Towards Better Object Oriented Software Designs With Quality Function Deployment

M. ELBoushi [1]
Jet Propulsion Laboratory
S. Zawacki [2]
Jet Propulsion Laboratory
E.Domb [3]
GOAL/QPC

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

1 Abstract

NASA's new direction is towards "faster, better, cheaper" space missions. At the Jet Propulsion Laboratory (JPL), home of NASA's interplanetary unmanned missions, we are meeting this challenge in many ways. One such effort is the Advanced Multi Mission Operations System (AMMOS), a ground software intensive system currently in development that will replace the costly past method of re-engineering and redeveloping a ground system for each new JPL mission. AMMOS will be used to support all future JPL missions, from interplanetary trips to Mars and Saturn, to the "Mission to Planet Earth".

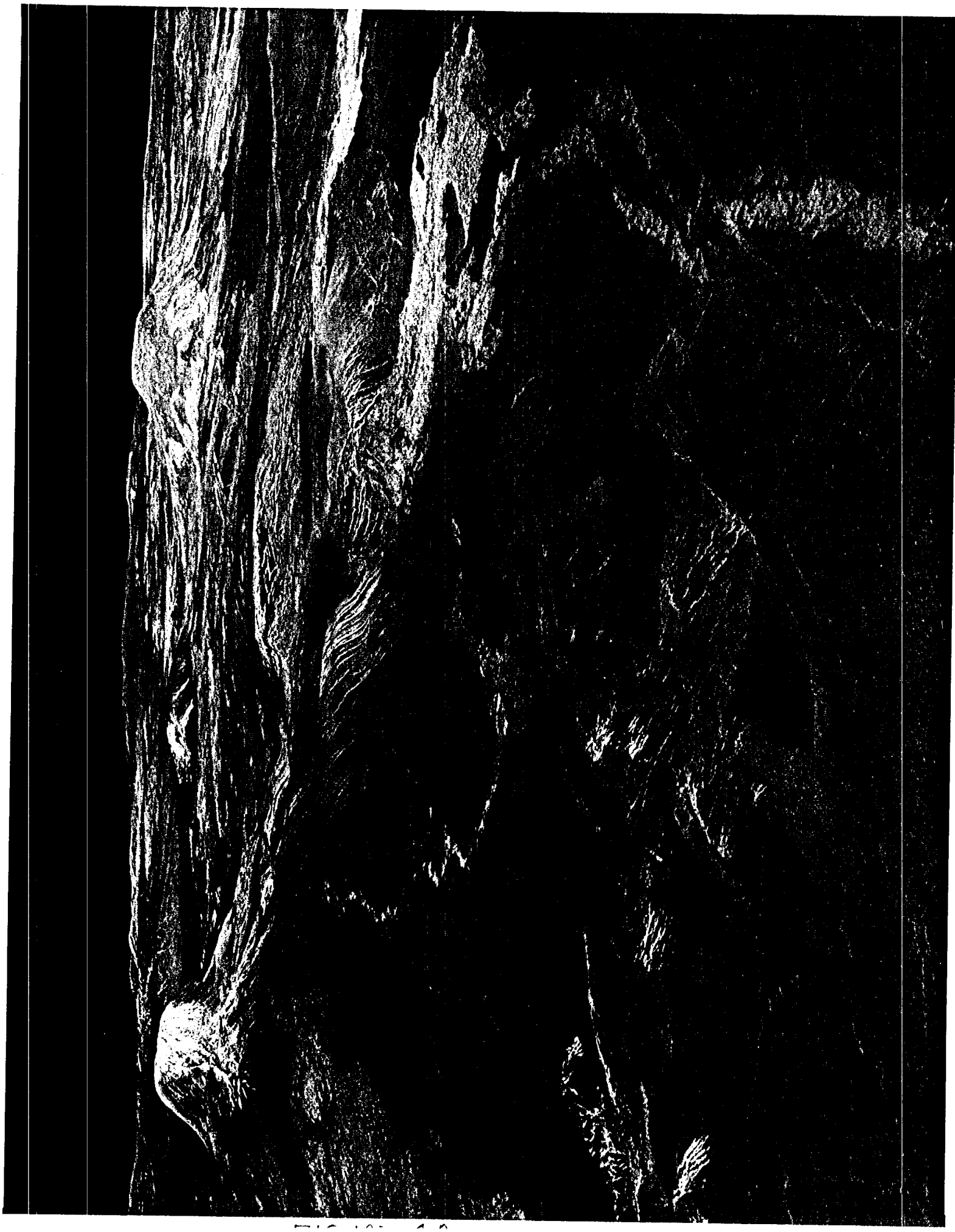
Quality Function Deployment (QFD) was used for the requirements and design analysis of one program in the Sequence Subsystem of AMMOS. This program, REVIEW, is meant to provide spacecraft sequencers a tool to make electronic reviews of the files produced by other sequence programs. The target customer was a small, yet diverse internal group spanning multiple scientific and engineering disciplines as well as individual mission "cultures". QFD provided a methodical approach of capturing the voice of the Customer (VOC) and played an important role in developing and tracing requirements. The QFD process is currently being extended and used for our object oriented design (OOD) of REVIEW, wherein important objects and classes are being identified and traced to the original VOC. This information will be used in prioritizing the development order.

