: *magazine production*):

1. Test case ID: *184*
2. CommonKADS model: *Organizational model*
3. Input variables: *primary process (refer to Figure 23)*
4. Test setup values: -
5. Test execution steps: *run system and call magazine production function.*
6. Expected solution: *system will enter the magazine production mode.*
7. System's solution: -
8. Local Importance: $(1+5+5+3)/4= 3.5$

9. Number of execution times: 0

10. Informal description: *this test case deals with process from UML activity diagram.*

Other test cases are introduced similar to this test case to test the system performance and compare it to the diagrams presented in the models. Test cases 183 to 188 represent the diagram entities; these test cases test the knowledge base and the system. It is interesting to note here that a small system like the housing KBS generated 188 test cases. The CBTCR tool is needed to reduce the number of executed test cases to minimize time and effort.

All the test cases are entered into the excel spread sheet of the CBTCR tool. Thus, CBTCR tool selects the appropriate set of test cases at each iteration. Test cases are executed on the knowledge-based system. While refining the system, the knowledge engineer might add more test cases of newly added features or refined parts of the system. Furthermore, the knowledge engineer would remove test cases for removed tasks or transactions. Results of running the test cases are introduced in Chapter 7.

6.5 Summary

In this chapter the housing KBS is introduced including the CommonKADS models. The diagrams and the system's worksheets are presented. System development is introduced. Moreover, the 188 test cases extracted from the system are listed. Next chapter describes the experiments conducted and their results. The full code for the housing KBS is included in Appendix B of this dissertation. The full code for the CBTCR tool is available for research upon request from the developer.

CHAPTER 7: EXPERIMENTAL EVALUATION OF MAVERICK

This chapter describes the experiments conducted to evaluate MAVERICK. The experiments consist of three main sets. The first set of results is obtained by inserting errors into the housing KBS by the developer (the author). The second set of results is obtained by inserting errors into the system by seven objective human test subjects. The third experiment is comparing MAVERICK qualitatively to other validation methods for knowledge-based systems. The experimental setup, procedures and results for the three sets are discussed in this chapter. All the experiments are performed using the housing KBS, MAVERICK and the CBTCR software tool.

7.1 Introduction

The first step of this experimental research is to validate a knowledge-based system through CommonKADS using MAVERICK. The housing KBS in Chapter 6 is used for that purpose. After developing the system with CommonKADS, MAVERICK should be able to find errors and invalid aspects in the system. The performance of MAVERICK is then qualitatively compared to other two validation methods from literature. The detailed steps of the experiments in this chapter are:

- 1- Design the housing KBS using CommonKADS (introduce all the models and diagrams) (described in Chapter 6)
- 2- Extract test cases from the context, analysis and design models and all the diagrams. (see in Chapter 6)
- 3- Implement the housing KBS incrementally
- 4- Run test cases for each iteration using the CBTCR automated tool

- 5- Record time and effort required for each one of the previous steps and for every iteration of validation and development
- 6- After all the steps of development are complete, four types of errors were seeded in the system by the developer. Errors were seeded into: worksheets, diagrams, design and implementation
- 7- Using the CBTCR tool, run test cases and check whether the MAVERICK method detects the errors.
- 8- Guide seven human subjects on inserting additional different errors into the housing KBS.
- 9- Using the CBTCR tool, run test cases and check if the MAVERICK detects the errors.
- 10- Record the results and the resources' consumption
- 11- Qualitatively validate the system with other two validation methods (VIVA and EMBODY) to compare with MAVERICK
- 12- Record the results and the resources' consumption for the VIVA and EMBODY

Steps 1 through 5 are described in Chapter 6. Steps 6 and 7 are introduced in the section 7.2 of this chapter. Steps 8, 9 and 10 introduced in section 7.3 while steps 11 and 12 are introduced in section 7.4.

7.2 Experiment #1: Detection of Errors seeded by the developer

This section describes the first experiment and the first set of results. The purpose of this experiment is to measure the ability of the validation method to capture different types of errors in the system at different stages of development.

7.2.1 Experimental Setup

As introduced in Chapter 5, one of the stages of MAVERICK is inspection validation. It is performed manually on the system's worksheets and diagrams. This stage includes inspection of the CommonKADS models by the expert and the knowledge engineer to reduce the number of failed test cases and thus the number of errors at the iterative validation phase. Inspection validation is not performed in this experiment, because this experiment aims to measure the capability of CBTCR in detecting errors, not manually by the expert. To comprehensively test this system, errors had to be entered at different levels of abstraction. Errors are inserted in the system by the developer in four categories:

1. Errors inserted in the worksheets: these are errors in representing the domain and then in defining the problem in the worksheets.
2. Errors inserted in the diagrams: these errors relate to seeding wrong representations of the real world in UML and DFD diagrams.
3. Errors inserted in the design: these errors relate to the design model.
4. Errors inserted in implementation and knowledge base: inconsistencies in the implementation of the worksheets into code (modifying the code so it doesn't reflect requirements). Additionally, errors in the knowledge where they don't reflect the real world knowledge.

Twenty errors were inserted by the developer in these four categories. These errors are discussed in the experimental procedure in sub-section 7.2.2.

7.2.1.1 System validity before errors insertion

It is essential to ensure that the housing KBS was valid and clear from errors before the developer and the human subjects' errors were inserted. The worksheets and the diagrams introduced in the previous chapter were all based on a solid benchmark: the worksheets and content from the book [9]. The CommonKADS models were carefully checked manually by the author, section by section to ensure their correct representation of the system.

The housing KBS was built based on the worksheets. While developing the system, each transition was carefully checked and all the related test cases were executed. As a result, all the extracted test cases were executed on the system in multiple iterations. This was possible because the number of test cases was relatively small. The housing KBS is a small KBS and checking all the rules and the logic of the system was feasible in a timely manner. This made it possible to execute all the test cases. The test cases cover all the possible execution paths for the knowledge-based system. For bigger systems, the number of test cases could reach up to millions, however, which would make it impossible to execute all the possible paths. After executing all test cases, no error was found in the housing KBS.

7.2.2 Experimental Procedure

Upon validation of base version, the housing KBS system and models were purposely modified at all levels with incorrect code/knowledge. The following errors were inserted into the housing KBS at the four defined levels defined below:

1. Errors in the worksheets: in real life, resident assignment is performed by the assigner.

The first error seeded here is having OM3 have resident assignment by the data typist.

The second error in this category is improperly removing magazine production from the input tasks section in TM1 for task 2. The third error in this category is not permitting the assigner to have access to the assessment criteria or the knowledge. (Total errors = 3).

2. Errors in the diagrams: first error here was to remove the action “*evaluate*” from the inference structure of the knowledge and replace it with a wrong inference “*select*” (refer to Figure 27). The second seeded error removed the residence specific constraints from the diagram (refer to Figure 28). Third error for this category changed the relation in Figure 30 from *many to 1* to *1 to many*. (Total errors = 3).
3. Errors in the design: Error seeded in design changed the location of the seed class and linked it to the data entry class (refer to Figure 33). (Total errors = 1).
4. Errors in implementation and knowledge base: First error in implementation is seeding a number of random calculation errors in the priority calculator; this is done with nine different errors. Second error gives incorrect access to wrong agents by giving them same user names and same access roles. This is done for the three users, which results in three errors. The last error in this category is to call the function/object responsible for application assessment before calling the function responsible order application data. (Total errors = 13).

These 20 errors are described in Table 42. The errors were inserted using the Java Eclipse platform. The code files were opened and the code was manually modified. After the 20 errors were inserted, the CBTCR tool is used to select test cases for every validation iteration. Test cases are manually executed on the system to detect these errors.

7.2.2.1 Error detection

Errors are detected using the predefined set of test cases. As it is discussed in previous chapters, in the test cases format, each test case includes an “expected result” and a “system result”. After executing any test case, if the system’s result is different than the expected result, this indicates an error in the system.

7.2.2.2 Error location identification

Each test case is designed to test a very specific location in the knowledge-based system. In CommonKADS, the design model smoothly transforms the context and analysis models to implementation. This means that each of the models is visible in the design of the system and therefore, visible in the implementation. Having extracted the test cases from the models and because each test case is associated with a model, it is easy to realize the location of the error in the system. Results for this set of experiments are introduced in the next sub-section.

7.2.3 Experimental Results

In this sub-section, the system’s ability to find errors is evaluated, recorded and discussed. The 20 developer-seeded errors were searched for by the CBTCR tool and the results are recorded in Table 42.

Validation in MAVERICK is performed in iterations. Iterations are part of the spiral validation in MAVERICK. Spiral implementation is accompanied with iterative validation where a set of test cases is executed for each iteration. Five iterations were needed to allocate the errors. Eighteen (18) errors were detected from the 20 inserted errors at different iterations and with different test cases. Based on the formula presented in Chapter 5 for N, N is equal to 25 in the first iteration for this system.

Table 42: Results for errors inserted into the system by the developer

Error	Category	Description	Detected?	Iteration	Test case
1	Worksheets	OM3 have resident assignment by the data typist	Yes	1	18
2	Worksheets	Removing magazine production from the input tasks section in TM1 for task 2	Yes	1	17
3	Worksheets	Not permitting the assigner to have access to the assessment criteria	Yes	1	16
4	Diagrams	Remove action “ <i>evaluate</i> ” from the inference structure of the knowledge and replace it with a wrong inference “ <i>select</i> ”	No	N/A	N/A
5	Diagrams	Removing the residence specific constraints from the diagram	Yes	1	26
6	Diagrams	Changing the relation in Figure 30 from <i>many to 1</i> to <i>1 to many</i>	Yes	5	187
7	Design	Changing the location of the seed class and link it to the data entry class	No	N/A	N/A
8	KB	Wrong access to assigner	Yes	5	1
9	KB	Wrong access to data typist	Yes	3	185
10	KB	Wrong access to magazine editor	Yes	3	184
11	KB	Numeric error for assignments in rules. Family members rule.	Yes	1	16
12	KB	Numeric error for assignments in rules. Age rule.	Yes	2	64
13	KB	Numeric error for assignments in rules. Income rule.	Yes	1	40
14	KB	Numeric error for assignments in rules. Area rule.	Yes	1	17
15	KB	Numeric error for assignments in rules. Rent rule.	Yes	1	18
16	KB	Numeric error for assignments in rules. Family members’ categories.	Yes	1	18
17	KB	Numeric error for assignments in rules. Family members and age.	Yes	1	34
18	KB	Numeric error for assignments in rules. Family members and income.	Yes	1	34
19	KB	Numeric error for assignments in rules. Income and age.	Yes	1	34
20	Code	Call the function/object responsible for application assessment before calling the function responsible order application data	Yes	1	25

The CBTCR tool selects 25 test cases based on the CBTCR algorithm described in Chapter 5. $N = (\text{Number of test cases} - \text{Number of rules}) / \text{Project size}$

$N = (188 - 40) / 6 = 24.6 = 25$. During every iteration, the values of LI, GI, MW and result change. Test cases are flagged and sorted accordingly.

Appendix C shows the parameters for every validation iteration. The test cases are sorted and displayed in Appendix C in the following format:

'TestCaseID'-'CommonKADSMModel'-'LocalImportance'-'NumberofRuns'-'ModelWeight'-'GlobalImportance'-'Result'

The result is either 0 (test case failed), 1 (test case succeeded) or 2 (not executed). The model number is either 0, 1 (organization model), 2 (task model), 3 (agent model), 4 (knowledge model), 5 (communication model) or 6 (design model). After the test cases are sorted, the first 25 test cases are displayed to the user.

Test cases' IDs generated by the CBTCR tool for the first iteration were:

16, 17, 18, 19, 24, 25, 26, 27, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50

Test cases ID numbers generated for the second iteration were:

51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75

Test cases ID numbers generated for the third iteration were:

10, 11, 12, 13, 14, 15, 28, 29, 30, 31, 32, 33, 184, 185, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86

Test cases ID numbers generated for the fourth iteration were:

87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111

Test cases ID numbers generated for the fifth iteration were:

1, 2, 3, 4, 5, 6, 7, 8, 9, 187, 188, 186, 20, 21, 22, 23, 112, 113, 114, 115, 116, 117, 118, 119, 120

The listed test cases are executed on the system. Some test cases were able to detect errors (test cases listed in the table of results). After executing these five validation iterations, several behaviors for the CBTCR tool were observed and they are discussed next.

7.2.4 Discussions

To find the 20 errors, five validation iterations were performed. It is fair to expect that some of these errors would have been detected previously by the early manual inspection validation stages. For example, errors 7 and 4 were not found by MAVERICK; they might have been detected by the early design and the analysis validation phases. Nevertheless, thirteen errors were found after the first iteration. One error was found after the second iteration. Two errors were found after the third iteration. Finally, two errors were found after the fifth iteration. The reason that the majority number of errors were found in the first iteration is because the design of the test cases and the local importance factor. In the universal set of test cases, many of the test cases come from the knowledge model. All the test cases presenting the different variations of applicants to the system had a high local importance and thus were selected by the CBTCR tool. That is why it took five iterations to get to all the other models' test cases. It is worth considering that the two un-detected errors were seeded into the diagrams. These errors were not found in the diagrams and weren't directly reflected into the system rules. All errors seeded into the knowledge base were detected. The validity of the knowledge in the worksheets was a good substitute for the defects found in the diagrams. In the housing KBS, the number of test cases for models such as the communication model or the organizational model was low because of the nature of the system. The CBTCR tool works better if test cases for different models don't have a difference in their number, the less the difference is, the better CBTCR works. Most importantly, the number of iterations would have been lower if there have been an equal or nearly equal

number of test cases with high importance in all the models, not only KM or AM as it is the case in the housing KBS. The housing KBS doesn't have a large OM, CM or DM so the errors in these models were hard to reach because their global importance was low. CBTCR selects better test cases when models have less difference in the number of test cases in each of them.

On the other hand, the validation percentage displayed in the tool didn't reflect the real validity of the system during certain stages. This method treats all the models as being equal in importance. For instance, the validity of the communication model is set to be equal in importance to the validity of the organizational model even though in most cases the number of test cases for a model is more or less than the other models. The validation percentage in the tool only gives the average validity of the models. It is suspected that bigger systems could yield better results and would fit into CBTCR better; the more test cases there are the better this tool will work. Test case reduction depends on many factors. The design of the test cases importance and which test case is in which model highly affects the process and the results. In CBTCR, if a test case failed, the model from which this test case is from will have a very high importance in the next iteration. If there are many errors in many different models, it would take many iterations to find all the errors. The knowledge engineer should have the experience and competence to skip some test cases if they are very similar.

Another important thing to point out here is that some test cases help in identifying a number of errors. For example, a test case such as #16 represents core functionality and helps in finding many errors after the first iterations. This is because of the successful sorting of the test cases. The test case that addresses the problem specifically was only considered the one to detect the error, although in many cases, fixing one error would lead to finding more related errors in

the same scope. This was done with the errors seeded by the human subjects. Errors seeded by the human test subjects are introduced in the next section.

7.3 Experiment #2: Errors Seeded by the Human Test Subjects

The process of inserting errors into the system by objective human test subjects is the main body of evidence for this research. In the previous section, results were described for errors inserted by the developer. In statistics, a population of 30 is enough to get a valuable result from an experiment. In this experiment, 100 errors were inserted into the system. 80 errors were inserted by the human test subjects. These are in addition to the 20 errors discussed in the previous section. First, the experimental setup is discussed.

7.3.1 Experimental Setup

To experiment more with MAVERICK, different number of errors were inserted by seven human subjects. Seven copies of this housing KBS were created, one copy for each human subject. The seven human subjects are students at the University of Central Florida (UCF) with different majors and degrees. Each human test subject was given a color code.

1. Red: PhD student in Computer Engineering
2. Yellow: PhD student in Computer Engineering
3. Blue: PhD student in Computer Engineering
4. White: PhD student in Computer Engineering
5. Orange: Undergraduate student in Computer Science
6. Brown: Undergraduate student in Electrical Engineering
7. Grey: Undergraduate student in Computer Engineering

A description of the system was provided to each one of the human subjects. They were given a short presentation on how the housing KBS functions. They were shown the screens, the classes and the knowledge base. They insert errors based on the following guidelines:

1. Errors could be inserted anywhere in the system
2. Errors inserted could be anything from: redundant rules, inconsistent logic, invalid information, or any modification to the system that makes it not representative of the real world system.
3. The human subject can modify certain access rules to make them perform in a wrong fashion
4. The human subject can modify the priority calculator to make them perform in a wrong fashion
5. Errors were to be kept secret from the developer until they were discovered by MAVERICK or undiscovered if MAVERICK claims a valid system

These guidelines were defined and communicated to them by the developer. The human test subjects then inserted the errors into their copy of the system based on the procedure discussed next.

7.3.2 Experimental Procedure

The errors were inserted using the Java Eclipse platform. Each test subject inserted the errors in different places of their copy of the system. In all, 80 errors were inserted by the seven human test subjects. After errors are seeded, a report was written by each test subject describing her/his errors. This report was not provided to the developer until after the CBTCR tool was used to find the errors. The report was sent to a neutral third party and not revealed to the developer. There

was no indication on what errors were seeded before running test cases and attempting to detect the errors. Test cases were executed on each system modified by each human subject separately. Results were also recorded separately by the developer. Another report was written by the developer to identify the detected errors. Errors are presented next as they were reported by the test subjects.

The detailed list of inserted and detected errors per human subject is introduced next in the experimental results. Each error indicates the test case by which it was identified. Additionally, a full list of test cases for the housing KBS can be found in Appendix A. The legitimacy of the errors is discussed next.

7.3.2.1 The legitimacy of the errors

To fairly evaluate MAVERICK, it is important to eliminate errors that don't address validation. Not legitimate errors fall into two categories:

- 1. Verification errors**
- 2. Syntax errors**

Some errors were strictly verification errors and not validation errors, which is outside the scope of MAVERICK. MAVERICK was able to trace some of the verification errors inserted, but not all of them. MAVERICK is designed to check for errors that deal with representing the real world, as a validation method. Some users didn't have a clear understanding on the difference between validation and verification, thus, inserted verification defects. These errors were considered not legitimate. Verification errors are identified based on the definition given in Chapter 1. As the definition states, any error that deals with the consistency or the completeness of the knowledge base is a verification error. On the contrary, errors that present the real-world are considered validation errors. Some other errors were strictly syntax errors in the code, for

which the knowledge-based system wouldn't compile or run with. These errors are not considered validation errors either. They are run-time and syntax errors. In the following sections, each inserted error will be indicated to as legitimate (validation error) or not legitimate (verification or syntax error) based on what is discussed in this sub-section.

7.3.3 Experimental Results

Experimental results are introduced per test subject. For each test subject, the list of inserted and detected errors is discussed. The list of errors inserted was taken from the reports written by the human test subjects after inserting the errors. The line of code in which the error was inserted is indicated for all the errors. The complete code with errors locations is in Appendix B. The Part of code where errors were inserted is colored in green. The location of the error is indicated to by the color code and the error number. For instance: Red #4 (indicates to the location of error number 4 inserted by the test subject Red) and Blue #1 (indicates to the location of error number 1 inserted by the test subject Blue).

7.3.3.1 Errors inserted by test subject Red

Table 43 lists the errors inserted by test subject Red. Test subject red inserted nine errors of which four were detected by MAVERICK. Five errors were not detected. Out of these five errors, one error should have been caught, number 7. This error represents a logical change to the system which reflects the validity of the system. In error 7, AND was modified to OR, which effects the logic of the knowledge. For the other four undetected errors, test subject red inserted one verification error (#1) and three syntax error (#5, #2 and #3). These errors are considered not legitimate.

Table 43: Errors inserted by test subject Red

Error #	Line #	Description	Detected?
1	328	Set to true	No
2	515	Removed equal sign	No
3	517	Removed equal sign	No
4	524	Added a line to reset counter	Yes
5	526	Removed 'else' and changed greater than sign to '=' sign	No
6	91	Added code to set rent to -2	Yes
7	72 – 90	Changed logical “and” to logical “or”	No
8	67 - 69	Changed the area number	Yes
9	50-55	Family is randomly assigned a number from 0 to 50	Yes

Errors 4, 6, 8 and 9 were detected by MAVERICK. Errors detected for color code Red:

1- Adding an invalid rule “if income and age < 5 then rent equals -2”

Only one applicant is assigned for an assignment. This error is caused by an added rule: income<5 and age<5, that is always the case and rent is assigned to -2! This is an invalid value. (Detected by: Test case 16)

2- Adding the following to the code ”- -ACounter”

Adding - -ACounter after ACounter++. Applicants number not increasing. (Detected by running the application)

3- Number of family members defined by a random generator

Family members = 30 while is entered 3 or 4, error found in KB, random generator. (Detected by: Test case 16)

4- Swap two rules for family and Area in the knowledge base

Change in rules for family and Area. Family of 2 is getting a home for a family of 4 and vice versa. (Detected by: Test case 36)

7.3.3.2 Errors inserted by test subject Blue

Table 44 lists the errors inserted by test subject Blue. Test subject blue inserted six errors of which three were detected by MAVERICK. Three errors were not detected. Out of these three errors, one error should have been caught, number 2. This error represents a minor change in one of the rules to the system that reflects the validity of the system. In error 2, a ">=" sign was changed to ">". For the other two undetected errors, test subject blue inserted one verification error (#1) and one syntax error (#3). These errors are considered not legitimate.

Table 44: Errors inserted by test subject Blue

Error #	Line #	Description	Detected?
1	509	ggg was assigned 2 instead of 1 (1 was the correct value).	No
2	507	if (Integer.parseInt(jTextFieldfamily.getText()+"") >= 8)fff=5; was replaced by: if (Integer.parseInt (jTextFieldfamily .getText()+"") > 8)fff=5;	No
3	709	returnedResults = kb.maincontroller (ARRAYa,ARRAYr, ACounter,RCounter); was replaced by: returnedResults = kb.maincontroller(ARRAYa,ARRAYr, RCounter, ACounter);	No
4	82	if (Income == 3 && Age == 1) Rent = 3; was replaced by: if (Income == 1 && Age == 1) Rent = 3;	Yes
5	92	if (Rent == 0 && Area ==0){return false;} was replaced by: if (Rent == 0 && Area ==0 && Rent != 0){return false;}	Yes
6	85	if (Income == 3 && Age == 4) Rent = 3; was replaced by if (Income == 3 && Age < 4) Rent = 3;	Yes

Errors 4, 5 and 6 were detected by MAVERICK. Errors detected for color code Blue:

1- Array of applicants counter is swapped with array of residences counter

This caused a Compilation/Debug error. Array out of bounds: wrong counter for wrong array. (Detected by: Test case 10)

2- Wrong rule for income and rent

Income = 1 is getting rent = 3 (Detected by: Test case 40)

3- Changing the Boolean, *reassign()* is always called!

Secondary assignments are assigned to what should have been initial assignments.

Reason: re-assign function is called for all the cases for no reason. (Detected by: Test case 101)

7.3.3.3 Errors inserted by test subject Yellow

Table 45 lists the errors inserted by test subject Yellow. Test subject yellow inserted eight errors of which three were detected by MAVERICK. Five errors were not detected. Out of these five errors, one error should have been caught, number 4. This error represents a minor change in the logic flow of the system. The size of the array defines the number of residences, the error changed a variable I to 0 in that array. The full code with the error locations is included in Appendix B of this dissertation. This error affects the validity of the system. For the other four undetected errors, test subject yellow inserted one verification error (#8) and three syntax errors (#1, #5 and #6). These errors are considered not legitimate.

Table 45: Errors inserted by test subject Yellow

Error #	Line #	Description	Detected?
1	185	LISTofRESIDENTS.add("7-1000-545-D") changed to LISTofRESIDENTS.add("7-1000-545-d")	No
2	500	Set fff = (Integer.parseInt(jTextFieldfamily.getText()+""))- 5	Yes
3	521	Set iii = 2	Yes
4	705	ARRAYa[1][i]=SecretlistIncome.get(i); changed to ARRAYa[1][i]=SecretlistIncome.get(0);	No
5	106	R[0][i]=50; changed to R[0][i]=49; in ASSIGN function	No
6	115	R[0][i]=50; changed to R[0][i]=4; in ReASSIGN function	No
7	72	In RulesKB, added Area = Family % 2;	Yes
8	94	In RulesKB, added Income = Rent % Income;	No

On the other hand, errors 2, 3 and 7 were detected by MAVERICK. Errors detected for color code Yellow:

1- Error in rule, is assigning *family-5* to number of family members instead of *family*.

Family members' value is minus value if less than category 5 (Detected by: Test case 35)

2- Error is adding a rule for income to be in category 2 every time.

Income is always between 100000 and 500000! (Detected by: Test case 41)

3- Add modulo to Area and Income.

