Prototyping as a Tool in the Specification of User Requirements

Hassan Gomaa
General Electric Co.
Syracuse, New York

Douglas B.H. Scott
I.P. Sharp Associates Ltd.
Toronto, Ontario, Canada

Abstract

One of the major problems in developing new computer applications is specifying the user's requirements such that the Requirements Specification is correct, complete and unambiguous. Although prototyping is often considered too expensive, correcting ambiguities and misunderstandings at the specification stage is significantly cheaper than correcting a system after it has gone into production. This paper describes how a prototype was used to help specify the requirements of a computer system to manage and control a semiconductor processing facility. The cost of developing and running the prototype was less than 10% of the total software development cost.

1.0 Introduction

In the past few years, the importance of Requirements Specification in the software life cycle has become widely acknowledged. The objective of this phase is to define what the system will provide for the user. The design phase defines how it is to be achieved.

Prototyping can be a powerful tool in assisting the requirements specification process. However prototypes are rarely developed because they are considered too expensive. For example, a panel session at the Fourth International Conference on Software Engineering dismissed prototypes on the grounds of cost [1]. However, correcting ambiguities and misunderstandings at the specification stage is significantly cheaper than correcting a system after it has gone into production.

This paper describes how a prototype was used to help specify the requirements of a computer system to manage a semiconductor processing facility which produces integrated circuits. The cost of developing and running the prototype was less than 10% of the total software development cost.

2.0 Problems in Specifying User Requirements

The Requirements Analysis and Specification phase of the software life cycle involves analyzing the user's requirements and then specifying a system to satisfy them.

The procedure usually followed in specifying a new computer system is to write a Requirements Specification and submit it for the approval of the users. The main problem with this approach is that users find it extremely difficult to visualize how the system will actually perform in their own environment just by reading a paper specification. This is particularly the case if the system is to be used by a wide spectrum of users, all of whom have different (sometimes conflicting) requirements, and some of whom have no previous experience of computer systems. These users often find a written specification dull to read and difficult to understand. Hence they are uncertain whether or not the proposed system meets their requirements. It is also very difficult to determine whether a Requirements Specification is complete, correct, consistent and unambiguous.

Furthermore, these problems are compounded by the fact that experience has shown that errors in requirements specification are usually the last to be detected and the most costly to correct [2]. Many data processing departments spend as much time maintaining existing computer systems as developing new ones, and many so-called 'maintenance' projects are in fact projects established to correct errors in the original Requirements Specification.

In recognition of these problems, various methods and tools have been developed to assist the requirements specification process. These tools fall into two main categories, system specification languages and graphical tools, or some combination of the two.

CH1627-9/81/0000/0333$00.75 © 1981 IEEE

Use of a formal system specification language such as PSL (Problem Statement Language [3]) and RSL (Requirements Specification Language [4]) can help ensure that the specification is unambiguous and consistent. However, it cannot ensure that the specification is correct, i.e. the problem may have been correctly specified but it may not be the problem the user wants solved. Nor can it ensure that the specification is complete - a function or a set of functions may have been entirely omitted. One study identified that 20% of the errors in the requirements specification of several large software projects were due to missing information, and a further 30% were due to incorrect requirements [5].

Furthermore, a typical user who has to approve the requirements specification may well find it difficult to determine whether a specification written in a formal language does indeed satisfy his requirements.

For this reason other specification tools which have been developed are based on a graphical approach to specifying user requirements. A frequently used approach is based on information (or data) flow diagrams. This approach has been formalized in SADT [6], Structured Analysis [7] and SAMM [8], by adopting a hierarchical decomposition technique to decomposing the problem to be solved into several sub-problems with well defined interfaces between them.

One advantage of the graphical approach is that these diagrams are understandable to users who may then be able to provide valuable feedback to the developers. However, from the user's point of view, the best way to determine whether a system will satisfy his requirements is "hands-on" use of the system.