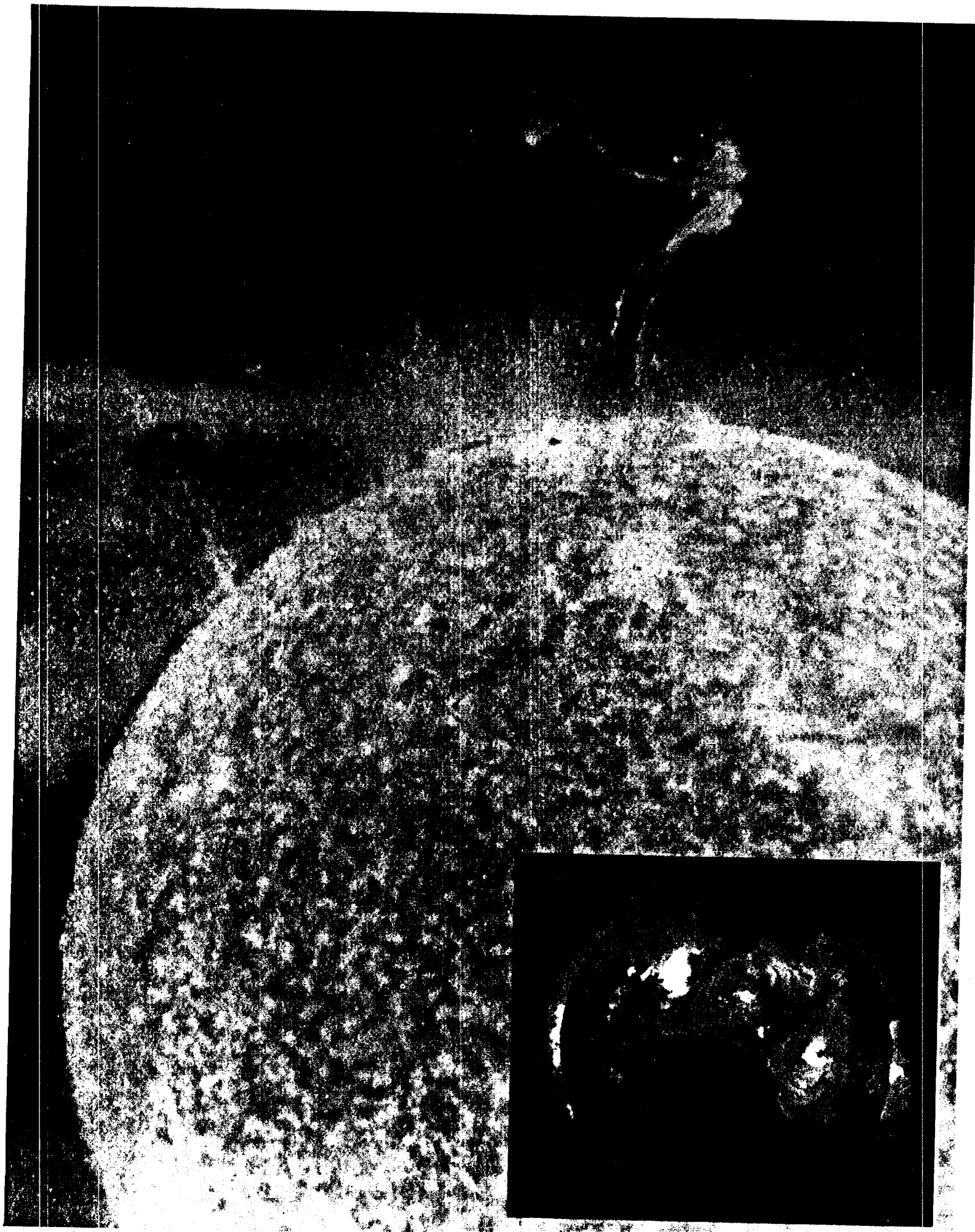
This paper shows how QFD focused our effort to produce an internal product for internal customers with diverse needs, and how it was expanded for use with modern 00D software technology

