A KNOWLEDGE ACQUISITION SCHEME FOR FAULT DIAGNOSIS IN COMPLEX MANUFACTURING PROCESSES

by

Asghar Motaabbed

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 9 2

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

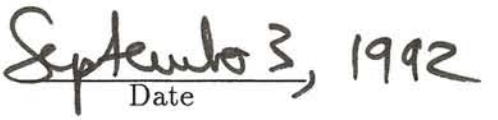
Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: 

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Francois E. Cellier
Associate Professor of
Electrical and Computer Engineering



Date

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Dr. Francois E. Cellier for his invaluable time, guidance, and support during my graduate school. I wish to express my gratitude and feelings of thankfulness to my manager, Jim Irwin for his support and guidance during my employment at Advanced Micro Devices, specially for his support of this thesis. My special thank goes to Mike Simpson at AMD for his invaluable time and his help in completing this work.

I would like to thank Professor Jerzy W. Rozenblit and Professor Harold G. Parks, who served on my committee, for their suggestions and help. I would like to thank John Zvonar at Advanced Micro Devices for offering me the opportunity for being part of the EDS project again.

This work was done as part of developing Expert Diagnostic System (EDS) at Advanced Micro Devices in Austin Texas. (Completion Date: 8-1990)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMINOLOGY INDEX

Component:	Component The smallest subdivision of the machine that the family manager chooses to identify. This could be a PC board, or it could be a resistor on that board.
DAR:	Data Analysis Routine.
Data:	Data denotes raw facts. It comprises the raw materials, the ingredients for information.
DSW:	Name of a family of equipments inside a fabrication facility.
FAB10:	Name of fabrication facility at AMD
FAB15:	Name of fabrication facility at AMD
Failure:	An incident that causes a machine to fail to perform its intended function.
Failure Code:	An internal code that the programs use to represent a unique failure.
Failure Description:	An English language description of a unique failure.
Failure Menu:	A hierarchical data structured containing Major-SubSystems, Minor-SubSystems, and Failure Components and their Failure Modes.
Failure Mode:	The manner in which a component has failed.
Facility:	The overall manufacturing environment can be decomposed into smaller environments, the so-called facilities, based on the nature of the involved equipment park and its intended functions.
Family:	A group of machines of like type and model that share a common failure menu.
Family Manager:	The equipment engineer or specialist assigned to create and oversee the content of a given family menu.
Family Menu:	Menus such as Failure Menu, Symptom Menu, and Repair Activity Menu.
Folder:	A menu choice that can be decomposed further into sub menus.
HDC:	Hierarchical Data Collection System - the software that displays the appropriate menu and collects event data, such as failure data.

Information:	Processed data become information. For example, failure data are collected by HDC and as a result, a failure description is formed. This description is considered information.
Ingress:	A Relational Data Base
Item:	Name used to describe the lowest component level in the menu.
Knowledge:	Knowledge (only factual knowledge is defined in this system) is defined as synthesized information. During synthesis, one piece of information is compared to other pieces of information, and they are combined in significant likes to form a pattern or chunk of knowledge.
Major Subsystem:	The largest subdivision of a machine that the family manager chooses to identify.
Minor Subsystem:	A machine subdivision contained within a major subsystem and containing components and/or other minor subsystems.
MMR:	Menu Maintenance Routines - the software used by the family managers to perform various housekeeping functions on the family menus.
Occurrence:	It is the number of frequency that an event has occurred in the past.
Other:	A special menu option chosen by maintenance personal in the absence of an appropriate menu selection, requires input of failure, symptom, and repair activity description, and is subject to approval by the equipment family manager.
Repair Activity:	It is activities done by maintenance personnel to fix the equipment.
Symptom:	Symptom is defined as equipment malfunction, quality disturbances, or process instability.
Symptomic Factor:	The Symptomic factors constitute a means of classification of symptoms.
Symptomic Classification:	The Symptomic Classification is used to decompose symptoms in a hierarchical fashion.
Workstream:	A Data Base Management System (Comets)

ABSTRACT

This thesis introduces the problem of knowledge acquisition in developing a Trouble Shooting Guide (TSG) for equipment used in integrated circuit manufacturing. TSG is considered as a first step in developing an Expert Diagnostic System (EDS). The research is focused on the acquisition and refinement of actual knowledge from the manufacturing domain, and a Hierarchical Data Collection (HDC) system is introduced to solve the problem of bottleneck in developing EDS.

An integrated circuit manufacturing environment is introduced, and issues relating to the collection and assessment of knowledge concerning the performance of the machine park are discussed. Raw data about equipment used in manufacturing environment is studied and results are discussed. A systematic classification of symptoms, failures, and repair activities is presented.

CHAPTER 1

Introduction

As industry grows and its technology evolves, manufacturing systems are becoming more and more complex. These systems frequently break down and then, they need to be diagnosed. As a result, the cost of troubleshooting and maintenance has become a crucial point in most of these manufacturing systems. On the other hand, accurate, timely, and reliable information about the equipment's behavior and the knowledge domain are essential for performing a proper diagnosis, and thus, the availability of an automated maintenance system seems highly desirable. Therefore, an Automated Expert Diagnostic System is highly recommended to bring these goals to bear. To make this system fully automated, three self maintenance parts are required: (i) a *Watchdog Monitor* to detect any error or discrepancy in the system; (ii) a *Self Maintenance Diagnoser Scheme* that can pinpoint the faulty component, and (iii) an *Automated Repair System* to perform the actual repair activity. This type of automated system may be feasible for new plants that are now under development (Figure 1.1 illustrates such a system). However, for currently existing plants, this approach may not be practical due to the absence of interfaces for Watchdog Monitors and Automatic Repair Systems. Retrofitting an existing manufacturing system with

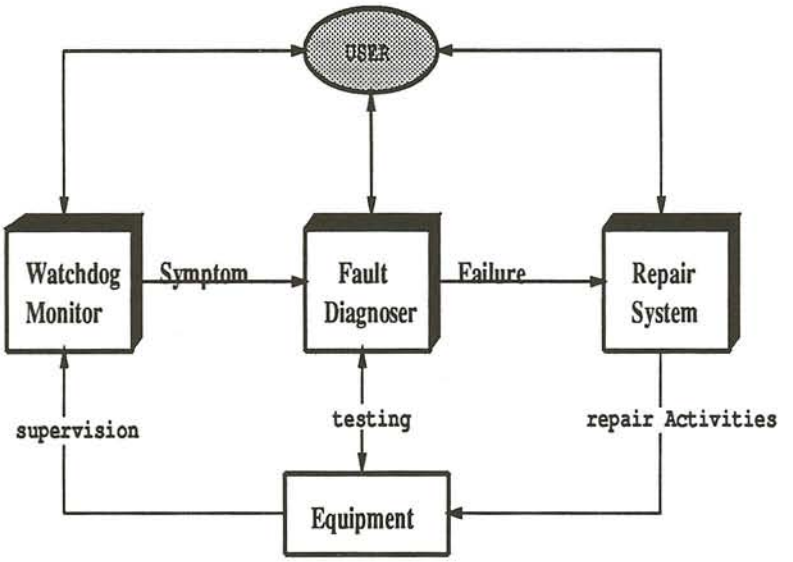


Figure 1.1: An Automated Expert Diagnostic System

the appropriate interfaces may often be almost as expensive as to replace the entire plant. Therefore, existing manufacturing systems need humans, such as operators and technicians, for guidance and maintenance. Figure 1.2 shows a Fault Diagnoser with its crew to show the first step toward automating the process of fault diagnosis in existing plants.

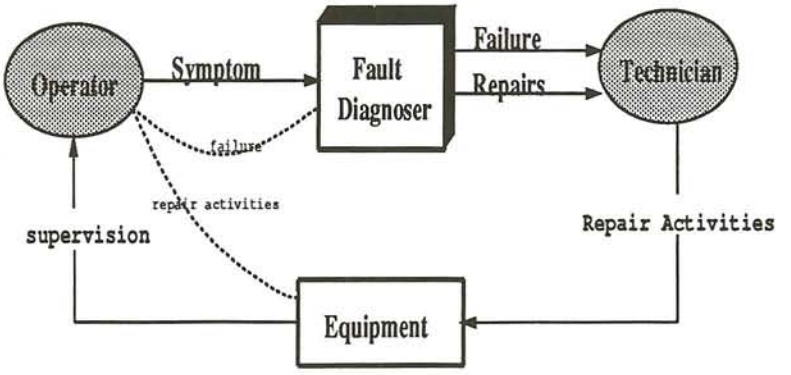


Figure 1.2: Fault diagnoser with human as maintenance and guidance

It is not the intent of this author to develop a fullfledged Expert Diagnostic System. Rather it is his intent to develop a semiautomated Knowledge Acquisition Scheme to collect and capture accurate information relating to equipment failures and their symptoms; and further to design a methodology to relate these pieces of information to each other to form a Knowledge Base for TroubleShooting Guide. It is envisioned that this TroubleShooting Guide forms the basis of an Expert Diagnostic System that could further revolutionize the way maintenance is practiced.

In developing the above Knowledge Acquisition Scheme, designing a tool that can automatically acquire knowledge would be desirable. Finding such a tool can be done in many different ways. The first way is to *do it by hand*. This method is expensive and slow. It requires craftspeople, custom designers, and MIS departments, which all cost money. Applicability of this approach is also limited to fairly simple processes. It would not be possible for a company to hire qualified permanent staff to perform this job in the context of an ever more project-oriented manufacturing environment with its increasingly complex machine park. On the other hand, if a company uses external and highly qualified consultants, the cost will be outrageous. Furthermore, handcrafting can take so long that the software is likely to be obsolete by the time it is ready for use.

The second way is to *think of everything in advance*. Such a package provides tables that are flexible enough to accommodate every foreseeable situation. This approach is feasible for applications that do not evolve, such as accounting and payroll.

Such software is not suited for a manufacturing environment where the data base evolves constantly due to inclusion of a new machine type, a new type of symptom, failure, and so on. Moreover, the original development of this software is difficult to maintain and tends to run like a "three-legged horse" (rovira, 1990).

Finally, the third way is to design an *adaptive software*. Adaptability of the software to an ever-changing environment would be the most important factor in its acceptance into a manufacturing environment. Such software must be able to maintain and modify its own tables and parameters in accordance with newly available knowledge. In this approach, only the (fairly application independent) mechanisms for modifying tables and parameters are frozen, whereas the (domain-specific) knowledge itself is not predetermined. The price to be paid for the generality and flexibility of this approach is a somewhat reduced execution speed in comparison with the other two approaches. However, this disadvantage is quite acceptable in the context of flexible manufacturing systems, where the cost of the diagnostic computer is, in any event, only a rather insignificant portion of the overall system cost, and where the cost of a non-operational machine due to an undiagnosed failure is far more significant than the cost paid for reliable and fast failure analysis.

In this thesis, the author will try to present the system development in three distinct stages: He first describes the development of a Hierarchical Menuing System to collect data; he then introduces a design strategy for collecting data items and relating the various facts to each other; and finally, he designs an Example-Based Knowledge

Base to complete the Trouble-Shouting-Guide as his final goal. Quick prototyping is used to establish the system architecture and its detailed requirements. Symptom Prototype, Repair Activity Prototype, and Diagnostic Prototype are among those components that determine the appropriate knowledge representation, man-machine interface, supporting hardware, and functional architecture of the system. A discussion of these prototypes is postponed to Chapter 4 in which all relevant terms are defined.

CHAPTER 2

Knowledge Acquisition and Design Principles

2.1 Introduction

In chapter one, the need for the development of an autonomous system was expressed and, in this context, a semiautomated expert diagnostic system was proposed. In this chapter, the author focuses on the development of such an expert diagnostic system, and he begins with the most difficult part, the *knowledge acquisition.* The author first explains the advantages of using an expert diagnostic system over the other two rivals, *human diagnosis* and a diagnostic system using *conventional programming* techniques. Next, he introduces the problem of knowledge acquisition, which constitutes from several different perspectives the primary bottleneck in the development of automated fault diagnosers as reported by various scholars in the field (Buchanan, 1983). Several approaches for knowledge acquisition are presented along with the modes in which their knowledge can be acquired. Furthermore, section 2.4 introduces several knowledge acquisition tools, including those that exist as components of expert system shells, those based on repertory grids, and finally those that serve to guide the interviewing of human experts. Section 2.5 explores types of

knowledge to be acquired from the manufacturing domain by an individual, such as an operator or a technician, and also provides an analysis of the structure of such knowledge. Finally, the chapter concludes with a description of the software design principles, and presents a step-by-step *development life cycle* of the overall system.

2.2 Expert System

In search for an efficient scheme for fault diagnosis, the author noticed that three general schemes of fault diagnosis have been employed in the past that differ in the kind of technology used. These schemes are referred to as *human diagnosis*, *conventional diagnosis*, and *expert system diagnosis*(Klue; Karel, G. and Kenner, M.,1989). Obviously, the human (manual) scheme is of not much interest in the context of this thesis. Its drawbacks are evident. Human experts are scarce and expensive. Their expertise is often specialized to one particular type of equipment. If only a few machines of that type are used in the manufacturing environment, it is economically infeasible to hire an expert. Thus, when the equipment breaks down, the expert has to be flown in, which adds to the cost of diagnosis and repair and prolongs the downtime of the equipment, which in turn again increases the cost. If sufficiently many machines are used, it is possible to hire an expert. Yet, the education of experts is time consuming and expensive, and the life span of most machines used in modern integrated circuit manufacturing is so short that the education to train a human expert on a new machine constitutes a significant portion of the life span of that

machine. On the other hand, conventional (code driven) programming techniques are inadequate for fault diagnosis in manufacturing. While conventional fault diagnosers provide for excellent run-time performance (speed), they are characterized by a low degree of flexibility. Thus, a real-time fault diagnoser for an aircraft (implemented as part of the autopilot) will probably be implemented using conventional programming techniques, since speed is of the essence and since the aircraft remains the same throughout its life cycle. However, in a manufacturing environment, the diagnostic knowledge is continuously evolving since new processes are being manufactured by existing machines and since new machines are either added to the machine park or replace existing machines in the machine park. Thus, a high degree of flexibility of the fault diagnoser is utterly important, and yet, the time constants of the manufacturing environment are sufficiently slow so that the overhead inherent in the (data driven) expert system approach can be tolerated. Thus, it becomes evident that the expert system approach to fault diagnosis is the most suitable among the three alternatives for the application in hand.

Sell (1985) divides *artificial intelligence* into two subdomains: *human cognition* and *intelligent artifacts.* Human cognition deals with how experts go about solving problems, and hence, human cognition characterizes the understanding of the workings of the experts' mind while solving a problem. Intelligent artifacts, on the other hand, are concerned with the specific pieces of knowledge of the expert used when a particular problem is being solved. Thus, human cognition deals primarily with

the *domain independent* meta-knowledge of the mechanisms of thinking, whereas the intelligent artifacts relate to the always *domain specific* pieces of information employed in the thinking process. In an expert system, these two types of knowledge are separated as well. Intelligent artifacts are stored as facts in the (domain specific) knowledge base, whereas the method of thinking is implemented in the (domain independent) inference engine. The process of building an expert system can be understood as the capturing of the domain knowledge of an expert and his thinking mechanisms in computer software in such a way that it can emulate the expert in arriving at solutions to the problems of interest. Present expert system techniques provide the ability to encode knowledge in usable software programs for giving advice on specific problems. Since the facts are stored as data entries in a knowledge base, new facts can be deduced and stored in the knowledge base without need for recompilation. However, little support is provided for generating the entries in the knowledge base. Yet, the power of an expert manufacturing system is derived from the knowledge it processes, not from the particular formalisms and inference schemes it employs in the construction of the system (Feigenbaum); that is why a *knowledge acquisition tool* plays a central role in the building of an expert system for capturing knowledge relating to an integrated circuit manufacturing plant.

2.3 Knowledge Acquisition: The Bottleneck

The design of an expert system usually faces several difficulties in the formalization of the problem, i.e., the selection of useful knowledge, the knowledge acquisition, the knowledge representation, and finally the knowledge utilization. Among the top difficulties, knowledge acquisition is recognized as the most critical element in the development of an expert system, and it is known to be the most problematic. Buchanan *et al.* (1983) describe knowledge acquisition as the *bottleneck* of the technology due to the inherent difficulty in retrieving knowledge from human experts (since humans are not fully aware of their own cognition mechanisms, and therefore don't have a complete and deliberate access to the intelligent artifacts stored in their memory), and due to the difficulty of appropriately translating and representing human expertise. Knowledge acquisition is defined as the *process of transforming data on expertise into an implementation formalism* (Kidd, 1987). Thus, this process consists of acquiring domain knowledge from the expert (knowledge elicitation) and formalizing the gathered knowledge for analysis (knowledge representation).

Traditionally, knowledge acquisition has been approached from two major vantage points: *manual* and *automated.* Figure 2.1 illustrates these two approaches.

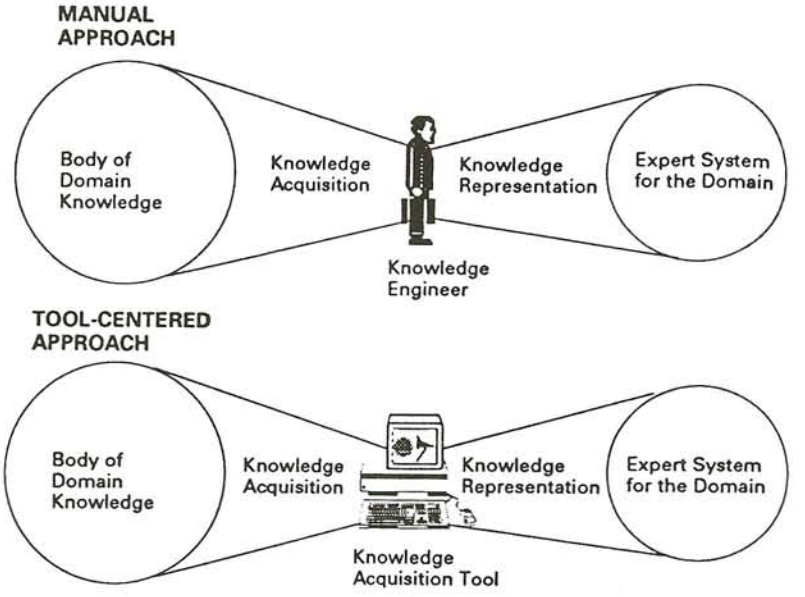


Figure 2.1: Two approaches for knowledge acquisition (McGraw)

The manual approach is both time consuming and costly. It takes time for a knowledge engineer to become familiar enough with the domain at hand to elicit knowledge from a domain expert in an efficient manner. Therefore, considerable research is being conducted in the development of *automated knowledge acquisition tools.* Programs that automate facets of the knowledge acquisition process range from those that support the knowledge engineer in conceptualizing or developing a model of the domain (Planet: Gains and Shaw, 1986; AQUINAS: Boose and Bradshaw 1987a,b; Kitten: Garg-Janardan and Slavendy, 1987; Kriton: Diederich, Ruhmann and May, 1987) to those that purport to automate the interviewing (Knack: Klinker, *et al.*, 1987a,b), domain modeling (More: Kahn, 1985) , and refinement processes (OposII: McGraw, Seale, 1987; Lotta: McGraw, Seale, 1987). Some tools have been developed for the purpose of researching the knowledge acquisition process itself to

identify more efficient approaches to knowledge acquisition (McGraw, 1989); others have been developed as commercially available stand-alone tools (MacSmarts: Rasmus, 1988; First-Class: Ruth, 1988). Marcus *et al.* (1985) report that the existing knowledge acquisition tools are mainly to provide two kind of assistance:

> *Increase the ease with which a domain expert can communicate his or her expertise*

and/or

> *Organize or proceduralize the knowledge that is communicated so that all relevant knowledge for a situation is exposed.*

There exist a number of different ways to approach knowledge acquisition (knowledge elicitation) for expert system development. Several techniques are widely used (Kidd, Slater, Welbank, Hart, waterman & Newel, Rozenblit): (i) *interviewing experts*, (ii) *learning by being told*, (iii) *learning by observation*, (iv) *conceptual sorting*, (v) *multidimensional scaling*, (vi) *machine induction*, (vii) *protocol analysis*, and (Viii) *explicit representation (KAR).* In reality, none of these techniques is used to the total exclusion of the others (McGraw, 1989). In some cases, a hybrid approach is adopted. In others, a single approach serves as primary tool for one phase of the development, while additional approaches are adopted for use in later stages. Buchanan *et al.* (1985) have investigated different modes that are used to acquire knowledge. They presented four different modes, each representing a variation of the manual

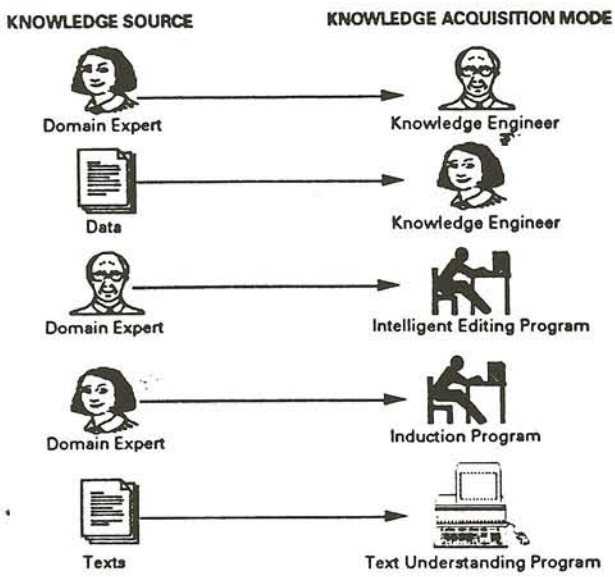


Figure 2.2: Knowledge acquisition modes (McGraw)

and automated perspective. Figure 2.2 illustrates five knowledge acquisition modes (McGraw, 1989), each of which varies along a continuum from manual to automated, based on the knowledge source and the transfer mechanism.

The first two knowledge acquisition modes are manual in nature while the third, fourth, and fifth modes reflect varying degrees of automation.