3.0 The Prototype Approach to Specifying User Requirements

The communication gap between the system developer and the user can be bridged by developing a prototype of the proposed system. With a prototype, the user can exercise the system just as though it were already operating in his own environment, and thereby provide vital feedback to the developer on the suitability of the specification.

Other branches of engineering frequently make use of prototypes, pilot plants and modelling techniques before putting complex systems into production. Prototyping can also help the software engineering process by producing application systems that are more user oriented.

A good time to develop a prototype is after a preliminary version of the Requirements Specification has been developed. At this stage, the developer believes he has a reasonable understanding of the user's problem and has completed his first serious attempt at proposing a system to satisfy the user's requirements.

The cost of making changes to a system increases significantly as the software development proceeds [2]. Consequently it is desirable to obtain the user feedback as early as possible in the software life cycle and definitely before the Requirements Specification is finalized. As the objective of the prototype is to maximize user feedback, the prototype should emphasize the user interface at the expense of lower level software which is not visible to the user.

After the prototype has been developed, users should be given ample opportunity to exercise it and to provide their feedback to the developers. Based on user feedback, further changes can be made to the prototype. Usually these changes are of a relatively minor nature, such as modifying output report formats or user prompts.

At some stage, which has to be carefully monitored and evaluated by user and developer, the benefits of further enhancements to the prototype are outweighed by the time and cost required for these modifications. At this stage, further development of the prototype is stopped. The temptation to continue development of the prototype after it has outlived its usefulness should be resisted. In particular, allowing the prototype to evolve into the delivered system is not recommended. The software engineering approach required to develop a production system is very different from that required to develop a prototype (see section 6). Hence maintainability is a much more important consideration in developing a production system than in a prototype.

Based on user feedback, the Requirements Specification is revised. This revised specification forms the basis from which the design and implementation of the system proceeds.

A prototype may also be used as a vehicle for training users on how to use the system. However, this requires that the prototype be kept up to date. The advantages of this approach have to be weighed against the additional cost of maintaining the prototype.

Prototyping is viewed as being complementary to other tools for analyzing and specifying user requirements. Thus a prototype can be used to help determine that the proposed system does in fact satisfy the user's requirements, i.e. to ensure that the specification is complete and correct. A requirements specification language may then be used to specify the user's requirements, ensuring that the specification is unambiguous and consistent.

4.0 Description of the Prototype

4.1 Introduction

The system under discussion, called PROMIS, is a Process Management and Information System for a semiconductor processing facility. PROMIS is a joint development of I.P. Sharp Associates and the General Electric Company, and the system is being installed in G.E.'s Solid State Applications Operation in Syracuse, N.Y..

PROMIS is a highly interactive system. Most user interactions with the system were explicitly prototyped. Some aspects of the system (such as interfacing with other computers) were not prototyped as it was felt that these requirements were generally clearly understood and would not benefit so much from prototyping.

The emphasis of the prototype was on prototyping user interactions with the system and not on performance. The prototype runs on an Amdahl 470 and the production system will run on a VAX 11/780. No attempt was made in the prototype to estimate what the performance of the production system would be. As the VAX has considerable spare capacity, performance was not considered a crucial factor.

4.2 Overview of the System

A semiconductor process (not to be confused with the software concept of a process) is the sequence of steps (typically 60-100) required to fabricate integrated circuit devices from bare silicon wafers (see Figure 1). In PROMIS, these processes are defined by a process engineer and stored on file.

A lot (i.e. batch) of silicon wafers (typically 25-50) is processed in the semiconductor processing facility as defined by the process. The processing for each process step is controlled by a process recipe which defines how a particular lot should be processed by the appropriate semiconductor equipment at a given work center. Each recipe consists of 10-15 operations and is displayed to the equipment operator on the work center CRT. A typical process structure is shown in Figure 2.

A lot is moved through the work centers of the facility as directed by the process it is executing.