2 Introduction

Spacecraft are composed of many different subsystems that are needed for their specific missions. Among these are spacecraft specific subsystems, such as attitude control, data storage and thermal control, as well as instrument subsystems that allow the spacecraft to carry out its scientific mission (see Figure 1). All these subsystems are controlled through the ground data system. The ground data system communicates with the spacecraft through sequences, which are sets of commands that are relayed to the spacecraft and loaded into it's on board memory. The spacecraft then executes these time ordered instructions. The format of a sequence is very strict and specific, and before it is uploaded to the spacecraft, there are a number of programs that are run to make sure that no spacecraft or mission damaging instructions are contained in the sequence. This is a very critical and lengthy process, which is currently done by many different groups using a variety of software tools and ad hoc methods, including manual inspection of computer printouts, BASIC "stripping" programs running on personal computers, C-shell or "Awk" scripts running on UNIX workstations, and so on. REVIEW is a new multi-mission development that is being created to address the ad hoc method of sequence inspection. Ultimately, REVIEW will offer sequence engineers and other potential users a tool that incorporates all the features they requested in a single multi-mission package that is easy to use, adapt, port and maintain. This is all within the new spirit of NASA's "faster, better, cheaper" space missions.

QFD was selected as the tool to use to gather the requirements and aid in the design analysis, primarily to serve as an experiment in the use of TQM tools to solve a real world problem that we face everyday at JPL. QFD was found to be a very good way of gathering requirements from many potential user groups with diverse needs and internal structures. This diversity was why a single sequence review tool wasn't created earlier. Another obstacle was that our customers (sequencers) had a particular way of reviewing sequences that was established decades ago. They also came from various technical backgrounds, and had personal preferences for computer platforms, operating systems and programming languages. We felt that QFD could help us systematically address each of these issues, plus the standard issues faced by all software developers.





3 Body

3.1 Preliminaries

The first QFD implementation problem we faced was the selection of the team members. Prior to selecting the REVIEW steering committee, a preliminary questionnaire was circulated to get an initial indication of who the potential customers were (see Figure 2). The questionnaire asked what the person's current duties were, where in the sequence generation process their duties lied, what platform and operating system' they currently used or were familiar with, how they went about reviewing their sequences, what they felt could better help them expedite this process, and finally, what they felt electronic review meant to them. From this questionnaire, a potential list of steering committee members was formed. We decided that the team should span the several spacecraft subsystems, JPL flight projects and the various software development disciplines. One important criterion for team membership was that the prospective team member had to have either worked with, or currently been working on sequences. To make our selection, we found it helpful to make a matrix of JPL flight project vs. spacecraft subsystem (Figure 3). We filled each location with a prospective team member name and took advantage of known overlap in order to achieve team balance and to keep the size down to 9 members.

Once the REVIEW steering committee was selected, we all took a class in QFD. The steering committees first priority was to develop requirements. To help drive out requirements, we developed 3 prototypes in parallel with the QFD exercise. These prototypes were built with the idea that they would demonstrate the feasibility of REVIEW and investigate important technological issues. One prototype used "perl", a UNIX script utility program, to develop and illustrate a primitive sequence review tool that did simple stripping. Stripping is the process which, based on a condition, extracts information contained in a sequence file. Another prototype was created in LISP, and took advantage of. it's artificial Intelligence capability to give a more complex illustration of a sequence review tool, Among this prototypes features was the ability to adapt it to any sequence file input format. This was an important goal because, in order for REVIEW to be successful, it had to be able to address the many different sequence file formats that existed for current as well as future space missions. The third and final prototype used two other UNIX utilities, "Lex" and "Yacc", to build a little language parser and lexical analyzer that was ultimately to be incorporated into the final tool. These tools were overseen by the QFD team in the later phases of their development. As time went on, these prototypes were combined with existing sequence generation tools (and other prototypes of future ones) to demonstrate their combined capability. Many demonstrations were made to potential customers, and the prototypes were in a state of continuous improvement throughout this whole process. Combined with the ongoing QFD exercise, the prototypes drove out all the requirements by challenging the customer to think about what an ideal electronic review tool would look like.