Most knowledge acquisition tools have been developed to circumvent the bottleneck in the process of building expert systems. Most developers would agree that the primary causes of this bottleneck are the difficulty of retrieving appropriate knowledge from human experts and of appropriately translating or representing human expertise. If tools could be developed such that a domain expert could sit at a workstation and encode his or her expertise directly, the bottleneck might be reduced. It has been suggested that, by using knowledge acquisition tools, there may be less "noise" introduced into the encoded knowledge (Friedland, 1981). "Noise" is

here defined as either erroneous information, missing information, poor description language, or unreliable data . Noise is often introduced in manual knowledge acquisition as a consequence of a misunderstanding between the domain expert and the knowledge engineer. However, the same type of noise can also be introduced as a consequence of a misunderstanding between the domain expert and the knowledge acquisition tool. For example, if a taxpayer (domain expert) uses a tax preparation program (knowledge acquisition tool) to generate his tax declaration (knowledge base), it is not evident that the generated tax declaration contains less "noise" than if he or she makes use of a CPA (knowledge engineer) who extracts the knowledge from the taxpayer using an expert interviewing technique, and then enters the extracted knowledge into the (probably identical) tax preparation program. The tax preparation program will ensure *consistency* among the data entries, not necessarily correctness. This consistency may, in fact, have the undesired side effect that incorrect data entries are less likely to be exposed later. Thus, the previously quoted conclusion does not seem plausible. It is a formidable task to design a knowledge acquisition tool that is able to reason about potential misunderstandings. To this end, it may in fact be desirable to equip the knowledge acquisition tool with *redundant data entry* to reduce potential noise by means of internal data consistency checks.

2.4 Knowledge Acquisition Tools

Unfortunately, The diversity in the focus of existing knowledge acquisition tools, the manner in which they have been implemented, and the domain in which they have have been tested, make a comparison of their features ineffective. Furthermore, the continued increase in the number of tools designed to automate some portion of the knowledge acquisition process prohibits the author from covering each tool. However, it is the intend of this author to introduce to reader to a few examples of different types of existing knowledge acquisition tools. This is done by dividing tools into three distinct categories.

In the first category, the author introduces two expert system shells that are being used as knowledge acquisition tools. These are powerful tools that have been developed for specific applications. For example, MacSmarts (Rasmus, 1988) is an expert system building tool developed by Cognition Technology for the Macintosh family of computers. MacSmarts combines the ability to design rule-based and example-based systems with a user-friendly interface and the ability to link rules and devices to other elements (e.g. graphics, text, spreadsheets, databases, and/or knowledge bases). While some domain experts may be able to input knowledge using the rule-based component, the factor that makes MacSmarts attractive as a knowledge acquisition tool is its example-based knowledge-base development component. After the domain expert has provided information on factors, choices, and advice, the ID3

algorithm (Quinlan, 1982) is used to translate the examples (factors, choices, advice) into an expert system knowledge base.

Another example of an expert system shell is the *1st-class* expert system (Ruth, 1988) It is a shell that can be used as knowledge acquisition tool in the development of expert systems. 1st-class has the capability to help the knowledge engineer to develop a knowledge base through sets of examples. This type of knowledge base is applicable to domains that have a classification system with historical and tabulated data. When the knowledge base is built, 1st-class will construct *rules* based on the examples saved in its knowledge base by applying one of the following methods: *optimize the rule*, *left-to-right optimized rule*, *progression of factors*, or *exhaustive left-to-right rule.* Then, it inspects rules for their validity and finally determines which questions to ask, and in what sequence they ought to be posed.

In the second category, the author presents knowledge acquisition tools that are used to build *domain models* based on Repertory Grids. These tools have been designed to automate (a portion of) the knowledge acquisition process. Knowledge acquisition research at BDM corporation has resulted in a design of BDM-KAT, (Blaxton, Geesey, Reeke, 1987). BDM-KAT provides the knowledge engineer with the ability to create an initial knowledge base. It consists of three modes: *clarification*, *prediction*, and *diagnosis.* In the clarification mode, BDM-KAT provides a framework that encourages the identification of domain concepts and categories, and the decomposition of domains into subprocesses. It builds a model of the expert's

conceptualization of the process, complete with graphically-depicted interrelationships and composite subprocesses. Once the clarification mode has been completed, the knowledge engineer may enter the *prediction* mode. The purpose of the prediction mode is to expose gaps and fill holes that remain in the knowledge base. When an acceptable mode has been developed, the knowledge structure can be refined in the *diagnosis* mode. In the *diagnosis* mode, the expert views the current state of the knowledge base in the form of a flow chart in which each major node represents a process.

Finally, in the third category, the author presents knowledge acquisition workbenches as a tool to acquire knowledge. While some of the knowledge acquisition tools may assist the user in creating a domain model, workbenches are designed to provide a mean to translate knowledge that has been previously captured into a representation scheme that can be used by the expert system under development. Systems such as AQUINAS (Boose, 1987), KRITON (Diederich, Ruhmann, May, 1987), and TDE (Kahn, 1987) all fit under this category. For example, among the top choices, Kriton is described as a *Hybrid Knowledge Acquisition Tool* that uses a variety of knowledge acquisition methods to elicit and capture knowledge from domain experts (Diedrich, Ruhmann and May, 1987). Kriton makes use of the domain experts' declarative and procedural knowledge as well as static, text-based knowledge. Its architecture supports the use of three knowledge elicitation methods, namely *automated interviewing*, *text analysis*, and *protocol analysis.*

2.5 Identifying and Tapping Knowledge

A primary task of the knowledge engineer is to capture and conceptualize what a domain expert knows and uses to solve problems. Thus, before the author can proceed to explain the acquiring of knowledge from domain experts, he must put himself in the pace of a knowledge engineer and provide a basic understanding of what knowledge is, how it is stored, and how domain experts access it. Therefore, the focus of this section is to define knowledge that is being acquired in the integrated circuit manufacturing environment by individuals, usually either *operators* or *technicians*, and explore this knowledge from various perspectives. Whenever possible, examples are given to support concepts. However, let us first define the two most common terminologies used in this context: *Knowledge Engineer* and *Domain Expert.*

Domain Expert

The domain expert is an individual selected for expertise in a given field and for his or her ability to communicate knowledge. In this project, there are three types of experts that each will have an important role in developing this expert system. They are the following:

Technician: A maintenance person responsible for diagnosing faults in manufacturing equipment.

Operator: An individual who operates the equipment, responsible for detecting that a fault has occurred.

Family Manager: The equipment engineer or specialist assigned to create and oversee the content of a given family menu such as the failure menu or the symptom menu.

Knowledge Engineer

The knowledge engineer is an individual responsible for structuring and/or constructuring an expert system. In this project, the author assumes the role of the knowledge engineer.

LAYERS OF KNOWLEDGE: Data, Information, Knowledge

Data

In this system, *data* denotes raw facts and concepts expressed in factual statements. Data comprises the raw materials, the ingredients for information. These ingredients are collected by the operators and technicians, and are stored in the *Workstream Data Base.* Section 4.1 presents the Workstream data base and its role in developing the expert system. Figure 2.3 illustrates a sample of data acquired from the manufacturing environment.

Information

Processed data become *information.* Information can result from analyzing the data based on some established criteria. Analyzed data are informative and can be useful for the maintainability of the manufacturing environment. Questions such as:

Sample Data	
Facility	Year
Entity	Operator
Begin-Date	End-Date
Begin-Time	Start-Date
Entity-Type	Event-Duration
Vendor	End-Time
Location	Shift
Event-type	Technician
*	*

Figure 2.3: Sample *data* acquired from manufacturing domain

What is the most common type of entity failure?

What is the total hours to repair a given entity?

What is the total hours to repair a given failure type?

What is the frequency of repair for a given failure type?

What is the frequency of repair for a given entity?

are among those that can be addressed here.

Information, in general, tells more about the *status* of a domain as questions are formed. For example, failure data are collected by the Hierarchical Data Collection System (HDC) and as a result, a failure-description is formed. This description is considered *information* while the individual entry about the faulty component is referred to as *data.* To clarify this concept better,

the author presents the following example: suppose an *inference bell* is the faulty component that caused a particular machine to crash and as a result, made the operator bring the machine down for the purpose of fault diagnosis and repair. After diagnosing the fault, the Technician uses HDC to collect the *failure* and its *failure mode.* As a result of this process, a failure description with the path: *Track assembly; inference assembly, inference bell, broken* is formed. Since this description tells us about the *status* of the inference bell, i.e., whether it is loose, broken, or misadjusted, the complete failure description is considered *information*, whereas the faulty component entry, *inference bell*, prior to the fault diagnosis is referred to as *data.* This information is first stored in the *Failure Menu Data Base*, and will then be transferred to the *Relational Data Base, Ingress* for further analysis. The failure description records provide a useful source of information for developing the knowledge base of the expert diagnostic system.

Knowledge

Philosophers, psychologists and artificial intelligence researchers have attempted to answer the question *what is knowledge? Knowledge* is defined in Webster's Dictionary as *Facts or ideas acquired by study, investigation, observation, or experience.* Heyes Roth (1983) contends that knowledge consists of description, relationship, and procedures. Fischer (1987) defines *knowledge* as sorted information, called a model,

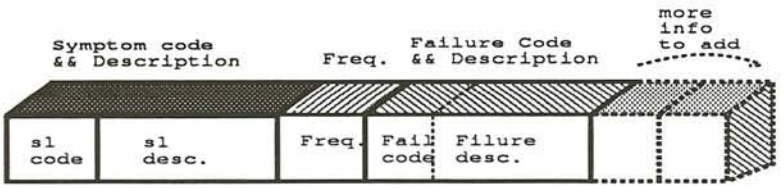


Figure 2.4: Chunk of knowledge

which is used by a person to interpret, predict, and appropriately respond to the outside world. Knowledge consist of relationships formed by a combination of *declarative (factual)* and *procedural (methodic)* statements that are called upon for the performance of an application. Tuthill (1990) defines *knowledge* as synthesized information. During synthesis, one piece of information is compared to other pieces of information, and they are combined in significant links to form a *pattern* or *chunk* of knowledge. In our application, the Symptom Description, Failure Description, and Repair Activity Description are examined, compared, and combined together to form a *chunk* of *knowledge.* Figure 2.4 illustrates the chunk of knowledge used in prototyping of this system.

It is important for readers to realize that the above chunk of knowledge is designed solely for the purpose of prototyping the system. This chunk contains the minimum amount of information and data necessary to build the prototype. This chunk needs to be completed by additional processed data as required. Most of the necessary information is organized in the Relational Data Base, Ingress.

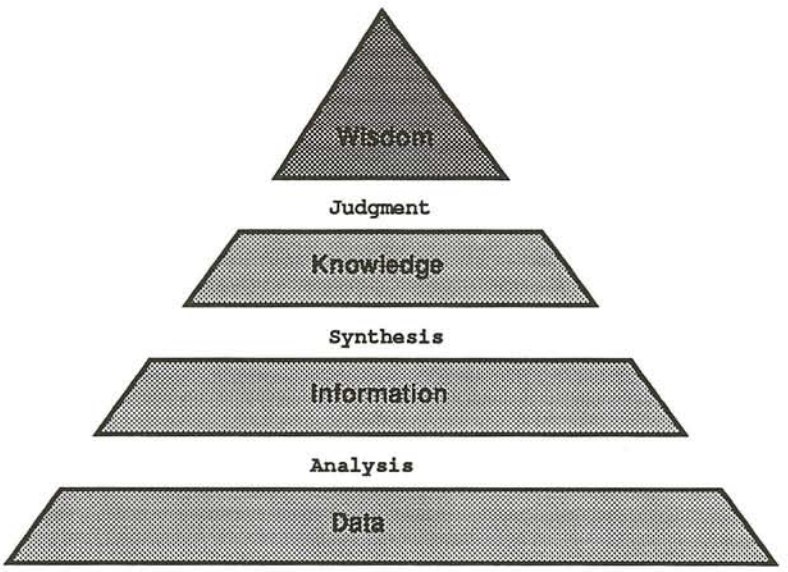


Figure 2.5: Pyramid of data (Tuthill)

Data, information, and knowledge form a *wisdom hierarchy.* Figure 2.5 illustrates the wisdom hierarchy. Data is aggregated and cathected with meaning as the hierarchy develops.

2.6 Design Principles

Eliciting knowledge from an expert domain in appropriate form is prohibitively difficult. However, while the expert cannot directly communicate his expertise and strategies, he can provide data and critical information about equipment performance via an *intelligent editing program.* A *learning program* can then process this information to induce rules or formulate strategies in further building the knowledge base for expert reasoning. For example, Michaleski (1980) has used AQ11 to formulate factual knowledge as rules that diagnose plant disease. In some cases, its rules were

more effective than the rules that were generated manually by a domain expert. The extent to which these strategies (rules) are representative of the complete solution depends on the goodness and completeness of the set of knowledge that is used to form the theory. Induced descriptions are guaranteed to be valid only for the facts from which the induction was made (Shapiro, 1987). Therefore, the integrity of the knowledge acquired by a knowledge acquisition tool must be one of this project's primary objectives.

This research has concentrated on the acquisition and refinement of actual knowledge from the manufacturing domain and does not deal with the problem of learning (inducing rules) or acquiring strategies for diagnosing a machine. Its primary objective is to develop an intelligent system that can acquire factual knowledge, such as symptoms, failures, repair activities, and all other data surrounding the operation of the manufacturing equipment, and is then able to relate these pieces of information together in building the knowledge base for expert reasoning. This knowledge can then be retrieved to assist the technician in troubleshooting problems. To develop a system that can fulfill all requirements of the users and also support the analysis of historical information about the equipment, the author decided to make use of a prototyping technique.

Why Prototyping?

Prototyping is a strategy for determining requirements for a system where the end user is not entirely clear up front what he or she really wants. User needs are extracted, presented, and developed by building a working model of the ultimate system quickly and in context and by letting the end user interact with that model, i.e., with the prototype. The technique differs significantly from *prespecification* in that once the software engineer has gained an initial understanding of the problem, he attempts to implement that understanding immediately. The resulting first-cut model serves as an anchor point to permit meaningful and unambiguous communication among all project participants. The definition of the system occurs through gradual and evolutionary discovery. It is the intent of this author to show step by step how this evolution took place.

Figure 2.6 illustrates the system development life cycle. At the top level, a *Hierarchical Data Collection* is designed to enable the user to collect data using specific *menus* available in the domain. This system also needs a Menu Maintenance Routine, MMR, that can maintain these menu data bases. At this stage of development, Failure Collection seems to be the most suitable candidate for the prototype due to the nature of its menu development. Section 4.2 describes these issues in more details.

The next stage of development is to study the symptoms and their characteristics. The main task is to study how symptoms can be classified based on the comments available in the Comet Data Base. Positional factors, behavioral factors, physical

factors, and circumstantial factors are all among the issues that need to be studied. Furthermore, the question of who should collect symptoms and when, is an important consideration due to its significant effects on the integrity of the symptom data base and the way relationships are formed with the failure data base.

Finally, it is to be investigated how symptoms are related to failures and in what manner they should be collected so that they can be easily and reliably related to failures for developing the *knowledge base*. Section 4.3 discusses these issues as the author develops the symptom prototype. HDC needs to be optimized in accordance with the result.

In the *Repair Activity Prototype*, a part of the expert system needs to be developed to collect repair activity comments from the Comet Data Base. The main objective is to study the nature of repair activities and their relationship with the failures. It is to be investigated how and in what manner these repair activities ought to be collected and related to their corresponding failures. The question of what constitutes a *valid* repair activity needs to be addressed since there exist plenty of repair activities that are irrelevant to the failure at hand.

At the final stage of the development life cycle, an *application program* needs to be designed to support operators and technicians with the *trouble shooting* task. In this step, all the application programs developed during the symptom and repair activity prototypes are put together. The goal of this application program is that for each given *symptom* it provides the list of *failures* together with their *frequencies* of

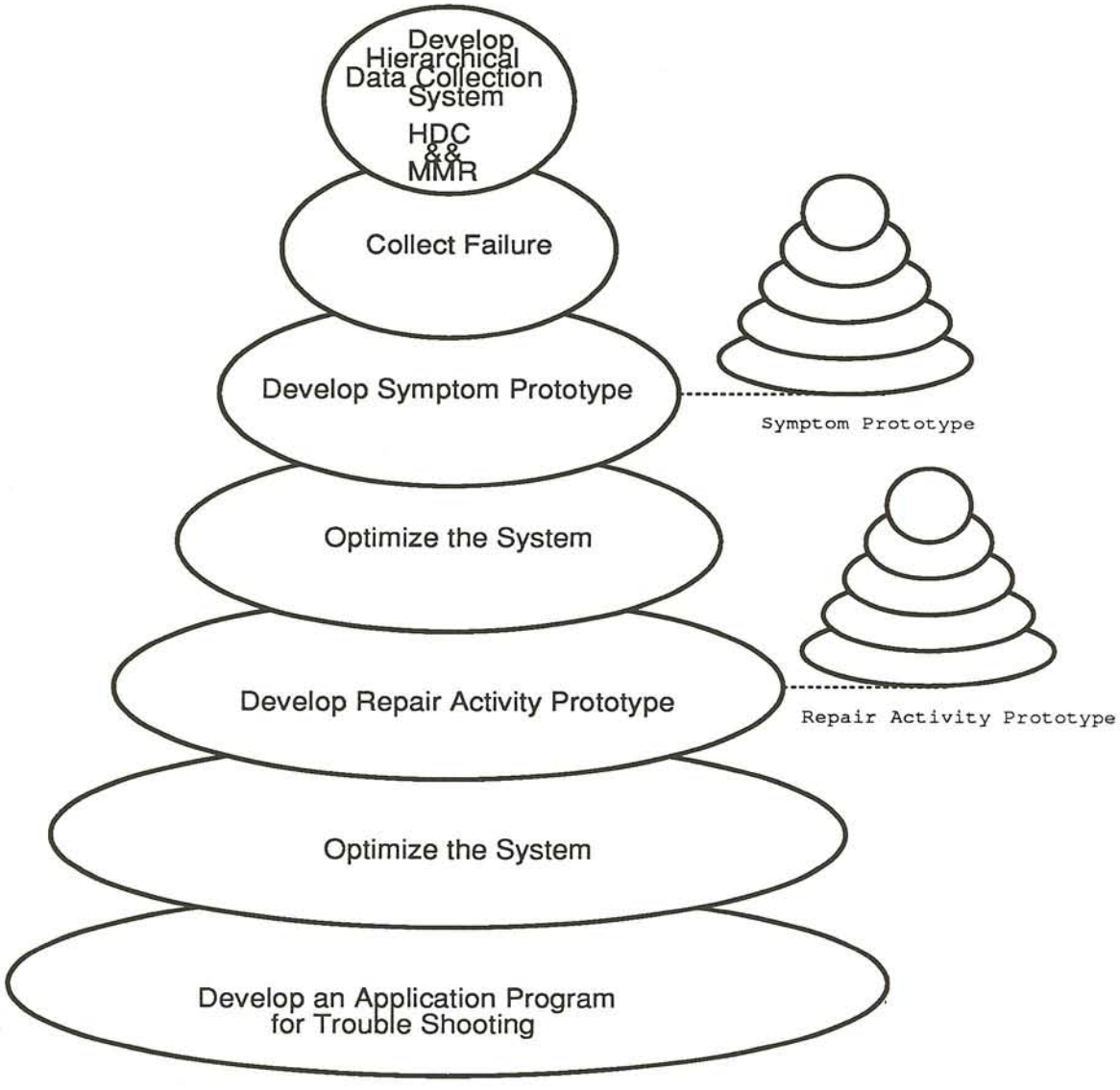


Figure 2.6: The system development life cycle

occurrence, and vice versa. It also provides a list of *repair activities* together with their *frequencies* of occurrence for each given *failure*. The application program is discussed in Section 4.5.

CHAPTER 3

Manufacturing Environment

3.1 Introduction

This chapter discusses the manufacturing environment and deals with important issues relating to collecting and measuring knowledge about performance of the machine park. A *System Entity Structure* (Zeigler, 1984) is introduced as a unique way to represent knowledge relating to the above environment and the activities performed on its machines. In Section 3.3, a manufacturing environment is introduced. Section 3.4 presents how *equipment time* needs to be categorized in order to be able to measure certain aspects of performance. A guideline for measuring performance is provided in Section 3.5. In Section 3.6, *event activity* is introduced as a means to keep track of the equipment activities. Issues concerning the involved Data Base Management Systems, such as Workstream and Ingress, as well as the Menu Data Base are discussed in Section 3.7. Finally, this chapter concludes with a discussion of collected comments about equipment behavior and activities that will be used to build the *Symptom Prototype* and *Repair Activity Prototype* explained in Chapter 4.

3.2 System Entity Structure

The System Entity Structure (SES) is a mechanism to describe hierarchically structured sets of objects and their interrelations (Zeigler, 1991). The SES is a labeled tree with attached variable types, i.e., a graphical object that describes the decompositions of systems into parts. In this chapter, the SES is used as a knowledge representation scheme that formalizes the modeling of systems in terms of only decomposition and specialization. Specialization is a mode of classifying objects and is used to express alternative choices for components in the system being modeled.

3.3 Manufacturing Environment

In this section, the author presents how the overall manufacturing environment can be decomposed into smaller environments, the so-called facilities, based on the nature of the involved equipment park and its intended functions. Figure 3.1 illustrates a decomposition of a manufacturing environment into many *facilities.* Each facility can be further decomposed into *families.* A family is defined to be a collection of equipment of similar types or copies of the same model.