4.3 Overview of the Prototype

PROMIS has four major functions:

- Process Management
- Lot Tracking
- Lot History Analysis
- Facility Monitoring

All of these functions, apart from Facility Monitoring, were explicitly prototyped. It was felt that the requirements of Facility Monitoring (monitoring ambient temperatures, gas flows, gas pressures etc.) were generally clearly understood and would not benefit so much from prototyping.

PROMIS is a highly interactive system. In the prototype, a menu was provided as the means by which users selected functions to be executed. The menu was implemented as a three level command tree (see Figure 3). The objective was to provide a menu that is friendly to both novice and experienced users.

To select a function from the menu, the user may type the entire command name (e.g. LOT PROCESSING) or, starting with the first character, any unique substring of the command name (e.g. LOT or L). If the substring is not unique the system points out the ambiguity and reissues the prompt.

An experienced user may enter multiple commands on the same line. For example, L T V means that LOT PROCESSING (L) will be selected from level 1, TRACK OPERATIONS (T) will be selected from level 2, and VIEW LOT RECIPE (V) will be selected from level 3 of the command tree. Once a function at the lowest level of the command tree has been selected, the user is presented with simple English language prompts for any data that is required to execute that function.

The major groups of functions available to users are:

a) Process Management

These functions allow process engineers to create and modify

semiconductor processes and their constituent recipes and operations. The process engineer is provided with a series of English language prompts to which he responds. For example to create a process, he is prompted for the overall characteristics of the process. He is then prompted for the recipe used at each process step.

b) Lot Tracking

These functions allow lots to be tracked in and out of work centers. Recipes are displayed at work center CRT's, enabling operators to process a lot on the work center equipment by following the operations described in the recipe.

The procedure for executing a process step at a work center is as follows:

1. TRACK IN. A lot is selected from the input queue for that work center and is tracked into the work center, i.e. transferred to the active list.

2. VIEW LOT RECIPE. The prototype automatically determines the recipe for that lot at that process step and displays it to the operator.

3. ENTER TEST RESULTS. If the recipe includes a test operation, the operator is prompted for each data item in turn, and the system performs range checks and pre-defined calculations on the test results.

4. TRACK OUT. The lot is tracked out of the work center, after which it appears in the input queue for the next work center.

c) Lot History Analysis

Each time a lot is tracked out of a work center a complete record of what happened to the lot is written out to file. This data may be later analyzed by a process engineer using an interactive query language to determine, for example, what factors affected lot yield. From this information, the engineer may modify this process with the objective of improving yield.

A more detailed description of the prototype is given in [9].

5.0 Experience with the Prototype

Users were given a two hour course and a practical demonstration of how to use the prototype. They were then given 2 weeks in which to experiment with the prototype.

Two methods were used to obtain the users' feedback: an evaluation form which they were asked to complete, and an on-line mechanism for entering (and reading) user comments. This allowed a user to enter his comments on-line, while using the prototype, and to review the comments of other users.

The main points revealed by developing and using the prototype were as follows:

1. In software development projects misunderstandings between developer and user tend to occur all too frequently, because of their different backgrounds. In this project, this type of misunderstanding was revealed through use of the prototype. For example, the developer thought that a semiconductor process could produce only one device, whereas it can produce many. This capability is required in the production system.

2. Ambiguities and inconsistencies in the Requirements Specification were identified, as the prototype was being developed, and corrected. Thus inconsistencies in the procedures for controlling process and recipe modification were discovered and corrected.

3. Omissions in the Requirements Specification were discovered when users asked for features which they considered essential, but were not available in the prototype. Examples of these omissions were the need for daily and weekly summary reports, and a user function to determine the lot to which an individual silicon wafer belongs.

4. Errors were discovered in the prototype which were due to incorrect or missing requirements. Thus in the prototype it was possible to start a lot before the process to fabricate it was fully defined. Furthermore it was

possible to track a lot out of a work center before test results had been entered. Both of these features should have been prohibited.

5. Some functions did not provide the user with the information he wanted. These were modified to provide additional or different information. An example of this is that although functions were provided to list the contents of individual processes and recipes, users requested the option of listing groups and summaries of processes and recipes.