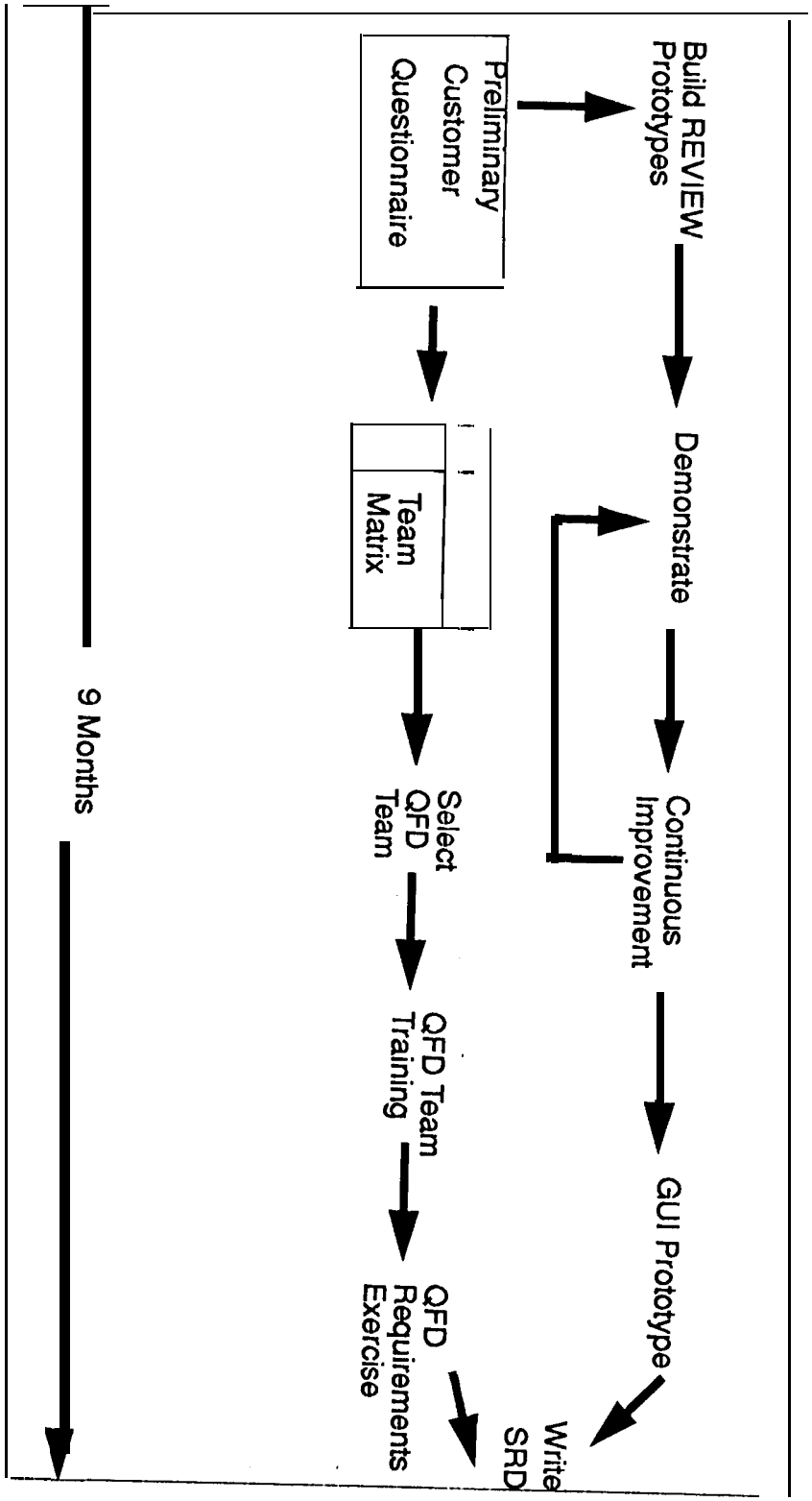


Figure 2: Overall QFD/Prototype Flow

Disciplines / Projects	Sequence Team	Science	Spacecraft Team	Mission Control Team	Mission Planning Team	Software Development
Voyager	D,I	G				D
Galileo		B	t			I
Topex	D					D
Mars observer	A			F	H	
Magellan		G				
Cassini		C				
Multl-Mission						

Key: A-1 Individual Team Members

Figure 3: Team Matrix

3.2 **Voice of The Customer (VOC)**

Next, a second more specific and detailed questionnaire had to be developed in order to capture the VOC. The questions on the VOC form grew out of interviews the steering committee conducted on each other as an exercise on the last day of our QFD class, and refinements during subsequent team meetings (Figure 4). The questions themselves were based on the"5 W's and an H" principle. The target customers that the team interviewed came from a cross section of sequencers from the various JPL missions (Figure 5).