Area (is always a low value, due to modulo) and income (is assigned to zero although mostly is not a zero, when income and rent are same category) assignments are different that what they should be! Area = family %2 and income = rent %income (Detected by: test case 64 for area and test case 129 for income)

7.3.3.4 Errors inserted by test subject White

Table 46 lists the errors inserted by test subject White.

Table 46: Errors inserted by test subject White

Error #	Line #	Description	Detected?
1	69	knowledgebase.java changed area assignment to 3	Yes
2	102	change == AREA to !=AREA	Yes
3	112	changed == to < for AREA-1	Yes
4	28	changed < a to <=a	Yes
5	26	ag=0 changed to ag=2	Yes
6	38	j++ changed to j--	Yes
7	50	== changed to !=	Yes
8	72	&& changed to	Yes
9	77	&& changed to &	Yes
10	515	MainApplet – <= changed to >=	Yes
11	521	iii changed to fff	Yes
12	178	SecretlistArea.add(1) changed to SecretlistArea.add(2)	No
13	495	Removed initial ! from if statement	Yes
14	524	changed ++ to --	Yes
15	526	changed false to true	No
16	555	changed “auser” to “a2user”	Yes
17	592	changed == null to !=null	Yes
18	690	changed [3] to [4]	Yes
19	515	changed iii to ggg	Yes
20	555	changed true to false	No

Test subject white inserted twenty errors of which seventeen were detected by MAVERICK. Three errors were not detected. All of the three errors were verification errors (#12, #15 and #20). More description about the detected errors is presented next. Errors detected for color code White:

1- Modifying Area in a rule.

Error was found in knowledge base, when family is set to 4, Area is set to 3 instead of 4.
(Detected by: Test case 44 and 45)

2- Applicants assigned when they shouldn't be assigned.

When income is from category 1 or age from category one, && switched to || in rule.
(Detected by: Test case 38 and 39)

3- “ANDing” error

Symbol & instead of && in rule (Detected by: Running and compiling the system)

4- Area too big assigned for some applicants

R[1][i]<Area-1 instead of R[1][i]==Area-1

(Detected by: Test case 73)

5- Income not assigned correctly

When income <=100000, ggg=2 and not iii=2. This generates a wrong value for the age and the income (Detected by: Test case 112, 113)

6- Error in code

Acounter—instead of Acounter++, wrong counter value for arrays holding information about applicants (Detected by: Compiling the system)

7- Error in interface

Jlabelerror: appearing during wrong times. Error asking for missing fields while there isn't any! Label set to visible = true (Detected by: Test case 11)

8- Login for assigner changed

Change of username (Detected by: Test case 1)

9- Login for assigner changed

Change of radio-button for assigner (Detected by: Test case 1)

10- Array of applicants has more than expected when assigning residences to applicants.

ARRAYa=new int[4] instead of int [3] (Detected by: Test case 13)

11- Data typist info saved without the need to enter a name

Error cause by changing a rule from “!=” to “==” (Detected by: Test case 18)

12- Income not assigned correctly

When income >=100000 instead of <= (Detected by: Test case 112, 113)

13- Null pointer error

Null pointer exception, change of “JButtonNewApp != Null” instead of “JButtonNewApp == Null” (Detected by: Compiling the first time)

14- If age is not set, age is assigned to 2

Age=2 instead of the default value of 0 (Detected by: Test case 67)

15- Always a missing ID for an applicant

Sign “=” found in the loop condition where it shouldn't be. From class KB.java for variable “a” i<=a (Detected by: Test case 13)

16- Not all residents are assigned a residence

CONTROL variable != false instead of CONTROL == false (Detected by: Test case 67)

17- Wrong area assignments

$R[1][i] \neq \text{Area}$ instead of $R[1][i] == \text{Area}$ (Detected by: Test case 67)

7.3.3.5 Errors inserted by test subject Orange

Table 47 lists the errors inserted by test subject Orange.

Table 47: Errors inserted by test subject Orange

Error #	Line #	Description	Detected?
1	72	Changed “Age” to “Family”	Yes
2	73	Changed “Age” to “Family”	Yes
3	74	Changed “Age” to “Family”	Yes
4	75	Changed “Age” to “Family”	Yes
5	77	Changed new Rent value to 4	Yes
6	77	Changed new Rent value to 4	Yes
7	130	Set $R[0][i]$ to 100 instead of 50	Yes
8	131	Set $R[0][i]$ to 5 instead of 50	Yes
9	87	Changed Income conditional value to 5	Yes
10	88	Changed Income conditional value to 5	Yes
11	89	Changed Income conditional value to 5	Yes
12	90	Changed Income conditional value to 5	Yes
13	92	Changed first IF condition to $\text{Family} == 1$	No
14	84	Changed $\&\&$ to $\ $	Yes
15	84	Changed $\&\&$ to $\ $	Yes

Test subject orange inserted fifteen errors of which fourteen were detected by MAVERICK. One syntax error was not detected (#13). More description about the detected errors is presented next.

Table 49 shows the detailed results for all the human subjects. Errors detected for color code Orange:

1- Mixed assignments for different family members input

Change in rule of “*income&age*” to “*income&family*” (Detected by: Test case 13)

2- Mixed assignments for different family members input

Change in rule of “*income&age*” to “*income&family*” (Detected by: Test case 13)

3- Mixed assignments for different family members input

Change in rule of “*income&age*” to “*income&family*” (Detected by: Test case 13)

4- Mixed assignments for different family members input

Change in rule of “*income&age*” to “*income&family*” (Detected by: Test case 13)

Note: Errors 1, 2, 3 and 4 are the same because the error is repeated 4 times in the knowledge base.

5- Change of rule for rent assignments

Change in rent assignments from 2 to 4 for income 2 and age category 1. Change of rule. (Detected by: Test case 40)

6- Change of rule for rent assignments

Change in rent assignments from 2 to 4 for income 2 and age category 4. Change of rule. (Detected by: Test case 64)

7- Age factor working in a wrong manner

Rule change: Income =3 || Age =1, 4 to && (Detected by: Test case 58)

8- Age factor working in a wrong manner

Rule change: Income =4 || Age =1, 4 to && (Detected by: Test case 55)

9- Income is set to group 5 which doesn't exist

When income= 4 and age = 1 rent = 4, income is changed to 5. (Detected by: Test case 2)

10- Income is set to group 5 which doesn't exist

When income= 4 and age = 2 rent = 4, income is changed to 5. (Detected by: Test case 2)

11- Income is set to group 5 which doesn't exist

When income= 4 and age = 3 rent = 4, income is changed to 5. (Detected by: Test case 2)

12- Income is set to group 5 which doesn't exist

When income= 4 and age = 4 rent = 4, income is changed to 5. (Detected by: Test case 2)

13- A residence is assigned multiple applicants

Limiting factor changed in code: R[0][i] = 50 switched to = 100. (Detected by: Test case 2)

14- A residence is assigned multiple applicants

Limiting factor changed in the code: R[0][i] = 50 switched to = 5. (Detected by: Test case 2)

7.3.3.6 Errors inserted by test subject Brown

Table 48 lists the errors inserted by test subject Brown.

Table 48: Errors inserted by test subject Brown

Error #	Line #	Description	Detected?
1	504	fff changed to ggg in main loop if statement #5	Yes
2	506	Eliminated if statement #6 from main loop	No
3	514	iii = 2 changed to iii = 5 on second if statement in main loop	Yes
4	68	Family changed to Rent in knowledge base 3 rd if statement	Yes
5	74	Rent changed to family in 8 th if statement	Yes
6	90	&& changed to in last if statement	Yes
7	127	Area+4 changed to area+7 in reassign function	Yes
8	83	== changed to != in 15 th if statement	Yes
9	511	<60 changed to <50 in main loop else if	Yes
10	516	<500000 changed to <550000 in main	Yes
11	68	3 changed to 8 in third if statement main loop	Yes
12	122	Rent changed to family in reassign function	Yes

Test subject brown inserted twelve errors of which eleven were detected by MAVERICK. One verification error was not detected (#2). More description about the detected errors for test human subject brown is presented next. Errors detected for color code Brown:

1- Rule changes in family and area relation assignments

Family changed to “Rent” for area =3 and family =3 (Detected by: Test case 48)

2- Rule changes in family and area relation assignments

Rent changed to “Family” for income =1 and age =2 (Detected by: Test case 64)

3- Rule changes in income and age relation assignments

Age == 3 is changed to *age* !=3 for rent =4 (Detected by: Test case 65)

4- More assignments to applicants (redundant applicants)

Rent && area changed to rent ||area (Detected by: Test case 16)

5- Wrong area assignment for applicants

Area+4 changed to area+7 (Detected by: Test case 16)

6- Wrong family member number assignment for applicants

Rent-5 changed to family-5 (Detected by: Test case 16)

7- Some families not assigned residences (if family have 6 members)

Missing rule, if *family* == 6 → *fff*=4 (Detected by: Test case 55)

8- Age entered = 57 is not getting right assignment

Age <50 instead of age<60 (Detected by: Test case82)

9- Family members = 8 is dealt with as = 2

Change in family members rule for category = 3 (Detected by: Test case82)

10- Income is set to a wrong category = 5!

Change iii to 5 instead of 2, category 5 doesn't exist (Detected by: Test case88)

11- Error for income <500000

Change of income from <500000 to <550000 (Detected by: Test case162)

7.3.3.7 Errors inserted by test subject Grey

Table 49 lists the errors inserted by test subject Grey. Test subject grey inserted ten errors of which eight were detected by MAVERICK. Two syntax errors were not detected (#8 and #1).

Table 49: Errors inserted by test subject Grey

Error #	Line #	Description	Detected?
1	47	In RULES KB added conditional statement	No
2	103	In KnowledgeBase created if statement for first assign	Yes
3	73	Changed condition (Income == 1 && Age == 2)	Yes
4	112	In ASSIGN, changed two i to i-1	Yes
5	84	For (Income == 3 && Age == 3) rent = 3	Yes
6	65	Age = ag-1	Yes
7	123	In REASSIGN, r has been changed to r*2	Yes
8	112	In ASSIGN, started for loop at i=1 instead of i=0	No
9	101	Removed case: if (Income == 4 && Age == 2) Rent = 4;	Yes
10	115	In ASSIGN, changed line RESULTS[1][app]=i; to RESULTS[1][app/2]=i;	Yes

More description about the detected errors is presented next. Furthermore, all details about the errors inserted are displayed in Tables 50 and 51. Errors detected for color code Grey:

1- Out of bounds exception

R changed to r * 2 (Detected by: Test case16)

2- Out of bounds exception

Assign is always called, deleted if-statement (Detected by: Test case16)

3- Age Category always wrong

Age = Age-1! (Detected by: Test case16)

4- When Age is in Category 3, applicant not assigned correctly

Income = 1, age = 2 → rent =2, age is switched to be equal to 3 (Detected by: Test case144)

5- System doesn't handle age for income = 4

Missing rule, income = 4 and age = 2, rent = 4 (Detected by: Test case159)

6- Syntax error: assigns even if when rent = 1 and area =0

Rent = 1 and area = 0 is added to KB (Detected by: Test case16)

7- Loop out of bounds exception

I-1 instead of I for assigner's loop controller (Detected by: Test case16)

8- Wrong assignments for applicants

Applicant 2 is getting what 4 should get; applicant 4 is getting what applicant 8 should get. Error in code app/2 instead of app in the knowledge base (Detected by: Test case16)

The results show that MAVERICK detected 60 out of 80 errors by the test subjects and 18 out of 20 errors by the developer. This leads to a total of 78 out of 100 (**78%**). All the errors were categorized into five main categories (Figure 37 shows the number of errors inserted for each category):

1. Errors that deal with the users of the system, their access rules and roles
2. Errors that manipulate the knowledge
3. Errors in the interface of the system and the code
4. Errors related to the CommonKADS models
5. Errors that are not legitimate

Out of the 80 inserted errors, 60 of them were detected by MAVERICK.

Overall, a total of 22 errors were not detected, 20 were undetected from the test subject errors and 2 from the 20 developer errors. 17 of the 22 undetected errors were deemed not legitimate (17/22 errors). The 17 errors were undetected because of their legitimacy (discussed earlier) in two categories:

1- Some errors were verification errors (7/17 not legitimate verification errors).

2- Some of the errors were syntax errors (10/17 not legitimate syntax errors).

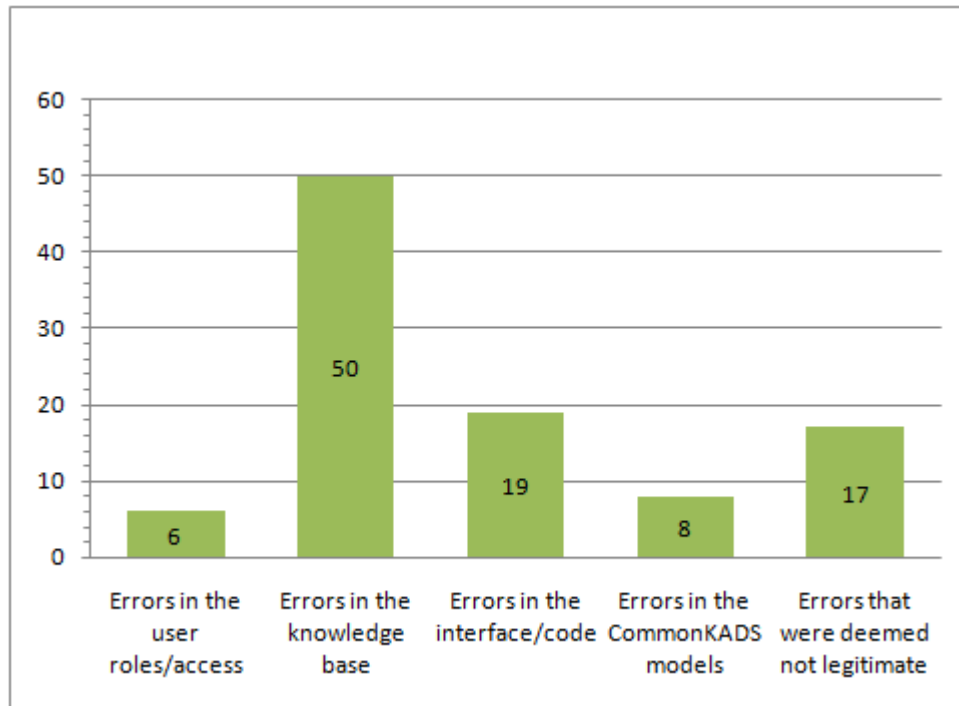


Figure 37: Bar chart for errors' types inserted into the system

Nevertheless, out of the 22 undetected errors, five were legitimate and should have been detected. Apart from the two undetected errors inserted by the developer, error number 7 from the Red human subject was not detected. The error was changing an AND to an OR in a rule. Additionally, error 2 from the Blue human subject was not detected, a \geq sign was changed to $>$. Finally, error number 4 from the Yellow user was not detected either, it was a misplaced function call. The three errors were very minor and hard to get to due to their low level of occurrence. The tool deviated towards more important parts of the knowledge-based system and couldn't track these validation errors. On the other hand, 17 errors were undetected and not legitimate, these

errors are categorized as either verification errors or syntax errors. Syntax errors are those that were inserted that would create a runtime or a compilation error. Verification errors are identified based on the definition provided in Chapter 1: “Verification is the process of ensuring that the intelligent system conforms to specifications, and its knowledge base is consistent and complete within itself” the following list discusses the reasoning behind this categorization for every error:

- 1) Verification error (#1) inserted by test subject Red: an “if statement” was set to true, which results in a true outcome every time it is checked. The “if statement” affects the logic of the program and its flow. This error results in an inconsistent outcome in the execution of the system. By having this error, the “false” case of the “if statement” doesn’t exist anymore resulting in incompleteness. Hence, this is a verification error.
- 2) Syntax error (#5) inserted by test subject Red: this error includes removing an “else” from the syntax of the code, this resulted in an unreachable code and a compilation error. This is a Java syntax error.
- 3) Syntax error (#2) inserted by test subject Red: an equal sign was removed from the initialization of the class. This created a runtime error when the object is reached in runtime. This error is categorized as a syntax error.
- 4) Syntax error (#3) inserted by test subject Red: similar to the previous error, an equal sign was removed from an initialize of an object in the main class, thus created a runtime error when this part of the code is reached while executing the system.
- 5) Verification error (#1) inserted by test subject Blue: the variable “ggg” represents the age categories of the applicant in the housing KBS. In the specifications of this system, it is

required to have four categories for ages 1, 2, 3 and 4 in the KBS. In this error, category 1 is deleted, resulting in a mismatch with the specifications, thus, a verification error.

- 6) Syntax error (#3) inserted by test subject Blue: this error consisted of switching the value of two counters that are passed to the method “maincontroller” in the class KB. The method maincontroller uses these counters for array sizes and loop counters. Therefore, switching the two values will result in an “array is out of bound” exception, which is a syntax error.
- 7) Verification error (#8) inserted by test subject Yellow: the variable “income” represents the income of the applicant in the housing KBS. In the specifications of this system, income is used to calculate the rent that the applicant is capable of paying. This value is compared with the rent, not the “rent % income” as the error presented. This results in a mismatch with the specifications, thus, a verification error.
- 8) Syntax error (#1) inserted by test subject Yellow: In this error, a new resident was added with rent area D. This was changed from D to d. The test subject reported this as an error, while it is not. The information about the applicants is not case sensitive. Therefore, not detecting this error doesn’t reflect validation; it is categorized under not legitimate syntax errors.
- 9) Syntax error (#5) inserted by test subject Yellow: In the code, the number 50 is used to represent an empty cell in the array; this is a coding style used to prevent null values in arrays and lists. The test subject inserted 49 instead of 50 in the array, which is not preferable, but is not a real error and most importantly, not a validation error. Therefore, this is categorized as a syntax error.

- 10) Syntax error (#6) inserted by test subject Yellow: This error is similar to the previous one. This time, the test subject inserted 4 instead of 50 in the array, which is not a real error. Therefore, this is categorized as a syntax error.
- 11) Verification error (#12) inserted by test subject White: this error modifies a variable passed to a method that controls the list of applicants. This list starts with 1 and should all be checked. If it is checked starting from 2, the process will be incomplete. This affects the completeness of the system; therefore, it is a verification error.
- 12) Verification error (#15) inserted by test subject White: this error changes the initial value assigned to a Boolean number/controller of the main class from “true” to “false”. This affects inconsistency with the specifications regarding the applicants’ assignment process. This error results in an inconsistent outcome in the execution of the class, thus is a verification error.
- 13) Verification error (#20) inserted by test subject White: this error is contrary to the previous one. The test subject changed a “false” to “true” in an initial Boolean value of the main class. This has the same effect of the previous error, thus is a verification error.
- 14) Syntax error (#13) inserted by test subject Orange: this error consists of removing an “if statement” that controls the value “family” in the application to “family == 1”. This indicated to a clear variance with the specifications which describe that number of family members might be 1, 2, 3, 4, 5 or more. This error assumes the number of family member is always 1, which is an obvious verification error.
- 15) Verification error (#2) inserted by test subject Brown: this error eliminates a rule from the rule base in class KB.java. This is an example of what is presented in the verification

definition introduced in this dissertation: the knowledge base needs to complete within itself. Eliminating a rule violates completeness; therefore, it is a verification error.

16) Syntax error (#1) inserted by test subject Grey: this error consists of adding a useless statement to the code. This statement doesn't affect the system in any shape or form. It is not relevant to the flow of the system and is unreachable. This error is not inserted into the knowledge base; therefore it is considered a syntax error.

17) Syntax error (#8) inserted by test subject Grey: this error proposes starting the loop at 1 and not 0. This would not let the "while loop" check for the first item in the array and therefore skip the first applicant's information. For the loop to be complete, it should check all the items. However, this error doesn't affect the KB or the validity of the system but the loop itself, therefore it is considered a syntax error.

Syntax errors were detectable because they result in an obvious compilation or runtime error that is not executable by the system; therefore, the system would halt or crash.

The verification errors were difficult to detect because MAVERICK doesn't look for them. These errors could be detected if a process was incorporated into MAVERICK that is performed prior to validation. This verification process would check for inconsistency and incompleteness and would compare the system against its specifications. If this was done, the verification errors would have been detected.

Figure 38 shows the errors inserted by all the users and the number of errors detected for each human subject.

The overall percentage of success for MAVERICK is **78%**. The undetected errors were due to three reasons discussed previously. However, only five of the errors were legitimate ones and should have been detected. This relation is illustrated in the pie chart (Figure 40).

If one only considers the legitimate errors, the numbers significantly look better for MAVERICK. Figure 41 shows only the legitimate errors inserted by the human subjects. Out of the 100 errors, 83 were legitimate errors and out of 83 legitimate errors, 78 were detected. This leads to a different percentage for MAVERICK's success in validating the system: $93.9\% \approx \mathbf{94\%}$.

Based on the above discussions, the **94%** percentage is used as the official percentage of this research work, not the **78%**. One might think that a verified system is not necessarily a valid system, which means that if the system has no verification or syntax errors, it doesn't mean that it meets the real world requirements, thus, not valid. On the other hand, a valid system also needs to be verified. Although it is true that a verified system is not a valid system and that a valid system is a verified system, MAVERICK is strictly a validation method and it is not fair to include all the different errors in the formula. Furthermore, although all errors are problematic in the real software development world, this research goal is to address validation only. That's why 94% is declared as the main result.

Figure 39 illustrates the errors that were detected and the ones that MAVERICK missed, from which some were legitimate and some weren't. Tables 50 and 51 introduce a summary of the specific results of the errors per human subject, based on the errors categories and the types of errors that are not legitimate.