There are many aspect of an equipment that can be studied. Those that are useful for this project such as *equipment state*, *equipment event* or *equipment hardware* are considered for further investigation. In this section, a hardware decomposition of equipment is presented. This decomposition is illustrated in Figure 3.2. Equipment can be decomposed into several *Major-SubSystems.* Each of these Major-SubSystems

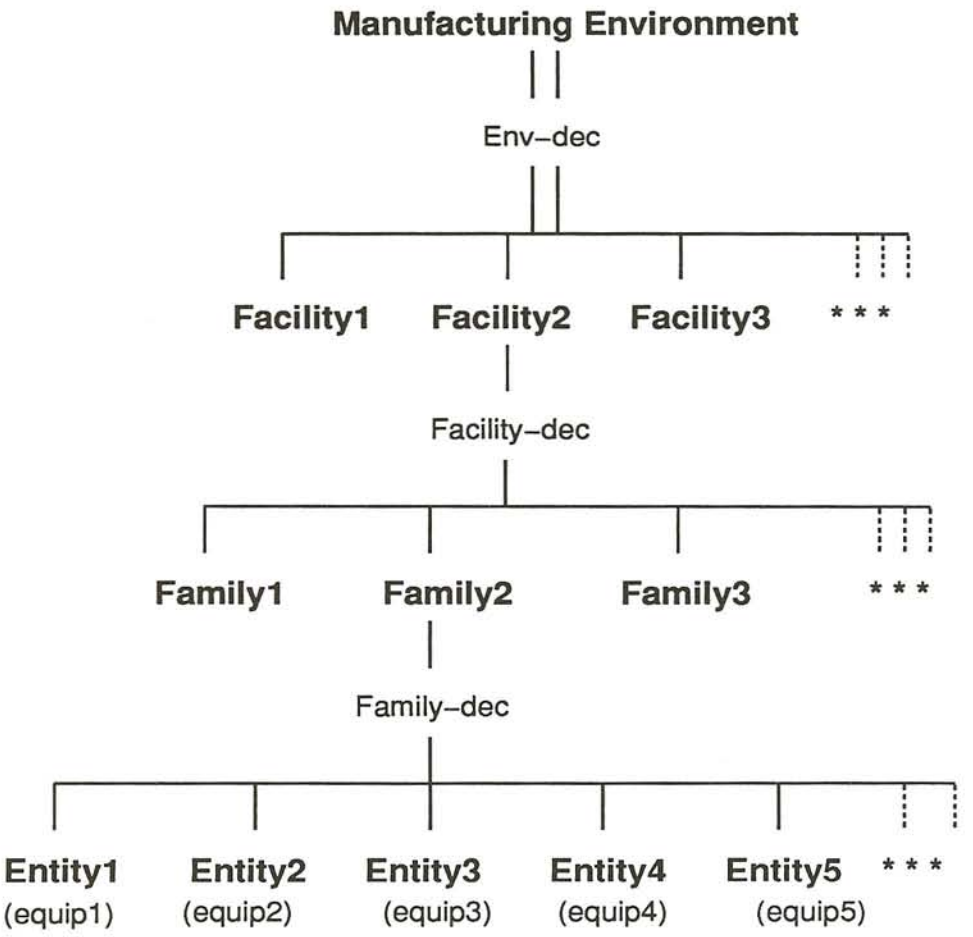


Figure 3.1: System Entity Structure of manufacturing environment

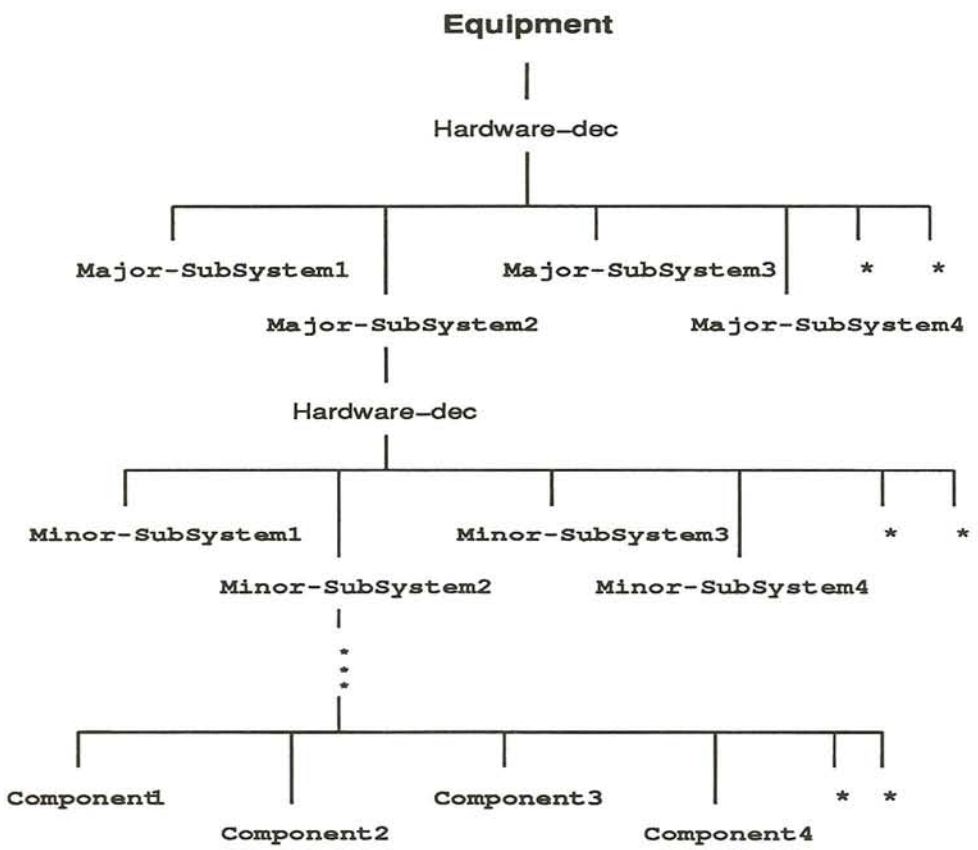


Figure 3.2: SES of hardware-decomposition of equipment

can be further decomposed into several *Minor-SubSystems.* Minor-SubSystems can be hierarchically decomposed into yet smaller subsystems if this is thought necessary by the family manager. The final decomposition will be into *Components.* Thus, a Component is the smallest subdivision of the machine that the family manager chooses to identify.

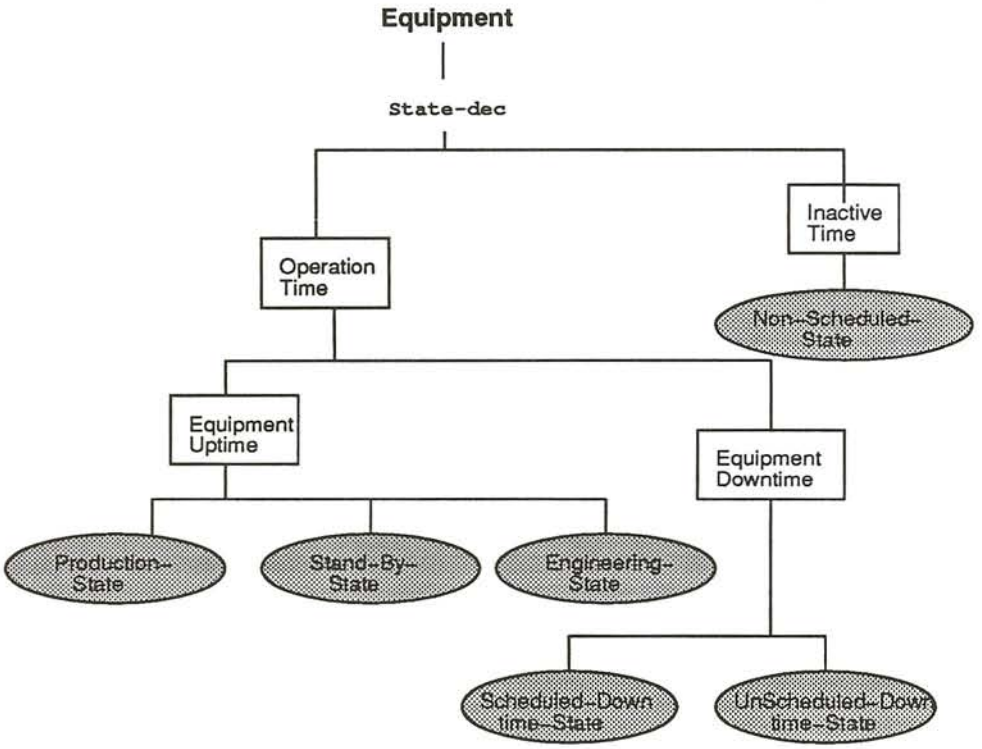


Figure 3.3: System Entity Structure of equipment states

3.4 Equipment States

In this section, the author defines how equipment time needs to be categorized into *states* in order to be able to measure certain aspects of equipment *performance.* These *states* are determined based on their functions, not by organization. Equipment time has been divided into six states into which all equipment conditions must fall. Each of the equipment states can be further divided into as many *sub-states* as necessary to achieve the desired equipment tracking resolution. Figure 3.3 illustrates the six states as they are grouped under *Operation Time*, *Equipment Downtime*, *Equipment Uptime*, and *Inactive Time.*

At the highest level of classification, Equipment Time can be divided into two groups, *Inactive Time* and *Operation Time. Inactive Time* is a period of time when the equipment is not scheduled to function, such as during the third shift, on weekends, or when the equipment is out of production due to a training session. The equipment is then considered to be in its *Non-Scheduled-State. Operation Time* categorizes equipment during time periods when it was intended to function. The total Operation Time is equivalent to the total time (Clock) minus the Non-Scheduled-State time. The Operation Time can be subdivided into two distinct groups, *Equipment Uptime* and *Equipment Downtime.*

Under the *Equipment Uptime* category, equipment can be classified under either of three different states: the *Production-State*, the *Stand-By-State*, and the *Engineering-State.* The *Production-State* is a period of time when the equipment performs its intended function, such as regular production, production test, and so on. The *Stand-By-State* is a period of time when the equipment is in a condition where it could perform its intended function but it is not operated. Situations which fall under this category include: no operator available, no product available, no support tools available, or waiting for the result of a running test. Finally, the *Engineering-State* is a period of time when the equipment is in a condition to perform its intended function, but it is used to conduct engineering experiments. New product evaluation runs fall under this category.

Under *Equipment Downtime*, equipment can be categorized into either an *Unscheduled-Downtime-State* or a *Scheduled-Downtime-State.* The *Unscheduled-Downtime-State* is a period of time when the equipment is not in a condition to perform its intended function due to an unexpected downtime event. Examples of situations that fit under this state include: (i) *Maintenance Delay*: a period during which equipment is waiting for personnel or parts, (ii) *Diagnosis*: the period of time needed by a technician to identify the faulty component, (iii) *Repair*: the time needed by a technician to repair the problem and bring the equipment back to a condition where it can perform its intended function, (iv) *Equipment test*: a period during which the technician operates the equipment to demonstrate its proper functionality. The *Scheduled-Downtime-State* is a period when the equipment is not in a condition to perform its intended function due to a planned downtime event. *Preventive Maintenance Delay*, and *Change of Consumable Chemicals* fall under this category. Preventive maintenance can be scheduled to occur at a certain time to reduce the likelihood of equipment failure during operation. Figure 3.3 uses the System Entity Structure to present the above concepts.

3.5 Performance Measurement Issues

This section provides guidelines for measuring the performance of equipment used in the manufacturing environment: *reliability*, *availability*, and *maintainability.* These

metrics are concentrated on the relationship of equipment failures to Production-State time, rather than the relationship of failures to total elapsed time (Clock). In the following sections, the author will define performance measurement issues for equipment in a manner to be consistent with the standards used in industry.

3.5.1 Equipment Reliability

Equipment reliability is defined to be the probability that the equipment performs its intended function for a specified period of time. The time used here is the Production-State and not the Clock. In the sequel, the author presents one reliability metric, MTBF, the mean time between failures.

MTBF - Mean Time Between Failures

MTBF is defined to be the average time the equipment performs its intended function between failures. It is the Production Time divided by the number of failures during that time.

$$MTBF = \frac{ProductTime}{\#ofFailure} \tag{3.1}$$

3.5.2 Equipment Availability

Equipment availability is defined as the probability that the equipment will be in a condition to perform its intended function when required. The author presents one metric of equipment availability, the *Operational Uptime.*

Operational Uptime

Operational Uptime is defined to be the percentage of time the equipment is in a condition to perform its intended function during the Operation Time.

$$Operational\ Uptime(\%) = \frac{EquipmentUptime}{OperationTime}\ X\ 100 \tag{3.2}$$

3.5.3 Equipment Maintainability

Equipment Maintainability is defined as the probability that the equipment will be retained in, or restored to, a condition where it can perform its intended function, within a specific period of time. The author presents one metric of equipment maintainability, MTTR, the mean time to repair.

MTTR: Mean time to repair

MTTR is the average time needed to repair a faulty component and bring the equipment back to a condition where it can perform its intended function. It is the total repair time divided by the number of failures during the considered time period.

$$MTTR = \frac{TotalRepairTime}{\#ofFailure} \tag{3.3}$$

3.6 Event Activity

Events are defined to represent time instants at which something is being done to or happens to a piece of equipment. Events are logged into the system in order to track the *activities* of the equipment. An event can change the state of the equipment. Events can be classified into many categories, such as *sign down an entity (sign down),log on to work on an entity (log on), log off to stop work on an entity (log off)* or *repair completed (repair complete).* Each of these categories includes many events as illustrated in Figure 3.4. This figure also shows the *Event-Type*, which defines the type of an event. For example, under the "MNT repair" event, if the Event-Type is set to *failure*, the technician may collect failure data, whereas if the Event-Type is *repair*, he can only collect data about repair activities. In the following sections, the author discusses each of the above categories in more detail.

3.6.1 Sign Down an Entity

A maintenance personnel or a process engineer may *sign down* an entity (equipment) for many different reason depending on the state of the equipment at the time. For example, any scheduled or unscheduled maintenance or any processing engineering work, or a material handling can be a valid reason to sign down an equipment. Therefore, special *events* has been defined to report the changes in the function of the machine as the time progress. Author presents the following events under which an equipment can be *signed down.*

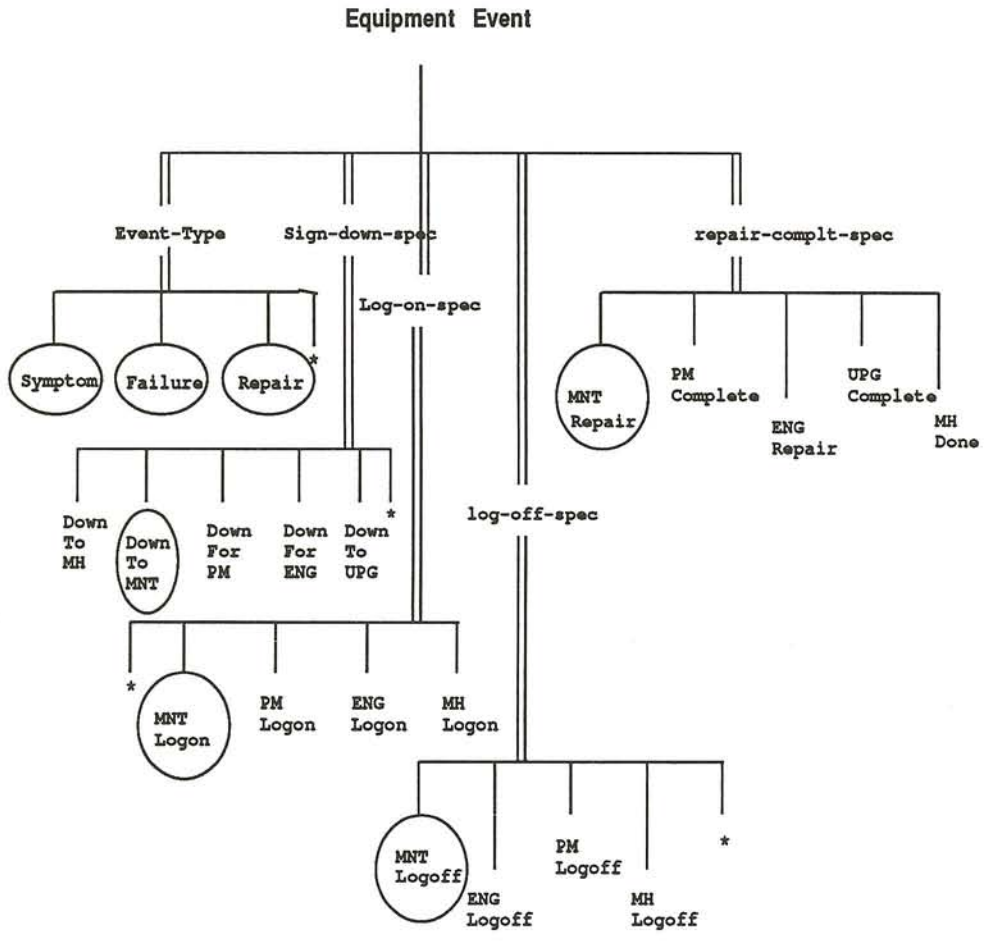


Figure 3.4: System Entity Structure representation of equipment events

Down to MNT	(maintenance after failure)
Down for PM	(preventive maintenance)
Down for ENG	(process engineering)
Down to UPG	(equipment upgrading)
Down to MH	(material handling)

By choosing one of the above events, the personnel can record the change in the status of the machine, and he may also provide additional information about the behavior of the machine, if appropriate. These data entries provide useful sources of information for analyzing the equipment reliability, maintainability, and availability, and furthermore, they can be used for analytical purposes relating to equipment maintenance resource management such as an Expert Diagnostic System or a Maintenance Cost Analysis.

3.6.2 Log on to start working on an entity

After an equipment is signed down, usually, it takes a while before a repair person starts working on the equipment. Most of the time, the necessary personnel, such as a technician, is not immediately available and must be called on duty. Therefore, there arises the need for a different event to report this change of activity on the machine in order to be able to distinguish between the time when the equipment was in a waiting status and the time that a maintenance person actually worked on the equipment. It is important to remember that knowledge of the precise time span

needed for equipment maintenance is critical for analyzing equipment performance. The following *log on* events are a few examples defined to report the start of work on a piece of equipment.

MNT Logon	(Start maintenance after failure)
PM Logon	(start preventive maintenance)
ENG Logon	(start process engineering)
MH Logon	(start material handling)

3.6.3 Log off to stop working on an entity

The maintenance person is sometimes interrupted before repair has been completed. This may happen due to a coffee or lunch break, or it may happen because the technician requires some parts that are not available, or because the technician is preempted to attend to another higher priority event. Such a change of activity must also be recorded in order to be able to calculate the true repair time. Therefore, a set of *events* has been designed to report the time at which work on the equipment was interrupted, and to report what activities were performed during the time of maintenance. Thus, *log off* events are used to either denote the completion of work (i.e., a successful maintenance), or the interruption of work due to any of the above scenarios.

MNT Logoff	(Stop of maintenance after failure)

PM Logoff	(stop of preventive maintenance)
ENG Logoff	(stop of engineering runs)
MH Logoff	(stop of material handling)

This discussion also sheds some additional light on the previously described *log on* events. Log on events are used to either denote the true beginning of a maintenance activity or the resuming of work after it had been previously interrupted.

3.6.4 Repair Completed

When maintenance work on a piece of equipment is finally completed, the maintenance personnel logs a *repair completed* event. This is used to report findings concerning the nature of the problem, such as the faulty component along with its failure mode, as well as the final repair activities. The following *events* are designed to fulfill these actions.

MNT Repair	(completion of maintenance after failure)
PM Complete	(completion of preventive maintenance)
ENG Repair	(completion of engineering runs)
UPG Complete	(completion of upgrading equipment)
MH done	(completion of material handling)

3.7 Data Base Management System (DBMS)

In this section, the author introduces three types of data bases, *Workstream*, *Ingress*, and *Menu*, used in this project. He first explains the function of each of the data bases as well as their importance. Finally, he discusses how the three data bases are related to each other.

3.7.1 Workstream DBMS

Workstream is an On-Line DBMS that uses a terminal as its primary means of interacting with the user. Workstream contains several independent modules, each of which is designed to carry out a specific task. For example, one module keeps track of a wafer during its production, while another module is designed to collect engineering parameters and data at wafer processing points. The module that this project deals with is used to track information regarding wafer process equipment and other entities used in the FAB, such as tools, fixtures or operators. A special task of this module, which is related to this thesis, is to aid the maintenance personnel, to collect information related to equipment behavior, its faulty component and its failure mode, and the repair activity performed on the machine. This is accomplished by use of the Hierarchical Data Collection system, HDC.

Data is collected through keyboard entry and is accomplished through "event logging" activities. When an operator or a technician performs a function, he or she must first log a proper event into Workstream. There is some concern about possible

inaccuracies in the collected data due to faulty time stamping. If it was left up to the technician if and when he or she logs the events into Workstream, some events would not be logged at all, while others would be logged after the fact thereby falsifying the time stamp. The system cannot accurately monitor total event/status time if events are logged after the fact. For collecting data such as a symptom, a failure, or a repair activity, a proper *Event-Type* as well as the event itself *must* be logged into Workstream before the HDC can be activated. For example, when collecting a failure, both the "MNT repair" event and the "failure" Event-Type must be logged into Workstream before HDC can be called. HDC then brings up the failure menu related to the family to which the equipment belongs. At this time, the technician may collect the failure by updating the menu or by adding another element to it. In the sequel, collected data are automatically reported back to Workstream for long term storage. All of these data are saved in a special *Area* designated as *Module.* Figure 3.5 illustrates the interaction between the Workstream DBMS and the Menu Data Base using HDC.