From the results of the VOC interviews the steering committee synthesized short statements of customer desires, necessities and functions of REVIEW. These statements were grouped with an affinity process, As the steering committee grouped these statements, we started to decide whether the statement was a Demanded Quality (DQ), Quality Characteristic (QC) or function. The DQ's were organized into 6 broad categories such as "Sequence Validation" and "Ease of Use", and then into additional sub-categories such as "Find stimuli of violations" and "Filter and re-order fields". Functions, and QC'S, which express in a quantitative manner how the developers will meet the customer DQ's, were synthesized by the team from the interviews and DQ's. This was the single biggest problem the team faced during the QFD application. We found that QC'S were not easy to express for software, and as a result the team spent several meetings shuffling items between the DQ, QC and function headings. Figure 6 shows the result of our efforts. Once the VOC was translated into the three groups, we started to build our matrices using QFD software.

3.3 **Demanded Quality Weight**

A third questionnaire had to be created to get the customer ranking (or weight) of each DQ. The third questionnaire posed a problem, because we were asking the customer to weight the importance of over 50 DQ's, and then subgroup these weights according to the rank they gave each DQ. We had to come up with a questionnaire that would be easy for the customer and still give us all the information we needed (see Figure 7). All the DQ's were put on the vertical axis, and all the weights on the horizontal axis. Weights were measured from 1 ("Most important") to 5 ("Least Important"). First, the customer ' was asked to scan the list of DQ's and rank them from 1 to 5. Next, we asked the customer to group each choice by weight, so that every DQ that the customer gave a weight of 1 to became a group. They then proceeded to rank this sub group from 1 to n, where n was the number of choices in weight group 1. This allowed the steering committee to compute an average score for each one of the DQ's