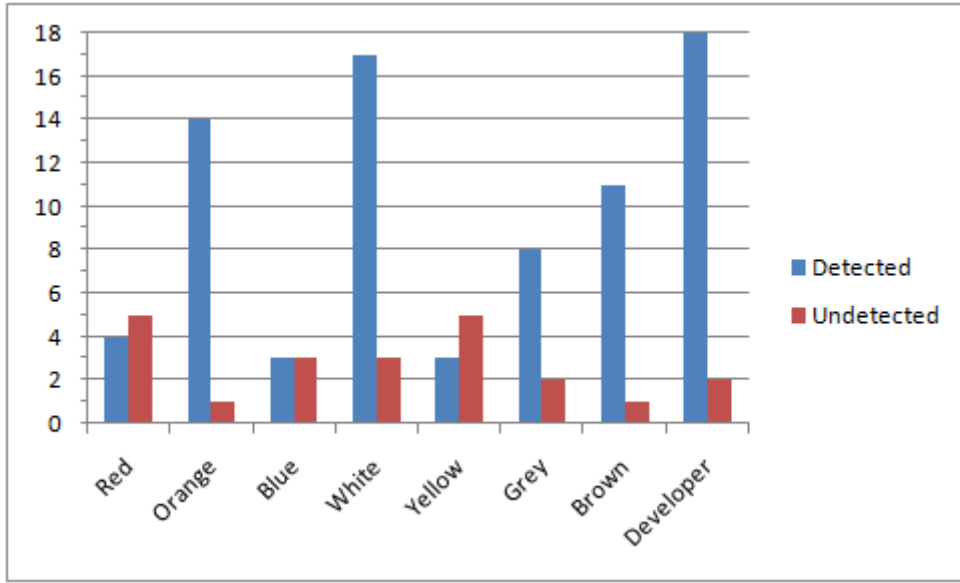


Figure 38: Bar chart for errors inserted by all human test subjects

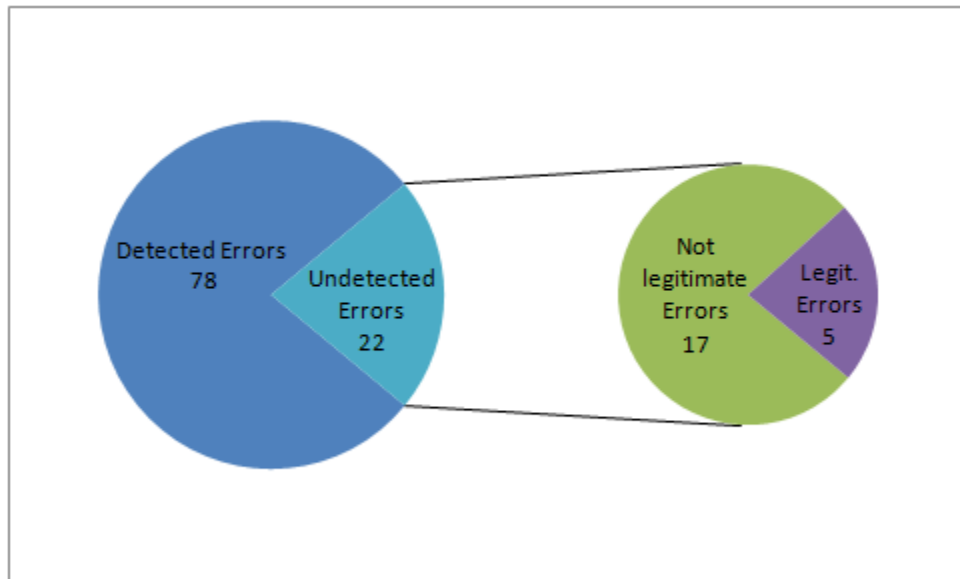


Figure 39: Pie charts for MAVERICK errors results

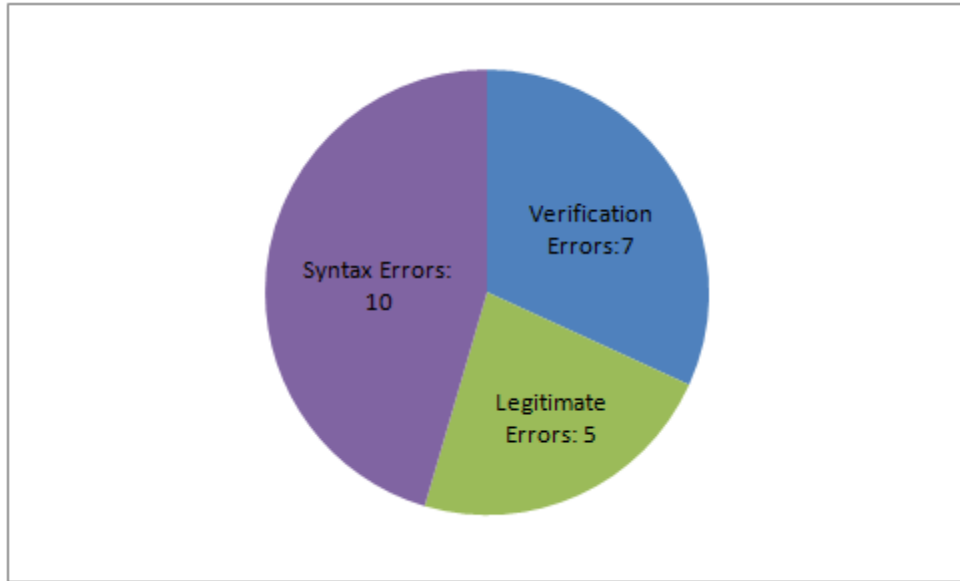


Figure 40: Pie chart for the 22 undetected errors

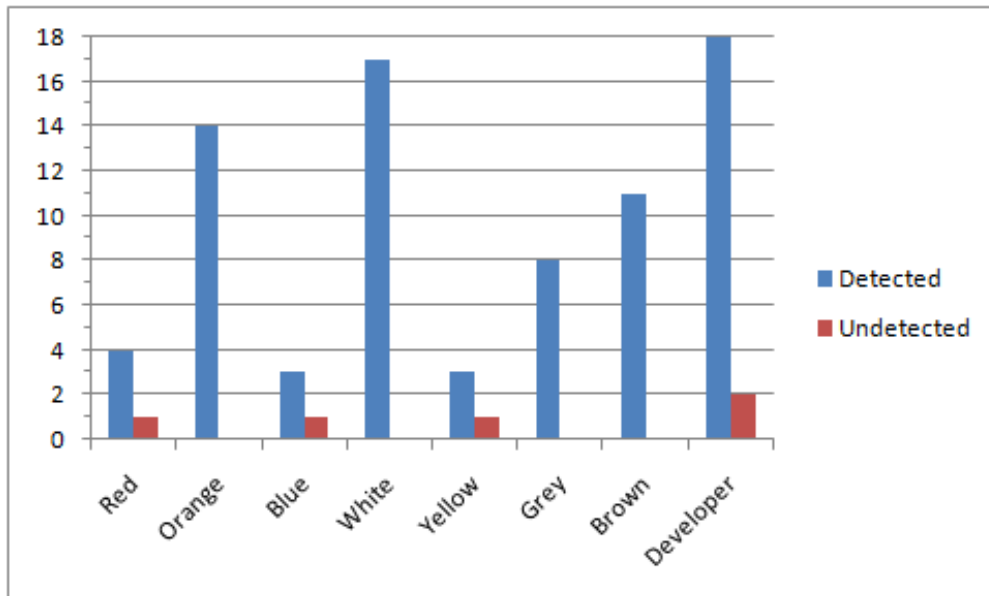


Figure 41: Bar chart for the legitimate errors inserted by all the users

Table 50: Not legitimate vs. legitimate errors inserted by human subjects

Human Subject	Detected/Total	Legitimate Errors	Syntax Errors	Verification Errors
Red	4/9	1	3	1
Blue	3/6	1	1	1
Yellow	3/8	1	3	1
White	17/20	0	0	3
Orange	14/15	0	1	0
Brown	11/12	0	0	1
Grey	8/10	0	2	0
Developer	18/20	2	0	0
<i>Total</i>	78/100	5	10	7

Table 51: Error categories inserted by the human subjects

Human Subject	Users/Roles	Knowledge	Interface/code	CommonKADS	Not legitimate
Red	0	3	2	0	4
Blue	0	2	2	0	2
Yellow	0	2	2	0	4
White	3	5	8	1	3
Orange	0	14	0	0	1
Brown	0	9	2	0	1
Grey	0	6	2	0	2
Developer	3	9	1	7	0
<i>Total</i>	6	50	19	8	17

7.3.4 Discussions (Statistical Analysis of Results)

Although the official result for MAVERICK in detecting errors is **94%**, in this discussion two results are considered, the **78%** and the **94%**.

For the experiments in this dissertation, each inserted error is either detected or undetected. To measure an actual confidence interval, a statistical interval analysis for the statistical population was performed. This analysis yields the probability of the occurrence of these results, regardless of the number of times the experiment is repeated. In this dissertation's scope, the confidence interval will indicate that if there were more errors, more human subjects

or more experiments, a percentage will be obtained within the interval. This experiment has two main outcomes for every error, detected or undetected. This illustrates its binomial nature. For such binomial problems where there are two options, it is suitable to use binomial proportion confidence interval.

Binomial proportion confidence interval uses the statistical sample provided and generalizes it. It allows for a sampling error which defines the outer bounds of the confidence interval. There exist several formulas for binomial proportion confidence intervals, but they all require binomial distribution. The normal approximation interval will be used here, the formula for that is:

$$p \pm Z_{1-\alpha/2} \sqrt{\frac{p(1-p)}{n}}$$

Where p is the probability, n the sample size and α is the sampling error rate. Alpha ranges from 0-1. The sampling error comes from using a sample of the data rather than all the data. The sampling error rate in this experiment is 0.05 (a generic sampling error for experiments of this nature), which makes our desirable percentage 95%. This makes $Z_{1-\alpha/2}$ equal to $Z_{0.975}$. To perform this analysis, the errors need to be grouped and we need to randomly selected errors and their outcome, 0 or 1, but in this experiment this step is done and we have an absolute percentage which is **78%** or **94%**. The 78% is discussed first.

For the first percentage, P is 0.78, n is 100, which represents the 100 errors and $Z_{0.975}$ is equal to 1.959964, which could be rounded to 1.96. The confidence interval formulas for this percentage are:

$$0.78 + 1.96 \sqrt{0.78(1 - 0.78) \div 100} \text{ And } 0.78 - 1.96 \sqrt{0.78(1 - 0.78) \div 100}$$

$$0.78 + 0.0812 \text{ and } 0.78 - 0.0812 = 0.8612 \text{ and } 0.6988$$

The confidence interval is 70% to 86%. This means that if the errors insertion experiment was repeated on any system similar to the housing KBS and with any number of errors, 95% (alpha) of the time the percentage of errors detected will be **70%** to **86%**.

More importantly, for the second percentage, P is 0.94, n is 83, which represents the 83 legitimate errors and $Z_{0.975}$ is equal to 1.959964, which could be rounded to 1.96. The official confidence interval formulas for MAVERICK are:

$$0.94 + 1.96 \sqrt{0.94(1 - 0.94) \div 83} \text{ And } 0.94 - 1.96 \sqrt{0.94(1 - 0.94) \div 83}$$

$$0.94 + 0.0510 \text{ and } 0.94 - 0.0510 = 0.991 \text{ and } 0.889$$

The official confidence interval is 89% to 99%.

This result means that if the errors insertion experiment was repeated on any system similar to the housing KBS and with any number of errors, 95% (alpha) of the time the percentage of errors detected will be **89%** to **99%**. This successful result gives us the interval of results for which this experiment can get and an assurance about the results of the experiment. This analysis concludes that this work provides confidence up to 99% that the **94%** error detection rate will be achieved when MAVERICK is used in any other similar KBS.

7.4 Experiment #3: Comparison of MAVERICK to Other Methods

This section introduces the last set of experiments for MAVERICK. This experiment is only qualitative. This section introduces two methods used for comparison purposes in the experiment #3. The two methods are used to validate the housing KBS and their consumption of resources is recorded and compared to the MAVERICK validation method results.

7.4.1 Experimental Setup

After inspecting multiple validation methods and considering many candidates, many of the methods were impossible to use to validate the housing KBS because of different reasons. Some of the methods strictly required their developed tools to be used and the tools were not available. Other methods had no useful guidance on how to implement the validation method on a knowledge-based system. Yet other methods were only useable within a specific domain, such as validation methods specialized in military or medical knowledge-based systems. Finally, some methods required test cases extraction; the process of extracting test cases couldn't be repeated without bias towards the set for the MAVERICK method. Therefore, all these methods were ruled out from consideration.

Two methods, however, were found suitable and had no constraints for using them in this validation experiment. The two methods are successful methods. They have been used before for validating knowledge-based systems and they use two different validation approaches. The two methods are VIVA [28] and EMBODY [53]. Both methods were reviewed previously in chapter 2. VIVA is a life-cycle independent validation method and EMBODY validates the system by embedding knowledge validation into the knowledge acquisition process.

To use EMBODY, a flow chart of the knowledge needs to be defined, as EMBODY uses a graphical representation of the knowledge for validation. EMBODY represents the real world knowledge and ensures that it meets the real world case, which is a sound definition of validation. In the EMBODY method, while the knowledge is being extracted, a model is being built that would be converted to the knowledge base. The idea in EMBODY is that if the model is valid then the knowledge base is. This is not perfectly sound, because of some errors occurring in the transition process.

VIVA is based on establishing a relation between the knowledge model, design and the code/KB without using a lifecycle. VIVA uses transformational links for the transformation between the knowledge model to the code or the design and between the design and the code and vice versa. VIVA uses structural links to link between objects within the knowledge model or the design. This structure is built to present all the contents of the knowledge-based system development process. After this is done, validation specifications are derived such as: correctness, completeness and existence. For validation, the structure of the system defined and the specifications are compared and mismatches are to be revised.

Both VIVA and EMBODY do not perform validation directly on the system, but rather on a model. Errors might occur when the model is transferred into the knowledge-based systems. In MAVERICK, validation is performed on the system as well as on the models. Furthermore, validation is performed on the transition process itself, which is incorporated in the CommonKADS design model. Additionally, both VIVA and EMBODY don't refine the system if errors are found, while MAVERICK does. VIVA and EMBODY are applied to the housing KBS; procedures and results are introduced in the next two sections.

7.4.2 Experimental Procedure

Comparing MAVERICK against two other validation methods is mostly qualitative and only numeric in terms of time consumed. Only the time consumed is compared and not the validity or errors insertion because:

1. MAVERICK used CommonKADS to regenerate the errors and validate. Comparison to VIVA and EMBODY is unfair because CommonKADS was not used in their cases.

2. Many errors were inserted in the CommonKADS models and diagrams, which is not possible for any other method than MAVERICK.
3. MAVERICK is incremental and is based on a lifecycle. VIVA and EMBODY are not.
4. MAVERICK was employed using the CBTCR tool. Tools for VIVA and EMBODY were not found to be publicly available.
5. The test application - the housing KBS - was built after generating the CommonKADS models and diagrams. This would have been a different process if VIVA or EMBODY had been used.

Any validation method selected would have the same limitations except possibly for reasons 3 and 4 above. Nevertheless reasons 1, 2 and 5 would apply to any other method because no validation method has ever used or been embedded into CommonKADS.

Based on the mentioned reasons, the housing KBS is validated using VIVA and EMBODY but only the time consumed by the developer is recorded and compared. No errors were inserted in the housing KBS for this experiment. This experiment was based only on following the processes defined in VIVA and EMBODY and recording time for that. The results for the three methods are discussed in the next section.

7.4.3 Experimental Results

The results of the experiments are illustrated in following three tables. In tables 52, 53 and 54, a star (*) is added to any step that is considered a validation step. Times consumed for validation in the three methods are introduced separately by summing the steps indicated to with a star (*). First, the steps for MAVERICK are presented in Table 52, VIVA is in Table 53 and EMBODY is in Table 54. The results for each method follow each table.

Table 52: General time consumed for MAVERICK

Stage name/ description	Time consumed
Context models development	6 hours
*Context Test Cases Extraction	4 hours
Analysis models development	5 hours
*Analysis Test Cases Extraction	5 hours
*Analysis validation	2 hours
Artifact models development	4 hours
*Design Test Cases Extraction	3 hours
*Design validation	1 hours
*Assign local importance for each test case	4 hours
*Fill test cases into the sheet of the CBTCR tool	11 hours
*Set all models' weights/assurance to 5	0 hours (Autonomous process)
*Calculate global importance and re-order	0 hours (Autonomous process)
Developing the system	32 hours (Total for all iterations)
*Select N number of test cases	0 hours (Autonomous process)
*Execute test cases on the system	8 hours (Total for all iterations)
*Recalculate global importance	0 hours (Autonomous process)
*Flag test cases based on results	0 hours (Autonomous process)
Refine system and go to next iteration	8 hours (Total for all iterations)

General total for building the housing KBS and validating it using MAVERICK is: **93 hours**

Total for validating a KBS using MAVERICK is $4+5+2+3+1+4+11+8 = \mathbf{38 \text{ hours}}$

EMBODY uses diagram-based validation and VIVA is a lifecycle-independent validation method that is based on traceability. Time consumed for VIVA and EMBODY are described in the next two tables:

Table 53: General time consumed for EMBODY

Stage name/ description	Time consumed
Knowledge acquisition and organization	10 hours (Manual process)
Developing the system	32 hours (This time is assumed to be the same for all methods)
*Using EMBODY flow charts	17 hours (Manual process)
*Representing knowledge EMBODY tabular format	12 hours (Manual process)
*Validating the system	18 hours (Manual process)
Refine system	8 hours (Manual process)

Total for building the housing KBS and validating it using EMBODY is: **97 hours**

Total for validating a KBS using EMBODY is $17+12+18 = 47$ hours

Table 54: General time consumed for VIVA

Stage name/ description	Time consumed
Knowledge acquisition and organization	17 hours (Manual process)
Developing the system	32 hours (This time is assumed to be the same for all methods)
*Using VIVA defined methods	15 hours (Manual process)
*Performing the VIVA link types for the system	8 hours (Manual process)
*Derivation of validation specification	10 hours (Manual process)
*Validating the system	16 hours (Manual process)
Refine system	8 hours (Manual process)

General total for building the housing KBS and validating it using VIVA is: **106 hours**

Total for validating a KBS using VIVA is $15+8+10+16 = 49$ hours

7.4.4 Discussions

The following conclusions are derived from the comparison introduced:

1. MAVERICK consumes less validation time than VIVA (less by 11 hours).
2. MAVERICK consumes less validation time than EMBODY (less by 9 hours).
3. Using MAVERICK within the KBS development process saves development and validation times.
4. MAVERICK consumes more time during the first stages but saves more time during the later stages.
5. The incremental fashion of MAVERICK means that it has more stages and phases.
6. Although MAVERICK is more complicated than EMBODY and VIVA, it uses less time and shows a clear guided path towards a valid system.

7. When the system was validated using VIVA and EMBODY, no indication on the validation percentage was provided, while in MAVERICK after the 93 hours, the previously discussed 94% was achieved.

MAVERICK is the first incremental life-cycle based validation method in knowledge-based systems. This work's contribution to the research community shows in the results presented.

7.5 Summary of Experiments

This chapter introduced all the results associated with the experiments performed for MAVERICK. Three kinds of experiments were presented. Errors were seeded into the system by the developer, by seven human subjects and MAVERICK was used to detect these errors. Furthermore, MAVERICK was compared to other validation methods. In the next chapter, summary, conclusions and future work are discussed.

CHAPTER 8: CONCLUSIONS AND FUTURE WORK

This chapter provides general conclusions and the overall results of this research. The significance of the results to the field is discussed. Furthermore, the global value of the validation method presented in this dissertation is evaluated. This chapter contains three sections: the first section summarizes the work done in this dissertation; the second section includes the conclusions reached, and the last section establishes ideas for future research.

8.1 Summary

This dissertation introduced a new validation method for knowledge-based systems. After reviewing some of the software disasters that happened in the past, it was concluded that more problems could occur if a rigorous process was not followed to fix errors and minimize loss. One of the most important phases in software development is software testing, validation and verification. This is equally true for knowledge-based systems. A definition was given to validation as the official definition used throughout this dissertation. This dissertation differentiates between validation approaches and validation methods. Validation approaches are general ways on validating a knowledge-based system. Validation methods include more specific processes and they can be used to obtain a valid system. This dissertation surveyed the existing validation methods in the literature and these methods were categorized and evaluated separately. An absence of validation methods built within a KBS lifecycle model was noted in the literature. Thus, the best known lifecycle models for knowledge-based systems development were introduced and described. The *defacto* standard for KBS lifecycle development, the CommonKADS model was chosen as the basis for this research.

MAVERICK is built within CommonKADS and consists of three main parts: test case extraction, inspection validation and context-based test case reduction (CBTCR). Context-based test case reduction steps include defining CommonKADS model's weights and using previous test case results to select the next set of test cases. To perform experiments on MAVERICK, a knowledge-based system was built based on the CommonKADS model and the MAVERICK validation steps. A tool was developed for MAVERICK to automate the process. A housing KBS was built within CommonKADS and based on MAVERICK. The CommonKADS models were introduced for the housing KBS and test cases were extracted based on the guidelines presented by MAVERICK. To evaluate the validation method, errors were inserted into the system to evaluate the ability of MAVERICK to track and detect errors. Errors were inserted by the developer as well as by seven unbiased human subjects. Errors inserted by the users were categorized and discussed. The results of detecting errors and error types were introduced. The final error detection result was 94%, which shows that MAVERICK is effective in detecting errors. The initial (not legitimate) rate was 78%, but as discussed previously, this only reflects the validation. If verification had been embedded into CommonKADS and MAVERICK looked for all types of errors, the results are expected to be better. Nevertheless, the official 94% is very close to the ultimate goal of 100% error detection. This and other issues are recommended as future work in another section of this Chapter.

MAVERICK was compared to other two validation methods, VIVA and EMBODY. The comparison to the other validation methods was mainly subjective and limited to measuring the time consumed to validate a knowledge-based system by each of the methods.

8.2 Conclusion

In a broad outline, the validation of a knowledge-based system is important for three main reasons: 1) minimize all kinds of losses, 2) the assurance of acceptance by the user/customer 3) and the commercialization of the system. In most cases, knowledge-based systems are built with a lifecycle model. Introducing validation within a lifecycle model would help to better meet the three mentioned goals. For systems that affect human life, it is necessary to have high level of validation and verification. The ultimate goal of detecting errors and having a valid system is, obviously 100%. While this number has not yet been achieved, it remains as the ultimate goal. However, 94% is considered as a successful result that conforms to the hypothesis presented in Chapter 3.

Introducing a validation method into CommonKADS was a difficult challenge. Many aspects needed to be considered for the MAVERICK validation method. These aspects include the fact that the method is incremental, built within a lifecycle model, based on CommonKADS and uses case testing validation approach. Assembling all these aspects in one validation method and making them fit together was quite challenging. Furthermore, performing experiments on the method from the point of view of all the aspects was difficult to achieve. Conducting this research led to a number of general observations and these observations are discussed next.

Different validation approaches produce different loads on the experts and the development team. Experts cannot perform classical validation by only examining inputs and outputs for the system. Such validation burdens the expert and the knowledge engineer. It is important to follow a defined process that is designed to increase system coverage. Using a lifecycle model is the most agreed upon way to do that. Thus, building a validation method into the lifecycle model minimizes the effort and time dedicated to define a process that is dedicated

for validation only. From a management point of view, planning and scheduling are essential. This validation method has been tailored to help management follow the process and evaluate the performance against deadlines. Because MAVERICK is performed in multiple phases and is included within the multiple phases of CommonKADS, it is clear where the phase starts and where it ends. This would help in planning and setting deadlines.

Management requires a solid and explicit organization for software development projects. MAVERICK defines a clear number of steps and elements. Every element in MAVERICK has a well defined name, location and documentation. In MAVERICK, all the models are identified, all the test cases are categorized, all the diagrams numbered and all the rules in the knowledge base are traced back to a certain part in the models. A well defined structure, yields to better communication and its usual consequent: better organization.

Although MAVERICK was specifically designed for CommonKADS and knowledge-based systems, some parts of it could be used for any software development process; specifically, context-based test case reduction, which is used to reduce the number of test cases. It is the first method that bases the selection of test cases on the previous set of results for knowledge-based systems. Using the results of previous validation iterations could be used in any testing process that uses the case testing approach.

CBTCR could be used by any testing team, especially that a tool was developed for it. To use this tool, the only needed step is to insert all the test cases into the excel sheet as the input. The CBTCR tool can then work with any type of software system. CBTCR tool could be used for testing and for maintenance too. Maintenance usually is costly and requires significant effort. To perform maintenance on a software system, traceability is the main difficulty. It is evident that to properly maintain a knowledge-based system, the requirements need to be well defined

and that they could be traced all the way down to the implementation level. In MAVERICK, the path for any element in the CommonKADS models is very clear from the models to the design model and down to implementation details. This is considered as a major advantage. In maintenance, most of the burden is on the expert and the knowledge engineer and user involvement is typically lacking. While this is true for most late phases in software development, it is not the case for the early requirements and knowledge extraction phase. Involving the user in validation and verification is not yet fully achieved in this research field. Involving the user is feasible because of MAVERICK's clear process towards reaching a valid system and the transparent nature of CommonKADS and CBTCR.

Automating the validation process is another aspect of this method. The definitive step in automating testing is to perform automated refinement. Automated refinement would lead to a full autonomous testing process. This would not only save time and effort but would insure that no human error will occur because of manual execution or refinement of tests.

Ultimately, the work presented in this dissertation aims to improve the general performance of knowledge-based systems. The goal is to attest the hypothesis introduced in Chapter 3. The hypothesis was based on the idea that MAVERICK's performance should be effective and efficient when compared to other validation methods. To be able to explicitly claim that MAVERICK is better than previous validation methods, it needs to be compared against all the previous validation methods for KBS. The reasons why this is not attainable were previously discussed in Chapter 7. MAVERICK introduces a new approach, which is building validation within an incremental lifecycle model. As it is discussed earlier, this has never been done before. Therefore, the comparisons performed against VIVA and EMBODY only show that MAVERICK's performance is indeed efficient. The errors' insertion experiments are performed

to identify the effectiveness of MAVERICK. The discussion on why some errors weren't detected is presented earlier. As the results display, MAVERICK is effective by a percentage equal to 94% and MAVERICK is more efficient (by time) than the two methods it is compared against. Finally, some of the ideas presented in this section are already included in MAVERICK, while others are recommended as future work. Future work is introduced in the next section.

8.3 Future Work

This section suggests future research in the field of validation of knowledge-based system based on the work presented in this dissertation. The future work recommendations for MAVERICK are:

1. Define a verification method and embed it into CommonKADS

Verification is essential in building knowledge-based systems. The experiments performed in this dissertation research showed that if verification was included in CommonKADS, the results might have been better. Verification targets different kind of problems in knowledge-based systems. Satisfying this would require an extensive study on how to incorporate verification means within the CommonKADS models. This would make the global evaluation of knowledge-based systems within CommonKADS more comprehensive. The ultimate goal of this future work recommendation is to increase the validation and verification percentage to 100%.

2. Apply MAVERICK to different-sized knowledge-based systems

MAVERICK was tested on only one knowledge-based system, the housing KBS, which is considered a small system. It would be interesting to observe how MAVERICK would perform with mid-size and large systems. Furthermore, applying MAVERICK to different systems in different disciplines would give MAVERICK more assurance on its usability. It would be

desirable to perform these experiments in industry, especially in a company that develops and uses knowledge-based systems. While the experiments in this dissertation support the successful functionality of MAVERICK, using CommonKADS and MAVERICK for real world projects would allow for a broader evaluation.