3.7.2 Ingress DBMS

Ingress is a relational data base with unique build-in utilities for creating forms for data queries and analysis. In Ingress, data are stored in tables, which can be merged and selected for analysis purposes. Furthermore, as a relational data base, it provides means for easy flexible english-like ad hoc data queries. Thus, Ingress is an ideal

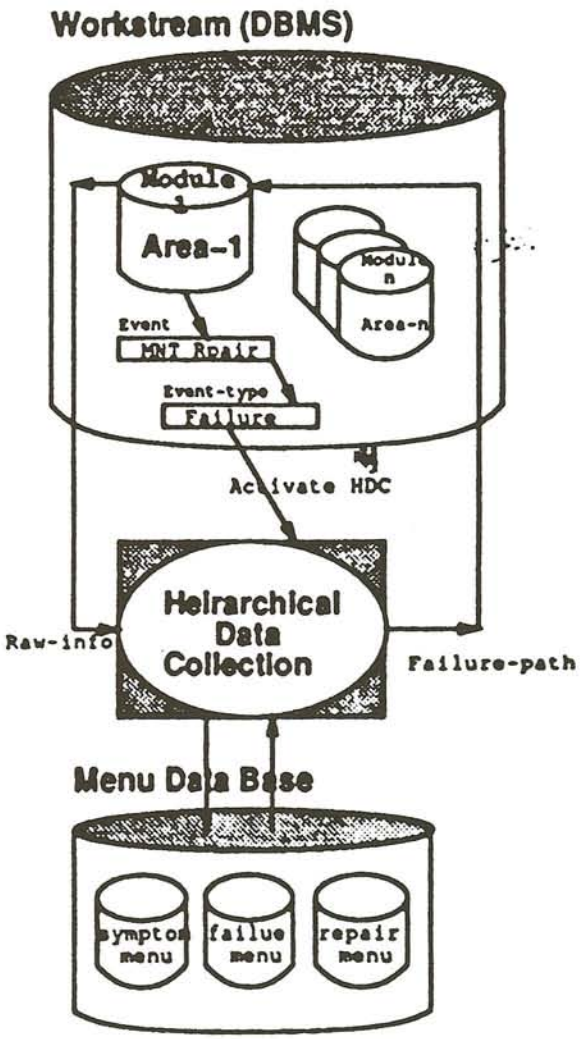


Figure 3.5: Interaction between Workstream and the Menu Data Base

system for relating data items to each other, for flexible data analysis, and for data presentation. While Ingress is a good DBMS choice for data analysis, Ingress would not be suitable for storing the menu data bases, because Ingress makes tremendous demands on computing resources, which make the data storage and retrieval process slow. There is also the danger that Ingress may not be available when needed. While this would be inconvenient for data analysis, it would be catastrophic for data collection. It would mean that either a repair activity would have to wait until Ingress becomes available or that data collection cannot be accomplished on-line. However, unavailability of the menu during an event invariably results in inaccurate time stamping of the collected data. Both alternatives are therefore unacceptable.

In Ingress, as discussed above, data can be tabulated and related to each other for the purpose of data analysis. Equipment reliability, availability, and maintainability can be addressed here. Furthermore, questions such as what is the total number of hours to repair a given entity or a given failure type? or what is the frequency of repair for a given entity? can be addressed here. This information can then be used for the development of the knowledge base of the Expert Diagnostic System.

Data can be extracted from Workstream and transferred to Ingress by another data conversion program, Extract, for the purpose of data analysis and (graphical) presentation. These data can also be used to build the knowledge base for the Expert Diagnostic System.

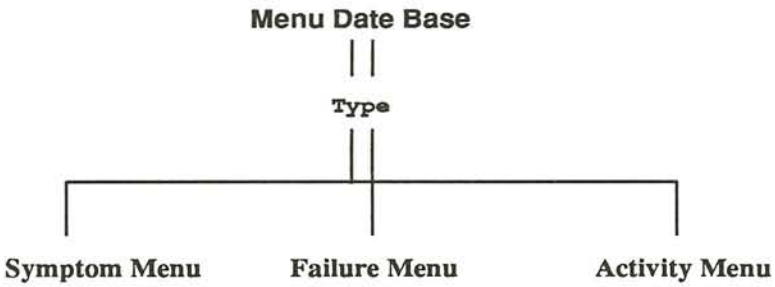


Figure 3.6: Specialization of Menu Data Bases

3.7.3 Menu Data Base

In section 3.3, Figure 3.1 illustrates the decomposition of a facility into numbers of families. It further shows that each family contains many pieces of equipment of similar or the same type and model. Therefore, each of these families requires a separate *Menu Data Base* to exploit the similarity between the entities contained in that family. Furthermore, there should exist separate menus for collecting symptoms, failures, and repair activities. Figure 3.6 illustrates these menus as specializations of the Menu Data Base.

The contents of these menus, how they are designed and structured, will be discussed in Chapter 4, Sections 2 through 4. The main purpose of these menus (symptom, failure, and repair activity) is to formalize knowledge related to equipment behavior, the faulty component and its failure mode, and a description of what was done to solve the problem. These menus contain these pieces of knowledge for each type of equipment. They are designed in a hierarchical fashion, and are accessed through the *Hierarchical Data Collection* (HDC) module. Figure 4.2 illustrates the

hierarchical decomposition of the failure menu. In addition to HDC, there exist utility routines, called *Menu Maintenance Routines* (MMRs), to maintain the content of these menus. MMR is used by the family managers to maintain equipment information as it is being updated by the operators and technicians. Since operators and technicians can make mistakes in reporting pieces of information, new data items are kept local until the respective family manager can validate the new information (i.e., changes of the menus). For this purpose, he uses MMR to report the updated menus back to Ingress.

Figure 3.7 illustrates the relationship of all 3 data bases through MMR, HDC, and the Extract program. HDC and MMR were designed by AMD in Austin, Texas. The participation of this author in the design and implementation of these two software products is limited to a few utility modules of the MMR software, and the enhancement of HDC as the symptom and repair activity prototypes were developed.

3.8 Comments in Data Base

In this section, the author discusses two types of *comments* collected by operators and technicians during the operation and maintenance of equipment. These comments, as well as other data, are collected through *Workstream* and have been saved in a designated area, *area-1*, of the data base. The first type of comment is a description of equipment behavior at the time of *sign down*, when an operator detected a discrepancy in machine behavior or product quality. When an equipment fails, the

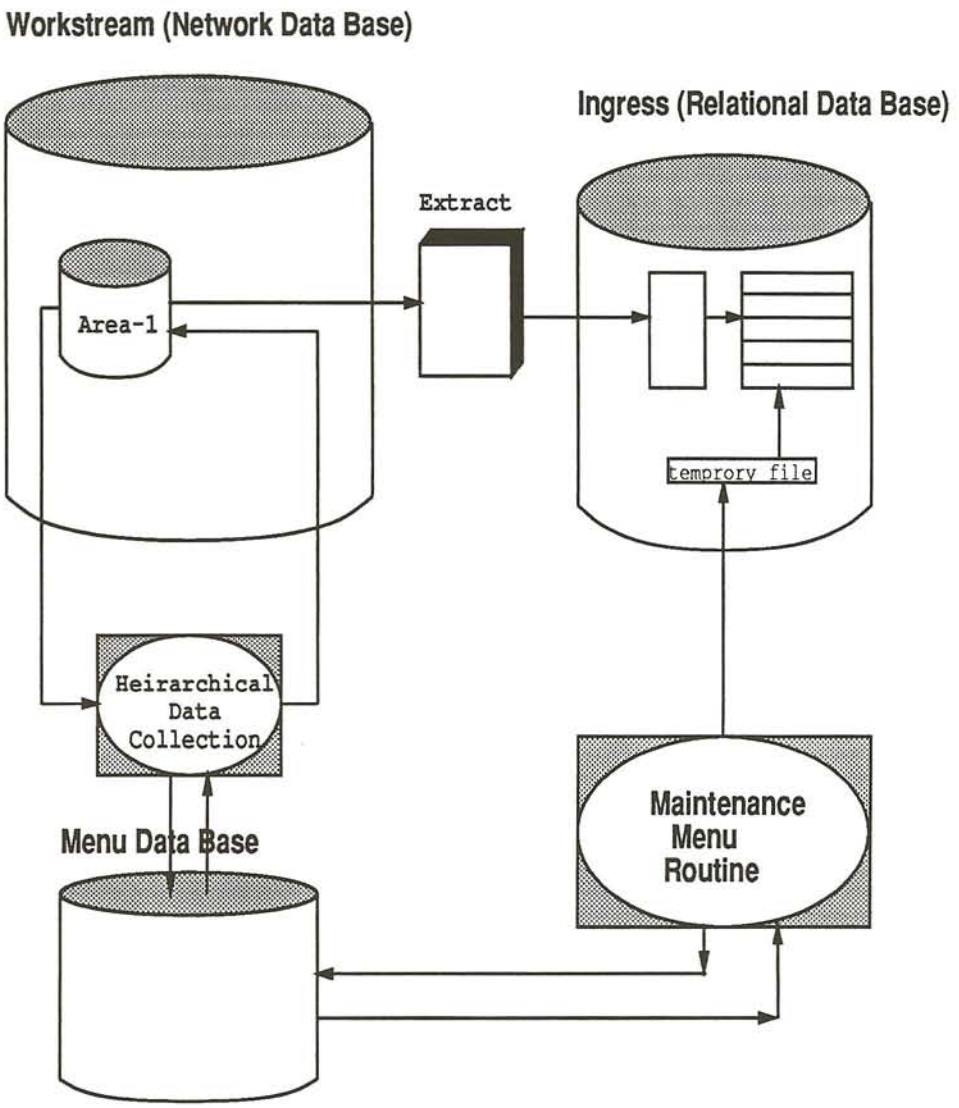


Figure 3.7: Workstream, Ingress, and Menus via. HDC and MMR

operator signs down the machine by logging a "Down to MNT" event into Workstream. He then reports the observed discrepancy in his own terms and language. These comments are referred to as *symptom comment* in Chapter 4.

The second type of comment is a description of activities executed by a technician in order to bring the machine back to a condition in which it can perform its intended function. These activity comments may be relevant or irrelevant to the real problem of the machine. Figure 3.8 presents sample information collected through Workstream using the events described in Chapter 3.

This chapter introduced the manufacturing environment and discussed issues relating to collecting and measuring knowledge about performance of the machine park. It also presented how "equipment time" and "equipment activity" need to be categorized in order to measure a certain aspect of equipment performance. Finally, this chapter concluded with a discussion of collected comments about equipment behavior and activities. In the next chapter, these comments will be utilized as a way to study characteristics of the symptom and repair activities and their relationships with failures.

```
Event: Down to MNT
Facility: FAB10
Family: DSW
Entity: DSW8
Status1: Wait for maintenance
Event-Type: Failure
Date: 03/12/90
Time: 04:12:46
Queue-time: xxx
Technician: Smith
Failure ID: 42
Symptom Comments: Failed prealign
        check.
```

```
Event: MNT Logon
Facility: FAB10
Family: DSW
Entity: DSW8
Status1: check
Event-Type: check
Date: 03/12/90
Time: 04:55:54
Technician: George
```

```
Event: MNT Logoff
Facility: FAB10
Family: DSW
Entity: DSW8
Status1: Wait
Date: 03/12/90
Time: 05:52:41
Technician: George
Repair Comments: Cleaned the Gears.
        Still has Y problem
```

```
Event: MNT Logon
Facility: FAB10
Family: DSW
Entity: DSW8
Status1: check
Event-Type: check
Date: 03/12/90
Time: 07:31:26
Technician: Smith
```

```
Event: MNT Repair
Facility: FAB10
Family: DSW
Entity: DSW8
Status1: Wait
Event-Type: Failure
Date: 03/12/90
Time: 08:04:27
Technician: Smith
Failure ID: 42
Comments: Prealigned T/A backslash and warn gears. Set up loading
          correction.
```

Figure 3.8: An example of collected information from Workstream

CHAPTER 4

Rapid Prototyping of Symptom and Repair Activity Knowledge Bases

4.1 Introduction

This chapter discusses the methodologies used to collect symptom, failure, and repair activity data. It also discusses how a knowledge base can be built along with its application program to implement a trouble shooting guide necessary to trouble shoot the equipment failure. Section 4.2 deals with issues concerning failures. The Hierarchical Data Collection System (HDC) for collecting data about failures along with the Maintenance Menu Routine (MMR) for maintaining those data are explained. Section 4.3 addresses issues relating to symptoms, and investigates important questions such as: What is a symptom? How can symptoms be classified? How can a symptom menu be developed? How can a symptom be related to the failure that caused it? How is a knowledge base built? and finally: How is the application program developed? A symptom prototype is built and discussed, and a specification for collecting symptoms and building a knowledge base is presented. Section 4.4 deals

with issues relating to repair activities. Important questions concerning repair activities are studied and summarized. A methodology for collecting data about repair activities and relating them back to a failure is presented. A repair activity prototype very similar to the symptom prototype is designed. This chapter concludes with a discussion of the overall application program, which is a collection of the application programs designed in Sections 4.3 and 4.4.

4.2 Failure Collection

4.2.1 Introduction

This section deals with issues concerning failures. First, the author explains what constitutes a failure and discusses the issue of single failures versus multiple failures. A strategy to build a failure menu is described in section 4.2.3. Finally, this section describes the *Hierarchical Data Collection* (HDC) System developed to collect data about failures, along with the *Menu Maintenance Routine* (MMR) which maintains the collected data.

4.2.2 Issues Concerning Failures

A failure is an incident that causes a machine to fail to perform its intended function. However, there are failures that are not machine related: for instance, a failure in the material handling process, or a failure caused by the environment, which can introduce a disturbance in the quality of the product. These failures are

Failure MOde	Component && Failure MOde
Out of adjustment	Flatfinder out of adjustment
Out of alignment	Paddle out of alignment
Out of tolerance	Temperature out of tolerance
Dirty	Backlash gear dirty
Worn out	Backlash gear worn out
Burned out	Light burned out
Defective	Horizental sync board defective
Loose	Cable to camera loose
Bad connection	Bad connection in microscope socket
Locked up	System locked up

Figure 4.1: Representative list of failure modes

classified under "Not a Machine Failure." Machine related failures can be described by a path showing the hierarchy of the systematic decomposition of the equipment's hardware beginning with its Major-SubSystem down to the faulty Component and its Failure Mode. A single component may have many failure modes. Each failure mode is a manner in which a component may fail. Consequences of a failure can be assessed for each failure mode. These may vary from inconvenience to serious down time, from rework to broken wafer. The criticality of each effect of a failure can then be used to weight the failure in developing the expert diagnostic system. Figure 4.1 provides an exemplary list of failure modes as they are used in menus.

One of the most important issues concerning the analysis of failures is the issue of a *single failure* versus *multiple failures.* When an equipment fails to perform its intended function, the expert diagnostic system (or the technician) may be presented with a list of logically related failures. The diagnostic question becomes whether multiple failures have indeed caused the machine to go out of production, or whether there exists one primary failure that initiated the problem while the remaining failures were consequence failures.

Consequence failures may be caused by a delay between the time when the original failure occurred and the time when the failure was recognized and the machine was signed down. For example, if a material handling failure occurred while transporting a wafer from one place to the next in the manufacturing process, the misaligned wafer may scratch and thereby damage a tool on its way. The original failure was the material handling failure, yet, there exists a consequence failure that may be reported independently. It is not always easy to relate these failure reports to each other.

However, consequence "failures" are also frequently being reported due to the nature of the repair activity initiated by the original failure. For example, in the event of a "leaking o-ring on a laser rod," to repair such a leak, the rod would have to be removed. To remove the rod, the mirrors at either end of the cavity would have to be removed. After drying the rod and replacing the ring, the mirrors need to be realigned, as would all optical components in the path to the wafer surface. Now, one scenario for the logging of the failure(s) over several shifts could be as follows:

The first technician to attack the problem might remove the rod, dry and remount it, then mount but not adjust the mirrors. At the end of the shift, he would log off and report the repair activities performed interpreting the failure as "o-ring leaked." The next technician might adjust the mirrors, log off and report his activities, identifying the failure mode as "mirror out of adjustment." Finally, the last technician to work on the equipment would then adjust the delivery optics bringing the equipment back to its normal condition. The failure in the eyes of the last technician would be "delivery optics out of adjustment." In summary, what should be considered as a failure? After interviewing many domain experts and family managers, all strongly agree that only one failure has occurred, namely: "o-ring leaked" even though three different "failures" were reported and could be counted. In the above case, three constituent actions have taken place as a consequence of one single failure.

Those interviewed strongly believe that in up to 99% of all cases, there exists a single root failure causing a machine to go out of production, and that it is hardly possible to find an example of multiple failures that occur simultaneously to the same machine without a common root failure. Thus, the capability of diagnosing simultaneous multiple *unrelated* failures is not an issue. However, the two examples presented in this section demonstrate clearly that consequence failures are quite common. The two types of consequence failures outlined above are quite different in nature. Yet, they have a similar appearance in the failure log: in both cases, several failures are

being reported that have a common cause. It is a diagnostically difficult problem to reconstruct the causal chain from the failure log.

4.2.3 Failure Menu

The *Failure Menu* is a hierarchical data structure organized in accordance with the hardware decomposition of the equipment, along with the failure modes for each component listed in that menu. In Section 3.3, a hardware decomposition of a piece of equipment was discussed, and a system entity structure was presented. Here, the author will describe how this hardware decomposition can be utilized along with other factors to develop a failure menu. In the first level of the menu, there exists a list of Major-SubSystems as they were originally recognized by the family manager. A Major-SubSystem is the largest subdivision of a machine. Each Major-SubSystem can be decomposed into several Minor-SubSystems. Each of these Major- or Minor-SubSystems is called a *folder*. Thus, folder is a name that describes a menu choice, a subsystem, which has another menu associated with it. A folder may contain several items or other folders. *Item* is a synonym for Component. It describes the lowest level of menu listing. At the lowest level of the menu, each item (Component) is associated with its failure modes. Figure 4.2 illustrates a hierarchical menu structure showing the above concepts. Figure 4.3 introduces a modified, simplified, truncated failure menu to show an example of a Major-SubSystem, Minor-SubSystem, Component, and Failure Mode. For example, AWH is a Major-SubSystem while TRANSFER ARM,

PREALIGNER, RECEIVE ELEVATOR, AND SEND ELEVATOR are considered to be its Minor-SubSystems. Furthermore, the paddle, transfer arm, backlash or warm gear, and motor speed are the Components of TRANSFER ARM (Minor-SubSystem). Failure Modes for these components are: out of adjustment, dirty, and worn, respectively.

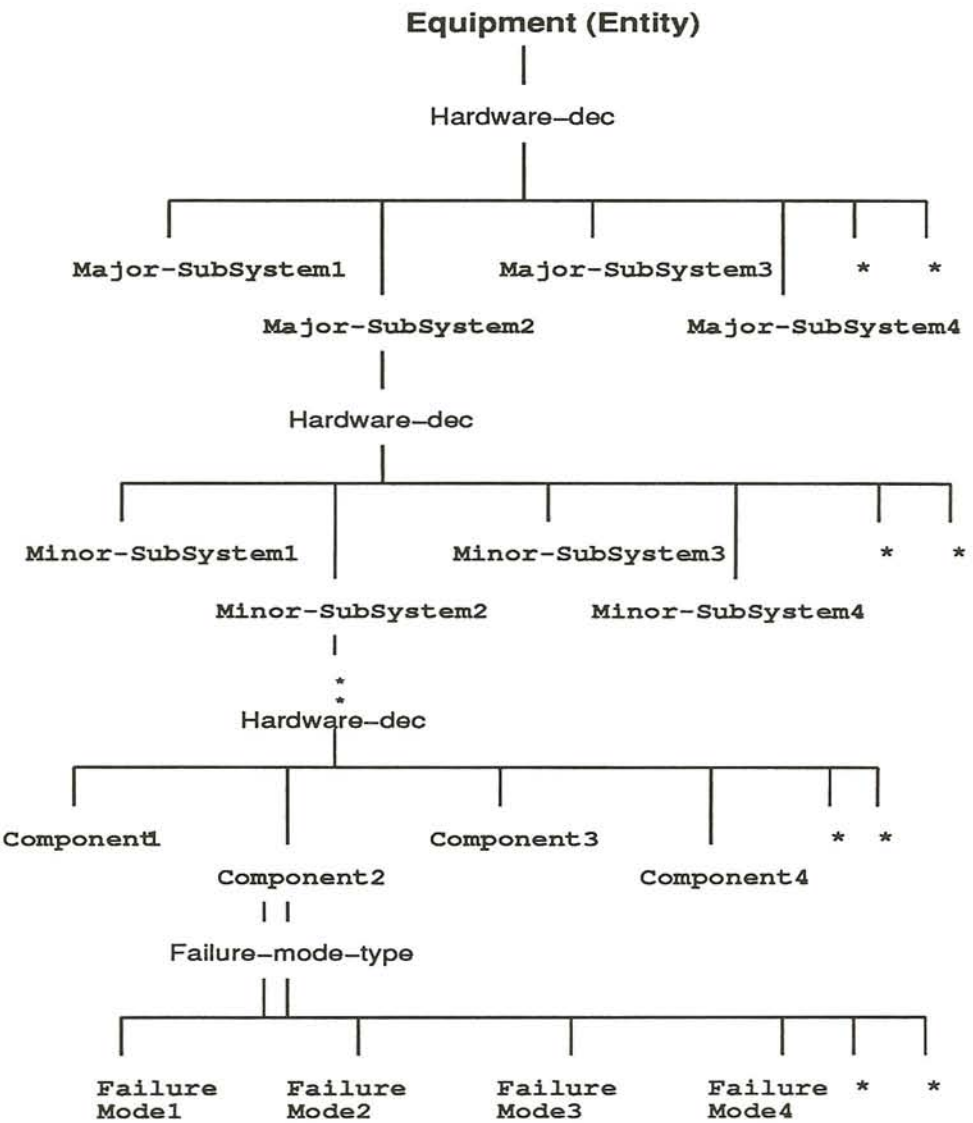


Figure 4.2: Systematic Classification of Failures

```
STAGES/CHUCK
        CHUCK/THETA
                  chuck dirty
                  theta drive jammed
                  theta home sensor defective
                  wafer present sensor out of adjustment
                  other
        STAGE MECHANICAL
                  barry table air hose leaks
                  barry table out of level
                  x motor defective
                  x motor gearbox dirty
                  other
AWH     TRANSFER ARM
                  paddle out of adjustment
                  transfer arm dirty
                  backlash or warm gear worn
                  motor speed out of adjustment
                  other
        PREALIGNER
                  prealign theta out of adjustment
                  spindle center out of adjustment
                  flatfinder drive motor defective
                  flatfinder sensor out of adjustment
                  motor speed out of adjustment
                  spindle height out of adjustment
                  wafer present sensor out of adjustment
                  other
        RECEIVE ELEVATOR
                  boat guides out of adjustment
                  cable defective
                  comb sensor out of adjustment
                  drive motor defective
                  index flag out of adjustment
                  motor speed out of adjustment
                  wafer present sensor outof adjustment
                  other
        SEND ELEVATOR
                  wafer preset sensor out of adjustment
                  boat guides out of adjustment
                  cable defective
                  comb sensor out of adjustment
                  drive motor defective
                  index falg out of adjustment
                  motor speed out of adjustment
                  wafer boat stuck
                  other
```

Figure 4.3: A Failure Menu

```
RMS
        PLATEN
                platen dirty
                platen out of adjustment
                platen vacuum solenoid defective
                align motor defective
                align sensor defective
                tower out of alignment
                other
        ELEVATOR
                bar code reader defective
                elevator motor defective
                lead screw sticking
                travel sensor out of adjustment
                bar code reader cable defective
                elevetor motor cable defective
                lead screw worn
                other

COLUMN/FOCUS
        ACS
                acs locked up
                acs out of adjustment
                monitor defective
                chasis defective
                sensor defective
                other
AWA/CCUTV
        CCU MODULE
                illuminator lamp burned out
                power cable defective
                camera control module defective
                canera defective
                ccu setup out of adjustment
                power supply out of adjustment
                vertical module out of adjustment
                vertical weep module defective
                other
```

Figure 4.3 - A Failure Menu (continued)

As the above figure indicates, there is an option "other" at the bottom of each folder. These options are to give operators or technicians the capability to attach new items or folders to the existing menu. Of course, these new items or folders will have to be checked and validated by the family manager, using the MMR software, for correctness and usefulness.