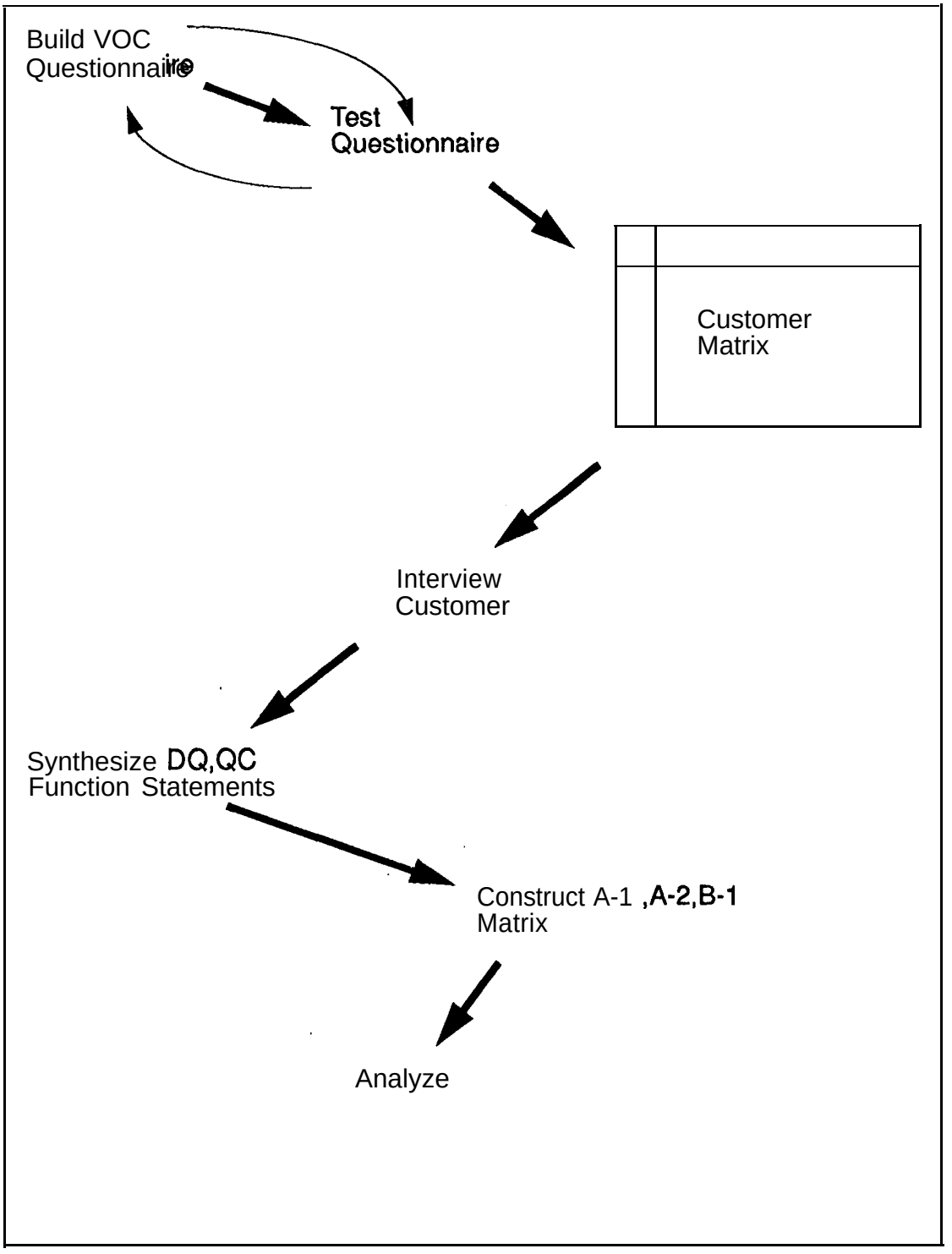


Figure 4: QFD Requirements Exercise

Spacecraft Subsystem / Flight Project	Sequencing Team	Science Planning	Nav gat	Flight Operations	Mission Planning	Spacecraft
VOYAGER	X	X	X	X	'X	X
GALILEO	X	X				
TOPEX			X	X		X
MARS OBSERVER	X	X	X	X	X	X
MAGELLAN	X			X	X	
CASSINI	X	X			X	
ULYSSES				X		
SIR-C		X				
MESUR					X	