3. Automate the remaining manual processes in MAVERICK and include them within the CBTCR tool.

MAVERICK was partially automated and partially manual. It would save time and effort if other steps could be automated. Fully automating the process from beginning to end is not currently possible. Although some steps could be automated, other steps are difficult to automate because of their dependency on the knowledge engineer and other human factors. Processes such as extracting the test cases, executing them on the system and evaluating the results could be performed autonomously by building a software tool. This would require broad research to understand programming languages and how to access different parts of the code. This could be done by building language parsers, which is its own research field.

4. Embed the MAVERICK idea into other lifecycle models for knowledge-based systems

Although this method was dedicated for CommonKADS, it would be a significant contribution to study whether MAVERICK works well with other lifecycle models. Performing this study would require full understanding of MAVERICK and a set of experiments to check if MAVERICK is compatible with other lifecycle models. There's no doubt that it would be necessary to make modifications to MAVERICK for each lifecycle model. MAVERICK would be modified to fit each lifecycle model. Performing this study would be motivating towards using MAVERICK with other software projects that use lifecycle models other than CommonKADS.

5. Proceed from automated testing towards automated repairing through MAVERICK

The ultimate goal of testing, validation and verification is not only to find all the errors in a system but also to fix them. MAVERICK partially performs automated testing, which helps in finding the errors in the system without manually looking for them. Although validation of knowledge-based systems and software systems in general has improved significantly, no method exists that performs full automated refinement. Eventually, this would lead to better evaluation of software systems.

Future work can be directed towards many directions. These future recommendations are within the scope of this research, specifically in validation of knowledge-based systems through a lifecycle model. More precisely, these recommendations are dedicated to build upon the main contribution of this dissertation, MAVERICK.

APPENDIX A: LIST OF TEST CASES FOR THE HOUSING KBS

TestCase ID	CommonKADS Model	Local Importance	Numberof Runs	CommonKADS Weight
1	1	2.75	0	50
2	1	2.75	0	50
3	1	2.75	0	50
4	1	2.75	0	50
5	1	2.75	0	50
6	1	2.75	0	50
7	1	2.75	0	50
8	1	2.75	0	50
9	1	2.75	0	50
10	2	3.75	0	50
11	2	3.75	0	50
12	2	3.75	0	50
13	2	3.75	0	50
14	2	3.75	0	50
15	2	3.75	0	50
16	3	5	0	50
17	3	5	0	50
18	3	5	0	50
19	3	5	0	50
20	3	0.5	0	50
21	3	0.5	0	50
22	3	0.5	0	50
23	3	0.5	0	50
24	4	5	0	50
25	4	5	0	50
26	4	5	0	50
27	4	5	0	50
28	5	3.5	0	50
29	5	3.5	0	50
30	5	3.5	0	50
31	5	3.5	0	50
32	5	3.5	0	50
33	5	3.5	0	50
34	4	4.75	0	50
35	4	4.75	0	50
36	4	4.75	0	50
37	4	4.75	0	50
38	4	4.75	0	50
39	4	4.75	0	50

40	4	4.75	0	50
41	4	4.75	0	50
42	4	4.75	0	50
43	4	4.75	0	50
44	4	4.75	0	50
45	4	4.75	0	50
46	4	4.75	0	50
47	4	4.75	0	50
48	4	4.75	0	50
49	4	4.75	0	50
50	4	4.75	0	50
51	4	4.75	0	50
52	4	4.75	0	50
53	4	4.75	0	50
54	4	4.75	0	50
55	4	4.75	0	50
56	4	4.75	0	50
57	4	4.75	0	50
58	4	4.75	0	50
59	4	4.75	0	50
60	4	4.75	0	50
61	4	4.75	0	50
62	4	4.75	0	50
63	4	4.75	0	50
64	4	4.75	0	50
65	4	4.75	0	50
66	4	4.75	0	50
67	4	4.75	0	50
68	4	4.75	0	50
69	4	4.75	0	50
70	4	4.75	0	50
71	4	4.75	0	50
72	4	4.75	0	50
73	4	4.75	0	50
74	4	4.75	0	50
75	4	4.75	0	50
76	4	4.75	0	50
77	4	4.75	0	50
78	4	4.75	0	50
79	4	4.75	0	50
80	4	4.75	0	50

81	4	4.75	0	50
82	4	4.75	0	50
83	4	4.75	0	50
84	4	4.75	0	50
85	4	4.75	0	50
86	4	4.75	0	50
87	4	4.75	0	50
88	4	4.75	0	50
89	4	4.75	0	50
90	4	4.75	0	50
91	4	4.75	0	50
92	4	4.75	0	50
93	4	4.75	0	50
94	4	4.75	0	50
95	4	4.75	0	50
96	4	4.75	0	50
97	4	4.75	0	50
98	4	4.75	0	50
99	4	4.75	0	50
100	4	4.75	0	50
101	4	4.75	0	50
102	4	4.75	0	50
103	4	4.75	0	50
104	4	4.75	0	50
105	4	4.75	0	50
106	4	4.75	0	50
107	4	4.75	0	50
108	4	4.75	0	50
109	4	4.75	0	50
110	4	4.75	0	50
111	4	4.75	0	50
112	4	4.75	0	50
113	4	4.75	0	50
114	4	4.75	0	50
115	4	4.75	0	50
116	4	4.75	0	50
117	4	4.75	0	50
118	4	4.75	0	50
119	4	4.75	0	50
120	4	4.75	0	50
121	4	4.75	0	50

122	4	4.75	0	50
123	4	4.75	0	50
124	4	4.75	0	50
125	4	4.75	0	50
126	4	4.75	0	50
127	4	4.75	0	50
128	4	4.75	0	50
129	4	4.75	0	50
130	4	4.75	0	50
131	4	4.75	0	50
132	4	4.75	0	50
133	4	4.75	0	50
134	4	4.75	0	50
135	4	4.75	0	50
136	4	4.75	0	50
137	4	4.75	0	50
138	4	4.75	0	50
139	4	4.75	0	50
140	4	4.75	0	50
141	4	4.75	0	50
142	4	4.75	0	50
143	4	4.75	0	50
144	4	4.75	0	50
145	4	4.75	0	50
146	4	4.75	0	50
147	4	4.75	0	50
148	4	4.75	0	50
149	4	4.75	0	50
150	4	4.75	0	50
151	4	4.75	0	50
152	4	4.75	0	50
153	4	4.75	0	50
154	4	4.75	0	50
155	4	4.75	0	50
156	4	4.75	0	50
157	4	4.75	0	50
158	4	4.75	0	50
159	4	4.75	0	50
160	4	4.75	0	50
161	4	4.75	0	50
162	4	4.75	0	50

163	4	4.75	0	50
164	4	4.75	0	50
165	4	4.75	0	50
166	4	4.75	0	50
167	4	4.75	0	50
168	4	4.75	0	50
169	4	4.75	0	50
170	4	4.75	0	50
171	4	4.75	0	50
172	4	4.75	0	50
173	4	4.75	0	50
174	4	4.75	0	50
175	4	4.75	0	50
176	4	4.75	0	50
177	4	4.75	0	50
178	4	4.75	0	50
179	4	4.75	0	50
180	4	4.75	0	50
181	4	4.75	0	50
182	4	4.75	0	50
183	4	4.75	0	50
184	1	3.5	0	50
185	1	3.5	0	50
186	3	3.5	0	50
187	4	3.5	0	50
188	4	3.5	0	50

TestCaseID	GlobalImportance	Result	InputVariables
	137.5	2	login information for the agent: assigner
1	137.5	2	login information for the agent: assigner
2	137.5	2	login information for the agent: assigner
3	137.5	2	login information for the agent: Datatypist
4	137.5	2	login information for the agent: Datatypist
5	137.5	2	login information for the agent: Datatypist
6	137.5	2	login information for the agent: Magazine producer
7	137.5	2	login information for the agent: Magazine producer
8	137.5	2	login information for the agent: Magazine producer
9	187.5	2	log in as assigner and try to assign applicants
10	187.5	2	log in as datatypist and try to datatype applicants
11	187.5	2	log in as magazinier and try to publish
12	187.5	2	log in as assigner

13	187.5	2	log in as datatypist
14	187.5	2	log in as assigner
15	250	2	assign all residences and check manually the correctness of the assignments
16	250	2	publish magazine and check if it is published in the bin folder
17	250	2	type data in DT mode
18	250	2	check applications list, done by assigner...access all information
19	25	2	Check log for action task 1
20	25	2	Check log for action task 2
21	25	2	Check log for action task 3
22	25	2	Check log for action task 4
23	250	2	magazine is published with the right information
24	250	2	applicants assigned the right residences
25	250	2	all applicants found, no one missing
26	250	2	any correct information could be entered about either a residence or an applicant
27	175	2	transaction one: order application assessment
28	175	2	transaction two: obtain application data
29	175	2	transaction three: report decision
30	175	2	transaction 1
31	175	2	transaction 2
32	175	2	transaction 3
33	237.5	2	Income: 10000, family: 1 , Age: 18
34	237.5	2	Income: 10000, family: 2 , Age: 18
35	237.5	2	Income: 10000, family: 4 , Age: 18
36	237.5	2	Income: 10000, family: 6 , Age: 18
37	237.5	2	Income: 10000, family: 8 , Age: 18
38	237.5	2	Income: 10000, family: more than 8 , Age: 18
39	237.5	2	Income: 40000, family: 1 , Age: 18
40	237.5	2	Income: 40000, family: 2 , Age: 18
41	237.5	2	Income: 40000, family: 4 , Age: 18
42	237.5	2	Income: 40000, family: 6 , Age: 18
43	237.5	2	Income: 40000, family: 8 , Age: 18
44	237.5	2	Income: 40000, family: 8+ , Age: 18
45	237.5	2	Income: 100000, family: 1 , Age: 18
46	237.5	2	Income: 100000, family: 2 , Age: 18
47	237.5	2	Income: 100000, family: 4 , Age: 18
48	237.5	2	Income: 100000, family: 6 , Age: 18
49	237.5	2	Income: 100000, family: 8 , Age: 18
50	237.5	2	Income: 100000, family:8+ , Age: 18

51	237.5	2	Income: 250000, family: 1 , Age: 18
52	237.5	2	Income: 250000, family: 2 , Age: 18
53	237.5	2	Income: 250000, family: 4 , Age: 18
54	237.5	2	Income: 250000, family: 6 , Age: 18
55	237.5	2	Income: 250000, family: 8 , Age: 18
56	237.5	2	Income: 250000, family: 8+ , Age: 18
57	237.5	2	Income: 500000, family: 1 , Age: 18
58	237.5	2	Income: 500000, family: 2 , Age: 18
59	237.5	2	Income: 500000, family: 4 , Age: 18
60	237.5	2	Income: 500000, family: 6 , Age: 18
61	237.5	2	Income: 500000, family: 8 , Age: 18
62	237.5	2	Income: 500000, family: 8+ , Age: 18
63	237.5	2	Income: 10000, family: 1 , Age: 35
64	237.5	2	Income: 10000, family: 1 , Age: 45
65	237.5	2	Income: 10000, family: 1 , Age: 60
66	237.5	2	Income: 10000, family: 1 , Age: 60+
67	237.5	2	Income: 10000, family: 2 , Age: 35
68	237.5	2	Income: 10000, family: 2 , Age: 45
69	237.5	2	Income: 10000, family: 2 , Age: 60
70	237.5	2	Income: 10000, family: 2 , Age: 60+
71	237.5	2	Income: 10000, family: 4 , Age: 35
72	237.5	2	Income: 10000, family: 4 , Age: 45
73	237.5	2	Income: 10000, family: 4 , Age: 60
74	237.5	2	Income: 10000, family: 4 , Age: 60+
75	237.5	2	Income: 10000, family: 6 , Age: 35
76	237.5	2	Income: 10000, family: 6 , Age: 45
77	237.5	2	Income: 10000, family: 6 , Age: 60
78	237.5	2	Income: 10000, family: 6 , Age: 60+
79	237.5	2	Income: 10000, family: 8 , Age: 35
80	237.5	2	Income: 10000, family: 8 , Age: 45
81	237.5	2	Income: 10000, family: 8 , Age: 60
82	237.5	2	Income: 10000, family: 8 , Age: 60+
83	237.5	2	Income: 10000, family: more than 8 , Age: 35
84	237.5	2	Income: 10000, family: more than 8 , Age: 45
85	237.5	2	Income: 10000, family: more than 8 , Age: 60
86	237.5	2	Income: 10000, family: more than 8 , Age: 60+
87	237.5	2	Income: 40000, family: 1 , Age: 35
88	237.5	2	Income: 40000, family: 1 , Age: 45
89	237.5	2	Income: 40000, family: 1 , Age: 60
90	237.5	2	Income: 40000, family: 1 , Age: 60+
91	237.5	2	Income: 40000, family: 2 , Age: 35

92	237.5	2	Income: 40000, family: 2 , Age: 45
93	237.5	2	Income: 40000, family: 2 , Age: 60
94	237.5	2	Income: 40000, family: 2 , Age: 60+
95	237.5	2	Income: 40000, family: 4 , Age: 35
96	237.5	2	Income: 40000, family: 4 , Age: 45
97	237.5	2	Income: 40000, family: 4 , Age: 60
98	237.5	2	Income: 40000, family: 4 , Age: 60+
99	237.5	2	Income: 40000, family: 8+ , Age: 35
100	237.5	2	Income: 40000, family: 8+ , Age: 45
101	237.5	2	Income: 40000, family: 8+ , Age: 60
102	237.5	2	Income: 40000, family: 8+ , Age: 60+
103	237.5	2	Income: 40000, family: 6 , Age: 35
104	237.5	2	Income: 40000, family: 6 , Age: 45
105	237.5	2	Income: 40000, family: 6 , Age: 60
106	237.5	2	Income: 40000, family: 6 , Age: 60+
107	237.5	2	Income: 40000, family: 8 , Age: 35
108	237.5	2	Income: 40000, family: 8 , Age: 45
109	237.5	2	Income: 40000, family: 8 , Age: 60
110	237.5	2	Income: 40000, family: 8 , Age: 60+
111	237.5	2	Income: 100000, family: 1 , Age: 35
112	237.5	2	Income: 100000, family: 1 , Age: 45
113	237.5	2	Income: 100000, family: 1 , Age: 60
114	237.5	2	Income: 100000, family: 1 , Age: 60+
115	237.5	2	Income: 100000, family: 2 , Age: 35
116	237.5	2	Income: 100000, family: 2 , Age: 45
117	237.5	2	Income: 100000, family: 2 , Age: 60
118	237.5	2	Income: 100000, family: 2 , Age: 60+
119	237.5	2	Income: 100000, family: 4 , Age: 35
120	237.5	2	Income: 100000, family: 4 , Age: 45
121	237.5	2	Income: 100000, family: 4 , Age: 60
122	237.5	2	Income: 100000, family: 4 , Age: 60+
123	237.5	2	Income: 100000, family: 6 , Age: 35
124	237.5	2	Income: 100000, family: 6 , Age: 45
125	237.5	2	Income: 100000, family: 6 , Age: 60
126	237.5	2	Income: 100000, family: 6 , Age: 60+
127	237.5	2	Income: 100000, family: 8 , Age: 35
128	237.5	2	Income: 100000, family: 8 , Age: 45
129	237.5	2	Income: 100000, family: 8 , Age: 60
130	237.5	2	Income: 100000, family: 8 , Age: 60+
131	237.5	2	Income: 100000, family:8+ , Age: 35
132	237.5	2	Income: 100000, family:8+ , Age: 45

133	237.5	2	Income: 100000, family:8+ , Age: 60
134	237.5	2	Income: 100000, family:8+ , Age: 60+
135	237.5	2	Income: 250000, family: 1 , Age: 35
136	237.5	2	Income: 250000, family: 1 , Age: 45
137	237.5	2	Income: 250000, family: 1 , Age: 60
138	237.5	2	Income: 250000, family: 1 , Age: 60+
139	237.5	2	Income: 250000, family: 2 , Age: 35
140	237.5	2	Income: 250000, family: 2 , Age: 45
141	237.5	2	Income: 250000, family: 2 , Age: 60
142	237.5	2	Income: 250000, family: 2 , Age: 60+
143	237.5	2	Income: 250000, family: 4 , Age: 35
144	237.5	2	Income: 250000, family: 4 , Age: 45
145	237.5	2	Income: 250000, family: 4 , Age: 60
146	237.5	2	Income: 250000, family: 4 , Age: 60+
147	237.5	2	Income: 250000, family: 6 , Age: 35
148	237.5	2	Income: 250000, family: 6 , Age: 45
149	237.5	2	Income: 250000, family: 6 , Age: 60
150	237.5	2	Income: 250000, family: 6 , Age: 60+
151	237.5	2	Income: 250000, family: 8 , Age: 35
152	237.5	2	Income: 250000, family: 8 , Age: 45
153	237.5	2	Income: 250000, family: 8 , Age: 60
154	237.5	2	Income: 250000, family: 8 , Age: 60+
155	237.5	2	Income: 250000, family: 8+ , Age: 35
156	237.5	2	Income: 250000, family: 8+ , Age: 45
157	237.5	2	Income: 250000, family: 8+ , Age: 60
158	237.5	2	Income: 250000, family: 8+ , Age: 60+
159	237.5	2	Income: 500000, family: 1 , Age: 35
160	237.5	2	Income: 500000, family: 1 , Age: 45
161	237.5	2	Income: 500000, family: 1 , Age: 60
162	237.5	2	Income: 500000, family: 1 , Age: 60+
163	237.5	2	Income: 500000, family: 2 , Age: 35
164	237.5	2	Income: 500000, family: 2 , Age: 45
165	237.5	2	Income: 500000, family: 2 , Age: 60
166	237.5	2	Income: 500000, family: 2 , Age: 60+
167	237.5	2	Income: 500000, family: 4 , Age: 35
168	237.5	2	Income: 500000, family: 4 , Age: 45
169	237.5	2	Income: 500000, family: 4 , Age: 60
170	237.5	2	Income: 500000, family: 4 , Age: 60+
171	237.5	2	Income: 500000, family: 6 , Age: 35
172	237.5	2	Income: 500000, family: 6 , Age: 45
173	237.5	2	Income: 500000, family: 6 , Age: 60

174	237.5	2	Income: 500000, family: 6 , Age: 60+
175	237.5	2	Income: 500000, family: 8 , Age: 35
176	237.5	2	Income: 500000, family: 8 , Age: 45
177	237.5	2	Income: 500000, family: 8 , Age: 60
178	237.5	2	Income: 500000, family: 8 , Age: 60+
179	237.5	2	Income: 500000, family: 8+ , Age: 35
180	237.5	2	Income: 500000, family: 8+ , Age: 45
181	237.5	2	Income: 500000, family: 8+ , Age: 60
182	237.5	2	Income: 500000, family: 8+ , Age: 60+
183	175	2	log in as magazine editor
184	175	2	log in as data entry
185	175	2	enter an invalid application
186	175	2	input 2 residences and one applicant
187	175	2	input 2 applicants and one residence
188			

TestCaseID	ExecutionSteps
1	run system and login as assigner to magazine producing interface.
2	run system and login as assigner to residence assignment interface
3	run system and login as assigner to data entry interface.
4	run system and login as datatypist to magazine producing interface.
5	run system and login as datatypist to residence assignment interface
6	run system and login as datatypist to data entry interface.
7	run system and login as magazine to magazine data entry.
8	run system and login as magaziner to residence assignment interface
9	run system and login as magaziner to magazine interface.
10	log in as assigner and try to assign applicants
11	log in as datatypist and try to datatype applicants
12	log in as magaziner and try to publish
13	check access to list of applicants and residences
14	check access to applications
15	check access to db and residences
16	log in as assigner and assign all residences
17	log in as Magazine editor and publish
18	log in as DT and type data for new entity
19	log in as assigner and check residences and applications
20	log in, perform task and check log
21	log in, perform task and check log
22	log in, perform task and check log
23	log in, perform task and check log
24	log in, perform task and check the outcome

25	log in, perform task and check the outcome
26	log in, perform task and check the outcome
27	log in, perform task and check the outcome
28	log in and communicate
29	log in and communicate
30	log in and communicate
31	run transaction1
32	run transaction2
33	run transaction3
34	enter the input variables for the applicant
35	enter the input variables for the applicant
36	enter the input variables for the applicant
37	enter the input variables for the applicant
38	enter the input variables for the applicant
39	enter the input variables for the applicant
40	enter the input variables for the applicant
41	enter the input variables for the applicant
42	enter the input variables for the applicant
43	enter the input variables for the applicant
44	enter the input variables for the applicant
45	enter the input variables for the applicant
46	enter the input variables for the applicant
47	enter the input variables for the applicant
48	enter the input variables for the applicant
49	enter the input variables for the applicant
50	enter the input variables for the applicant
51	enter the input variables for the applicant
52	enter the input variables for the applicant
53	enter the input variables for the applicant
54	enter the input variables for the applicant
55	enter the input variables for the applicant
56	enter the input variables for the applicant
57	enter the input variables for the applicant
58	enter the input variables for the applicant
59	enter the input variables for the applicant
60	enter the input variables for the applicant
61	enter the input variables for the applicant
62	enter the input variables for the applicant
63	enter the input variables for the applicant
64	enter the input variables for the applicant
65	enter the input variables for the applicant

66	enter the input variables for the applicant
67	enter the input variables for the applicant
68	enter the input variables for the applicant
69	enter the input variables for the applicant
70	enter the input variables for the applicant
71	enter the input variables for the applicant
72	enter the input variables for the applicant
73	enter the input variables for the applicant
74	enter the input variables for the applicant
75	enter the input variables for the applicant
76	enter the input variables for the applicant
77	enter the input variables for the applicant
78	enter the input variables for the applicant
79	enter the input variables for the applicant
80	enter the input variables for the applicant
81	enter the input variables for the applicant
82	enter the input variables for the applicant
83	enter the input variables for the applicant
84	enter the input variables for the applicant
85	enter the input variables for the applicant
86	enter the input variables for the applicant
87	enter the input variables for the applicant
88	enter the input variables for the applicant
89	enter the input variables for the applicant
90	enter the input variables for the applicant
91	enter the input variables for the applicant
92	enter the input variables for the applicant
93	enter the input variables for the applicant
94	enter the input variables for the applicant
95	enter the input variables for the applicant
96	enter the input variables for the applicant
97	enter the input variables for the applicant
98	enter the input variables for the applicant
99	enter the input variables for the applicant
100	enter the input variables for the applicant
101	enter the input variables for the applicant
102	enter the input variables for the applicant
103	enter the input variables for the applicant
104	enter the input variables for the applicant
105	enter the input variables for the applicant
106	enter the input variables for the applicant

148	enter the input variables for the applicant
149	enter the input variables for the applicant
150	enter the input variables for the applicant
151	enter the input variables for the applicant
152	enter the input variables for the applicant
153	enter the input variables for the applicant
154	enter the input variables for the applicant
155	enter the input variables for the applicant
156	enter the input variables for the applicant
157	enter the input variables for the applicant
158	enter the input variables for the applicant
159	enter the input variables for the applicant
160	enter the input variables for the applicant
161	enter the input variables for the applicant
162	enter the input variables for the applicant
163	enter the input variables for the applicant
164	enter the input variables for the applicant
165	enter the input variables for the applicant
166	enter the input variables for the applicant
167	enter the input variables for the applicant
168	enter the input variables for the applicant
169	enter the input variables for the applicant
170	enter the input variables for the applicant
171	enter the input variables for the applicant
172	enter the input variables for the applicant
173	enter the input variables for the applicant
174	enter the input variables for the applicant
175	enter the input variables for the applicant
176	enter the input variables for the applicant
177	enter the input variables for the applicant
178	enter the input variables for the applicant
179	enter the input variables for the applicant
180	enter the input variables for the applicant
181	enter the input variables for the applicant
182	enter the input variables for the applicant
183	enter the input variables for the applicant
184	transfer to the data entry mode with log in
185	transfere to application assessment mode
186	try to save and assign
187	try to assign, applicant assigned 2 residents? WRONG
188	try to assign, residence assigned 2 applicants? WRONG

TestCaseID	ExpectedSolution
1	system does not allow access.
2	system allows access
3	system does not allow access.
4	Stop Access
5	Stop access
6	Give access
7	Stop Access
8	Stop access
9	Give access
10	assigner success in assigning applicants (not checking for validity of the assignment process here)
11	datatypist success in typing
12	magaziner could access publishing
13	access to applicants list displayed on the interface
14	applications available
15	access to lists displayed on the interface
16	all correct residences are assigned to correct applicants
17	magazine published in bin folder
18	data could be typed in and saved
19	assigner could access all current information
20	Info found in log
21	Info found in log
22	Info found in log
23	Info found in log
24	task performed correctly
25	task performed correctly
26	task performed correctly
27	task performed correctly
28	transaction goes through
29	transaction goes through
30	transaction goes through
31	transaction results are correct
32	transaction results are correct
33	transaction results are correct
34	applicant assigned an apartment based on variables
35	applicant assigned an apartment based on variables
36	applicant assigned an apartment based on variables
37	applicant assigned an apartment based on variables
38	applicant assigned an apartment based on variables

162	applicant assigned an apartment based on variables
163	applicant assigned an apartment based on variables
164	applicant assigned an apartment based on variables
165	applicant assigned an apartment based on variables
166	applicant assigned an apartment based on variables
167	applicant assigned an apartment based on variables
168	applicant assigned an apartment based on variables
169	applicant assigned an apartment based on variables
170	applicant assigned an apartment based on variables
171	applicant assigned an apartment based on variables
172	applicant assigned an apartment based on variables
173	applicant assigned an apartment based on variables
174	applicant assigned an apartment based on variables
175	applicant assigned an apartment based on variables
176	applicant assigned an apartment based on variables
177	applicant assigned an apartment based on variables
178	applicant assigned an apartment based on variables
179	applicant assigned an apartment based on variables
180	applicant assigned an apartment based on variables
181	applicant assigned an apartment based on variables
182	applicant assigned an apartment based on variables
183	applicant assigned an apartment based on variables
184	log in and out smoothly, full access to new mode and no prblems or errors generated.
185	log in and out smoothly, full access to new mode and no prblems or errors generated.
186	the system should not allow saving
187	applicant assigned one residence only!
188	applicant assigned one residence only!