6. Users were able to give valuable feedback as to which features of the system they found difficult or confusing to use. Users found certain command names confusing. Certain terminology such as 'queue' was also found confusing. Prompts requesting input included data type definitions (such as I6 for a six digit integer number) which were intended to help users. In fact these had the opposite effect and consequently were removed. Users also requested the capability to exit from a function prematurely and to return to the state they were at before entering the function.

7. In some cases the user was not sure how he wanted certain functions implemented. Developing the prototype helped him come to a decision on these. One of the major areas prototyped was the historical analysis of lot data. It was necessary to have a flexible and usable procedure. Although the proposed approach was flexible, it was not clear how usable it would be for process engineers. Some difficulty was experienced in using these features on the prototype and so the procedure is being modified.

8. Developing the prototype provided valuable insight to the developers on how the system should be designed, in particular how the system should be structured, how files and data should be structured and which algorithms should be used.

As user feedback was received, some relatively minor modifications were made to the prototype, primarily in the areas of report formats and user prompts. This enabled users to experience the results of their comments quickly. After five iterations, further development was cut off as it was felt that the primary benefits had been obtained from using the prototype.

A detailed analysis was then made of the user comments. Based on this, the Requirements Specification was significantly revised.

6.0 Implementation Aspects of the Prototype

The prototype was developed in APL by three people over a three month period. The total manpower spent on the prototype was 7 man months, which represents about 6% of the total estimated development effort of 10 man years. In addition, approximately $12,500 was spent in timesharing costs.

The main reasons why the development costs of the prototype were kept to this level are as follows:

a) Emphasis of the Prototype

The prototype emphasized the user interface to the system. The objective was to get the most benefit (i.e. user feedback) from the prototype. Consequently, not all aspects of the system were prototyped. In particular, frequently executed user functions were emphasized, not the exception conditions. Data validation, error handling and logging were not as comprehensive in the prototype as they are planned to be in the production system. Certain subsystems such as system recovery and facility monitoring were not prototyped at all, because they have little or no interaction with the user. Performance considerations were also ignored in the prototype.

b) Size of the Prototype

The prototype was developed by 3 people whereas the production system is being developed by a team of 12 people. A much more flexible and less formal development approach was therefore used for the prototype than that required for the production system.

c) APL Language

APL is an interpretive language with many powerful features for manipulating multi-dimensional arrays. Consequently APL programs are often much shorter than

programs written in other languages. Because APL is interpretive, APL programmers tend to find it relatively fast to test their programs.

APL has often been criticized for resulting in unmaintainable code [10]. Whereas there is some justification for this criticism, it is possible to write readable programs in APL by observing a disciplined approach and a good programming style [11]. However, maintainability is by no means as important a consideration in a prototype as it is in a production system.

d) APL System

The APL system used, SHARP APL[12], has a number of features which assist program development. It has a file system which is easy to use. It also has a powerful report formatting utility, which facilitates writing and changing of sample reports, a valuable capability in any prototyping exercise. Furthermore, backup and recovery procedures are built into the system.

7.0 Conclusions

The use of a prototype was found to be an excellent method of evaluating the Requirements Specification.

The circulation of a written Requirements Specification to process engineers, management personnel, facility operators and supervisors yielded virtually no useful feedback. Unfortunately, the specification for such a system has to be quite lengthy to be complete, and prospective users often don't find the time to evaluate a written specification thoroughly. Even if they have the time, the task of carefully scrutinizing a 120 page document is difficult and boring, and is therefore not performed with the enthusiasm that is most conducive to good results.

Exercising a prototype system is interesting. Prospective users find it much easier to evaluate a working system than a paper document, and as a result are much more likely to obtain a system that will meet their requirements.

The cost of developing and running this prototype was less than 10% of the total software development cost. This demonstrates clearly that prototyping can be done economically. It is believed that this is also true for many other systems.