Key: X marks an interviewed customer

Figure 5: Customers Interviewed

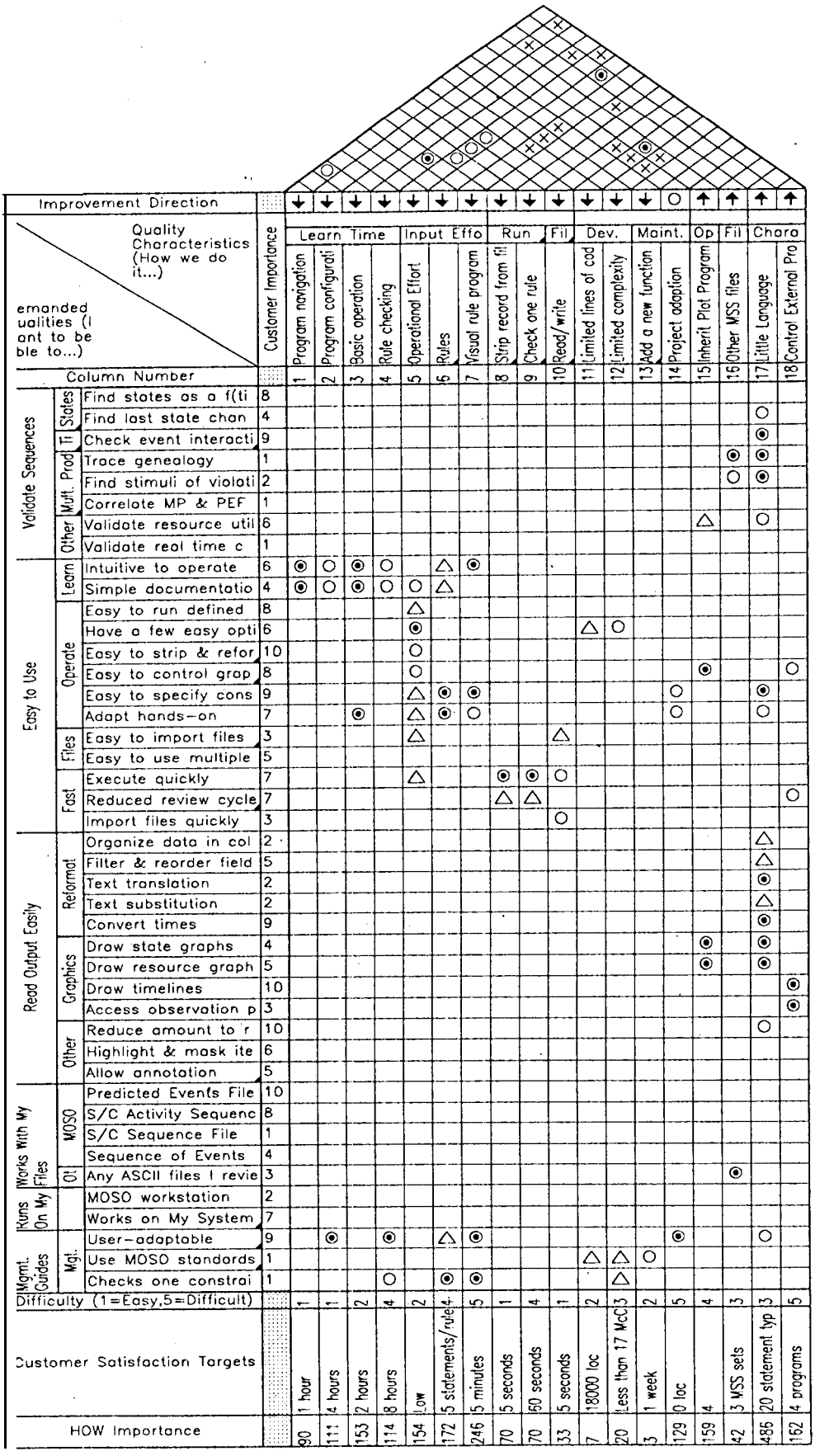


Figure 6: Quality Characteristics vs. Demanded Qualities

Demanded Qualities \ Weight	1	2	3	4	5
finds state (as a function of time)	2				
finds last state change	3				
check S/C event interaction		3			
verify observation implementation	1				
trace genealogy			1		
find stimuli of constraint violation			3		
correlate mission plan and PEF		2			
Validate resource utilization				2	
Validate fault protection F/P		1			
Validate real time (R/T) commands					2
Reduce amount to read			2		
Highlight and mask items				1	
time conversions	4				
text translation and substitution					1
state graphs		4			
Observation plots	5				

A Weight of 1 is ranked “Most important’ and weight 5 “Least Important”

Figure 7: Questionnaire of Customer Importance

4 Matrices

4.1 A-1, A-2, B-1 Matrix

Part of the standard QFD methodology is building matrices, the first of which is generally the A-1 requirements matrix, or DQ's vs. QC's. We quickly realized that the resulting matrix was too large, and that we would be consumed by the n-squared effect while analyzing our correlations. We dealt with this problem by reducing and/or combining some of our DQ's and QC's. For example, our customers gave us a long laundry list of computers and operating systems that they wanted REVIEW to run on, and these eventually found their way onto the DQ list. However, the first and second questionnaire clearly showed that the overwhelming customer favorite was an AMMOS Unix workstation. Therefore, we reduced the DQ list from a list of 5 computers and 5 operating systems to just "MOSO workstation" and "Works on my system". This eliminated eight rows from the DQ list! Once we completed this compression, we reduced the matrix to 42 DQ's by 18 QC's, leaving us a more manageable correlation job.

As mentioned previously, the biggest problem we had using QFD, including the matrix development, was cliff iculty in expressing QC'S for software, Of the 18 QC'S in our final matrix, 7 deal with user interaction time, either time spent in learning the operation of REVIEW or input effort. Although these times are important to users, and represent eminently testable QC'S they do not tell us much about measuring how well we meet many of the DQ's. We think that the remaining 11 QC'S fill this gap but we are not sure, and will be looking for evidence of this during our implementation.

Another trap we fell into was wasting too much time worrying about the empty columns and rows in our correlation matrix. We fretted for nearly a week about our lack of correlations for the "MOSO workstation" and "Works on my system" DQ's. Although we were obviously missing a QC to measure how well we met the DQ, we didn't need it! Why? Because we knew we had to meet those customer DQ's, and furthermore we had already conceived of a simple way to do it, so why worry about this missing QC?

4.2 Result

We found that using the QFD process took about the same time as the old way that we used to gather requirements. This could be attributed in part to the fact that this was our first time using QFD, so it was as much a learning process as it was an exercise. However, the results we got from QFD were much better than the previous method we had used. We found out that QFD is a good tool to drive out customer requirements and measure their importance, and our performance will only get better as we get more experience at using the QFD tool.

5 Software Requirements Document (SRD)

Since the QFD methodology is too terse for an SRD, we had to expand what we learned from the matrices to write the document. Our first attempt consisted of translating the list of DQ's, functions and QC'S into plain English.

This approach, although simple and quick, was rejected because the resulting requirements document was too hard to read. The problem was that our lists of qualities and functions did a good job of summarizing user requirements, but did not provide the reader with much of a feel for the functionality of the REVIEW product.