To develop such a menu, a text file containing the Major-Subsystems, the Minor-SubSystems, and related Components along with their Failure Modes needs to be edited. A software has been developed to take this text file and create a data structure that HDC can work with. The menu can evolve as users collect data.

4.2.4 Hierarchical Data Collection

The *Hierarchical Data Collection* (HDC) system is software that has been linked to Workstream, and enables maintenance personnel to collect data related to equipment performance. This software can be called from Workstream to display any one of the menus such as the symptom, failure or repair activity menu, depending on the event the machine is in. HDC assists the user in stepping through the various levels of the menu. Each level is presented as a separate screen or window from which a choice must be made. When a field is selected, its *number of occurrence* is updated. The number of occurrence is the frequency of the given field which indicates how many time that field was selected. The frequency will be incremented upon its selection. All selections from previous level menus will be displayed as a reminder and will form

the path that describes the collected data. This path will be sent back to Workstream for storing and further analysis. In the case of the absence of an adequate field, HDC allows the user to enter his own description by selecting the "other" option. *Other* allows the user to enter a conditionally valid description at the next and possibly subsequent levels of the menu. As a result, a mail message will be forwarded to the supervisor and/or equipment engineer who will then be required to follow up either by validating the new description, or by reassigning it to an existing description.

The following example of a failure collection using HDC is given to provide a better understanding of how these steps may be taken. A failure description may be collected when a technician has identified the faulty component and also has completed the repair activities, and as a result, the equipment is restored to functioning. At that point, the technician may log the "MNT Repair" event into Comets and set the event type to *failure* in order to activate HDC in the appropriate mode. HDC will display a specific failure menu related to the family to which the equipment belongs. At the first level of the menu, the user is faced with a list of Major-SubSystems and needs to make a selection that leads to the faulty component. At this point in time, the user is aware of which component has failed, and he also knows which Major- or Minor-SubSystem the faulty component belongs to. If there is any doubt or confusion as to which selection would be best to make, the *search* utility may be used to bring up all plausible paths that contain such a component related to the specific failure. As a selection is made, the next screen (menu) appears for further selection. When the

selection is completed, the selected path will stored as the failure description. The *failure description* is then sent back to Workstream for storage and further analysis. A graphical presentation of Workstream, HDC, and the menus is given in Figure 3.7.

4.2.5 Menu Maintenance Routine

The *Menu Maintenance Routine* (MMR) facility is a collection of several utility modules. This code has primarily been developed to maintain the collected data in the menus. Several utilities were developed that are useful to a variety of different users, from the family managers down to the programmer, such as *Add*, *Validate*, *Make*, *Outline*, *Maintain*, or *ItemEdit*. A few utilities and their related benefits are as follows: **Add**: is a module that allows a family manager to add a new item or folder to an existing menu file. **Clean** is a module that works as a garbage collector; it crunches the file after the family manager has invalidated an item or folder. **ItemEdit** is software that can be used by a programmer or an expert who is knowledgeable about the menu files. This software maintains existing items and folder records at the data structure level. It allows programmers to dynamically update item and folder information at the data structure level. This software also maintains the menu file data integrity in case of bugs in production utilities. It is also helpful in testing new utilities and fixes. **Maintenance utility** can be used by family managers to dynamically update and manipulate existing menu information for items and folders. **Validate** is software that a family manager uses to validate a

new entry entered by maintenance personnel. He can delete, reassign, or modify the entry. In case of any modification, the changes will be reported back to Ingress to update the data in the data base. Figure 3.7 illustrates the complete cycle of data collection starting including Comets, HDC, the menus, MMR, and Ingress.

4.3 Symptom Prototype and Knowledge Base

4.3.1 Introduction

This section introduces issues related to the collection of *symptoms*. A study of symptoms for equipment belonging to the DSW family of FAB15 is presented.

Questions such as: What constitutes a symptom? and: What are the characteristics of a symptom? are investigated. A methodology for classifying symptoms has been developed and field-tested. Sections 4.3.3 and 4.3.4 present this methodology. *Symptomic factor* and *Symptomic classification* are used as a systematic way to classify symptoms. In Section 4.2.4, question concerning the *who*, *how* and *when* of symptom collection are studied. The relationship between symptoms and failures is also addressed. This section presents furthermore the symptom prototype built to support the specifications for collecting symptoms, for relating symptoms to failures, and for building a knowledge base. Finally, the specification (requirement definition) for collecting symptoms is also presented in this section.

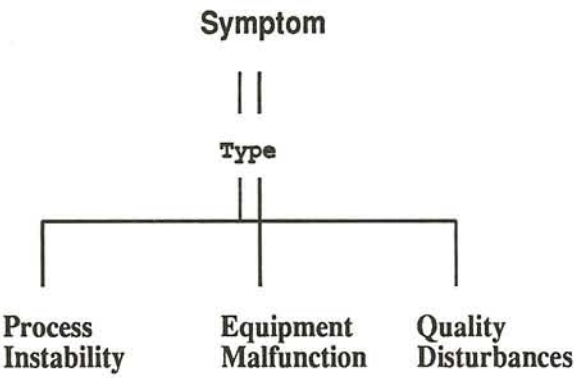


Figure 4.4: Specialization of symptoms

4.3.2 What is a Symptom?

A symptom is defined as *a sign or indication of something* [Webster dictionary]. In our context, a symptom sometimes indicates an *equipment malfunction* that is caused by a faulty component. On the other hand, many symptoms can be related to an *instability in the processing* of the product, i.e., they are related to problems in material handling. Again others may be characterized as problems with product quality. They often are related to equipment calibration problems. For example, a quality disturbance can be caused by environmental factors such as changes in humidity or temperature in the chamber. Such factors can influence the calibration of equipment, and thereby reduce the quality of the produce. Figure 4.4 illustrates the types of symptoms represented through a system entity structure specialization.

It is very difficult to pinpoint a symptom by keeping track of the hardware decomposition of equipment, even if the observed symptom results from an equipment malfunction. This difficulty is due to the lack of operator knowledge at the time of

sign down (Down to MNT). The operator does not know at that time which is the faulty component. It is also clear that a hardware decomposition would not be useful in pinpointing the symptom in the case of either of the other two symptom types: the process instability and the quality disturbances. Therefore, there is a need for a methodology to systematically classify symptoms, which can accomplish the above task.

In pursuing this question, the author has studied a large amount of comments about equipment behavior that were collected during sign down and that were stored in the data base. These comments reflect the operators' explanation of observations about equipment behavior and product, expressed in each operator's own terms and language. Below are a few examples of such comments:

Gates window jumps up and down
Will not send wafer to prealigner
Picture bad
Bad picture
No focus
Bad focus
No vacuum on chuck
Won't send wafer out
Fatal disk error
Paddle will not take wafer from chuck to receiver boat.
Inconsistent alignment
Arm won't transport wafer
Arm won't drop wafer on chuck
Misaligning wafer
Exposure went out
Double image

Receiver paddle won't release wafer

The investigation of such comments led to the creation of a methodology for systematic classification of symptoms using two new concepts that will be called

Symptomic factors and *Symptomic classification.* These are defined in the next two sections.

4.3.3 Symptomic Factor

The *Symptomic factors* constitute a means of classification of symptoms. Using Symptomic factors, a symptom can be classified into many categories. Each new category conveys a more precise picture than the previous one. Symptomic factors vary for different symptoms. They are based on the equipment's functional, physical, observational, and procedural characteristics. A typical Symptomic factor decomposition for an equipment family is shown in Figure 4.5.

4.3.4 Symptomic Classification

The *Symptomic classification* is used to decompose symptoms in a hierarchical fashion. At the highest level of the hierarchy, symptoms are described in their most general form. As we proceed down the hierarchy, the symptoms become more specialized and thereby provide a better and more precise picture. Figure 4.6 illustrates a Symptomic classification of problems for a representative family in the FAB.

At the first level of this hierarchy, problems are classified into several types including: *alignment problem*, *handling problem*, *process problem*, *focus problem*, *control system problem*, *environmental problem*, etc. This decomposition characterizes a symptom in a very general fashion. Each of these categories can be further specialized (classified) into several subcategories based on the their symptomatic factors.

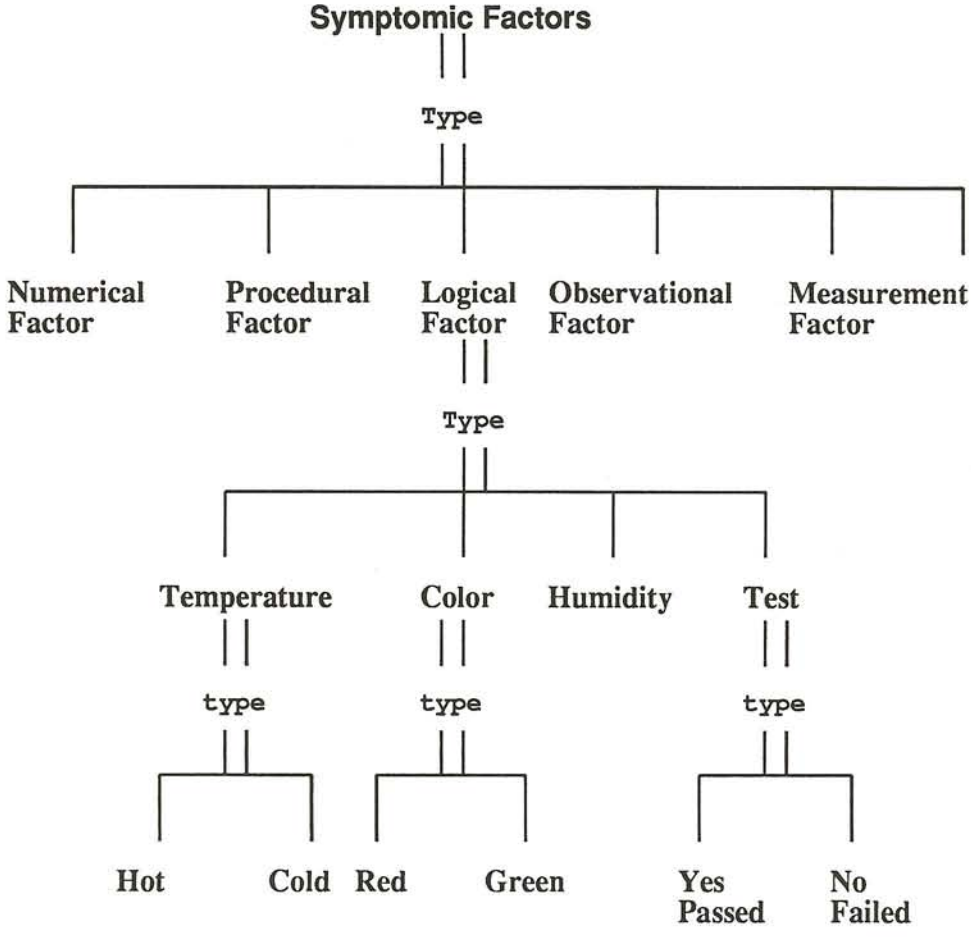


Figure 4.5: Symptomic factor decomposition for an equipment family

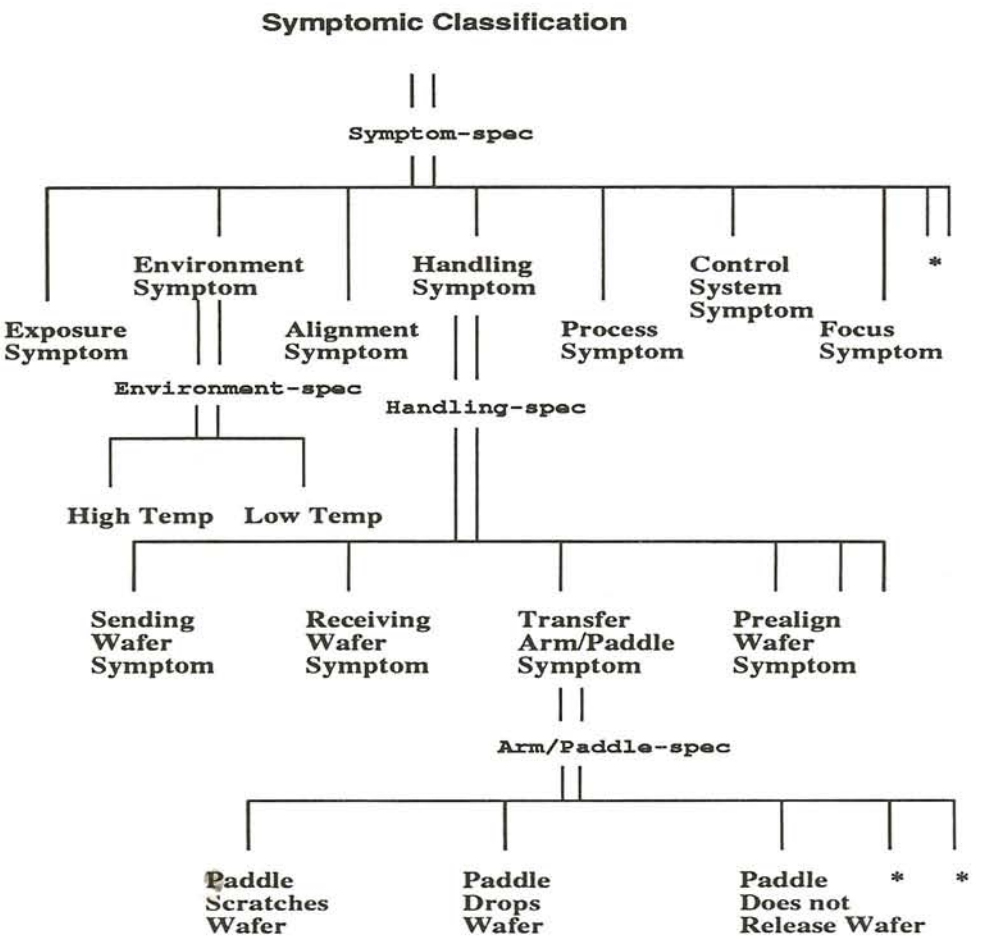


Figure 4.6: Systematic classification of symptoms

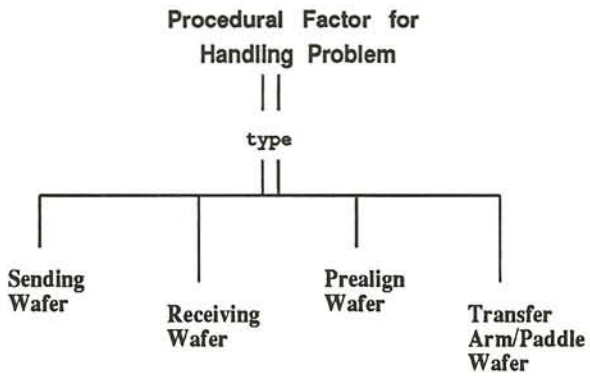


Figure 4.7: Procedural factor decomposition for wafer handling

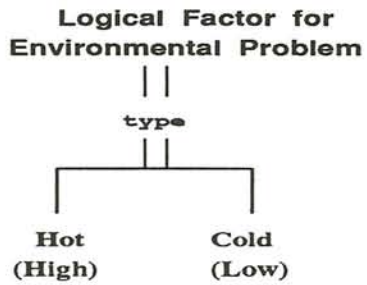


Figure 4.8: Logical factor decomposition for environmental problem

For example, handling problems can be classified into *sending wafer problem*, *receiving wafer problem*, *prealign wafer problem*, *transfer arm/paddle problem*, etc. This classification is based on the *procedural Symptomic factors* that have been specified for handling problems. A procedural Symptomic factor decomposition for the above example is shown in the Figure 4.7.

Another example of a classification at the first level is the classification of *environmental problems* into problems of *Hot* and *Cold*. For this example, the Symptomic factor decomposition is recognized to be a *logical factor decomposition* as shown in Figure 4.8.

At the second level of the hierarchy, each symptom can be further classified (specialized) into several categories. Figure 4.6 shows the classification of *transfer arm/paddle* into *paddle scratches wafer*, *paddle drops wafer*, and *paddle does not release wafer.* The Symptomic factor for this classification was recognized to be *procedural.* The classification can be continued if the family manager recognizes the existence of a Symptomic factor that can categorize the symptom further in a more clear and precise manner.

Symptoms are the most precise and clear messages that can indicate a malfunction. They are always in their most specialized form and are located at the bottom level of the hierarchy. For instance, in the previous example, "paddle scratches wafer" is considered to be a symptom, which is a specialization of "transfer arm/paddle." The path from the top level down to the bottom level is called the *symptom description.* For example, the symptom description for the above example would be:

handling symptom; transfer arm/paddle symptom; paddle scratches wafer

As the above description indicates, a symptom in its general form would not be very useful in diagnosing an equipment failure. A Symptomic factor is the result of a further classification of symptom. In this example an observational factor, such as to watch how the handling process takes place, has lead the operator to classify the handling symptom more precisely as "transfer arm/paddle symptom." Off course, this classification is only feasible for equipment that is covered by glass where an operator or technician can observe the process of handling the wafer. The machines

in FAB10 are of the type with glass cover. However, there exists equipment without a glass cover (e.g. in FAB15) and, as a result, the process of wafer handling cannot be observed. In this case, a numerical code displayed on the equipment's monitor can be helpful in further classifying the problem. These codes are usually documented in the equipment reference manual. Each of these codes can specify whether a time out was due to a problem with "sending elevator" or "receiving elevator," etc. This Symptomic factor is referred to as *numerical Symptomic factor.* An example of a symptom description using the numerical Symptomic factor would be:

AWH ERRORS; code 24

Symptom menus for FAB10 and FAB15 are illustrated in Figures 4.10 and 4.11.

4.3.5 Relating Symptoms to Failures

This section investigates important issues such as *who* should collect symptoms, *when* should symptoms be collected, and *how* to relate symptoms, especially multiple symptoms, to the corresponding failure. In pursuing the above investigation, this author has developed several programs to extract, for further study, symptom comments, which are available in the data base. This particular study was done on two different FABs, one using two months of data and the other using six months of data. One program designed to relate a failure to its corresponding symptom comment. The program simply searches the data base for a specific failure (failure ID) which was collected during a "MNT repair" event. Upon its finding, the search

will be continued for its corresponding "Down to MNT" event in order to collect the symptom comment. This process will be continued until all episodes are tested. A few sample results are shown in Table 4.1.

Table 4.1 - Failure with corresponding symptoms comments

Failure ID Failure Description	181 DSW; AWA/CCUTV; CCU MODULE; ccu setup out of adjustment
Comments	focus problem on screen focus walk focus is bad focus walk, not across Texas either need to adjust gate angle key won't lock on left gate not locking on lamp fluctuation balck line across screen bad picture on screen, crooked and blurry bad picture picture not clear gates won't lock on
Failure ID Filure Description	6 DSW; AWH; PREALIGNER; flatfinder sensor out of adjustment
Comments	prealigner shaking machine won't start prealign no working prealign not relating wafer correctly prealign won't release wafer
Failure ID Failure Description	27 DSW; AWH; SEND ELEVATOR; wafer present sensor out of adjustment
Comments	won't send wafer out sender boat sensor not working won't send wafer out sender elevator will not go down
Failure ID Failure Description	3 DSW; PREALIGNER; motor speed out of adjustment
Comments	wafer won't start keeps timing out on every wafer inconsistent alignment

These examples clearly indicate how many times a particular failure has occurred during a period of six months. The number of occurrences is equivalent to the number of comments listed under the failure description. Furthermore, while these failures usually cause similar symptoms, there are occasions when this is not the case. Table 4.1 clearly shows such an example. Furthermore, some failures have a tendency to occur more often than others. This would suggest a need for prioritizing the *preventive maintenance* of equipment in order to forestall high incidence failures. Table 4.2 and Table 4.3 summarize the results of two separate studies from two different FABs, analyzing six months and two months of data, respectively.