APPENDIX B: JAVA CODE FOR THE HOUSING KNOWLEDGE-BASED SYSTEM

//KnowledgeBase.JAVA

```
import javax.swing.JButton;

public class KnowledgeBase {

    //Sort by applicant date then choose based on the following criteria:
    //Location will be assigned automatically
    //Area is associated with Family
    //Rent/Location is associated with Income
    //Age with Associated with Rent/Location
    int Family=0; //1,2,3,4,5 --> 1,23,45,67,8
    int Income=0; //1,2,3,4 --> 0-40,40-100,100-500,500+
    int Age=0; //1,4,3,2 --> 0-20,20-40-,40-60,60+

    int Area=0; //1,2,3,4,5 --> 600,800,1000,1200,1200+
    int Rent=0; //1,2,3,4 --> 545,750,1047,1047+

    int RESULTS [][];

    boolean CONTROL = false;

    public int [][] maincontroller(int [][] A, int [][] R, int a, int r)//get
    array of applicants, array of residences, and number of applicants
    {
        RESULTS = new int [2][a];
        int fm=0,in=0,ag=0; //White #5

        for (int i=0; i<a; i++)//fill applicants ID's //White #4
        {
            RESULTS[0][i] = i;
            RESULTS[1][i] = 50;//EMPTY, NOT ASSIGNED YET
        }

        for (int i=0; i<a; i++)
        {
            System.out.print("\n"+"-----"+" \n");

            for (int j=0;j<a;j++)//White #6
            {
                System.out.print("RESULTS:"+RESULTS[0][j]+", "+RESULTS[1][j]+" \n");
            }

            ag=A[0][i];
            fm=A[1][i];
            in=A[2][i];
            //Grey #1
            RULES_KB(fm,in,ag);

            CONTROL = ASSIGN(a, R, r, i);

            if (CONTROL == false) //White #7
            {
                ReASSIGN(a,R, r, i);
            }
            //Red #9
        }
        // return rent and area
    }
}
```

```

return RESULTS;
}

private boolean RULES_KB(int fm, int in, int ag) // Assignment rules
{
Family = fm;
Income = in; Yellow #7
Age = ag; //Yellow #8
//Grey #6

if (Family == 1)Area = 1; //Red #8
//White #1
if (Family == 2)Area = 2;
if (Family == 3)Area = 3; //Brown #4
//Brown #11
if (Family == 4)Area = 4;
if (Family == 5)Area = 5;

if (Income == 1 && Age == 1) Rent = 1; //Red #7
//Orange #1
if (Income == 1 && Age == 3) Rent = 2; //White #8
//Orange #2
if (Income == 1 && Age == 2) Rent = 2; //Brown #5
//Orange #3
//Grey #3
if (Income == 1 && Age == 4) Rent = 1;
//Orange #4
if (Income == 2 && Age == 1) Rent = 2; //Orange #5
//Orange #6
if (Income == 2 && Age == 3) Rent = 3; //White #9
if (Income == 2 && Age == 2) Rent = 3;
if (Income == 2 && Age == 4) Rent = 2;

if (Income == 3 && Age == 1) Rent = 3; //Blue #4
if (Income == 3 && Age == 3) Rent = 4; //Orange #14
//Brown #8
//Grey #5
if (Income == 3 && Age == 2) Rent = 4; //Orange #15
if (Income == 3 && Age == 4) Rent = 3; //Blue #6
if (Income == 4 && Age == 1) Rent = 4; //Orange #9
if (Income == 4 && Age == 2) Rent = 4; //Orange #10
//Grey #9
if (Income == 4 && Age == 3) Rent = 4; //Orange #11
if (Income == 4 && Age == 4) Rent = 4; //Orange #12

if (Rent == 0 && Area ==0){return false;} //Blue #5
else {return true;} //Brown #6

//21 rules out of 40 rules
} //Red #6
//Grey #2
private boolean ASSIGN (int a, int [][] R, int r, int app)
{

```

```

for (int i=0; i<r;i++) //Grey #8
{
if (R[0][i]==Rent && R[1][i]==Area) //rule 38
//Grey #4
//Orange #13
{
RESULTS[1][app]=i;
//Grey #10
//remove residence from array
R[0][i]=50;//not to be taken again
R[1][i]=50;//not to be taken again
return true;

}
else if ((R[0][i]==Rent+1|R[0][i]==Rent-1|R[0][i]==Rent-
2|R[0][i]==Rent+2|R[0][i]==Rent-3|R[0][i]==Rent+3)
&& (R[1][i]==Area+1|R[1][i]==Area-1|R[1][i]==Area+2|R[1][i]==Area-
2|R[1][i]==Area+3|R[1][i]==Area-3)) // rule 39 //White #2
{//Yellow #5
//White #3
RESULTS[1][app]=i;
R[0][i]=50;//not to be taken again //Orange #7
R[1][i]=50;//not to be taken again //Orange #8
return true;
}}
return false;
}
private void ReASSIGN(int a,int[][] R,int r, int app)
{//Brown #12
for (int i=0; i<r;i++)
{ //Grey #7
if ((R[0][i]==Rent+4|R[0][i]==Rent-4|R[0][i]==Rent-5|R[0][i]==Rent+5)
&& (R[1][i]==Area+4|R[1][i]==Area-4|R[1][i]==Area+5|R[1][i]==Area-
5))//rule 40 //Brown #7
{//Yellow #6
RESULTS[1][app]=i;
R[0][i]=50;//not to be taken again
R[1][i]=50;//not to be taken again
}}}}

```

//MainApplet.JAVA

```

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Label;
import java.awt.Point;
import java.awt.Rectangle;
import java.util.LinkedList;

import javax.swing.ButtonGroup;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;

```

```

import javax.swing.JPasswordField;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import java.awt.GridBagLayout;
import java.io.BufferedWriter;
import java.io.FileWriter;

public class MainApplet extends JApplet {

private JPanel jContentPane = null;
private JPanel jPanelLogin = null;
private Label labeltext1 = null;
private Label labelusername = null;
private Label labelpassword = null;
private Label labelERROR = null;
private JRadioButton jRadioButtonassigner = null;
private JRadioButton jRadioButtondatatypist = null;
private JLabel loginas = null;
private JLabel jLabeltitle = null;
private JTextArea jTextAreaDESC = null;
private JScrollPane jScrollPaneDESC = null;
private JPanel jPanelAssigner = null;
private JPanel jPanelDatatypist = null;
private JLabel jLabelassigner = null;
private JLabel jLabeldatatypist = null;
private JLabel jLabeltitle1 = null;
private JLabel jLabeltitle2 = null;
private ButtonGroup bgroup = null; // @jve:decl-index=0:
private ButtonGroup bgroup1 = null; // @jve:decl-index=0:
private ButtonGroup bgroup2 = null; // @jve:decl-index=0:
private ButtonGroup bgroup3 = null; // @jve:decl-index=0:

private JButton jButtonsaveandnew = null;
private JButton jButtonSignIn = null;
private JLabel jLabelname = null;
private JLabel jLabelage = null;
private JLabel jLabeldate = null;
private JLabel jLabelmem = null;
private JLabel jLabelincome = null;
private JButton jButtonNEWAPP = null;
private JPanel jPanelresident = null;
private JLabel jLabelarea = null;
private JLabel jLabelloc = null;
private JButton jButtonassingAUTO = null;
private JLabel jLabeltext = null;
private JLabel jLabelliclistapplicants = null;
private JLabel jLabelliclistofresidences = null;
private JScrollPane jScrollPaneapplicants = null;
private JScrollPane jScrollPaneresidents = null;
private JTextArea jTextAreaapplicants = null;
private JTextArea jTextAreaREsidents = null;

```

```

public LinkedList<String> LISTofAPPLICANTS= new LinkedList<String> (); //
@jve:decl-index=0:
public LinkedList<String> LISTofRESIDENTS= new LinkedList<String> (); //
AREA_RENT_LOCATION // @jve:decl-index=0:

public LinkedList<Integer> SecretlistRent= new LinkedList<Integer> (); //
@jve:decl-index=0:
public LinkedList<Integer> SecretlistArea= new LinkedList<Integer> (); //
@jve:decl-index=0:
public LinkedList<Integer> SecretlistFamily= new LinkedList<Integer> ();
// @jve:decl-index=0:
public LinkedList<Integer> SecretlistIncome= new LinkedList<Integer> ();
// @jve:decl-index=0:
public LinkedList<Integer> SecretlistAge= new LinkedList<Integer> (); //
@jve:decl-index=0:

private String AREA=""; // @jve:decl-index=0:
private String RENT=""; // @jve:decl-index=0:
private String LOCATION=""; // @jve:decl-index=0:
private JLabel jLabeltext66 = null;
private JButton jButtonADDRES = null;
private JRadioButton jRadioButtonA = null;
private JRadioButton jRadioButtonB = null;
private JRadioButton jRadioButtonC = null;
private JRadioButton jRadioButtonD = null;
private JRadioButton jRadioButtonE = null;
private JRadioButton jRadioButton600 = null;
private JRadioButton jRadioButton800 = null;
private JRadioButton jRadioButton1000 = null;
private JRadioButton jRadioButton1200 = null;
private JRadioButton jRadioButtonMORE1200 = null;
private JRadioButton jRadioButtonrent1 = null;
private JRadioButton jRadioButtonrent750 = null;
private JRadioButton jRadioButtonrent1047 = null;
private JRadioButton jRadioButtonrent1047more = null;
private JTextField jTextFieldname = null;
private JTextField jTextFielddage = null;
private JTextField jTextFielddate = null;
private JTextField jTextFieldfamily = null;
private JTextField jTextFieldincome = null;
private JLabel jLabeltextshow = null;
private JLabel jLabelsaved = null;
private JLabel jLabelerror = null;
private JTextField jTextFieldusername = null;
private JPasswordField jPasswordFieldpass = null;
private boolean setresidence = false;
private JLabel jLabelmissing = null;
private JButton jButtonongetapplicants = null;

int rrr=0;
int aaa=0;
int fff=0;
int iii=0;
int ggg=0;
String ResultsString="Apartments are not assigned yet! Contact Assigner!";
// @jve:decl-index=0:

```

```

int RCounter = 15; //0-14 are assigned statically
int ACounter = 1; //0 is assigned statically

private KnowledgeBase kb = new KnowledgeBase(); // @jve:decl-index=0:
private JLabel jLabeltext77 = null;
private JScrollPane jScrollPaneresults = null;
private JTextArea jTextAreaResults = null;
private JRadioButton jRadioButtonmagazineeditor = null;
private JPanel jPanelMagaziner = null;
private JLabel jLabelmagaziner = null;
private JButton jButtonPrintMAG = null;
private JLabel jLabelmessgaeformagazine = null;
private JTextArea jTextAreacredits = null;
/**
This is the xxx default constructor
*/
public MainApplet() {
super();
}

/**
This method initializes this
*
@return void
*/
public void init() {
this.setSize(900, 508);
this.setContentPane(getJContentPane());
bgroup = new ButtonGroup();
bgroup.add(jRadioButtondatatypist);
bgroup.add(jRadioButtonassigner);
bgroup.add(jRadioButtonmagazineeditor);

bgroup1 = new ButtonGroup();
bgroup2 = new ButtonGroup();
bgroup3 = new ButtonGroup();

bgroup1.add(jRadioButton600);
bgroup1.add(jRadioButton800);
bgroup1.add(jRadioButton1000);
bgroup1.add(jRadioButton1200);
bgroup1.add(jRadioButtonMORE1200);

bgroup2.add(jRadioButtonrent1047);
bgroup2.add(jRadioButtonrent1);
bgroup2.add(jRadioButtonrent1047more);
bgroup2.add(jRadioButtonrent750);

bgroup3.add(jRadioButtonA);
bgroup3.add(jRadioButtonB);
bgroup3.add(jRadioButtonC);
bgroup3.add(jRadioButtonD);
bgroup3.add(jRadioButtonE);

//add initial residences (random) AREA-RENT-LOCATION//15 here

```

```

LISTofRESIDENTS.add("0-600-545-
A");SecretlistRent.add(1);SecretlistArea.add(1); //White #12
LISTofRESIDENTS.add("1-600-1048-
B");SecretlistRent.add(1);SecretlistArea.add(4);
LISTofRESIDENTS.add("2-600-545-
E");SecretlistRent.add(1);SecretlistArea.add(1);
LISTofRESIDENTS.add("3-1000-545-
A");SecretlistRent.add(3);SecretlistArea.add(1); //Yellow #1
LISTofRESIDENTS.add("4-1201-750-
D");SecretlistRent.add(5);SecretlistArea.add(2);
LISTofRESIDENTS.add("5-1201-545-
B");SecretlistRent.add(5);SecretlistArea.add(1);
LISTofRESIDENTS.add("6-800-750-
E");SecretlistRent.add(2);SecretlistArea.add(2);
LISTofRESIDENTS.add("7-1000-545-
D");SecretlistRent.add(3);SecretlistArea.add(1);
LISTofRESIDENTS.add("8-800-750-
B");SecretlistRent.add(2);SecretlistArea.add(2);
LISTofRESIDENTS.add("9-1200-1047-
B");SecretlistRent.add(4);SecretlistArea.add(3);
LISTofRESIDENTS.add("10-1000-1048-
B");SecretlistRent.add(3);SecretlistArea.add(4);
LISTofRESIDENTS.add("11-1201-1047-
C");SecretlistRent.add(5);SecretlistArea.add(3);
LISTofRESIDENTS.add("12-600-750-
C");SecretlistRent.add(1);SecretlistArea.add(2);
LISTofRESIDENTS.add("13-800-545-
B");SecretlistRent.add(2);SecretlistArea.add(1);
LISTofRESIDENTS.add("14-1200-750-
C");SecretlistRent.add(4);SecretlistArea.add(2);

LISTofAPPLICANTS.add("0-"+Feras+"-"+26+"-"+04/02/10+"-"+7+"-
"+"20,000");
SecretlistFamily.add(3);SecretlistIncome.add(1);SecretlistAge.add(2);
}

/**
This method initializes jContentPane
*
@return javax.swing.JPanel
*/
private JPanel getJContentPane() {
if (jContentPane == null) {
jLabeltitle2 = new JLabel();
jLabeltitle2.setBounds(new Rectangle(377, 9, 132, 32));
jLabeltitle2.setForeground(Color.white);
jLabeltitle2.setText("The");
jLabeltitle2.setHorizontalAlignment(SwingConstants.CENTER);
jLabeltitle2.setFont(new Font("Dialog", Font.BOLD, 24));
jLabeltitle1 = new JLabel();
jLabeltitle1.setBounds(new Rectangle(380, 56, 132, 32));
jLabeltitle1.setForeground(Color.white);
jLabeltitle1.setText("Housing");
jLabeltitle1.setHorizontalAlignment(SwingConstants.CENTER);
jLabeltitle1.setFont(new Font("Dialog", Font.BOLD, 24));
jLabeltitle = new JLabel();

```

```

jLabeltitle.setBounds(new Rectangle(376, 107, 136, 38));
jLabeltitle.setFont(new Font("Dialog", Font.BOLD, 24));
jLabeltitle.setForeground(Color.white);
jLabeltitle.setHorizontalAlignment(SwingConstants.CENTER);
jLabeltitle.setText("Application");
jContentPane = new JPanel();
jContentPane.setLayout(null);
jContentPane.setBackground(new Color(0, 0, 51));
jContentPane.add(getJPanelLogin(), null);
jContentPane.add(jLabeltitle, null);
jContentPane.add(getJScrollPaneDESC(), null);
jContentPane.add(getJPanelAssigner(), null);
jContentPane.add(getJPanelDatatypist(), null);
jContentPane.add(jLabeltitle1, null);
jContentPane.add(jLabeltitle2, null);
jContentPane.add(getJPanelMagaziner(), null);
jContentPane.add(getJTextAreacreidts(), null);
}
return jContentPane;
}

/**
This method initializes jPanelLogin
*
@return javax.swing.JPanel
*/
private JPanel getJPanelLogin() {
if (jPanelLogin == null) {
loginas = new JLabel();
loginas.setText("Sign in as:");
loginas.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC, 12));
loginas.setForeground(new Color(0, 0, 51));
loginas.setBounds(new Rectangle(249, 13, 69, 16));
labelERROR = new Label();
labelERROR.setBounds(new Rectangle(119, 105, 118, 19));
labelERROR.setForeground(Color.red);
labelERROR.setText("ERROR: Invalid info");
labelERROR.setVisible(false);
labelpassword = new Label();
labelpassword.setBounds(new Rectangle(4, 75, 65, 23));
labelpassword.setText("Password:");
labelusername = new Label();
labelusername.setBounds(new Rectangle(3, 46, 71, 23));
labelusername.setText("User Name:");
labeltext1 = new Label();
labeltext1.setBounds(new Rectangle(2, 2, 78, 35));
labeltext1.setFont(new Font("Dialog", Font.BOLD, 18));
labeltext1.setText("Sign in:");
jPanelLogin = new JPanel();
jPanelLogin.setLayout(null);
jPanelLogin.setBackground(Color.white);
jPanelLogin.setSize(new Dimension(362, 143));
jPanelLogin.setLocation(new Point(7, 8));
jPanelLogin.setToolTipText("Please log in!");
jPanelLogin.add(labeltext1, null);
jPanelLogin.add(labelusername, null);
jPanelLogin.add(labelpassword, null);
}
}

```



```

jPanelLogin.add(labelERROR, null);
jPanelLogin.add(loginas, null);
jPanelLogin.add(getJRadioButtonatypist(), null);
jPanelLogin.add(getJRadioButtonassigner(), null);
jPanelLogin.add(getJButtonSignIn(), null);
jPanelLogin.add(getJTextFieldusername(), null);
jPanelLogin.add(getJPasswordFieldpass(), null);
jPanelLogin.add(getJRadioButtonmagazineeditor(), null);
}
return jPanelLogin;
}

/**
This method initializes jRadioButtonassigner
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonassigner() {
if (jRadioButtonassigner == null) {
jRadioButtonassigner = new JRadioButton();
jRadioButtonassigner.setText("Assigner");
jRadioButtonassigner.setBackground(Color.white);
jRadioButtonassigner.setBounds(new Rectangle(247, 35, 83, 21));
}
return jRadioButtonassigner;
}

/**
This method initializes jRadioButtonatypist
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonatypist() {
if (jRadioButtonatypist == null) {
jRadioButtonatypist = new JRadioButton();
jRadioButtonatypist.setText("Data Typist");
jRadioButtonatypist.setBackground(Color.white);
jRadioButtonatypist.setBounds(new Rectangle(247, 59, 92, 24));
}
return jRadioButtonatypist;
}

/**
This method initializes JTextAreaDESC
*
@return javax.swing.JTextArea
*/
private JTextArea getJTextAreaDESC() {
if (JTextAreaDESC == null) {
JTextAreaDESC = new JTextArea();
JTextAreaDESC.setText("Description: The housing application is used by the
Dutch \ngovernment to assign apartments to citizens who apply for
one.\nThis process is called residence distribution and citizens \nwho
want an apartment must apply to be a potential applicant. \nEvery two
weeks a magazine is published and distributed in \nthe Netherlands with a
list of residences for which people can \napply. Another published
proceeding has the names of the \nwinning applicants. To be eligible for a

```

```

residence, applicants \nmust meet a number of criteria. Residents must
consider \nthe number of family members in their household, \nresidence
category in regards to the area where the residence \nis located and a
fair relation between the applicant's income \nand the rent of the
apartment. Other parameters may apply for \nevery specific
case/applicant.");
jTextAreaDESC.setToolTipText("Description of KBS");
jTextAreaDESC.setEditable(false);
}
return jTextAreaDESC;
}

/**
This method initializes jScrollPaneDESC
*
@return javax.swing.JScrollPane
*/
private JScrollPane getJScrollPaneDESC() {
if (jScrollPaneDESC == null) {
jScrollPaneDESC = new JScrollPane();
jScrollPaneDESC.setLocation(new Point(522, 7));
jScrollPaneDESC.setSize(new Dimension(362, 143));
jScrollPaneDESC.setToolTipText("DEscription about the KBS");
jScrollPaneDESC.setViewportViewView(getJTextAreaDESC());
}
return jScrollPaneDESC;
}

/**
This method initializes jPanelAssigner
*
@return javax.swing.JPanel
*/
private JPanel getJPanelAssigner() {
if (jPanelAssigner == null) {
jLabelerror = new JLabel();
jLabelerror.setBounds(new Rectangle(131, 238, 86, 16));
jLabelerror.setForeground(Color.red);
jLabelerror.setEnabled(true);
jLabelerror.setText("Missing Fields!");
jLabelerror.setVisible(false);
//Red #1
jLabelsaved = new JLabel();
jLabelsaved.setBounds(new Rectangle(12, 238, 107, 16));
jLabelsaved.setForeground(Color.red);
jLabelsaved.setEnabled(true);
jLabelsaved.setText("Applicant Saved!");
jLabelsaved.setVisible(false);
jLabelincome = new JLabel();
jLabelincome.setText("Income Per Year (In Euro):");
jLabelincome.setLocation(new Point(5, 165));
jLabelincome.setEnabled(false);
jLabelincome.setSize(new Dimension(152, 16));
jLabelmem = new JLabel();
jLabelmem.setText("Number of Family Memebers:");
jLabelmem.setLocation(new Point(5, 125));
jLabelmem.setEnabled(false);

```

```

jLabelmem.setSize(new Dimension(169, 16));
jLabeldate = new JLabel();
jLabeldate.setText("Applying Date (MM/DD/YY):");
jLabeldate.setLocation(new Point(5, 84));
jLabeldate.setEnabled(false);
jLabeldate.setSize(new Dimension(153, 16));
jLabelage = new JLabel();
jLabelage.setText("Age:");
jLabelage.setLocation(new Point(5, 61));
jLabelage.setEnabled(false);
jLabelage.setSize(new Dimension(31, 16));
jLabelname = new JLabel();
jLabelname.setBounds(new Rectangle(5, 37, 38, 16));
jLabelname.setEnabled(false);
jLabelname.setText("Name:");
jLabeldatatypist = new JLabel();
jLabeldatatypist.setBounds(new Rectangle(8, 9, 147, 23));
jLabeldatatypist.setFont(new Font("Dialog", Font.BOLD, 14));
jLabeldatatypist.setForeground(new Color(0, 153, 0));
jLabeldatatypist.setEnabled(false);
jLabeldatatypist.setText("Data Typist's Screen:");
jPanelAssigner = new JPanel();
jPanelAssigner.setLayout(null);
jPanelAssigner.setBounds(new Rectangle(7, 160, 437, 260));
jPanelAssigner.setToolTipText("Data typist only screen");
jPanelAssigner.setBackground(Color.lightGray);
jPanelAssigner.setEnabled(false);
jPanelAssigner.add(jLabeldatatypist, null);
jPanelAssigner.add(getJButtonsaveandnew(), null);
jPanelAssigner.add(jLabelname, null);
jPanelAssigner.add(jLabelage, null);
jPanelAssigner.add(jLabeldate, null);
jPanelAssigner.add(jLabelmem, null);
jPanelAssigner.add(jLabelincome, null);
jPanelAssigner.add(getJButtonNEWAPP(), null);
jPanelAssigner.add(getJPanelresident(), null);
jPanelAssigner.add(getJTextFieldname(), null);
jPanelAssigner.add(getJTextFielddage(), null);
jPanelAssigner.add(getJTextFielddate(), null);
jPanelAssigner.add(getJTextFielddfamilly(), null);
jPanelAssigner.add(getJTextFielddincome(), null);
jPanelAssigner.add(jLabelsaved, null);
jPanelAssigner.add(jLabelerror, null);
}
return jPanelAssigner;
}

/**
This method initializes jPanelDatatypist
*
@return javax.swing.JPanel
*/
private JPanel getJPanelDatatypist() {
if (jPanelDatatypist == null) { //White #13
jLabelText77 = new JLabel();
jLabelText77.setText("Results:");
jLabelText77.setLocation(new Point(19, 216));