The feedback obtained from using the prototype was found to be extremely valuable, and more than justified its development. The changes made to the Requirement Specification, as a result of user feedback obtained through using the prototype, were of relatively low cost and would have been considerably more expensive to make had they been left to later in the software life cycle.

8.0 Acknowledgements

The authors gratefully acknowledge the major contributions of J.R. DeBolt, R.F. Johnston and I. Griggs in the Requirements Specification of PROMIS and the development of the prototype. Thanks are also due to L.J. Lambert and C.A. Shaw for their assistance in the Requirements Specification and the evaluation of the prototype.

9.0 References

1. W.P. Dodd,"Prototype Programs", IEEE Computer, February 1980.

2. B. Boehm,"Software Engineering - R and D Trends and Defense Needs" in Research Directions in Software Technology, MIT Press, 1978.

3. D. Teichrow and E. Hershey,"PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Transactions on Software Engineering, January 1977.

4. T. Bell, D. Bixler and M. Dyer,"An Extendable Approach to Computer Aided Software Requirements Engineering", IEEE Transactions on Software Engineering, January 1977.

5. T. Bell and T. Thayer,"Software Requirements: Are They Really a Problem?", Proceedings of the 2nd International Software Engineering Conference, October 1976.

6. D. Ross and W. Schoman,"Structured Analysis for Requirements Definition", IEEE Transactions on Software Engineering, January 1977.

7. T. DeMarco,"Structured Analysis and System Specification", Yourdon Press, 1978.

8. S. Stephens and L. Tripp,"Requirements Expression and Verification Aid, Proceedings of the 3rd International Conference on Software Engineering, May 1978.

9. H. Gomaa and D. Scott,"An APL Prototype of a Management and Control System for a Semiconductor Fabrication Facility", Proceedings of the 1980 APL Users Meeting, October 1980.