Table 4.2 - Partial distribution list of failures during six month period

Number of Occurrences	Major-SubSystem Failed
93	Transfer Arm
21	Receive Elevator
17	Send Elevator
14	Electronic Chassis
257	Not a Machine Failure

(a)

# of occurrences	Failure Description
257	Not a Machine Failure
108	No problem found
36	Process problem
30	Down to incorrect category

(b)

Table 4.3 - Partial distribution list of failures during two month period

Name	# of Occurrence of failure
Total number of MNT repair	263
Not a Machine Failure	63
Machine Failure	201

(a)

# of Occurrences	Failure description
63	Not a machine failure
23	No problem found
12	Process problem
6	Down to incorrect category
11	Operator error
4	Software problem

(b)

From the above, a striking and very interesting finding is that a high percentage of equipment *sign downs* was not a result of actual equipment failure, rather it was due to "not a machine failure." Furthermore, a large percentage of the "not a machine failure" situations were attributed to *operator error*, *down to incorrect categories*, *no problem found* or *process problem.* For example, Table 4.3a illustrates that during two months of operation there were 264 occurrences of MNT Repair, out of which the number of "not a machine failure" accounted for 63. This corresponds to about

25% of the total number of machines signed down. Table 4.3b illustrates the 5 most common causes for 63 "not a machine failure". In the six months study, Table 4.2a indicates that while problems with *transfer arm* occurred 93 times, problems with *send elevator* occurred 17 times, problems with *receive elevator* occurred 21 times, problems with *not a machine failure* occurred 257 times. The partial distribution (the most 3 common) of "not a machine failure" is shown in Table 4.2b which indicates that out of 257 "not a machine failure" there are 108 occurrence of "no problem found", 36 occurrence of "process problem", and 30 occurrence of "down to incorrect category".

One of the most important issues dealing with symptoms is *who* should collect the symptom and *when*. This issue is of utmost importance due to its effects on the integrity of symptom collection. Since the operator is the one who operates the equipment and observes any indication of faulty performance, such as equipment malfunction, processing break down, or quality disturbance, the operator would be the most suitable candidate to report this event. The symptom collection should take place when the operator signs down the equipment, i.e., during *Down to MNT*. The collected symptoms can be very useful for the technician in processing the fault diagnosis. The technician can use the expert diagnostic system to display the most probable cause of any reported symptom.

However, while it seems most logical that the operator should carry out the task of symptom collection, there are concerns over operator judgment and inconsistency in

selecting the most appropriate symptom. It may be a difficult task for the operator to interpret Symptomic factors such as *logical* or *observational* in determining the true symptom, and the result could be questionable. For example, in those pieces of equipment where the operator cannot see the exact handling process, it would be difficult for him or her to pinpoint the appropriate symptom. Furthermore, the study shows that in many situations the reported symptom had nothing to do with the failure. This kind of error can be very harmful. A bad symptom not only corrupts the knowledge base, but it also provides the technician with misleading information in his search for the faulty component. In interviewing the domain experts, there is agreement that in most cases, such as in processing problems, the technician will have a lot to say about the true symptom and will be able to clarify the ambiguity that the operator had during the sign down of the equipment.

For the above reasons, *validation* of the operator-collected symptom is vital in achieving a credible knowledge base and maintaining its integrity. Validation in this context is different from the one done by the family manager using the "validate" utility of MMR. At this point, the technician judges the operator's selection and interpretation of the symptoms for validity, integrity, and relevance to the failure. Therefore, an *intermediate storage* is proposed so that symptoms can be saved, and repair activities collected, during each "Down to MNT" and "Log off" event, respectively. In this way, information can easily and quickly be retrieved to be displayed for the technician whenever a "Down to MNT" or "MNT repair" event is logged into

Workstream. Such information needs to be displayed whenever a technician logs a "MNT Repair" event to collect the failure. Validation can be done by simply answering a "yes" or "no" to the prompt regarding each previously collected symptom. In the case of an invalid symptom, the technician may collect a new symptom, and a "mailing" system needs to be designed to inform the family manager of the discrepancy between the operator's and the technician's views of the symptom. The family manager would then need to investigate this discrepancy, and educate both the operator and the technician concerning the outcome of the investigation. At times it may be necessary to add another option to the menu or set up policies that both types of personnel can follow. It is important that both the operator and the technician have the same understanding of each item in the menu and its meaning. Whenever a validation takes place, the result should be reflected coherently in the Menu, Workstream, and Ingress data bases.

To carry out the process of validation, an intermediate storage is necessary to save both symptoms and repair activities for quick retrieval during the "Down to MNT" and "MNT repair" events. In this way, the technician knows exactly what was reported as the original symptom and what kind of actions were previously taken to fix the problem. This information acts as a guide to each successive technician working on the problem, informs him or her what needs to be done when he or she logs the "Down to MNT" event, and also helps him or her identify the true failure when he (she) logs the "MNT Repair" event. For example, in the previously mentioned

scenario involving an o-ring leak, displaying the prior symptoms and repair activities simplifies the job for the second and third technicians: it becomes easier to see what remains to be done in order to complete the job, and it aids the identification of the true failure. Figure 5.1 illustrates the type of information that needs to be displayed during each event.

4.3.6 Symptom Prototype

The symptom prototype has been developed to experiment with different ways of specification, requirement definition, symptom collection, symptom/failure relation, the knowledge base and its application program, and finally, in order to obtain an opportunity to examine the trouble shooting guide in advance. The prototype can be used to specify the necessary interface connections along with other necessary requirements of the trouble shooting guide.

The prototype extracts comments from the data base and translates comments into a real symptom description by using the symptom and failure menus. The prototype then relates these symptom descriptions to their corresponding failure descriptions as it builds a knowledge base. As a result, a knowledge base for (at this point only single) symptoms and failures is developed, and a methodology to extend this knowledge base for multiple symptoms is discussed.

In the final stage of this prototype, an application program is designed to assist the maintenance personnel in trouble shooting equipment problems. The application program displays all the failures related to a given symptom in descending order of occurrence frequencies. It is also possible to display all symptoms related to any given failure. In this way, the technician can start with the reported symptom, display failures that may cause that symptom, then pick the most likely failure, display all symptoms that usually accompany this failure, and check whether other (unreported) symptoms can also be observed. If this is not the case and if he or she believes that

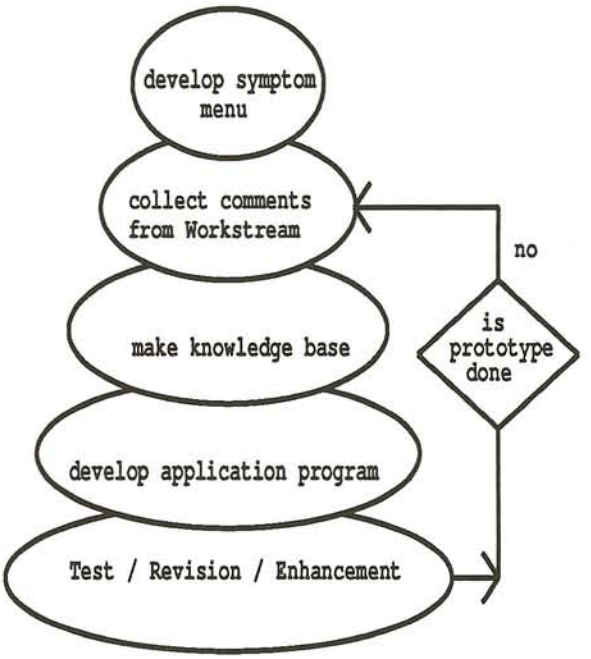


Figure 4.9: Development cycle for symptom prototype

these symptoms should be present, the most likely failure is rejected, and the list of symptoms for the next most likely failure in the list are displayed.

The prototype was developed based on two months of data available for the DSW family of FAB15. Figure 4.9 illustrates the development cycle for the symptom prototype. The next paragraph discusses issues involved in each phase of this development cycle.

During the first cycle, *develop symptom menu*, a strategy for developing a symptom menu is discussed, and symptom menus for the DSW family of FAB15 and FAB10 are constructed and presented. (Sections 4.3.3 and 4.3.4 present Symptomic factors and a Symptomic classification as a systematic way to classify symptoms) Figures 4.10 and Figure ?? illustrate the above two menus. The FAB10 menu relates

to equipment with a glass cover so that the operator can observe the wafer handling process. As a result, there is an option in Figure 4.10 that describes problem with wafer handling: *won't send wafer, won't receive wafer, won't prealign wafer, transfer arm/paddle problem.* But in FAB15, equipment without a glass cover is used and thus, the wafer handling process cannot be visually observed by the maintenance personnel. Therefore, these symptoms must be classified based on some numeric codes displayed on the equipment monitor. Each of these code describes a particular handling problems described above. Figure 4.11 classifies these symptoms under AWH ERROR as awh time out, code 24, and code 29.

The symptom menu is simply a hierarchical data structure that can assist the maintenance personnel in describing symptoms. The symptom description is a path that begins at the main menu level and that becomes progressively more specific as the maintenance personnel makes selections while proceeding along the various hierarchical levels of the menu.

In the second development cycle, *collect comments from Workstream*, this author has developed a program to extract comments from Workstream and to translate these comments into symptom descriptions. The symptom and failure menus have been used to assist this process. Figure 4.12 illustrates such a process. These symptoms will be validated and then will be related to their corresponding failures. This process is done by identifying and extracting the entire *episode.* An episode is the complete cycle of maintenance starting from "Down to MNT" and ending with "MNT

```
WAFER ALIGNMENT
          inconsistent alignment
          AWA failed
          AWA locked up
          compansator problem
          key will not lock on
          other
WAFER HANDLING
          won't send wafer
          won't receive wafer
          won't prealign wafer
          no vacuum on chuck
          transfer arm/paddle problem
          other
EXPOSURE PROBLEM
          gates and windows problem
          not exposing wafer
          over exposure
          double image
          lamp intensity incorrect
          other
FOCUS PROBLEM
          bad picture
          ccu problem
          no picture
          other
ENVIRONMENTAL PROBLEM
          TEMPERATURE
                    low
                    high
                    other
          other
CONTROL SYSTEM PROBLEM
          disk error
          computer lock up
          key board is not working
          other
STAGES
          stages problem
          stages time out
          other
CHUCK SPOTS
          chuck spots/dirty chuck
          other
```

Figure 4.10: Symptom Menu for Fab10

```
AWA PROBLEM
        awa does not exist
        camera setup required
        other
AWH ERRORS
        awh timeout
        code 24
        code 29
        other
CONTROL SYSTEM PROBLEM
        computer lock up
        key board won't work
        monitor/screen problem
        other
RMS PROBLEM
        align done interrupt not received
        bar code error
        detectors did not balance in time
        fork extend error
        fork retract error
        fork vacuum error
        inventory not working
        no reticle detected on fork
        no reticle detected on platen
        obstruction between elevator and turntable
        platen vacuum error
        reticle alignment is false
        reticle detected error
        reticle rotation error
        other
auto focus failure
awl problem
dfas problem
exposure problem
high particle counts
hot spots
ias timeout
lamp temperature out of tolerance
scratching wafers
send/receive elevator problem
other
```

Figure 4.11: Symptom Menu for Fab15

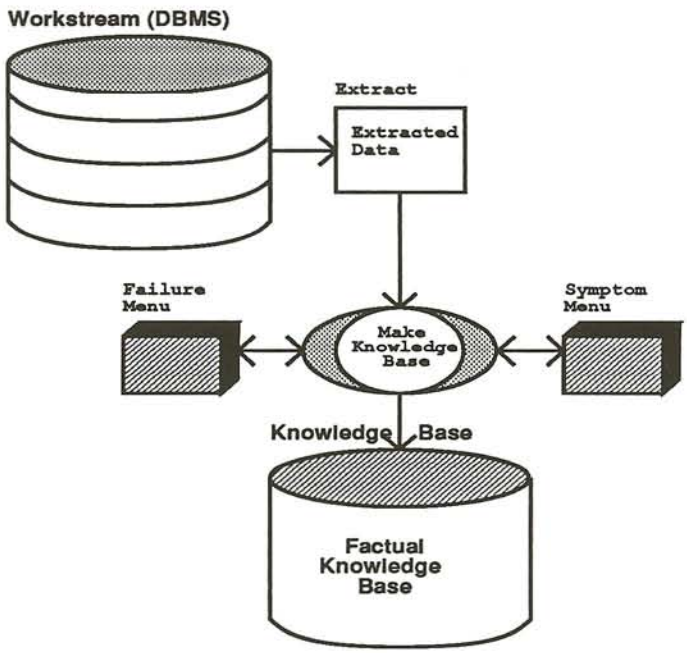


Figure 4.12: Process of building the knowledge base

Repair," which can include many "MNT Log on" and "MNT Log off" events. Comments were validated for correctness and relevance of comments to failures through interviews with domain experts and family managers. A knowledge base was constructed as the above process continued.

The third level of the development cycle is dedicated to issues concerning the knowledge base. A knowledge base was constructed for relating a single symptom to its failure. This knowledge base is known to be an *example-based knowledge base.* Each example, which is being referred to as *factual knowledge*, simply defines the relationship between a failure and its corresponding symptoms or vice versa. It is simply a record that holds related information such as symptom description, failure description, frequency of occurrence, and other necessary information. In Chapter 2, Section

2.5, the data, information, and knowledge were described in detail. It is important to notice that what is being built here is only a presentation of factual knowledge. To acquire procedural knowledge, such as rules, from the above knowledge base, one may develop programs to automatically create such rules. This represents major work toward the development of an expert diagnostic system, which falls outside the scope of this thesis. An example of a single data structure for knowledge used to build knowledge base is presented in Figure 2.4. This structure can be expanded as necessary to utilize more information. For example, data specifying a "time stamp" would be essential in augmenting the knowledge base with a *time model* of equipment performance. Also issues relating to multiple symptoms can be addressed with simple modifications of this data structure, provided the family managers agree to collect multiple symptoms using the following procedure: A *user interface* from the HDC side is necessary to let the user collect symptoms as often as needed. Multiple symptoms can then be related to their corresponding failure in one package.

During the final prototype development cycle, an application program was designed to complete the Trouble Shooting Guide. Two separate programs were developed that work off the knowledge base and display the list of failures for any given symptom and the list of symptoms for any given failure, respectively. Figure 4.13 and Figure 4.14 illustrate the results of the these two programs. In these figures, symptoms or failures are arranged according to their frequency of occurrence. These application programs are in their most simple forms and apply the most fundamental

Symptom: Send/Receive elevator problem

Freq.	Failure Description
(9)	AWH; Bad send puck cable
(8)	AWH; TRANSFER ARM; T/a falg out of adjustment
(4)	AWH; SEND ELEVATOR; Wafer present sensor out of adjustment

Symptom: AWH ERROR; Awh timeout

Freq	Failure Description
(7)	AWH; RECEIVE ELEVATOR; Boat guides out of adjustment
(5)	AWH; TRANSFER ARM; Paddle out of alignment
(3)	Not a Machine Problem; Facility Problem

Figure 4.13: Result of application program: symptom vs. failure

principles. As the knowledge base is augmented using other factors such as time stamps, this program can be expanded to utilize all the properties and resources available.

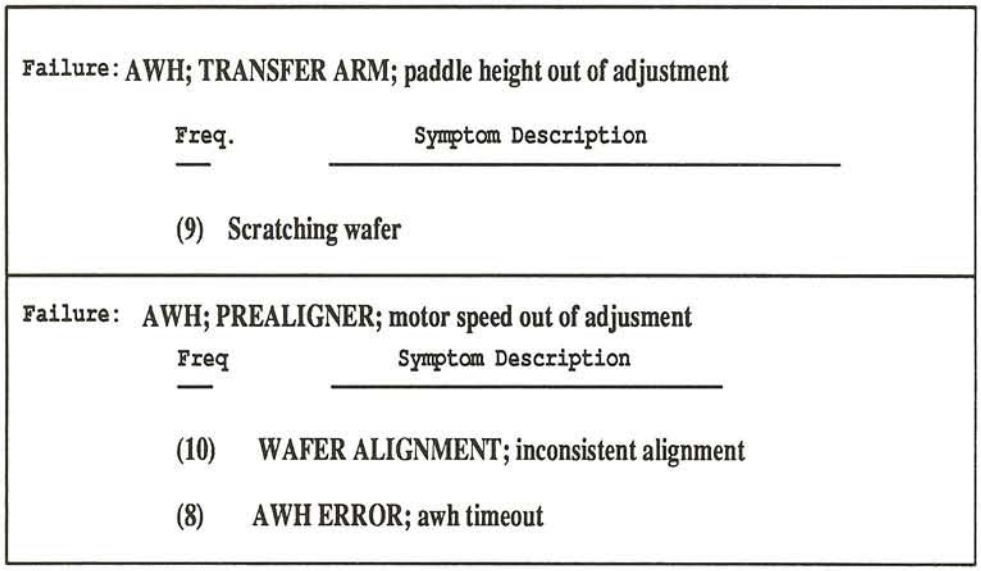


Figure 4.14: Result of application program: failure vs. symptom

This prototype was tested for equipment used in the DSW family of Fab15. Each testing required further revision in the prototype and as a result of this process, a methodology to collect symptoms and specifications to build a knowledge base were characterized. Questions were addressed such as: What is a symptom? when, is it to be collected, how, and by whom? A methodology for classifying symptoms was developed as well as software to develop the symptom menu from a text file. It was recognized that symptoms must be validated through the use of "Short Term Memory," and a methodology for doing this was designed. HDC was enhanced for collecting symptoms, connection interfaces between Ingress, Workstream, and MMR were specified, the need for a "browser" to browse through the menus in diagnostic mode was determined, and also other considerations are the result of the above study and prototype. This prototype concludes with the specification for collecting

symptoms and also introduces two different ways to build a knowledge base to trouble shoot equipment problems.

Methodology and specification to collect symptoms

1. The symptom menu needs to be structured in a hierarchical fashion similar to the data structure for the failure menu, based on recognized principles such as Symptomic factors and a Symptomic classification. A conversion program already exists to translate the symptom menu text file into a structured file that HDC can use. Two different menus for the DSW families of FAB10 and FAB15 have already been developed.

2. A user interface that enables maintenance personnel to activate HDC as often as they need.

3. The symptom menu needs to have an "Other" option in the second and subsequent menu levels. "Other" will be allowed as a conditionally valid description in the menu. Selection of "other" automatically results in an electronic mail message being sent to the supervisor and/or equipment engineer, who will then be required to follow up either by validating the new description, or by reassigning it to an existing description. If an "other" symptom is validated as belonging to a new category, the menu is automatically augmented to include the new symptom type.

4. Symptom descriptions need to be collected as a required part of logging transaction during the "Down to MNT" event by an operator.

5. Collected symptoms need to be displayed during each "Down to MNT" and "MNT Repair" events.

6. Symptoms need to be *validated* for their integrity, correctness, and relation to the failure during "MNT Repair". (This validation is different from the one performed by family manager through the use of "validate utility.") If a collected symptom is irrelevant to an identified failure, a new symptom needs to be collected. A mail message must inform the family manager of the discrepancy for further investigation. A consequent implementation of this procedure promises to minimize the gap between the operator's and the technician's interpretation of observed symptoms and their selected paths in the symptom menu.

7. In case of an invalid symptom, the symptom menu needs to be updated. A newly validated symptom must be reported to Workstream and Ingress.

8. The use of Short Term Memory to temporarily store collected symptoms is necessary for preventing a contamination of the permanent data bases with non-validated information, but is also important for easy and quick retrieval of information for display during the "Down to MNT" and "MNT Repair" events.

Two approaches to build a knowledge base

It is recognized that the best way to build the knowledge base for the Trouble Shooting Guide is to use the validated data available in Ingress. This data has been validated (a validation interface already exists between MMR and Ingress), it is well tabulated, and different pieces of information can easily be related to each together for building the knowledge base. Another alternative is to adopt the way the prototype was constructed, and acquire the the knowledge directly from Workstream. This approach requires a lot of redundant work to duplicate features already contained in/for Ingress: such as the connection interface between MMR and Ingress, the validation procedures available for Ingress implementing the validation done by the family manager, the existence of maintenance episodes and tabulated data in Ingress (Workstream data is poorly organized), a need for extra linkage of data collection from Workstream, etc. The following graphical representation illustrates the above assertions.

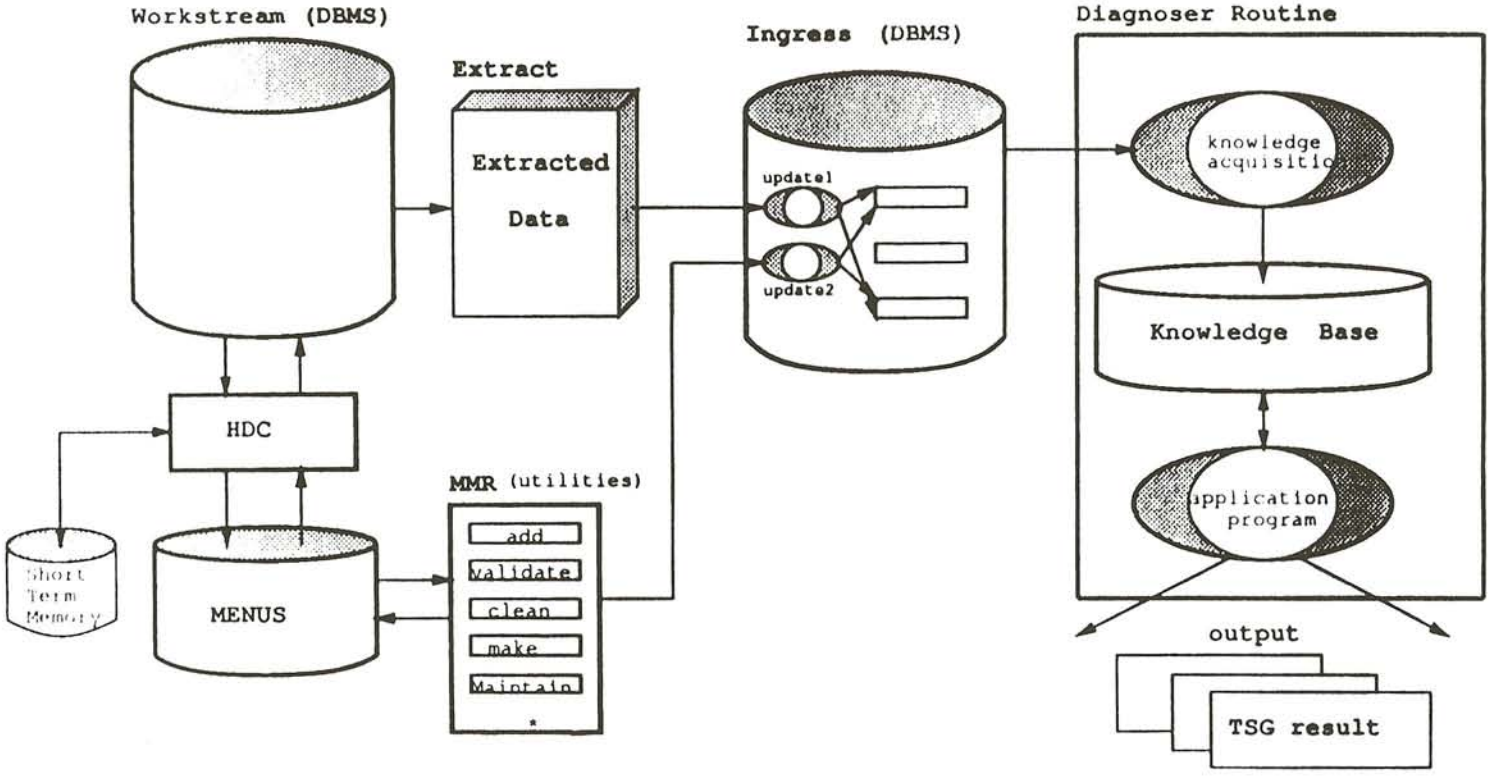


Figure 4.15: Trouble Shooting Guide using Ingress data

4.4 Repair Activity

4.4.1 Introduction

This section investigates issues relating to repair activities. *Activity comments* have been extracted from the Workstream data base and are studied for their characteristics and relevance to the failures. A methodology to analyze repair activities is introduced. A repair activity menu similar to the one developed for the symptom prototype has been developed. Finally, specifications for collection of repair activities and the development of a knowledge base are presented.