```

```

jLabelText77.setForeground(Color.blue);
jLabelText77.setSize(new Dimension(49, 16));
jLabelText77.setVisible(false);
jLabellistofresidences = new JLabel();
jLabellistofresidences.setText("List of Residences (Area-Rent-
Location):");
jLabellistofresidences.setLocation(new Point(17, 141));
jLabellistofresidences.setForeground(new Color(0, 0, 51));
jLabellistofresidences.setEnabled(false);
jLabellistofresidences.setSize(new Dimension(231, 16));
jLabellistapplicants = new JLabel();
jLabellistapplicants.setBounds(new Rectangle(16, 75, 107, 16));
jLabellistapplicants.setForeground(new Color(0, 0, 51));
jLabellistapplicants.setEnabled(false);
jLabellistapplicants.setText("List of Applicants:");
jLabelText = new JLabel();
jLabelText.setBounds(new Rectangle(291, 40, 125, 16));
jLabelText.setForeground(Color.red);
jLabelText.setEnabled(true);
jLabelText.setText("Applicants Assigned!");
jLabelText.setVisible(false);
jLabelassigner = new JLabel();
jLabelassigner.setBounds(new Rectangle(8, 6, 136, 23));
jLabelassigner.setFont(new Font("Dialog", Font.BOLD, 14));
jLabelassigner.setForeground(new Color(0, 153, 0));
jLabelassigner.setEnabled(false);
jLabelassigner.setText("Assigner's Screen:");
jPanelDatatypist = new JPanel();
jPanelDatatypist.setLayout(null);
jPanelDatatypist.setLocation(new Point(451, 160));
jPanelDatatypist.setSize(new Dimension(437, 260));
jPanelDatatypist.setToolTipText("Assigner only screen");
jPanelDatatypist.setBackground(Color.lightGray);
jPanelDatatypist.setEnabled(false);
jPanelDatatypist.add(jLabelassigner, null);
jPanelDatatypist.add(getJButtonassingAUTO(), null);
jPanelDatatypist.add(jLabelText, null);
jPanelDatatypist.add(jLabellistapplicants, null);
jPanelDatatypist.add(jLabellistofresidences, null);
jPanelDatatypist.add(getJScrollPaneapplicants(), null);
jPanelDatatypist.add(getJScrollPaneresidents(), null);
jPanelDatatypist.add(getJButtongetapplicants(), null);
jPanelDatatypist.add(jLabelText77, null);
jPanelDatatypist.add(getJScrollPaneresults(), null);
}
return jPanelDatatypist;
}

/**
This method initializes jButtonsaveandnew
*
@return javax.swing.JButton
*/
private JButton getJButtonsaveandnew() {
if (jButtonsaveandnew == null) {
jButtonsaveandnew = new JButton();
jButtonsaveandnew.setBounds(new Rectangle(4, 206, 62, 27));

```

```

jButtonsaveandnew.setEnabled(false);
jButtonsaveandnew.setText("Save");
jButtonsaveandnew.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent e) {
System.out.println("actionPerformed()"); // TODO Auto-generated Event stub
actionPerformed()
if((!(jTextFieldname.getText().equals(""))&&!(jTextFielddage.getText().eq
uals(""))&&!(jTextFielddate.getText().equals(""))))
&&!(jTextFielddfamily.getText().equals(""))&&!(jTextFielddincome.getText(
).equals(""))))
{LISTofAPPLICANTS.add(ACounter+"-"+jTextFieldname.getText()+"+"+"-
"+jTextFielddage.getText()+"+"+"-"+
jTextFielddate.getText()+"+"+"-"+jTextFielddfamily.getText()+"+"+"-
"+jTextFielddincome.getText()+"");

if (Integer.parseInt(jTextFielddfamily.getText()+"")==1)fff=1; //Red #2
if (Integer.parseInt(jTextFielddfamily.getText()+"")==2)fff=2; //Yellow #2
if (Integer.parseInt(jTextFielddfamily.getText()+"")==3)fff=2; //Red #3
if (Integer.parseInt(jTextFielddfamily.getText()+"")==4)fff=3; //Brown #1
if (Integer.parseInt(jTextFielddfamily.getText()+"")==5)fff=3;
if (Integer.parseInt(jTextFielddfamily.getText()+"")==6)fff=4;
if (Integer.parseInt(jTextFielddfamily.getText()+"")==7)fff=4;
if (Integer.parseInt(jTextFielddfamily.getText()+"")>=8)fff=5; //Blue #2
//Brown #2
if (Integer.parseInt(jTextFielddage.getText()+"")<=20)ggg=1; //Red #4
else if (Integer.parseInt(jTextFielddage.getText()+"")<=40)ggg=2;
//Red #5 //Blue #1
//White #10
else if (Integer.parseInt(jTextFielddage.getText()+"")<60)ggg=3;
else if (Integer.parseInt(jTextFielddage.getText()+"")>=60)ggg=4; //Brown #9

if (Integer.parseInt(jTextFielddincome.getText()+"")<=40000)iii=1; //Yellow #3
//Brown #3
//White #19
else if (Integer.parseInt(jTextFielddincome.getText()+"")<=100000)iii=2;
else if (Integer.parseInt(jTextFielddincome.getText()+"")<500000)iii=3;
//White #11
//Brown #10
else if (Integer.parseInt(jTextFielddincome.getText()+"")>=500000)iii=4;
//16 rules out of 40 rules

SecretlistFamily.add(fff);
SecretlistIncome.add(iii);
SecretlistAge.add(ggg);

ACounter++; //White #14
jLabelsaved.setVisible(true);
jLabelerror.setVisible(false); //White #15
System.out.print("LISTofAPPLICANTS: ");
System.out.print(LISTofAPPLICANTS);
}
else {jLabelerror.setVisible(true);}
}
});
}
}

```

```

return jButtonsaveandnew;
}

/**
This method initializes jButtonSignIn
*
@return javax.swing.JButton
*/
private JButton getJButtonSignIn() {
if (jButtonSignIn == null) {
jButtonSignIn = new JButton();
jButtonSignIn.setBounds(new Rectangle(279, 115, 72, 22));
jButtonSignIn.setText("Sign in");
jButtonSignIn.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent e) {

jLabelmessgaeformagazine.setVisible(false);
System.out.println("actionPerformed()"); // TODO Auto-generated Event stub
actionPerformed()
System.out.println(jTextFieldusername.getText());

//Passwords and access
if ((jRadioButtonassigner.isSelected()==true)&& //White #20
//White #16
(jTextFieldusername.getText().toString().equals("auser"))&&(jPasswordField
pass.getText().toString().equals("apass")))
{
enableassigner();
disabledatatypepist();
disablemagazine();
labelERROR.setVisible(false);
}
else if ((jRadioButtonmagazineeditor.isSelected()==true)&&
(jTextFieldusername.getText().toString().equals("muser"))&&(jPasswordField
pass.getText().toString().equals("mpass")))
{
disableassigner();
disabledatatypepist();
enablemagazine();
labelERROR.setVisible(false);
}
else if ((jRadioButtondatatypepist.isSelected()==true)&&
(jTextFieldusername.getText().toString().equals("duser"))&&(jPasswordField
pass.getText().toString().equals("dpass")))
{
enabledatatypepist();
disableassigner();
disablemagazine();
labelERROR.setVisible(false);
}
else {labelERROR.setVisible(true);disableassigner();disabledatatypepist();}
}
});
jButtonSignIn.setText("Sign in");

}

```

```

return jButtonSignIn;
}

/**
This method initializes jButtonNEWAPP
*
@return javax.swing.JButton
*/
private JButton getJButtonNEWAPP() {
if (jButtonNEWAPP == null) { //White #17
jButtonNEWAPP = new JButton();
jButtonNEWAPP.setBounds(new Rectangle(71, 206, 120, 27));
jButtonNEWAPP.setEnabled(false);
jButtonNEWAPP.setText("New Applicant");
jButtonNEWAPP.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent e) {
System.out.println("actionPerformed()"); // TODO Auto-generated Event stub
actionPerformed()
jTextFieldname.setText("");
jTextFieldage.setText("");
jTextfielddate.setText("");
jTextfielddfamily.setText("");
jTextfielddincome.setText("");
jLabelsaved.setVisible(false);
jLabelerror.setVisible(false);
}
});
}
return jButtonNEWAPP;
}

/**
This method initializes jPanelresident
*
@return javax.swing.JPanel
*/
private JPanel getJPanelresident() {
if (jPanelresident == null) {
jLabelmissing = new JLabel();
jLabelmissing.setBounds(new Rectangle(18, 227, 37, 15));
jLabelmissing.setForeground(Color.red);
jLabelmissing.setText("Error!");
jLabelmissing.setVisible(false);
jLabeltextshow = new JLabel();
jLabeltextshow.setBounds(new Rectangle(12, 210, 46, 16));
jLabeltextshow.setForeground(Color.red);
jLabeltextshow.setEnabled(true);
jLabeltextshow.setText("Saved!");
jLabeltextshow.setVisible(false);
jLabeltext66 = new JLabel();
jLabeltext66.setText("Rent:");
jLabeltext66.setLocation(new Point(7, 110));
jLabeltext66.setEnabled(false);
jLabeltext66.setSize(new Dimension(38, 16));
jLabelloc = new JLabel();
jLabelloc.setText("Location:");
jLabelloc.setLocation(new Point(7, 166));
}
}

```

```

jLabelloc.setEnabled(false);
jLabelloc.setSize(new Dimension(62, 16));
jLabelarea = new JLabel();
jLabelarea.setText("Area:");
jLabelarea.setLocation(new Point(7, 5));
jLabelarea.setEnabled(false);
jLabelarea.setSize(new Dimension(38, 16));
jPanelresident = new JPanel();
jPanelresident.setLayout(null);
jPanelresident.setBounds(new Rectangle(232, 9, 196, 244));
jPanelresident.setBackground(Color.white);
jPanelresident.setEnabled(false);
jPanelresident.add(jLabelarea, null);
jPanelresident.add(jLabelloc, null);
jPanelresident.add(jLabelText66, null);
jPanelresident.add(getJButtonADDRES(), null);
jPanelresident.add(getJRadioButtonA(), null);
jPanelresident.add(getJRadioButtonB(), null);
jPanelresident.add(getJRadioButtonC(), null);
jPanelresident.add(getJRadioButtonD(), null);
jPanelresident.add(getJRadioButtonE(), null);
jPanelresident.add(getJRadioButton600(), null);
jPanelresident.add(getJRadioButton800(), null);
jPanelresident.add(getJRadioButton1000(), null);
jPanelresident.add(getJRadioButton1200(), null);
jPanelresident.add(getJRadioButtonMORE1200(), null);
jPanelresident.add(getJRadioButtonrent1(), null);
jPanelresident.add(getJRadioButtonrent1047(), null);
jPanelresident.add(getJRadioButtonrent1047more(), null);
jPanelresident.add(getJRadioButtonrent750(), null);
jPanelresident.add(jLabelTextshow, null);
jPanelresident.add(jLabelmissing, null);
}
return jPanelresident;
}

/**
This method initializes jButtonassingAUTO
*
@return javax.swing.JButton
*/
private JButton getJButtonassingAUTO() {
if (jButtonassingAUTO == null) {
jButtonassingAUTO = new JButton();
jButtonassingAUTO.setBounds(new Rectangle(157, 34, 129, 29));
jButtonassingAUTO.setEnabled(false);
jButtonassingAUTO.setText("Auto Assign All");
jButtonassingAUTO.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent e) {
System.out.println("actionPerformed()"); // TODO Auto-generated Event stub
actionPerformed()
}
});
}
return jButtonassingAUTO;
}

int [][] ARRAYa = new int [3][ACounter];!--> age,family and income
//White #18
int [][] ARRAYr = new int [2][RCounter];!--> area and income
int [][] returnedResults = new int [1][ACounter];

```



```

for (int i=0; i <RCounter ;i++)
{
ARRAYr[0][i]=SecretlistRent.get(i);
ARRAYr[1][i]=SecretlistArea.get(i);
}

ResultsString = "Results: ";

for (int i=0; i <ACounter;i++)
{
ARRAYa[0][i]=SecretlistFamily.get(i);
ARRAYa[1][i]=SecretlistIncome.get(i); //Yellow #4
ARRAYa[2][i]=SecretlistAge.get(i);
}

returnedResults = kb.maincontroller(ARRAYa,ARRAYr,ACounter,RCounter);
for (int i=0; i<ACounter;i++) //Blue #3
{
ResultsString= ResultsString+ "ApplicantID: "+ returnedResults[0][i];
ResultsString= ResultsString+ "-ResidenceID: "+ returnedResults[1][i];
ResultsString= ResultsString+ " ----- ";
}
System.out.print(ResultsString+"\n");
jTextAreaResults.setText(ResultsString);
jLabelText.setVisible(true);
jLabelText77.setVisible(true);
jScrollPaneResults.setVisible(true);
jTextAreaResults.setVisible(true);

// DISPLAY RESULTS IN TEXT AREA RESULTS
}
});
}
return jButtonassingAUTO;
}

/**
This method initializes jScrollPaneapplicants
*
@return javax.swing.JScrollPane
*/
private JScrollPane getJScrollPaneapplicants() {
if (jScrollPaneapplicants == null) {
jScrollPaneapplicants = new JScrollPane();
jScrollPaneapplicants.setLocation(new Point(16, 95));
jScrollPaneapplicants.setSize(new Dimension(411, 40));
jScrollPaneapplicants.setViewportViewView(getJTextAreaapplicants());
}
return jScrollPaneapplicants;
}

/**
This method initializes jScrollPaneresidents
*
@return javax.swing.JScrollPane
*/
private JScrollPane getJScrollPaneresidents() {

```

```

if (jScrollPaneResidents == null) {
jScrollPaneResidents = new JScrollPane();
jScrollPaneResidents.setLocation(new Point(16, 163));
jScrollPaneResidents.setViewportViewView(getJTextAreaResidents());
jScrollPaneResidents.setSize(new Dimension(411, 37));
}
return jScrollPaneResidents;
}

/**
This method initializes jTextAreaApplicants
*
@return javax.swing.JTextArea
*/
private JTextArea getJTextAreaApplicants() {
if (jTextAreaApplicants == null) {
jTextAreaApplicants = new JTextArea();
jTextAreaApplicants.setEditable(false);
jTextAreaApplicants.setEnabled(false);
}
return jTextAreaApplicants;
}

/**
This method initializes jTextAreaResidents
*
@return javax.swing.JTextArea
*/
private JTextArea getJTextAreaResidents() {
if (jTextAreaResidents == null) {
jTextAreaResidents = new JTextArea();
jTextAreaResidents.setEditable(false);
jTextAreaResidents.setSize(new Dimension(408, 37));
jTextAreaResidents.setEnabled(false);
}
return jTextAreaResidents;
}

/**
This method initializes jButtonADDRESS
*
@return javax.swing.JButton
*/
private JButton getJButtonADDRESS() {
if (jButtonADDRESS == null) {
jButtonADDRESS = new JButton();
jButtonADDRESS.setBounds(new Rectangle(64, 212, 129, 28));
jButtonADDRESS.setEnabled(false);
jButtonADDRESS.setText("Add Residence");
jButtonADDRESS.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent e) {
System.out.println("actionPerformed()"); // TODO Auto-generated Event stub
actionPerformed()
}
});
}
return jButtonADDRESS;
}

if (jRadioButtonA.isSelected()==true)LOCATION="A";
else if (jRadioButtonB.isSelected()==true)LOCATION="B";
else if (jRadioButtonC.isSelected()==true)LOCATION="C";

```

```

else if(jRadioButtonD.isSelected()==true)LOCATION="D";
else if(jRadioButtonE.isSelected()==true)LOCATION="E";

if(jRadioButtonrent1.isSelected()==true){RENT="545"; rrr=1;}
else if(jRadioButtonrent1047.isSelected()==true){RENT="1047"; rrr=2;}
else if(jRadioButtonrent750.isSelected()==true){RENT="750"; rrr=3;}
else if(jRadioButtonrent1047more.isSelected()==true){RENT="1048"; rrr=4;}

if(jRadioButton600.isSelected()==true){AREA="600";aaa=1;}
else if(jRadioButton800.isSelected()==true){AREA="800";aaa=2;}
else if(jRadioButton1000.isSelected()==true){AREA="1000";aaa=3;}
else if(jRadioButton1200.isSelected()==true){AREA="1200";aaa=4;}
else if(jRadioButtonMORE1200.isSelected()==true){AREA="1201";aaa=5;}

if (AREA!=""&&RENT!=""&&LOCATION!="")
{LISTofRESIDENTS.add(RCounter+"-"+AREA+"-"+RENT+"-"+LOCATION);

SecretlistRent.add(rrr);
SecretlistArea.add(aaa);

RCounter++;
jLabeltextshow.setVisible(true);
System.out.print("LISTofRESIDENTS: ");
System.out.print(LISTofRESIDENTS);
jLabelmissing.setVisible(false);
jLabeltextshow.setVisible(true);}

else {jLabelmissing.setVisible(true);jLabeltextshow.setVisible(false);}
}
});
}
return jButtonADDRESS;
}

/**
This method initializes jRadioButtonA
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonA() {
if (jRadioButtonA == null) {
jRadioButtonA = new JRadioButton();
jRadioButtonA.setBounds(new Rectangle(72, 165, 38, 21));
jRadioButtonA.setBackground(Color.white);
jRadioButtonA.setEnabled(false);
jRadioButtonA.setText("A");
}
return jRadioButtonA;
}

/**
This method initializes jRadioButtonB
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonB() {
if (jRadioButtonB == null) {

```

```

jRadioButtonB = new JRadioButton();
jRadioButtonB.setText("B");
jRadioButtonB.setLocation(new Point(106, 165));
jRadioButtonB.setBackground(Color.white);
jRadioButtonB.setEnabled(false);
jRadioButtonB.setSize(new Dimension(33, 21));
}
return jRadioButtonB;
}

/**
This method initializes jRadioButtonC
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonC() {
if (jRadioButtonC == null) {
jRadioButtonC = new JRadioButton();
jRadioButtonC.setText("C");
jRadioButtonC.setLocation(new Point(137, 164));
jRadioButtonC.setBackground(Color.white);
jRadioButtonC.setEnabled(false);
jRadioButtonC.setSize(new Dimension(33, 23));
}
return jRadioButtonC;
}

/**
This method initializes jRadioButtonD
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonD() {
if (jRadioButtonD == null) {
jRadioButtonD = new JRadioButton();
jRadioButtonD.setBounds(new Rectangle(71, 186, 33, 24));
jRadioButtonD.setBackground(Color.white);
jRadioButtonD.setEnabled(false);
jRadioButtonD.setText("D");
}
return jRadioButtonD;
}

/**
This method initializes jRadioButtonE
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonE() {
if (jRadioButtonE == null) {
jRadioButtonE = new JRadioButton();
jRadioButtonE.setBounds(new Rectangle(103, 185, 33, 24));
jRadioButtonE.setBackground(Color.white);
jRadioButtonE.setEnabled(false);
jRadioButtonE.setText("E");
}
return jRadioButtonE;
}

```

```

}

/**
This method initializes jRadioButton600
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButton600() {
if (jRadioButton600 == null) {
jRadioButton600 = new JRadioButton();
jRadioButton600.setBackground(Color.white);
jRadioButton600.setSize(new Dimension(103, 21));
jRadioButton600.setLocation(new Point(46, 4));
jRadioButton600.setEnabled(false);
jRadioButton600.setText("0-600 sq");
}
return jRadioButton600;
}

/**
This method initializes jRadioButton800
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButton800() {
if (jRadioButton800 == null) {
jRadioButton800 = new JRadioButton();
jRadioButton800.setText("600-800 sq");
jRadioButton800.setLocation(new Point(46, 23));
jRadioButton800.setBackground(Color.white);
jRadioButton800.setEnabled(false);
jRadioButton800.setSize(new Dimension(101, 24));
}
return jRadioButton800;
}

/**
This method initializes jRadioButton1000
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButton1000() {
if (jRadioButton1000 == null) {
jRadioButton1000 = new JRadioButton();
jRadioButton1000.setText("800-1000 sq");
jRadioButton1000.setLocation(new Point(46, 44));
jRadioButton1000.setBackground(Color.white);
jRadioButton1000.setEnabled(false);
jRadioButton1000.setSize(new Dimension(103, 24));
}
return jRadioButton1000;
}

/**
This method initializes jRadioButton1200
*
@return javax.swing.JRadioButton

```

```

*/
private JRadioButton getJRadioButton1200() {
if (jRadioButton1200 == null) {
jRadioButton1200 = new JRadioButton();
jRadioButton1200.setText("1000-1200 sq");
jRadioButton1200.setLocation(new Point(46, 63));
jRadioButton1200.setBackground(Color.white);
jRadioButton1200.setEnabled(false);
jRadioButton1200.setSize(new Dimension(109, 24));
}
return jRadioButton1200;
}

/**
This method initializes jRadioButtonMORE1200
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonMORE1200() {
if (jRadioButtonMORE1200 == null) {
jRadioButtonMORE1200 = new JRadioButton();
jRadioButtonMORE1200.setText("1200++ sq");
jRadioButtonMORE1200.setLocation(new Point(46, 82));
jRadioButtonMORE1200.setBackground(Color.white);
jRadioButtonMORE1200.setEnabled(false);
jRadioButtonMORE1200.setSize(new Dimension(119, 24));
}
return jRadioButtonMORE1200;
}

/**
This method initializes jRadioButtonrent1
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonrent1() {
if (jRadioButtonrent1 == null) {
jRadioButtonrent1 = new JRadioButton();
jRadioButtonrent1.setText("E 545");
jRadioButtonrent1.setSize(new Dimension(65, 17));
jRadioButtonrent1.setLocation(new Point(46, 110));
jRadioButtonrent1.setEnabled(false);
jRadioButtonrent1.setBackground(Color.white);
}
return jRadioButtonrent1;
}

/**
This method initializes jRadioButtonrent750
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonrent750() {
if (jRadioButtonrent750 == null) {
jRadioButtonrent750 = new JRadioButton();
jRadioButtonrent750.setText("E 750");
jRadioButtonrent750.setBounds(new Rectangle(110, 110, 56, 18));
}
}

```

```

jRadioButtonrent750.setEnabled(false);
jRadioButtonrent750.setBackground(Color.white);
}
return jRadioButtonrent750;
}

/**
This method initializes jRadioButtonrent1047
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonrent1047() {
if (jRadioButtonrent1047 == null) {
jRadioButtonrent1047 = new JRadioButton();
jRadioButtonrent1047.setText("E 1047");
jRadioButtonrent1047.setSize(new Dimension(64, 18));
jRadioButtonrent1047.setLocation(new Point(46, 130));
jRadioButtonrent1047.setEnabled(false);
jRadioButtonrent1047.setBackground(Color.white);
}
return jRadioButtonrent1047;
}

/**
This method initializes jRadioButtonrent1047more
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonrent1047more() {
if (jRadioButtonrent1047more == null) {
jRadioButtonrent1047more = new JRadioButton();
jRadioButtonrent1047more.setText("E 1047++");
jRadioButtonrent1047more.setSize(new Dimension(78, 19));
jRadioButtonrent1047more.setLocation(new Point(110, 129));
jRadioButtonrent1047more.setEnabled(false);
jRadioButtonrent1047more.setBackground(Color.white);
}
return jRadioButtonrent1047more;
}

/**
This method initializes jTextFieldname
*
@return javax.swing.JTextField
*/
private JTextField getJTextFieldname() {
if (jTextFieldname == null) {
jTextFieldname = new JTextField();
jTextFieldname.setBounds(new Rectangle(45, 38, 172, 17));
jTextFieldname.setText("Name");
jTextFieldname.setEnabled(false);
}
return jTextFieldname;
}

/**
This method initializes jTextFieldage