10. E. Dijkstra,"The Humble Programmer", Communications ACM, Vol. 15, No. 10, 1972.

11. B. Kernighan and W. Plauger,"The Elements of Programming Style", McGraw Hill, Second Edition, 1978.

12. P. Berry,"Sharp APL Reference Manual", I.P. Sharp Associates, 1979.

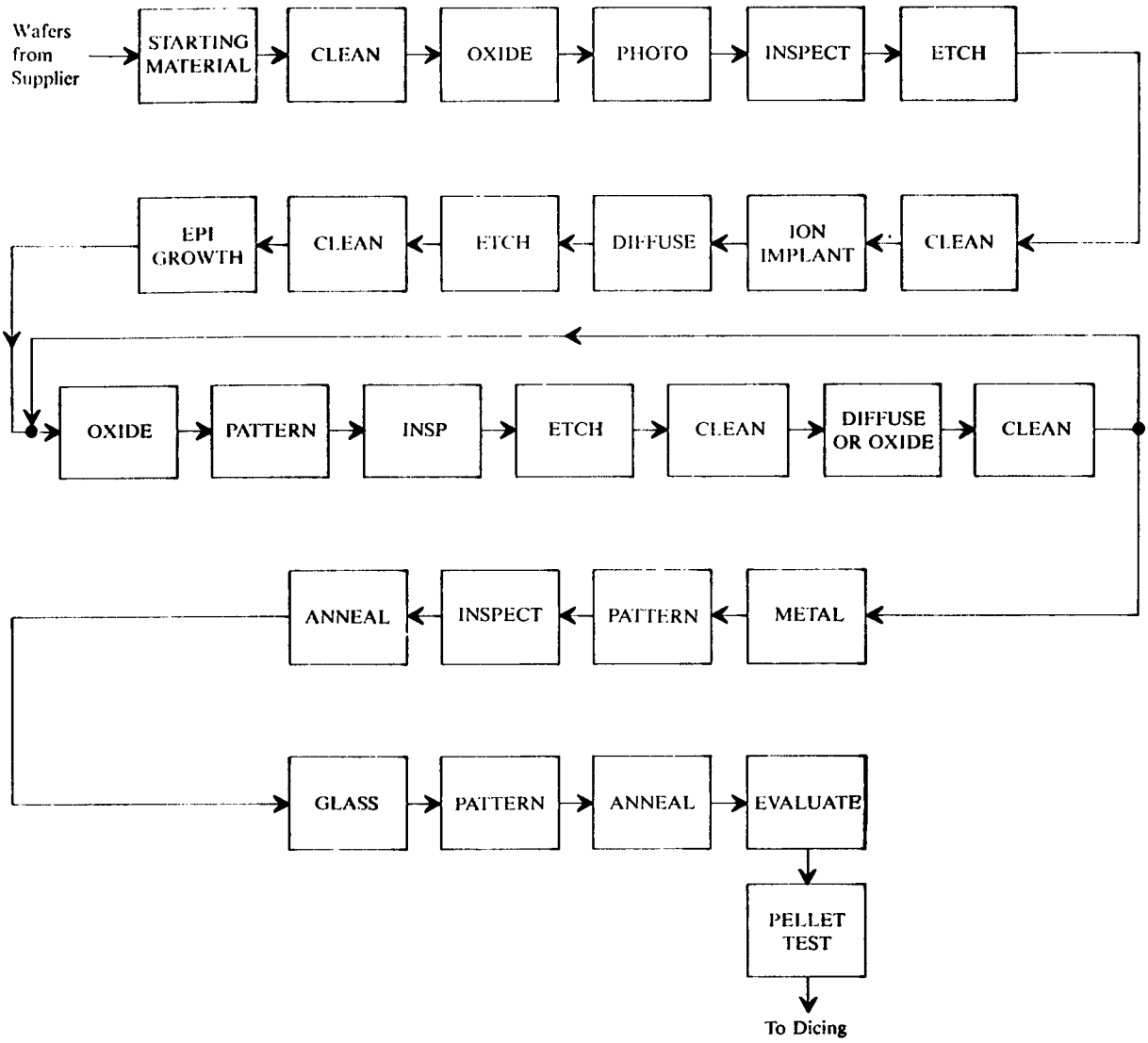

FIGURE 1 Example of Semiconductor Process

FIGURE 2

PROCESS STRUCTURE

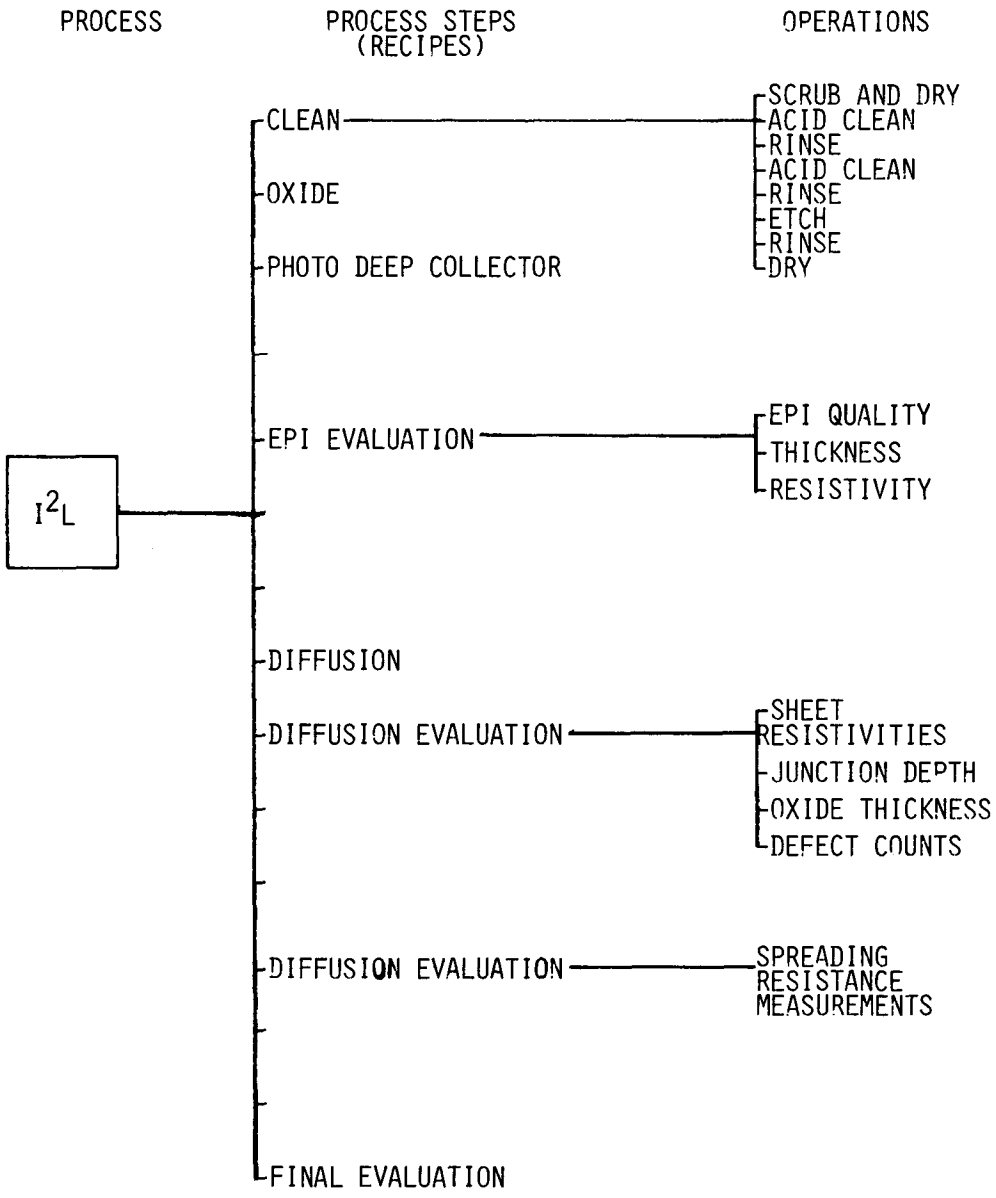

FIGURE 3

PROTOTYPE MENU STRUCTURE

LEVEL 1	LEVEL 2	LEVEL 3
PROCESS MANAGEMENT	PROCESS FUNCTIONS	ADD PROCESS
		CHANGE PROCESS
		DELETE PROCESS
		LIST PROCESS
		FULL-LIST PROC
		REC-LIST
		F-REC-LIST
		B-REC-LIST
		RECIPE XREF
	RECIPE FUNCTIONS	ADD RECIPE
		CHANGE RECIPE
		DELETE RECIPE