4.4.2 Repair Activity

Repair activities are defined as actions performed by maintenance personnel to bring a signed down machine back to a state in which it can perform its intended function. In Section 4.2.2, the author introduced the *failure mode* as a way to describe how a component has failed. The repair activity, here, is defined as the procedures taken to correct the failure component. These activities can be relevant or irrelevant to the failure mode specified in the collected failure description. A validation process needs to be designed to allow maintenance personnel to validate repair activities after completion of the diagnosis. To understand the relationship between failures and repair activities, as defined above, the author has developed a program to extract the repair activity comments from the history file, and has related them to their corresponding failure descriptions. These results were studied in order to gain a clear

definition of repair activities and also to find a way for developing a repair activity menu. Table 4.4 shows activity comments related to corresponding failures.

Table 4.4 - Failure and its related repair activity

ID	Failure Description	Log off Comment	MNT repair Comment
95	DSW; RMS; PLATEN; platen dirty	* Cleaned VAC lines	* Replaced platen vacuum sensor * Cleaned platen
			* Cleaned platen
			* Cleaned platen
			* Cleaned platen
80	DSW; COLUMN/FOCUS; ACS; acs out of adjustment	* Calibrated ACS	* Reset ACS
		* Replaced leveling value	* Set up ACS unit
			* Reset ACS
64	DSW; AWA/CCUT; CCU MODULE; computer setup defective	* Replaced Lamp	* Replaced power supply to monitor * Replaced cable to monitor
181	DSW; AWA/CCUTV; CCU MODULE; ccu setup out of adjustment		* Adjust target voltage and focus
			* Adjust illuminator
			* Adjusted AWA system and CCU setup
			* Setup CCU
			* Adjusted CCU and gate angels
			* setup CCU

An important issue concerning a repair activity is that, during the task of diagnosis, the maintenance personnel may have difficulties identifying the failure. As the technician trouble-shoots a repair, he may become involved with activities unrelated to the original problem. He may notice that some wiring is loose or some other parts were out of adjustment, and may decide to take action to repair these simultaneously. However, these additional corrections are unrelated to the failure that caused the machine to go down. Regardless of their relevance to the reported failure (symptom), all these repair activities need to be collected at the "Log off" event. A Short Term Memory as proposed in Section 4.3 is a must to store these collected data for quick retrieval in order to inform new personnel of prior activities (since individual repair activities may stretch over several days and may be performed by different technicians) and also to assist the technician(s) in validating the collected data. The collected data needs to be validated during the "MNT Repair" after the failure component has been identified and all the repair activities have been completed. Validation can be done by the technician by answering "yes" or "no" to the prompt regarding each collected repair activity. The technician would be the best-qualified person to judge the relevance of a collected activity to a reported failure since he or she is the domain expert who knows the machine and its functions best. For example, in the case of the "o-ring leak" failure, the mirror had to be removed which then called for subsequent readjustment. However in another situation, the

mirror will be out of place due to vibration and also needs readjustment. Here, the first mirror adjustment may not be collected while the second would.

4.4.3 Repair Activity Menu

This section discusses the development of a repair activity menu. The repair activity menu is designed as a hierarchical data structure based on the hardware decomposition of the equipment, just as in the case of the failure menu. A set of "repair activity verbs" needs to be attached to the menu to express the way the component has been repaired. A typical *Repair Activity Verb* list is prepared and shown in Table 4.5.

Table 4.5 - List of Repair Activity Verb

Verb-ID	Verb	Verb-ID	Verb
1	adjusted/setup	11	powered down
2	aligned	12	powered up
3	calibrated	13	raised
4	cleaned	14	rebooted
5	corrected	15	rebuild
6	decreased	16	reinstalled/installed
7	enlarged	17	repaired
8	increased	18	replaced
9	leveled	19	reseated
10	lubricated	20	tightened

In developing the repair activity menu, the author notes that adding up to 20 repair activity verbs to each existing component in the menu is not practical. Some menus have already grown to 900 hundred failure descriptions in six months of operation, i.e.

to 900 failure components. Adding another level of 20 options to each component is definitely not a good idea. Therefore, a better approach uses a menu for repair activity which is a hierarchical data structure based on the hardware decomposition of the equipment starting with the Major-SubSystems, followed by the Minor-SubSystems, and finally the Failure Components. However, repair activity verbs also need to be incorporated along with the selected path from this menu to form the *repair activity description.* Creating such a menu, as with the failure menu, would be time consuming, especially for the existing FABs where a failure menu has been used for a long period of time. Therefore, a special software was developed to create such a menu from the existing Failure Menu. Table 4.6 shows a menu for the DSW family in FAB15.

Table 4.6 - Repair Activity Menu

```
STAGES/CHUCK
        CHUCK/THETA
                  chuck
                  theta drive
                  theta home sensor
                  wafer present sensor
                  other
        STAGE MECHANICAL
                  barry table air hose
                  barry table
                  x motor
                  x motor gearbox
                  other
AWH     TRANSFER ARM
                  paddle
                  transfer arm
                  backlash or warm gear
                  motor speed
                  other
        PREALIGNER
                  prealign theta
                  spindle center
                  flatfinder drive motor
                  flatfinder sensor
                  motor speed
                  spindle height
                  wafer present sensor
                  other
        RECEIVE ELEVATOR
                  boat guides
                  cable
                  comb sensor
                  drive motor
                  index flag
                  motor speed
                  wafer present sensor
                  other
        SEND ELEVATOR
                  wafer preset sensor
                  boat guides
                  cable
                  comb sensor
                  drive motor
                  index falg
                  motor speed
                  wafer boat
                  other
```

Table 4.6 - Repair Activity Menu (continued)

```
RMS
        PLATEN
                platen
                platen vacuum solenoid
                align motor
                align sensor
                tower
                other
        ELEVATOR
                bar code reader
                elevator motor
                lead screw
                travel sensor
                bar code reader cable
                elevator motor cable
                other

COLUMN/FOCUS
        ACS
                acs
                monitor
                chassis
                sensor
                other

AWA/CCUTV
        CCU MODULE
                illuminator lamp
                power cable
                camera control module
                camera
                ccu setup
                power supply
                vertical module
                vertical weep module
                other
```

It is proposed that when HDC selects the path through the repair activity menu, thereafter the screen containing the list of repair activity verbs needs to be displayed in order to further select the proper verb. Thus, the repair activity description would be the selected path from the menu in addition to the selected verb. The repair activity description will have a code number equivalent to:

$$\text{Repair-Activity-Description-ID} = (\text{Activity-ID} - 1) * 20 + \text{verb-ID} \quad (1)$$

It is proposed that the interface between Menu and activity-list be done from the HDC side.

4.4.4 Repair Activity Prototype

The repair activity prototype is developed in a similar way to the one for the symptom prototype. The development cycle is similar to the one previously explained (Figure 4.9). It extracts repair comments from the Workstream data base and translates them into the repair activity description. Repair comments will first be validated through domain experts for integrity, correctness, and relevance to the failure. These comments will be related to their corresponding failure description to build a knowledge base. Since the interface between the repair activity menu and the repair activity verb list was not available, most of this process was done manually. The development cycle for the repair activity prototype can be described as follows:

During the first cycle, *develop repair activity menu*, software was developed to create a repair activity menu from the existing failure menu. Also a list of *repair verbs* was prepared along with a strategy to generate a path for repair activity descriptions. The list of repair activity verbs, the repair activity menu, and the strategy are presented in Table 4.5, Table 4.6, and equation (1), respectively. During the second and third cycle, a program was developed to extract repair activity comments from the Workstream data base and relate them to their corresponding failure. These comments were validated manually through domain experts and a knowledge base was constructed. The structure of this knowledge base is similar to the one developed for the single symptom/failure knowledge base.

During the final prototype development cycle, an application program (similar to the one for symptom/failure) was developed to carry out the final task of the trouble shooting guide. This program would help the technician in the FAB to display the list of repair activities for a selected failure description. This program is in its most simple form and applies the most fundamental principles within the trouble shooting guide. Figure 4.16 illustrates the result of such a program. A *browser* is further necessary to complete the application program for production. The browser lets the user browse through the repair activity menu to select options and finally displays the results developed. The browser is simply a front/end truncated HDC.

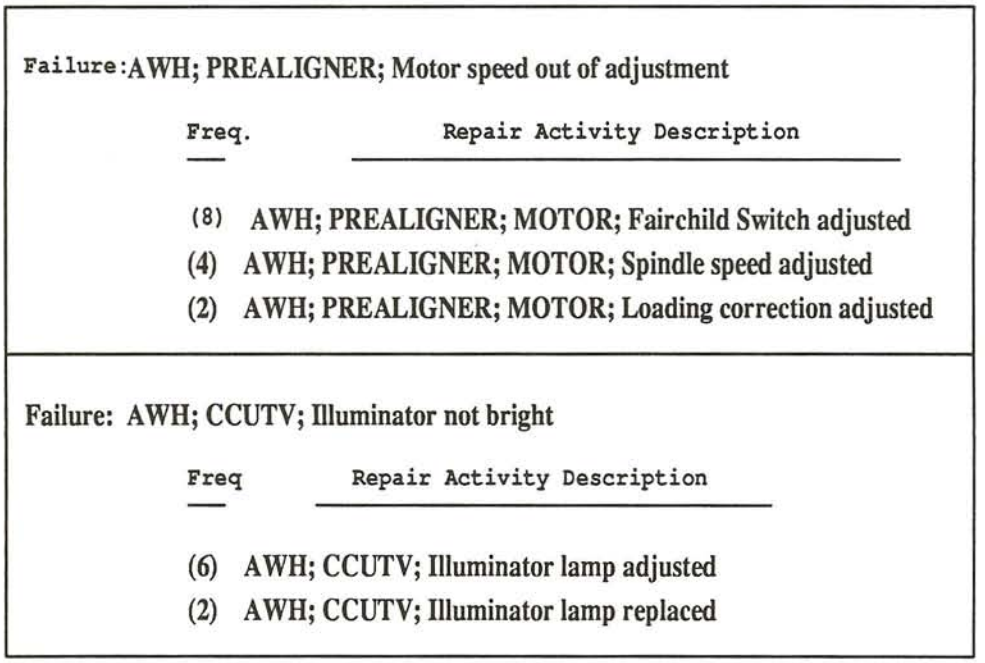


Figure 4.16: Result of the application program for repair activity prototype

Methodology and specification to collect repair activities

1. The repair activity menu needs to be developed as a hierarchical data structure based on the hardware decomposition of the equipment. Its data structure is similar to the failure and symptom menus. A text file can be generated from the failure menu using the software developed in Section 4.4.3, and finally this text file can be further converted into a structured file that HDC can use.

2. A special interface needs to be created to display the repair activity verbs after each selection made from the menu, and also to calculate the ID number for the repair activity description.

3. A user interface needs to be developed to let the maintenance personnel collect multiple repair activities during either "Log off" or "MNT repair" events. This simply lets the maintenance personnel call HDC as often as necessary.

4. Repair activities need to be collected during each "Log off" event (if such exists) and "MNT repair" event after the technician has completed the task of diagnosis and has returned the machine to a state in which it can perform its intended function.

5. The collected repair activities during "Log off" need to be displayed at the "Down to MNT" event. This will provide the technician with information as to what has been achieved so far and possibly what needs to be done in addition to complete the work.

6. Short Term Memory, suggested in Section 4.3, is necessary to temporarily store the collected repair activities for quick retrieval. This will assist the process of validation.

7. Repair activities collected during the "Log off" event need to be validated by the last technician during the "MNT repair". This can be done by answering "yes" or "no" to each prompt regarding collected repair activities. In case of an invalid repair activity, the menu needs to be updated accordingly. Validated repair activities need to be reported to Workstream.

4.5 Diagnoser (Application Program)

This section discusses the proposed *application program* required to carry out the task of implementing the Trouble Shooting Guide. It consists of actually adding together the three application programs developed in Sections 4.2 and 4.3 along with a *browser* that enables maintenance personnel to browse through each menu to examine different selections for possible answers. The browser can be implemented as a front/end truncated HDC. In order to activate the Trouble Shooting Guide, the maintenance personnel needs to log a "diagnostic" event into Workstream. This requires adding "diagnostic" events to the Workstream data structures and properly checking them in HDC. The Trouble Shouting Guide consists of three functional modes: (i) For a given *symptom*, it displays a list of possibly related failures according to their frequency of occurrence, (ii) for a given *failure*, it displays a list of potentially related symptoms according to their frequency of occurrence, and (iii) for a given *failure*, it displays a list of repair activities that may possibly fix the failure according to their frequency of occurrence. Whenever a mode is selected, the browser will display a proper menu from which a selection can be made. This selection will be routed to a knowledge base for pattern matching, and a proper list of findings will be displayed. The proposed interface is shown in Figure 4.17.

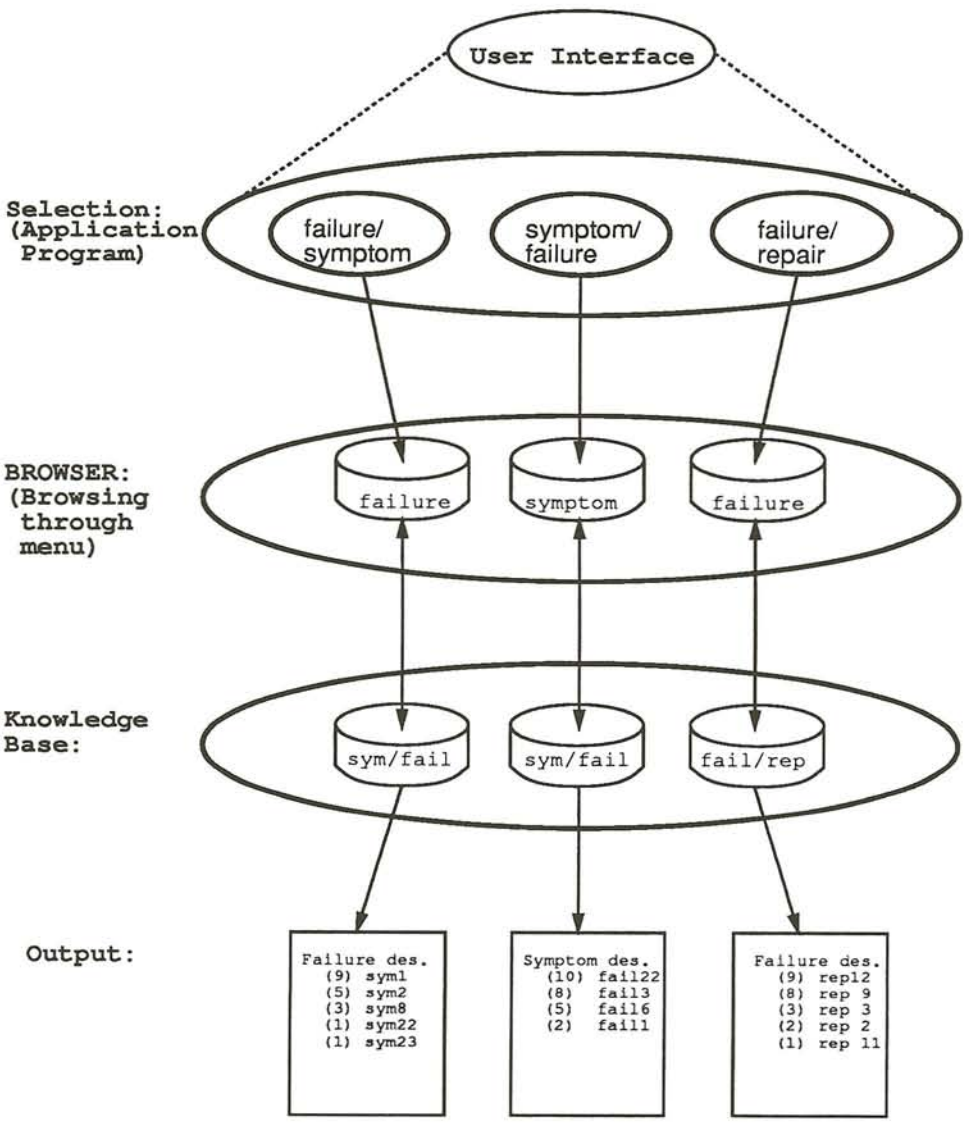


Figure 4.17: Diagnoser shell

CHAPTER 5

Results

A manufacturing environment was introduced and issues relating to collecting and measuring knowledge about performance of the machine parked were discussed. It was also discussed how "equipment time" and "equipment activity" need to be categorized in order to measure certain aspects of equipment performance. Figure 5.1 illustrate the event activities designed under which an operator or technician is able to collect and validate data relating to equipment performance. These figures summarize five different events along with their activities: Down to MNT, MNT Logon, Diagnostic, MNT Logoff, MNT Repair.

A maintenance personnel, operator, may sign the equipment down during the "Down to MNT" event as a result of equipment malfunction, instability in the processing, or quality disturbances. During this event, the operator is to collect symptoms using HDC and the symptom menu. He may also call the diagnostic system if he feels the failure is minor and can be fixed easily using Trouble Shooting Guide. Using this procedure, there would be no need to call upon the technician for assistance. The main purpose of the "MNT Logon" is to report the exact time maintenance personnel start to work on the equipment. This distinguishes between the time when

the equipment was in a waiting status and the time a maintenance person actually worked on the equipment. During this event, symptoms collected by the operator and the repair activities done by prior technicians can be displayed. This information would be essential to a new technician as what has been done to the equipment and/or what else needed to be done to complete the work. During the "Diagnostic" event, maintenance personnel may call the Trouble Shooting Guide to trouble shoot the problem. The episode information (symptoms, and prior repair) need to be displayed to facilitate the trouble shooting.

MNT Logoff is designed so that a technician can report his repair activities before he leaves his maintenance activity. This event will be logged into the system only when the task of diagnosis is not finished and equipment is not yet back to the state in which it can do its intended functions. This event is typically used when a technician takes a lunch break or leaves his post for other reasons. Display of episode information would be informative rather necessary.

When equipment is completely fixed and back to the state in which it can do its intended function, the technician would log the "MNT repair" event into the system to accomplish several functions. Collected data are validated by answering "Yes" or "No" questions. If an error identified, he may collect symptom on the spot. He also collects the final repair activities and validates the collected repairs(if any) by simply answering "Yes" or "No" questions. Finally, he may collect the failure that he thinks caused the machine to be signed down. Display of the episode information is

necessary for validation purposes and also to assist the technician in making a right judgment in selecting the real cause of the failure.