```

```

*
@return javax.swing.JTextField
*/
private JTextField getJTextFieldage() {
    if (jTextFieldage == null) {
        jTextFieldage = new JTextField();
        jTextFieldage.setLocation(new Point(45, 62));
        jTextFieldage.setEnabled(false);
        jTextFieldage.setText("0");
        jTextFieldage.setSize(new Dimension(172, 17));
    }
    return jTextFieldage;
}

/**
This method initializes jTextFieldddate
*
@return javax.swing.JTextField
*/
private JTextField getJTextFieldddate() {
    if (jTextFieldddate == null) {
        jTextFieldddate = new JTextField();
        jTextFieldddate.setLocation(new Point(5, 104));
        jTextFieldddate.setEnabled(false);
        jTextFieldddate.setText("MM/DD/YY");
        jTextFieldddate.setSize(new Dimension(216, 17));
    }
    return jTextFieldddate;
}

/**
This method initializes jTextFielddfamily
*
@return javax.swing.JTextField
*/
private JTextField getJTextFielddfamily() {
    if (jTextFielddfamily == null) {
        jTextFielddfamily = new JTextField();
        jTextFielddfamily.setLocation(new Point(5, 144));
        jTextFielddfamily.setEnabled(false);
        jTextFielddfamily.setText("1");
        jTextFielddfamily.setSize(new Dimension(212, 17));
    }
    return jTextFielddfamily;
}

/**
This method initializes jTextFielddincome
*
@return javax.swing.JTextField
*/
private JTextField getJTextFielddincome() {
    if (jTextFielddincome == null) {
        jTextFielddincome = new JTextField();
        jTextFielddincome.setLocation(new Point(5, 185));
        jTextFielddincome.setEnabled(false);
        jTextFielddincome.setText("0");
    }
}

```



```

jTextFieldincome.setSize(new Dimension(212, 17));
}
return jTextFieldincome;
}

//ACCESS INTERFACES

private void enablemagazine(){
jLabelmagaziner.setEnabled(true);
jButtonPrintMAG.setEnabled(true);
jLabelmessgaeformagazine.setEnabled(true);
}
private void disablemagazine(){
jLabelmagaziner.setEnabled(false);
jButtonPrintMAG.setEnabled(false);
jLabelmessgaeformagazine.setEnabled(false);}

private void disabledatatypist(){
jPanelAssigner.setEnabled(false);
jLabeldatatypist.setEnabled(false);
jLabelname.setEnabled(false);
jLabelage.setEnabled(false);
jLabeldate.setEnabled(false);
jLabelincome.setEnabled(false);
jLabelmem.setEnabled(false);
jTextFielddage.setEnabled(false);
jTextFielddate.setEnabled(false);
jTextFielddfamilly.setEnabled(false);
jTextFielddincome.setEnabled(false);
jPanelresident.setEnabled(false);
jTextFielddname.setEnabled(false);
jButtonsaveandnew.setEnabled(false);
jButtonNEWAPP.setEnabled(false);
jLabelarea.setEnabled(false);
jLabeltext66.setEnabled(false);
jLabelloc.setEnabled(false);
jButtonADDRES.setEnabled(false);
jPanelresident.setEnabled(false);

jRadioButton1000.setEnabled(false);
jRadioButton600.setEnabled(false);
jRadioButton800.setEnabled(false);
jRadioButton1200.setEnabled(false);
jRadioButtonMORE1200.setEnabled(false);
jRadioButtonA.setEnabled(false);
jRadioButtonB.setEnabled(false);
jRadioButtonC.setEnabled(false);
jRadioButtonD.setEnabled(false);
jRadioButtonE.setEnabled(false);
jRadioButtonrent1.setEnabled(false);
jRadioButtonrent1047.setEnabled(false);
jRadioButtonrent1047more.setEnabled(false);
jRadioButtonrent750.setEnabled(false);
}

private void enabledatatypist(){
jPanelAssigner.setEnabled(true);

```

```

jLabeldatatypist.setEnabled(true);
jLabelname.setEnabled(true);
jLabelage.setEnabled(true);
jLabeldate.setEnabled(true);
jLabelincome.setEnabled(true);
jLabelmem.setEnabled(true);
jTextFielddage.setEnabled(true);
jTextFielddate.setEnabled(true);
jTextFielddfamily.setEnabled(true);
jTextFielddincome.setEnabled(true);
jPanelresident.setEnabled(true);
jTextFielddname.setEnabled(true);
jButtonsaveandnew.setEnabled(true);
jButtonNEWAPP.setEnabled(true);
jLabelarea.setEnabled(true);
jLabeltext66.setEnabled(true);
jLabelloc.setEnabled(true);
jButtonADDRES.setEnabled(true);
jPanelresident.setEnabled(true);

jRadioButton1000.setEnabled(true);
jRadioButton600.setEnabled(true);
jRadioButton800.setEnabled(true);
jRadioButton1200.setEnabled(true);
jRadioButtonMORE1200.setEnabled(true);
jRadioButtonA.setEnabled(true);
jRadioButtonB.setEnabled(true);
jRadioButtonC.setEnabled(true);
jRadioButtonD.setEnabled(true);
jRadioButtonE.setEnabled(true);
jRadioButtonrent1.setEnabled(true);
jRadioButtonrent1047.setEnabled(true);
jRadioButtonrent1047more.setEnabled(true);
jRadioButtonrent750.setEnabled(true);
}

private void disableassigner(){
jLabelassigner.setEnabled(false);
jButtonassingAUTO.setEnabled(false);
jLabellistapplicants.setEnabled(false);
jLabellistofresidences.setEnabled(false);
jScrollPaneResidents.setEnabled(false);
jScrollPaneapplicants.setEnabled(false);
jTextAreaapplicants.setEnabled(false);
jTextAreaREsidents.setEnabled(false);
jPanelDatatypist.setEnabled(false);
jButtongetapplicants.setEnabled(false);
jTextAreaResults.setEnabled(false);
jScrollPaneResults.setEnabled(false);
jLabeltext77.setEnabled(false);
}

private void enableassigner(){
jLabelassigner.setEnabled(true);
jButtonassingAUTO.setEnabled(true);
jLabellistapplicants.setEnabled(true);

```

```

jLabellistofresidences.setEnabled(true);
jScrollPaneResidents.setEnabled(true);
jScrollPaneApplicants.setEnabled(true);
jTextAreaApplicants.setEnabled(true);
jTextAreaResidents.setEnabled(true);
jPanelDatatypist.setEnabled(true);
jButtonGetApplicants.setEnabled(true);
jTextAreaResults.setEnabled(true);
jScrollPaneResults.setEnabled(true);
jLabelText77.setEnabled(true);
}

/**
This method initializes jTextFieldUsername
*
@return javax.swing.JTextField
*/
private JTextField getJTextFieldUsername() {
if (jTextFieldUsername == null) {
jTextFieldUsername = new JTextField();
jTextFieldUsername.setSize(new Dimension(165, 20));
jTextFieldUsername.setLocation(new Point(75, 48));
}
return jTextFieldUsername;
}

/**
This method initializes jPasswordFieldPass
*
@return javax.swing.JPasswordField
*/
private JPasswordField getJPasswordFieldPass() {
if (jPasswordFieldPass == null) {
jPasswordFieldPass = new JPasswordField();
jPasswordFieldPass.setLocation(new Point(75, 77));
jPasswordFieldPass.setSize(new Dimension(165, 20));
}
return jPasswordFieldPass;
}

/**
This method initializes jButtonGetApplicants
*
@return javax.swing.JButton
*/
private JButton getJButtonGetApplicants() {
if (jButtonGetApplicants == null) {
jButtonGetApplicants = new JButton();
jButtonGetApplicants.setLocation(new Point(15, 34));
jButtonGetApplicants.setText("Get All Data");
jButtonGetApplicants.setEnabled(false);
jButtonGetApplicants.setSize(new Dimension(129, 29));
jButtonGetApplicants.addActionListener(new java.awt.event.ActionListener()
{
public void actionPerformed(java.awt.event.ActionEvent e) {
System.out.println("actionPerformed()"); // TODO Auto-generated Event stub
actionPerformed()
}
}
}
}

```

```

jTextAreaapplicants.setText(LISTofAPPLICANTS+"");
jTextAreaREsidents.setText(LISTofRESIDENTS+"");
}
});
}
return jButtonongetapplicants;
}

/**
This method initializes jScrollPaneResults
*
@return javax.swing.JScrollPane
*/
private JScrollPane getJScrollPaneResults() {
if (jScrollPaneResults == null) {
jScrollPaneResults = new JScrollPane();
jScrollPaneResults.setBounds(new Rectangle(71, 210, 356, 46));
jScrollPaneResults.setVisible(false);
jScrollPaneResults.setViewportView(getJTextAreaResults());
}
return jScrollPaneResults;
}

/**
This method initializes JTextAreaResults
*
@return javax.swing.JTextArea
*/
private JTextArea getJTextAreaResults() {
if (jTextAreaResults == null) {
jTextAreaResults = new JTextArea();
jTextAreaResults.setBackground(Color.cyan);
jTextAreaResults.setVisible(false);
}
return jTextAreaResults;
}

/**
This method initializes JRadioButtonMagazineEditor
*
@return javax.swing.JRadioButton
*/
private JRadioButton getJRadioButtonMagazineEditor() {
if (jRadioButtonMagazineEditor == null) {
jRadioButtonMagazineEditor = new JRadioButton();
jRadioButtonMagazineEditor.setBackground(Color.white);
jRadioButtonMagazineEditor.setSize(new Dimension(109, 21));
jRadioButtonMagazineEditor.setLocation(new Point(247, 87));
jRadioButtonMagazineEditor.setText("Magazine Ed");
}
return jRadioButtonMagazineEditor;
}

/**
This method initializes JPanelMagazine
*
@return javax.swing.JPanel

```

```

*/
private JPanel getJPanelMagaziner() {
if (jPanelMagaziner == null) {
jLabelmessgaeformmagazine = new JLabel();
jLabelmessgaeformmagazine.setBounds(new Rectangle(167, 36, 237, 16));
jLabelmessgaeformmagazine.setText("Magazine printed to file. Check bin
folder.");
jLabelmessgaeformmagazine.setEnabled(false);
jLabelmessgaeformmagazine.setVisible(false);
jLabelmagaziner = new JLabel();
jLabelmagaziner.setBounds(new Rectangle(4, 4, 190, 19));
jLabelmagaziner.setFont(new Font("Dialog", Font.BOLD, 14));
jLabelmagaziner.setForeground(new Color(0, 153, 0));
jLabelmagaziner.setText("Magazine Editor's Screen:");
jLabelmagaziner.setEnabled(false);
jPanelMagaziner = new JPanel();
jPanelMagaziner.setLayout(null);
jPanelMagaziner.setBackground(Color.lightGray);
jPanelMagaziner.setLocation(new Point(7, 426));
jPanelMagaziner.setSize(new Dimension(436, 73));
jPanelMagaziner.add(jLabelmagaziner, null);
jPanelMagaziner.add(getJButtonPrintMAG(), null);
jPanelMagaziner.add(jLabelmessgaeformmagazine, null);
}
return jPanelMagaziner;
}

/**
This method initializes jButtonPrintMAG
*
@return javax.swing.JButton
*/
private JButton getJButtonPrintMAG() {
if (jButtonPrintMAG == null) {
jButtonPrintMAG = new JButton();
jButtonPrintMAG.setBounds(new Rectangle(30, 29, 131, 30));
jButtonPrintMAG.setEnabled(false);
jButtonPrintMAG.setText("Print Magazine");
jButtonPrintMAG.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent e) {
System.out.println("actionPerformed()"); // TODO Auto-generated Event stub
actionPerformed()

jLabelmessgaeformmagazine.setVisible(true);
// write console to notepad!!

FileWriter fWriter = null;
BufferedWriter writer = null;
try {
fWriter = new FileWriter("MAGAZINE.txt");
writer = new BufferedWriter(fWriter);
//WRITES 1 LINE TO FILE AND CHANGES LINE
writer.write("Begining of Magazine! \n \n");
writer.write(ResultsString);
writer.newLine();
writer.write("End of Magazine!");
writer.newLine();

```

```

writer.close();
} catch (Exception e1) {
}
}
});
}
return jButtonPrintMAG;
}

/**
This method initializes JTextAreacredits
*
@return javax.swing.JTextArea
*/
private JTextArea getJTextAreacredits() {
if (jTextAreacredits == null) {
jTextAreacredits = new JTextArea();
jTextAreacredits.setBounds(new Rectangle(452, 425, 436, 73));
jTextAreacredits.setToolTipText("Credits");
jTextAreacredits.setText(" This testing tool is developed for academic
research purposes at the \n University of Central Florida (UCF),
Intelligent Systems Lab (ISL).\n The housing application is used to
experiment MAVERICK. \n Developer: Feras A. Batarseh - 2010 ©");
jTextAreacredits.setEditable(false);
jTextAreacredits.setLineWrap(true);
jTextAreacredits.setWrapStyleWord(true);
jTextAreacredits.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC, 12));
}
return jTextAreacredits;
}

}

```

//End of Code

APPENDIX C: EXPERIMENTAL VALIDATION ITERATIONS

In this appendix, the test cases are sorted and displayed in the following format:

'TestCaseID'-'CommonKADSMModel'-'LocalImportance'-'NumberofRuns'-'ModelWeight'-'GlobalImportance'-'Result'

Iteration #1:

'16.0'-'3.0'-'5.0'-'1.0'-'50.0'-'250.0'-'2.0'-'
'17.0'-'3.0'-'5.0'-'1.0'-'50.0'-'250.0'-'2.0'-'
'18.0'-'3.0'-'5.0'-'1.0'-'50.0'-'250.0'-'2.0'-'
'19.0'-'3.0'-'5.0'-'1.0'-'50.0'-'250.0'-'2.0'-'
'24.0'-'4.0'-'5.0'-'1.0'-'50.0'-'250.0'-'2.0'-'
'25.0'-'4.0'-'5.0'-'1.0'-'50.0'-'250.0'-'2.0'-'
'26.0'-'4.0'-'5.0'-'1.0'-'50.0'-'250.0'-'2.0'-'
'27.0'-'4.0'-'5.0'-'1.0'-'50.0'-'250.0'-'2.0'-'
'34.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'35.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'36.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'37.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'38.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'39.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'40.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'41.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'42.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'43.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'44.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'45.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'46.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'
'47.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-'

'48.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-
'49.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-
'50.0'-'4.0'-'4.75'-'1.0'-'50.0'-'237.5'-'2.0'-
'51.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'52.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'53.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'54.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'55.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'56.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'57.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'58.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'59.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'60.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'61.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'62.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'63.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'64.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'65.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'66.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'67.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'68.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'69.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'70.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'71.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'72.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'73.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'74.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-

'75.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'76.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'77.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'78.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'79.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'80.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'81.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'82.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'83.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'84.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'85.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'86.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'87.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'88.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'89.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'90.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'91.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'92.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'93.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'94.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'95.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'96.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'97.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'98.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'99.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'100.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'101.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-

'102.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'103.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'104.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'105.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'106.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'107.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'108.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'109.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'110.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'111.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'112.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'113.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'114.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'115.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'116.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'117.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'118.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'119.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'120.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'121.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'122.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'123.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'124.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'125.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'126.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'127.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'128.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-

'129.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'130.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'131.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'132.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'133.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'134.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'135.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'136.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'137.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'138.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'139.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'140.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'141.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'142.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'143.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'144.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'145.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'146.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'147.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'148.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'149.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'150.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'151.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'152.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'153.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'154.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'155.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-

'156.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'157.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'158.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'159.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'160.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'161.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'162.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'163.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'164.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'165.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'166.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'167.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'168.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'169.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'170.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'171.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'172.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'173.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'174.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'175.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'176.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'177.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'178.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'179.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'180.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'181.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-
'182.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-

'183.0'-'4.0'-'4.75'-'0.0'-'50.0'-'237.5'-'2.0'-'
'10.0'-'2.0'-'3.75'-'0.0'-'50.0'-'187.5'-'2.0'-'
'11.0'-'2.0'-'3.75'-'0.0'-'50.0'-'187.5'-'2.0'-'
'12.0'-'2.0'-'3.75'-'0.0'-'50.0'-'187.5'-'2.0'-'
'13.0'-'2.0'-'3.75'-'0.0'-'50.0'-'187.5'-'2.0'-'
'14.0'-'2.0'-'3.75'-'0.0'-'50.0'-'187.5'-'2.0'-'
'15.0'-'2.0'-'3.75'-'0.0'-'50.0'-'187.5'-'2.0'-'
'28.0'-'5.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'29.0'-'5.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'30.0'-'5.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'31.0'-'5.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'32.0'-'5.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'33.0'-'5.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'184.0'-'1.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'185.0'-'1.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'186.0'-'3.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'187.0'-'4.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'188.0'-'4.0'-'3.5'-'0.0'-'50.0'-'175.0'-'2.0'-'
'1.0'-'1.0'-'2.75'-'0.0'-'50.0'-'137.5'-'2.0'-'
'2.0'-'1.0'-'2.75'-'0.0'-'50.0'-'137.5'-'2.0'-'
'3.0'-'1.0'-'2.75'-'0.0'-'50.0'-'137.5'-'2.0'-'
'4.0'-'1.0'-'2.75'-'0.0'-'50.0'-'137.5'-'2.0'-'
'5.0'-'1.0'-'2.75'-'0.0'-'50.0'-'137.5'-'2.0'-'
'6.0'-'1.0'-'2.75'-'0.0'-'50.0'-'137.5'-'2.0'-'
'7.0'-'1.0'-'2.75'-'0.0'-'50.0'-'137.5'-'2.0'-'
'8.0'-'1.0'-'2.75'-'0.0'-'50.0'-'137.5'-'2.0'-'
'9.0'-'1.0'-'2.75'-'0.0'-'50.0'-'137.5'-'2.0'-'

'20.0'-'3.0'-'0.5'-'0.0'-'50.0'-'25.0'-'2.0'-
'21.0'-'3.0'-'0.5'-'0.0'-'50.0'-'25.0'-'2.0'-
'22.0'-'3.0'-'0.5'-'0.0'-'50.0'-'25.0'-'2.0'-
'23.0'-'3.0'-'0.5'-'0.0'-'50.0'-'25.0'-'2.0'-

Iteration #2:

'24.0'-'4.0'-'5.0'-'1.0'-'87.0'-'435.0'-'1.0'-
'25.0'-'4.0'-'5.0'-'1.0'-'87.0'-'435.0'-'1.0'-
'26.0'-'4.0'-'5.0'-'1.0'-'87.0'-'435.0'-'1.0'-
'27.0'-'4.0'-'5.0'-'1.0'-'87.0'-'435.0'-'1.0'-
'34.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'35.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'36.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'37.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'38.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'39.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'40.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'41.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'42.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'43.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'44.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'45.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'46.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'47.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'48.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'49.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'50.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'1.0'-
'51.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-

'52.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'53.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'54.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'55.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'56.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'57.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'58.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'59.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'60.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'61.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'62.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'63.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'64.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'65.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'66.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'67.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'68.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'69.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'70.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'71.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'72.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'73.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'74.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'75.0'-'4.0'-'4.75'-'1.0'-'87.0'-'413.25'-'2.0'-
'76.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'77.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'78.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-

'79.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'80.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'81.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'82.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'83.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'84.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'85.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'86.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'87.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'88.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'89.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'90.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'91.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'92.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'93.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'94.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'95.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'96.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'97.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'98.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'99.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'100.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'101.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'102.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'103.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'104.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'105.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-

'106.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'107.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'108.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'109.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'110.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'111.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'112.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'113.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'114.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'115.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'116.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'117.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'118.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'119.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'120.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'121.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'122.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'123.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'124.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'125.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'126.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'127.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'128.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'129.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'130.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'131.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'132.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-

'133.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'134.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'135.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'136.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'137.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'138.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'139.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'140.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'141.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'142.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'143.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'144.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'145.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'146.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'147.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'148.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'149.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'150.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'151.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'152.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'153.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'154.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'155.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'156.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'157.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'158.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'159.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-

'160.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'161.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'162.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'163.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'164.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'165.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'166.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'167.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'168.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'169.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'170.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'171.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'172.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'173.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'174.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'175.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'176.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'177.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'178.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'179.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'180.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'181.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'182.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'183.0'-'4.0'-'4.75'-'0.0'-'87.0'-'413.25'-'2.0'-
'10.0'-'2.0'-'3.75'-'0.0'-'100.0'-'375.0'-'2.0'-
'11.0'-'2.0'-'3.75'-'0.0'-'100.0'-'375.0'-'2.0'-
'12.0'-'2.0'-'3.75'-'0.0'-'100.0'-'375.0'-'2.0'-

'13.0'-'2.0'-'3.75'-'0.0'-'100.0'-'375.0'-'2.0'-
'14.0'-'2.0'-'3.75'-'0.0'-'100.0'-'375.0'-'2.0'-
'15.0'-'2.0'-'3.75'-'0.0'-'100.0'-'375.0'-'2.0'-
'28.0'-'5.0'-'3.5'-'0.0'-'100.0'-'350.0'-'2.0'-
'29.0'-'5.0'-'3.5'-'0.0'-'100.0'-'350.0'-'2.0'-
'30.0'-'5.0'-'3.5'-'0.0'-'100.0'-'350.0'-'2.0'-
'31.0'-'5.0'-'3.5'-'0.0'-'100.0'-'350.0'-'2.0'-
'32.0'-'5.0'-'3.5'-'0.0'-'100.0'-'350.0'-'2.0'-
'33.0'-'5.0'-'3.5'-'0.0'-'100.0'-'350.0'-'2.0'-
'184.0'-'1.0'-'3.5'-'0.0'-'100.0'-'350.0'-'2.0'-
'185.0'-'1.0'-'3.5'-'0.0'-'100.0'-'350.0'-'2.0'-
'187.0'-'4.0'-'3.5'-'0.0'-'87.0'-'304.5'-'2.0'-
'188.0'-'4.0'-'3.5'-'0.0'-'87.0'-'304.5'-'2.0'-
'16.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'17.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'18.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'19.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'1.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'2.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'3.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'4.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'5.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'6.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'7.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'8.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'9.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'186.0'-'3.0'-'3.5'-'0.0'-'56.0'-'196.0'-'2.0'-

'20.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'21.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'22.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'23.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-

Iteration #3:

'10.0'-'2.0'-'3.75'-'1.0'-'100.0'-'375.0'-'2.0'-
'11.0'-'2.0'-'3.75'-'1.0'-'100.0'-'375.0'-'2.0'-
'12.0'-'2.0'-'3.75'-'1.0'-'100.0'-'375.0'-'2.0'-
'13.0'-'2.0'-'3.75'-'1.0'-'100.0'-'375.0'-'2.0'-
'14.0'-'2.0'-'3.75'-'1.0'-'100.0'-'375.0'-'2.0'-
'15.0'-'2.0'-'3.75'-'1.0'-'100.0'-'375.0'-'2.0'-
'24.0'-'4.0'-'5.0'-'1.0'-'71.0'-'355.0'-'1.0'-
'25.0'-'4.0'-'5.0'-'1.0'-'71.0'-'355.0'-'1.0'-
'26.0'-'4.0'-'5.0'-'1.0'-'71.0'-'355.0'-'1.0'-
'27.0'-'4.0'-'5.0'-'1.0'-'71.0'-'355.0'-'1.0'-
'28.0'-'5.0'-'3.5'-'1.0'-'100.0'-'350.0'-'2.0'-
'29.0'-'5.0'-'3.5'-'1.0'-'100.0'-'350.0'-'2.0'-
'30.0'-'5.0'-'3.5'-'1.0'-'100.0'-'350.0'-'2.0'-
'31.0'-'5.0'-'3.5'-'1.0'-'100.0'-'350.0'-'2.0'-
'32.0'-'5.0'-'3.5'-'1.0'-'100.0'-'350.0'-'2.0'-
'33.0'-'5.0'-'3.5'-'1.0'-'100.0'-'350.0'-'2.0'-
'184.0'-'1.0'-'3.5'-'1.0'-'100.0'-'350.0'-'2.0'-
'185.0'-'1.0'-'3.5'-'1.0'-'100.0'-'350.0'-'2.0'-
'34.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'35.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'36.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'37.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-

'38.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'39.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'40.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'41.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'42.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'43.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'44.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'45.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'46.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'47.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'48.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'49.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'50.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'51.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'52.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'53.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'54.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'55.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'56.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'57.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'58.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'59.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'60.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'61.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'62.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'63.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-
'64.0'-4.0'-4.75'-1.0'-71.0'-337.25'-1.0'-

'65.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'66.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'67.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'68.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'69.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'70.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'71.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'72.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'73.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'74.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'75.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'1.0'-
'76.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'77.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'78.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'79.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'80.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'81.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'82.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'83.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'84.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'85.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'86.0'-'4.0'-'4.75'-'1.0'-'71.0'-'337.25'-'2.0'-
'87.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'88.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'89.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'90.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'91.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-

'92.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'93.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'94.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'95.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'96.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'97.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'98.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'99.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'100.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'101.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'102.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'103.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'104.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'105.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'106.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'107.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'108.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'109.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'110.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'111.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'112.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'113.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'114.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'115.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'116.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'117.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'118.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-

'119.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'120.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'121.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'122.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'123.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'124.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'125.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'126.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'127.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'128.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'129.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'130.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'131.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'132.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'133.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'134.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'135.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'136.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'137.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'138.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'139.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'140.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'141.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'142.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'143.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'144.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'145.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-