		LIST RECIPE
		F-REC-LIST
		B-REC-LIST
		RECIPE XREF
		OPERATION XREF
	OPERATION FUNCTIONS	ADD OPERATION
		DELETE OPN
		LIST OPN
		FULL-LIST OPN
		BRIEF-LIST OPN
		OPERATION XREF
LOT PROCESSING	MANAGING LOTS	START NEW LOT
		SPLIT A LOT
		REWORK LOT
		BACK UP LOT
		DISPLAY A LOT
		EDIT A LOT
	TRACK OPERATIONS	INTO WORK CENTER
		VIEW LOT RECIPE
		OUT OF WORK CENTER
		ATTACH TO WC
		WORK CENTER DISPLAY
		EQUIPMENT DISPLAY
		FIND LOT
		ENTER TEST RESULTS
	REPORTS	LOT
		FULL LOT
OUTPUT DATA	STATUS	LOT
		EQUIPMENT
		WORK CENTER
		WAFER INVENTORY
		TANKS GAS
		BACK LOG
	LOT HISTORY	DESCRIBE
		REPORT
	OTHER HISTORY	ALARM HISTORY
		FACILITY HISTORY
MESSAGE	DIRECTORY	
	SEND	
	COMMENTS	DESCRIBE
		WRITE
		READ