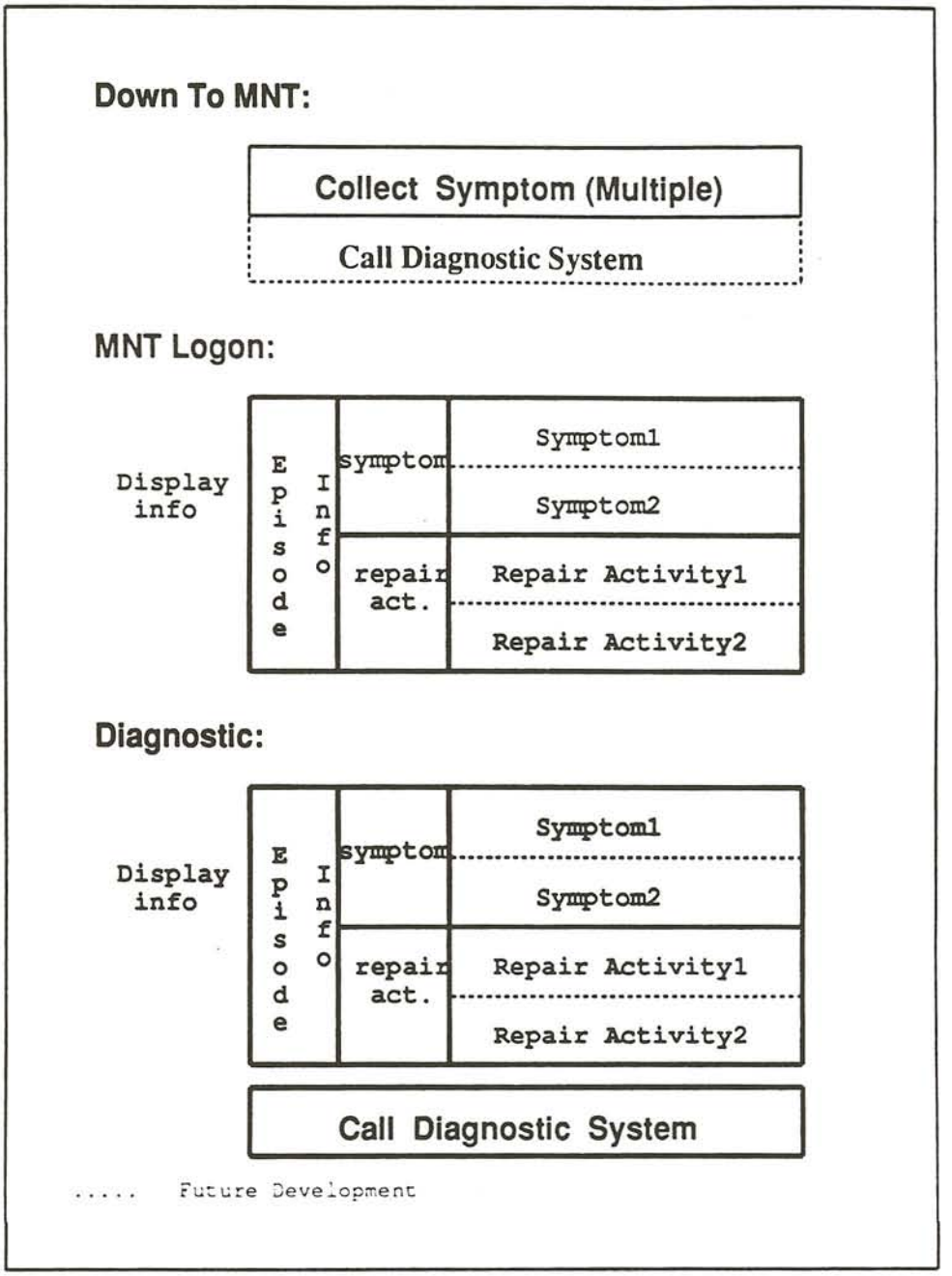


Figure 5.1: System's event activity protocols

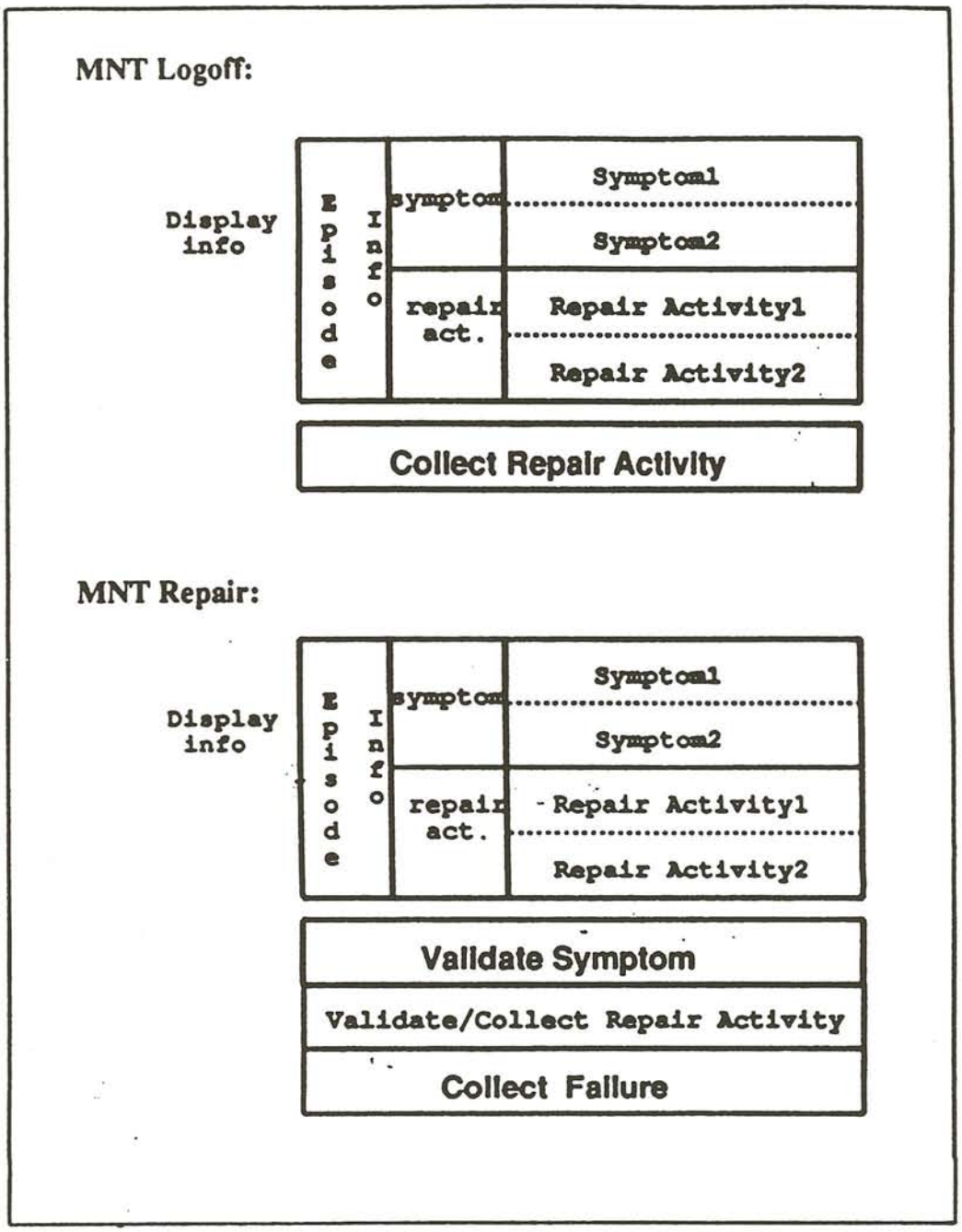


Figure 5.1: System's event activity protocols (continued)

Quick prototyping was used to establish the system architecture and its detailed requirements. In these prototypes, issues concerning symptom and repair activity were studied. What symptom is and what constitutes repair activity were defined and presented. Methodology and specification for collection of symptom and repair activity were developed and presented. Sections 4.3.6 and 4.4.4 present these methodologies respectively. Symptom and repair activity prototypes were developed to determine appropriate knowledge representation, man-machine interface, supporting hardware, and functional architecture of the system. A system architecture is presented in Figure 4.17. The results of both prototypes are presented in Figures 4.13, 4.14, and 4.16.

As prototypes were completed, HDC needed to be enhanced to carry out the task of collecting symptom and repair activity. Figure 5.2 illustrates the need for a *shell* around HDC. Inside this shell, a *decision box* receives the proper "event" and "event-type" from Workstream and decides which action: such as collection of symptom, failure, repair, or diagnosis (use of Trouble Shooting Guide) need to be taken. A user interface is also needed for multiple collection of symptom or repair activity. The shell also illustrates in which fashion collection and validation of data take place. For example, when "event" is "Down to MNT" and "event-type" is "symptom", then HDC will invoke symptom menu.

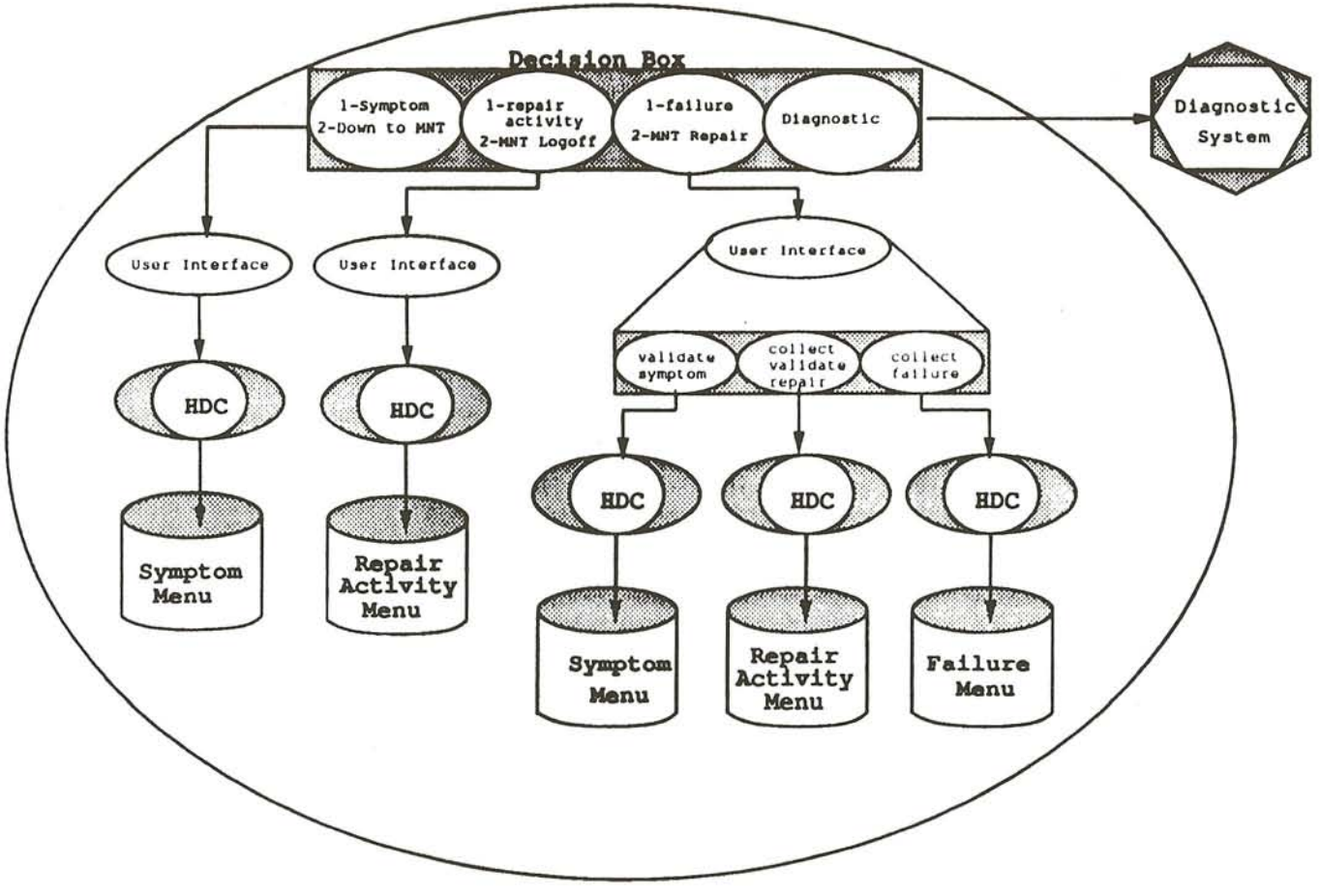


Figure 5.2: HDC Shell

Trouble Shooting Guide consists of knowledge acquisition routines, Example-Based(case-based) knowledge base, and an application program. Knowledge acquisition will extract well-formed, tabulated, and validated knowledge from Ingress and update the knowledge bases. There exist two kinds of knowledge bases: symptom-failure and failure-repair. The application program simply employs a logical but non-intelligent interface method on the knowledge base. Trouble Shooting Guide gives all possible failures to a set of symptoms or vise versa. It also suggests all sets of repair activities which should be taken to fix a failure.

One of the most important issue concerning the Trouble Shooting Guide is the speed of TSG and how soon its knowledge base needs to be updated with new information. It was strongly suggested that the process of updating the knowledge base should be done through a program which is to be triggered whenever Ingress receives new data from Workstream and is validated by MMR. This would require that another program be triggered whenever an episode is completed at Short Term Memory. Program simply transfers the episode to Workstream and forwards it further to Ingress. This way we have separated the process of collecting and validating information from the process of updating knowledge base. This avoids the problem of permitting maintenance personnel to have access to knowledge base to collect and update data which would be highly risky and could jeopardize the integrity of the knowledge base.

Finally, this paper discussed the pyramid of data in this system. It explained how data, information and knowledge were formed and shaped. Figure 5.3 illustrates such a progress systematically.

Conclusion

The designed Trouble Shooting Guide is the first step toward the development of Expert Diagnostic System. It assists operators and technicians in problem isolation and repair activities for equipment used in Integrated Circuit Manufacturing. It isolates the failure and suggests appropriate repair activities. TSG will suggest all possible failures for a set of symptoms according to frequency of occurrence. It also provides a display listing of repair activities which one need to fix the failure. TSG would be useful tool in the training of new technicians. It can be used by an operator to detect failure in case of a minor problem, and to suggest the proper repair activities. This shortens the diagnosis time by not requiring the call of a technician. It assists technicians to diagnose equipment failure more accurately and quickly. TSG acquires knowledge from domain experts (operator, technician, family managers) through a series of automated interviews; as a result it solves the bottle neck of building an Expert System. It also partially automates the process of validating and updating its own knowledge. This system is applicable to all kinds of equipment available within a company.

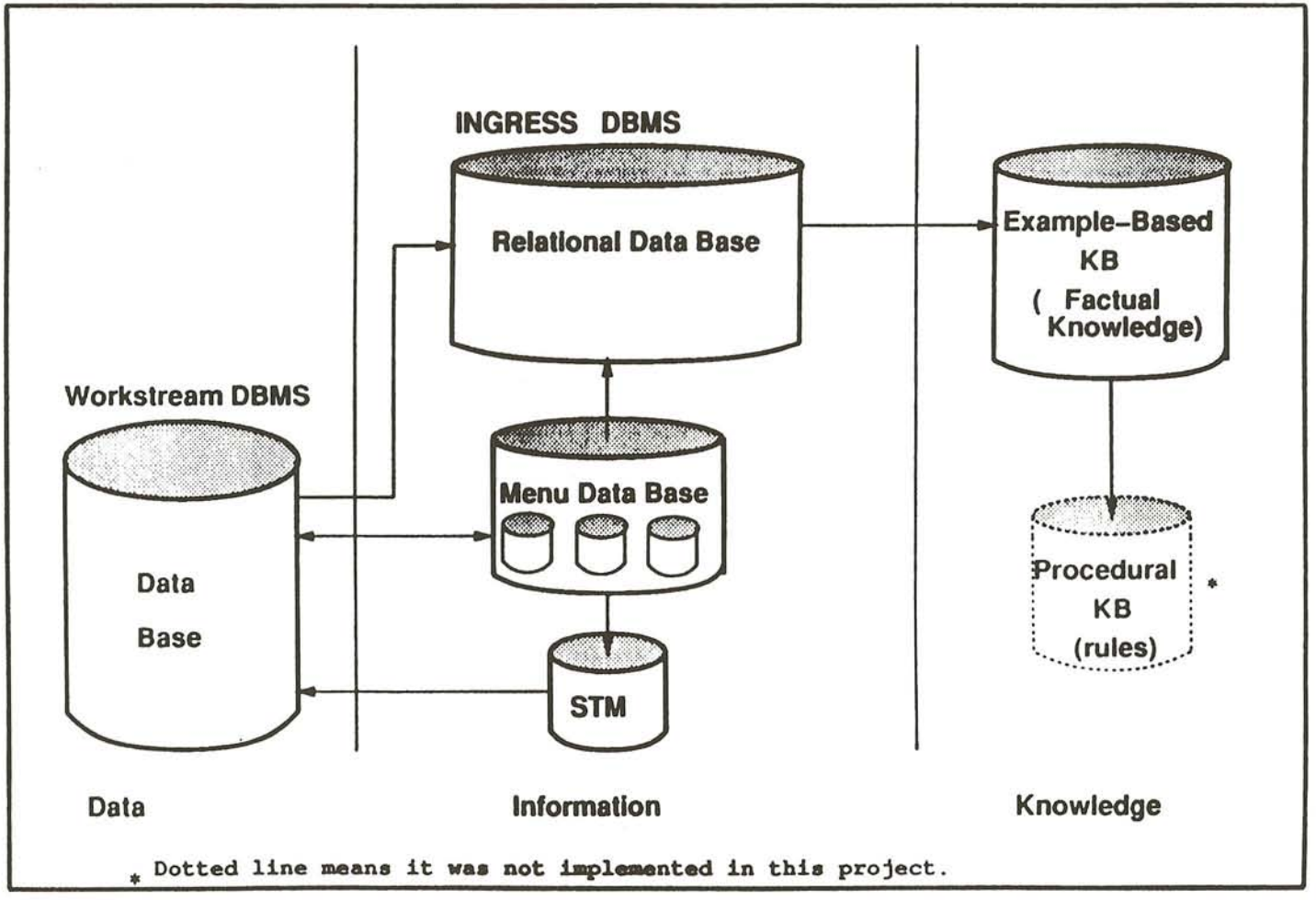


Figure 5.3: System Representation of data, information, and knowledge

REFERENCES

[1] Blaxton, T., R. Geesey, and L. Reeker. "A knowledge acquisition tools for process applications." In proceedings of the 5th intelligence AI symposium, (1987), Washington, DC.

[2] Boose, J., "User of repertory grid-centered knowledge acquisition tools for knowledge-based system." Proceeding of the 2nd AAAI knowledge acquisition for knowledge-based systems workshop, Banff, Canada, October 1987.

[3] Boose, J., and Bradshaw. "Expertise transfer and complex problems: Using Aquinas as a knowledge acquisition workbench for expert systems." Special issue in the 1st AAAI knowledge acquisition for knowledge-based systems workshop, 1986, part 1, International Journal of Man-Machine studies, 26:1, 1987a.

[4] Buchanan, B. "Artificial Intelligence: Toward machines that think". Encyclopedia Britannica Yearbook of Science and the Future, 1985.

[5] Buchanan, B., D. Barstow, R. Bechtal, J. Bennett, W. Clancey, E. Kulikowski, T. Mitchell, and D. Waterman. "Constructing an Expert System." In F. Hayes-Roth, D. Watermann, and D. Lenat, eds. Building Expert Systems, Reading, MA: Addison-Wesley, 1983

[6] Diedrich, J., I. Ruhmann, and M. May. "KRITON: A knowledge acquisition tool for expert systems." Special issue on the 1st AAAI knowledge Acquisition for knowledge-based systems workshop, 1986, part1, International Journal of Man-Machine studies, 26:1 (1987)

[7] Friedland, P. "Acquisition of procedural knowledge from domain experts". Proceedings of the International Joint Conference on artificial Intelligence, pp. 856-861, 1981

[8] Gains, B., and M. Shaw. "Interactive elicitation of knowledge from experts," Future Computing Systems, 1:2 (1986)

[9] Gary-Janardan and G. Salvendy. "A conceptual framework for knowledge elicitation." Special issue on the 1st AAAI knowledge acquisition for knowledge-based systems workshop, 1986, part4, International Journal of Man-Machine Studies, 27:1 (1987)

[10] Hart , A., Knowledge Acquisition for Expert System, Kogan Page, London, 1986.

[11] Hayes-roth, F., D. Waterman, and D. Lenat. Building Expert Systems. Reading, MA: Addison-Wesley, 1983

[12] Kahn, G. S. "From applicationshell to knowledge acquisition system," Proceeding of the tenth International Joint Conference on Artificial Intelligence (IJCAI'87), vol.1, 1987, pp. 355-358

[13] Kahan, G., S. Nowlan, and Mcdermott., "An intelligent knowledge acquisition tool." Proceedings of AAAI'85, 1985, pp. 581-584

[14] Karel, G., and Kenner M., 3M Company. Intellinews, Winter/Spring 1989, Volume 5, No. 1

[15] Kidd, A.L., Knowledge Acquisition for Expert Systems: A partial handbook, Plenum Press. New York. 1987

[16] Klinker, G., J. Bentolila, S. Genetet, M. Grimes, and J. McDermott. "Knack-Report-driven knowledge acquisition." Proceedings of the 1st AAAI knowledge acquisition for knowledge-based system workshop, International Journal of Man-Machine Studies, 1987a

[17] Klinker, G., C. Boyd, S. Genetet, and J. Medermott. "KNACK for knowledge acquisition." Proceedings of AAAI'87, 1987b, pp. 488-493

[18] Mcgraw, K., K. Harrison-briggs. "Knowledge acquisition: Principles and Guidelines," Prentice Hall, New Jersey, 1989

[19] Mcgraw, K., and Seal. "Structured knowledge acquisition for combat aviation." In Proceedings of NAECON'87, vol. 4, (may 1987a), pp. 1340-1348, Dayton, Ohio

[20] Mcgraw, K., and M. Seal. "Multiple expert knowledge acquisition methodology: MEKAM." Proceedings of the Third Australian Conference on Applications for

Expert Systems, The New South Wales Institute of Technology, Sydney, May 1987b, pp. 165-197

[21] Michaleski, R., "Knowledge acquisition through conceptual clustering. A theoretical framework and algorithm for partitioning data into conjunctive concept." In International Journal of policy Analysis and Information Systems, 4:3 (1980), pp. 219-243

[22] Quinlan, J.R., "Learning efficient classification procedures and their applications to chess end-games." In Michalski, R., G. Carbonell, and T. Mitchell, eds. Machine Learning: An Artificial Intelligence Approach. Palo Alto, CA, 1982

[23] Rasmus, D.W., "Expert Input." MacUser (January 1988), 136-150

[24] Ronira, C., "Acquiring knowledge: Finding the Tool is the Trick." AI Expert; July 1990.

[25] Rozenblit, J.W., and HU, J., "Knowledge Acquisition Based on Explicit Representation." *Expert System with Applications*, Vol. 3, pp. 303-315, 1991.

[26] Rozenblit, J.W., and Zeigler, B.P., "Design and Modeling Concepts." *International Encyclopedia of Robatics Application and Automation.* New York: John Wiley and Sons.

[27] Ruth, C.S., "Developing Expert System using 1st-class." Mitchell Publishing, 1988

[28] Sell, P.S., Expert Systems: A practical introduction, Macmillan, Basingstoke, 1985

[29] Slater, P.E., Building Expert System: Cognitive Emulation, Ellis Harwood, New York, 1987

[30] Tuthill, G.S., Knowledge Engineering, concepts and practices for knowledge base systems. TAB Books, 1990

[31] Welbank, M., Martlesham Consulting Services, A review of knowledge acquisition techniques.

[32] Zeigler, B.P., Multifacetted Modeling and Discrete Event Simulation, Academic Press.

[33] Zeigler, Bernard. P., Object-Oriented Simulation with Hierarchical, Modular Models. Academic Press. 1990