'146.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'147.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'148.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'149.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'150.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'151.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'152.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'153.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'154.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'155.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'156.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'157.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'158.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'159.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'160.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'161.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'162.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'163.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'164.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'165.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'166.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'167.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'168.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'169.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'170.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'171.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'172.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-

'173.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'174.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'175.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'176.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'177.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'178.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'179.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'180.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'181.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'182.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'183.0'-'4.0'-'4.75'-'0.0'-'71.0'-'337.25'-'2.0'-
'16.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'17.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'18.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'19.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'1.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'2.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'3.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'4.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'5.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'6.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'7.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'8.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'9.0'-'1.0'-'2.75'-'0.0'-'100.0'-'275.0'-'2.0'-
'187.0'-'4.0'-'3.5'-'0.0'-'71.0'-'248.5'-'2.0'-
'188.0'-'4.0'-'3.5'-'0.0'-'71.0'-'248.5'-'2.0'-
'186.0'-'3.0'-'3.5'-'0.0'-'56.0'-'196.0'-'2.0'-

'20.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'21.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'22.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'23.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-

Iteration #4:

'24.0'-'4.0'-'5.0'-'1.0'-'64.0'-'320.0'-'1.0'-
'25.0'-'4.0'-'5.0'-'1.0'-'64.0'-'320.0'-'1.0'-
'26.0'-'4.0'-'5.0'-'1.0'-'64.0'-'320.0'-'1.0'-
'27.0'-'4.0'-'5.0'-'1.0'-'64.0'-'320.0'-'1.0'-
'34.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'35.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'36.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'37.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'38.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'39.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'40.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'41.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'42.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'43.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'44.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'45.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'46.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'47.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'48.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'49.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'50.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'51.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-

'52.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'53.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'54.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'55.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'56.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'57.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'58.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'59.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'60.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'61.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'62.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'63.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'64.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'65.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'66.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'67.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'68.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'69.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'70.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'71.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'72.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'73.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'74.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'75.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'76.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'77.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'78.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-

'79.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'80.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'81.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'82.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'83.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'84.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'85.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'86.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'1.0'-
'87.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'88.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'89.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'90.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'91.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'92.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'93.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'94.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'95.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'96.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'97.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'98.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'99.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'100.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'101.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'102.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'103.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'104.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'105.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-

'106.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'107.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'108.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'109.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'110.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'111.0'-'4.0'-'4.75'-'1.0'-'64.0'-'304.0'-'2.0'-
'112.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'113.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'114.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'115.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'116.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'117.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'118.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'119.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'120.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'121.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'122.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'123.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'124.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'125.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'126.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'127.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'128.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'129.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'130.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'131.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'132.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-

'133.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'134.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'135.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'136.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'137.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'138.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'139.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'140.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'141.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'142.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'143.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'144.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'145.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'146.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'147.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'148.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'149.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'150.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'151.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'152.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'153.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'154.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'155.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'156.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'157.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'158.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'159.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-

'160.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'161.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'162.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'163.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'164.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'165.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'166.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'167.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'168.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'169.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'170.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'171.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'172.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'173.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'174.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'175.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'176.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'177.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'178.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'179.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'180.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'181.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'182.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'183.0'-'4.0'-'4.75'-'0.0'-'64.0'-'304.0'-'2.0'-
'184.0'-'1.0'-'3.5'-'1.0'-'82.0'-'287.0'-'1.0'-
'185.0'-'1.0'-'3.5'-'1.0'-'82.0'-'287.0'-'1.0'-
'16.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-

'17.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'18.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'19.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'1.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'2.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'3.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'4.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'5.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'6.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'7.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'8.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'9.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'187.0'-'4.0'-'3.5'-'0.0'-'64.0'-'224.0'-'2.0'-
'188.0'-'4.0'-'3.5'-'0.0'-'64.0'-'224.0'-'2.0'-
'186.0'-'3.0'-'3.5'-'0.0'-'56.0'-'196.0'-'2.0'-
'20.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'21.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'22.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'23.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'10.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'11.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'12.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'13.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'14.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'15.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'28.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-
'29.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-

'30.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-
'31.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-
'32.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-
'33.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-

Iteration #5:

'184.0'-'1.0'-'3.5'-'1.0'-'82.0'-'287.0'-'1.0'-
'185.0'-'1.0'-'3.5'-'1.0'-'82.0'-'287.0'-'1.0'-
'16.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'17.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'18.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'19.0'-'3.0'-'5.0'-'1.0'-'56.0'-'280.0'-'1.0'-
'24.0'-'4.0'-'5.0'-'1.0'-'48.0'-'240.0'-'1.0'-
'25.0'-'4.0'-'5.0'-'1.0'-'48.0'-'240.0'-'1.0'-
'26.0'-'4.0'-'5.0'-'1.0'-'48.0'-'240.0'-'1.0'-
'27.0'-'4.0'-'5.0'-'1.0'-'48.0'-'240.0'-'1.0'-
'34.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'35.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'36.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'37.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'38.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'39.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'40.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'41.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'42.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'43.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'44.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'45.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-

'46.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'47.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'48.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'49.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'50.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'51.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'52.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'53.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'54.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'55.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'56.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'57.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'58.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'59.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'60.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'61.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'62.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'63.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'64.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'65.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'66.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'67.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'68.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'69.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'70.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'71.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'72.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-

'73.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'74.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'75.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'76.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'77.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'78.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'79.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'80.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'81.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'82.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'83.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'84.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'85.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'86.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'87.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'88.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'89.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'90.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'91.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'92.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'93.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'94.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'95.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'96.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'97.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'98.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-
'99.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'1.0'-

'100.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'101.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'102.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'103.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'104.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'105.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'106.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'107.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'108.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'109.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'110.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'111.0'-4.0'-4.75'-1.0'-48.0'-228.0'-1.0'-
'112.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'113.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'114.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'115.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'116.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'117.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'118.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'119.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'120.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'121.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'122.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'123.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'124.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'125.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-
'126.0'-4.0'-4.75'-1.0'-48.0'-228.0'-2.0'-

'127.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'2.0'-
'128.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'2.0'-
'129.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'2.0'-
'130.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'2.0'-
'131.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'2.0'-
'132.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'2.0'-
'133.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'2.0'-
'134.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'2.0'-
'135.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'2.0'-
'136.0'-'4.0'-'4.75'-'1.0'-'48.0'-'228.0'-'2.0'-
'137.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'138.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'139.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'140.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'141.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'142.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'143.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'144.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'145.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'146.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'147.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'148.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'149.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'150.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'151.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'152.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'153.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-

'154.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'155.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'156.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'157.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'158.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'159.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'160.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'161.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'162.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'163.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'164.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'165.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'166.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'167.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'168.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'169.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'170.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'171.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'172.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'173.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'174.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'175.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'176.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'177.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'178.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'179.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'180.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-

'181.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'182.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'183.0'-'4.0'-'4.75'-'0.0'-'48.0'-'228.0'-'2.0'-
'1.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'2.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'3.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'4.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'5.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'6.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'7.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'8.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'9.0'-'1.0'-'2.75'-'0.0'-'82.0'-'225.5'-'2.0'-
'186.0'-'3.0'-'3.5'-'0.0'-'56.0'-'196.0'-'2.0'-
'187.0'-'4.0'-'3.5'-'0.0'-'48.0'-'168.0'-'2.0'-
'188.0'-'4.0'-'3.5'-'0.0'-'48.0'-'168.0'-'2.0'-
'20.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'21.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'22.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'23.0'-'3.0'-'0.5'-'0.0'-'56.0'-'28.0'-'2.0'-
'10.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'11.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'12.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'13.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'14.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'15.0'-'2.0'-'3.75'-'1.0'-'0.0'-'0.0'-'1.0'-
'28.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-
'29.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-

'30.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-

'31.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-

'32.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-

'33.0'-'5.0'-'3.5'-'1.0'-'0.0'-'0.0'-'1.0'-

END OF ITERATIONS

LIST OF REFERENCES

1. Sommerville, I. "Software Engineering" 8th edition, 2007, Chapter 4, published by Addison Wesley.
2. Dowson, M. "The Ariane 5 Software Failure", Proceedings of the ACM SIGSOFT software engineering note, Volume 22, Issue 2, March 1997, pp. 84
3. N. Leveson and C. Turner "An Investigation of the Therac-25 Accident", Proceedings of the IEEE Computer, Vol. 26, No. 7, July 1993, pp. 18-41
4. Clancy, T. "Chaos: A Standish Group Report", a report by the Standish Research Group, Year 1995.
5. Abdallah, K. Mohammad, T. and Louis, F. "Validation of Intelligent Systems: a Critical Study and a tool, Corus" Proceedings of the International Journal of Soft Computing 2007, pp.191-198.
6. The Merriam Webster Dictionary, Merriam Webster Incorporated 2009
<http://www.merriam-webster.com/>
7. Luger, G. F. "Artificial Intelligence: structures and strategies for complex problem solving" Fourth edition, Published by Addison Wesley 2002
8. Gonzalez, A.J. and Dankel, D. "The Engineering of Knowledge-Based Systems, Theory and Practice" published by Prentice Hall 1993
9. Schreiber, G. Akkermans, H. Anjewierden, A. De Hoog, R. Shadbolt, N. Van De Velde, W. and Wielinga, B. "Knowledge Engineering and Management-The CommonKADS Methodology", published by The MIT Press 2000

10. Anderson, C. Thelin, T. Runeson, P. Dzamashvili, N. "An Experimental Evaluation of Inspection and testing for Detection of Design Faults", Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE '03)
11. Adrion W., Branstad, M. and Cherniavski, J. "Validation, verification and testing of computer software". 1989 IEEE Transactions on Systems, Man and Cybernetics, 21 (2), pp. 293-301
12. Dolores, R. Wallace, R. and Fuji, R. "Software Verification and Validation: An Overview", Proceedings of IEEE Software, v.6 n.3, p.10-17
13. Zlatareva, N. "A framework for knowledge-based system validation, verification and refinement: the VVR system" Proceedings of FLAIRS-1992, Ft. Lauderdale, FL, pp. 10-14.
14. O'Keefe, R. , Balci, O. and Smith, E. "Validating Expert System Performance", IEEE 1987, Proceedings of the IEEE Expert, Volume 2, pp.81-90
15. US Department of Defense Directive DoDD 5000.59, 199x
16. Min, F. Ma, P. and Yang, M. "A Knowledge Based Method for the Validation of Military Simulation", Proceedings of the 2007 Winter Simulation Conference- IEEE, pp.1395-1402
17. IEEE Std. 610.12-1990, Glossary of Software Engineering Terminology, 1990
18. Gonzalez, A. J. and Barr, V. "Validation and Verification of Intelligent Systems – what are they and how are they different?" Proceedings of the Journal of Experimental & Theoretical Artificial Intelligence, October-2000, pp.407-420
19. Preece, A. "Evaluating Verification and Validation Methods in Knowledge Engineering" Micro-Level Knowledge Management, 2001, pp. 123-145

20. Balci, O. "Validation, Verification and Testing Techniques throughout the Life Cycle of a Simulation Study", Proceedings of the 1994 Winter Simulation Conference, pp.215-220
21. Karlsson, D. Eles, P. and Peng, Z. "Formal Verification in a Component-based Reuse Methodology" Proceedings of the ISSS conference 2002, pp. 156-161
22. Tadji, C. and Laroussi, T. "Dynamic Verification of an Object-Rule Knowledge base Using Colored Petri Nets" Proceedings of the conference of Systemic, Cybernetics and Informatics 2009, Volume 4, pp. 337-352
23. Wu, Q. Zhou, C. Wu, J. and Wang, C. "Study on Knowledge base Verification Based on Petri Nets" Proceedings of the International Conference on Control and Automation 2005, pp. 997-1001
24. Morell, L. "Use of Meta Knowledge in the Verification of Knowledge-Based Systems" Proceedings of the 1st International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems 1988, pp. 847-857
25. Marcos, M. Del Pobil, D. and Moisan, S. "Model-Based Verification of Knowledge-Based Systems: a Case Study" Proceedings of the IEEE Journal on Software 2000, pp. 163-167
26. Korany, A. Shaalan, K. Baraka, H. and Rafea, A. "An Approach for Automating the Verification of KADS-Based Expert Systems" Proceedings of the International Conference on Applied Informatics and Communications- (WSEAS)-2007, pp. 1-22
27. Tadji, C. and Laroussi, T. "Dynamic Verification of an Object-Rule Knowledge base Using Colored Petri Nets", Proceedings of the conference of Systemic, Cybernetics and Informatics 2009, Volume 4, pp. 337-352

28. Levy, A. and Rousset, M. C. "Verification of Knowledge bases Based on Containment Checking", Proceedings of the Journal of Artificial Intelligence - 1998, pp. 227-250
29. Al Korany, A., Shaalan, K., Baraka, H., and Rafea, A. "An Approach for Automating the Verification of KADS-Based Expert Systems", Proceedings of the 7th International Conference on Applied Informatics and Communications- (WSEAS)-2007, pp. 1-22
30. De Rougemont, M. "Random Verification of Knowledge-Based Systems with Uncertainty" Proceedings of the AAAI Conference, 1993, pp. 9-11
31. Murrel, S. and Plant, R. "A survey of Tools for the Validation and Verification of Knowledge-Based Systems: 1985-1995" Proceedings of the Decision Support Systems – 1997, pp. 307-323
32. Culbert, C. and Savely, R. "Expert system verifications & Validation" Proceedings of First AAAI Workshop on VV and Testing. Palo Alto, CA August 1988.
33. Williamson, K. and Dahl, M. "Knowledge- based reduction for verifying Rule Bases Containing Equations" In: Working Notes: 6 Workshop on V.V of KBS at AAAI, Washington, D.C., pp 66-71
34. Rousset, M. "On the consistency of Knowledge-bases: the COVADIS system" Proceedings of the ECA188 Conference pp 79-84, Manchen, Germany 1988
35. Cragun, B. Steudel, H. "A Decision Table Based Processor for Checking Completeness and Consistency" International Journal of Man Machine Studies, No. 26, 1987, pp633-648.
36. Coener, F. Berch-Capon, T. and Kent, A. "A Binary Encoded Incidence Matrix Representation to Support KBS Verification" Proceedings of the Working Notes 7 Workshop on V&V of KBS at AAAI 94, pp. 84-93

37. Zhang, D. and Nguyen, D. "PREPARE: A Tool for Knowledge-base Verification"
IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 6. December 1994,
pp983-989
38. Preece, A. Shinghal, R. and Batarekh, A. "Principles and Practice in Verifying Rule-
Based Systems", Proceedings of the Journal of Knowledge Engineering Review, Vol. 7,
No. 2, 1992. pp115-141.
39. Yu, X. and Biswas "CHECKER: An Efficient Algorithm for Knowledge-Based System
Verification" In, Proceedings of the Third International Conference on Engineering
Applications of A.I. IEA/AIE-90. 1990
40. Stachowitz, R. Chang, C. Stock, T. and Coombs, J. "Building Validation Tools for
Knowledge-based Systems" First Annual Workshop on Space Operations Automation
and Robotics (SOAR '87) Houston, Texas. August 5-7, 1987, pp. 209-216
41. Loiseau, S. "A Method for Checking and Restoring the Consistency of Rule bases"
Proceedings of the Int. Journal of Human-Computer Studies, 1994, 40, pp.425-442
42. Morell, L. "Use of Meta Knowledge in the Verification of Knowledge-Based Systems",
Proceedings of the 1st International Conference on Industrial and Engineering
Applications of Artificial Intelligence and Expert Systems 1988, pp. 847-857
43. Levy, A. and Rousset, M. "Verification of Knowledge bases Based on Containment
Checking" Proceedings of the Journal of Artificial Intelligence - 1998, pp. 227-250
44. Lee, S. and O'Keefe, R. "Developing a Strategy for Expert System Validation and
Verification", IEEE 1994, Proceedings of the IEEE Transaction on systems, Man and
Cybernetics, Volume 24, pp.643-655

45. Langlotz, C. Shortliffe, E.H. and Fagan, L.M. "Using Decision Theory to Justify Heuristics", Proceedings of the AAAI conference 1986, pp. 215-219
46. Slafer, L. (Boeing Satellite Systems) "Achieving Software Validation through Simulation", Proceedings of the Advanced Dynamics International User Conference (ADIUS), 2001.
47. Knauf, R. Tsuruta, S. and Gonzalez, A.J. "Toward Reducing Human Involvement in Validation of Knowledge- Based Systems", Proceedings of the IEEE transaction on Systems, Man and Cybernetics, Volume 37- January 2007, pp.120-131
48. O'Keefe, R. M. Balci, O. and Smith, E.P. "Validating Expert System Performance" Proceedings of the IEEE Expert, Volume 2, Issue 4, Jan. 1987 pp.81 – 90
49. Turing, A. "Computing machinery and intelligence" Proceedings of *Mind* 1950, LIX, Number 236, pp. 433-460
50. Onoyama, T. Oyanagi, K., Kubota, S. and Tsuruta, S. "Concept of Validation and Its Tools for Intelligent Systems", IEEE 2000, The Proceedings of the Digital Object Identifier, TENCON 2000, pp.394-399
51. Min, F. Yang, M. and Wang, Z. "An Intelligent Validation System of Simulation Model", Proceedings of the Fifth International IEEE Conference on Machine Learning and Cybernetics. August 2006, pp. 1459-1464
52. Zlatareva, N. "A Framework for Knowledge-Based Systems Verification, Validation and Refinement: The VVR System" Proceedings of the 5th FLAIRS conference 1992, pp.10-14

53. Kelbasa, H. "Context Refinement- Investigating the Rule Refinement Completeness of SEEK/SEEK 2" Proceedings of the 15th European Conference on Artificial Intelligence, 2004, pp. 123-154
54. Mengshoel, O. J. and Delab, S. "Knowledge Validation: Principles and Practice" Proceedings of the Journal of IEEE Experts Systems, 1993, pp. 62-68
55. Santos Jr., E and Dinh, H. "Consistency of Test Case in Validation of Bayesian Knowledge Bases", Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence – ICTAI 2004.
56. Ginsberg, A. Weiss, S. and Politakis, P. "SEEK2: A Generalized Approach to Automatic Knowledge-base Refinement" Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1985, pp. 367-374
57. Dibia-Barthelemy, J. Haemmerle, O. and Salvat, E. "A Semantic Validation of Conceptual Graphs" Proceedings of the journal of knowledge-based systems. 2006, pp. 498-511
58. Smith, S. and Kandel, A. "Validation of Expert Systems" Proceedings of the Third Florida Artificial Intelligence Research Symposium (FLAIRS) 1990, pp.197-201
59. Vinze, A. Vogel, D. and Nunamaker, J. "Performance evaluation of a knowledge-based system, a Validation Study" Proceedings of the Journal of Information and Management 1991, Elsevier Science Publishers, pp.225-236
60. Onoyama, T. and Tsuruta, S. "Validation Method for Intelligent Systems", Proceedings of the Journal of Experimental and Theoretical Artificial Intelligence 2000, pp.461-472

61. Min, F. Ma, P. and Yang, M. "A Knowledge Base Method for the Validation of Military Simulation", Proceedings of the 2007 Winter Simulation Conference- IEEE, pp.1395-1402.
62. Wu, C. and Lee, S. "KJ3- a tool assisting formal validation of knowledge-based systems" Proceedings of the Int. J. Human-Computer Studies, 2002, pp. 495-525
63. Gonzalez, A.J. Gupta, U. and Chianese, R. "Performance Evaluation of a Large Diagnostic Expert System Using a Heuristic Test Case Generator", Proceedings of the Engineering Applications for Artificial Intelligence 1996, Volume 9, pp. 275-284.
64. Abel, T., Knauf, R. and Gonzalez, A. J. "Generation of a minimal set of test cases that is functionally equivalent to an exhaustive set, for use in knowledge-based system validation" Proceedings of the 9th FLAIRS conference 1996, pp. 280-284
65. Abel, T. and Gonzalez, A. J. "Utilizing Criteria to Reduce a Set of Test Cases for Expert System Validation" Proceedings of the 10th FLAIRS conference 1997, pp.402-406
66. Mosquiera-Rey, E. and Moret-Bonillo, V. "Validation of Intelligent Systems: A Critical Study and a Tool" Proceedings of the Journal of Expert Systems with Applications, 2000, pp.1-16
67. Zlatareva, N. " Knowledge Refinement During Development and Field Validaiton of Expert Systems" Proceedings of the 11th International FLAIRS conference, 1998, pp. 467- 472
68. Abdallah, K. Mohammad, T. and Louis, F. "Validation of Intelligent Systems: a Critical Study and a tool, Corus" Proceedings of the International Journal of Soft Computing 2007, pp.191-198.

69. Zlatareva, N. and Preece, A. "State of the Art in Automated Validation of Knowledge-Based Systems", Proceedings of the journal of Expert Systems with Applications, 1994, pp.151-168
70. Lockwood, S. and Chen, Z. "Knowledge Validation of Engineering Expert Systems" Proceedings of the Journal of Advances in Software Engineering, 1995, pp. 97-104
71. Lee, S. and O'Keefe, R. "Developing a Strategy for Expert System Validation and Verification", IEEE 1994, Proceedings of the IEEE Transaction on systems, Man and Cybernetics, Volume 24, pp.643-655
72. O'Keefe, R. , Balci, O. and Smith, E. "Validating Expert System Performance", IEEE 1987, Proceedings of the IEEE Expert, Volume 2, pp.81-90
73. Mosqueira-Rey, E. and Moret-Bonillo, V. "Validation of intelligent systems: a critical study and a tool" proceedings of Expert Systems with Applications, 2000, pp. 1-16
74. Vermesan, A. and Hogberg, F. "Applicability of Conventional Software Verification and Validation to Knowledge Base Components: A Qualitative Assessment", Proceedings of the 5th European Symposium on Validation and Verification of Knowledge Based-Systems-Theory, Tools and Practice, (EUROVAV '99), pp. 343-364
75. Wells, S. "The VIVA Method: A Life Cycle Independent Approach to KBS Validation", Proceedings of the IEEE AAAI Conference 1993, pp.102-106.
76. Cook, D. "Software Processes? How Bohring?" Back Talk in the Magazine of Crosstalk, Journal of Defense Software Engineering, Vol. 23, No.1, 2010, pp. 39
77. Brazier, F. Keplicz, B. Jennings, N. and Treur, J. "DESIRE: Modeling Multi agent Systems In a Compositional Formal Framework" International Journal of Cooperative Information Systems, 1997, pp.1-29

78. Weitzel, J. and Kerscheberg, L. "Developing Knowledge-Based Systems: Reorganizing the System Development Life Cycle" Proceedings of ACM journal of Communications, 1989, pp. 481 – 488
79. Chandrasekaran, B. "Generic Tasks in Knowledge-Based Reasoning: High Level building Blocks For Expert Systems Design" Proceedings of the IEEE Journal of Expert Systems, 1989, pp. 23- 31
80. Schreiber, G. Weilinga, B. and De Hoog, R. "Common KADS: A Comprehensive Methodology for KBS Development", Proceedings of the International Conference on Computer Design (ICCD '94), IEEE 1994, pp.28-36
81. Fensel, D. Angele, D. and Studer, R. "Knowledge Acquisition and Representation language: KARL" Proceedings of the IEEE Transactions in Knowledge and Data Engineering, pp.1-93
82. Schreiber, G. Wielinga, B. Akkermans, H. Van de Velde, W. and Anjewierden, A. "CML: The CommonKADS Conceptual Language" Proceedings of the LNCS 1994.
83. Schreiber, G. Weilinga, B. and De Hoog, R. "Common KADS: A Comprehensive Methodology for KBS Development", Proceedings of the International Conference on Computer Design (ICCD '94), IEEE 1994, pp.28-36
84. Angele, J. Fensel, D. Landes, D. and Studer, R. "Developing Knowledge-Based Systems with MIKE" Proceedings of the Automated Software Engineering Journal, 1998, pp. 389-428
85. Vollebregt, A. and Lei, J. "A Study of PROFORMA, a Development Methodology For Clinical Procedures" Proceedings of AI in Medicine, 1999, pp. 1-22

86. Medina, M. Sanchez, A. and Casterlanos, N. "Ontological Agents Model Based on MASCommonKADS Methodology", Proceedings of the 14th International Conference on Electronics, Communication and Computers, (CONIELECOMP '04), pp. 260-263
87. Vob, W. and Karbach, W. "Implementing KADS Expertise Model with Model- K", Proceedings of the IEEE Expert: Intelligent Systems and their Applications, Computer Society Press, August '93, pp.74-81
88. Gonzalez, A. J. Stensrud, B. and Barret, G. "Formalizing context-based reasoning: A modeling paradigm for representing tactical human behavior", Proceedings of the International Journal of Intelligent Systems 2008, pp. 822-847