Our second, more successful approach, was to realize that the task of stating our requirements was going to be a lot simpler if we first carried out a couple of "pre-design" steps prior to writing requirements: First, a tentative Graphical User Interface (GUI) was designed. This gave us a chance to organize user-demanded features in a manner the user could relate to. We implemented this preliminary design in Microsoft Visual BASIC and made it available to potential users for feedback. Second, we needed to show a concrete example of a Little Language (LL), and explain how it relates to the desired functionality of REVIEW. This step actually required little effort, since a LL was already developed as part of the prototype effort. While this LL didn't meet all the requirements, it close enough to provide the reader with a sense of how the product operates.

With these precursor steps, the resulting SRD tells the story of how the user interacts with REVIEW and, along with the list of expanded DC? 'S, QC'S and functions, completed the REVIEW requirements process. Next, our attention turned to the design of REVIEW.

6 Object Oriented Design (OOD)

6.1 Methodology

Although we thought our use of QFD was over with the completion of the requirements process, we later learned we could make good use of the techniques during our 00D of REVIEW.

The method we used to design REVIEW was essentially the Class/Responsibility/Collaboration (CRC) approach described by Wirfs-Brock, et al. (see reference). The starting point of the design (see Figure 8) was the REVIEW SRD, which concentrates almost exclusively on the user's perspective of the program (as you might suspect). The requirements do not address how the program is supposed to accomplish the various tasks.

Therefore, we found that the SRD was not "rich enough" as a source of objects when it came to describing the inner workings of the program. In particular, it was difficult to try and write program operation scenarios that went beyond the user interface. We then decided to use the scenarios as a source of objects, in addition to a means to check the validity of the design. This is of course dangerous, since many design decisions could be made inadvertently while writing scenarios. We avoided this problem by keeping the sce-

narios as simple and "down to Earth" as possible and subjecting them to frequent scrutiny.

After writing five or six scenarios and looking at the objects that would be necessary to support them, it became clear that objects fell into well-defined classes, and that these classes should be organized into hierarchies using the inheritance scheme. The resulting classes provided our first "draft" of the design.

A "shell" prototype program, featuring all the classes, but only some of their responsibilities, was implemented in C++. This was done to validate our design and to make sure that the C++ compiler would not object to our inheritance scheme.

Inheritance, which had been the focus of our class-building effort, is only part of the story. It became clear that classes had a definite "personality" and that classes with similar personalities should be grouped in separate subsystems.This naturally led to the next phase in the design: organizing classes into subsystems (Figure 9).

In the next step of the design, we built two more prototypes. The first one was a refinement of the earlier "shell". Although this new version still did not do any useful work, it was able to print in indented, scenario-like style what it was doing. The second of these prototypes consisted of a MOTIF implementation of the "Define Strip" panel of the REVIEW user interface, This is probably the most complex graphic object in the GUI, As a result of all this prototype activity, we gained the confidence necessary to organize our preliminary classes into well-defined subsystems, We feel that our subsystem design is robust enough that it will survive any last-minute change to the class definitions, and we therefore look at our subsystem descriptions as the central part of our design.

6.2 Class Trace Back to SRD

With our classes defined, we did a trace back to the SRD. We made sure that every DQ,QC and Function was covered by a class and a subsystem. This helped flush out any missing aspect of our design. t

6.3 DQ vs. Classes

Now that we had an object oriented design with subsystems composed of classes and objects, how do we proceed in order to plan the work for the first delivery of REVIEW? After pondering this question, it occurred to us that QFD could give us an answer. We decided to make a matrix of DQ's vs. Classes and subsystems in the final design (Figure 10). Although an imposing matrix with some 1600 possible interactions, it was actually quite easy to complete because the notion of an object derived directly from the SRD, and hence directly from the DQ's and functions, was very concrete and easy for the programming staff to visualize, given the completed design. The only other "breakthrough" thinking that was needed was tore-define the meaning of the correlation weights. Instead of the classical "Strong", "Medium" and "Weak definition, which had little meaning in this context, we used the following definition that the programmers could more eas-

ily relate to:

Percent Object Code Requirement to Meet DQ	Weight
80-1 00%	9
30-79%	3
0-29%	1

Completing the matrix correlations told us what objects were required for each DQ. We could then choose the DQ's to be included in each delivery and see what we had to code.

7 Conclusion

QFD allowed us to address the VOC's. This was a process that was lacking in most of our software developments at JPL. With JPL's new directive of "faster, better, cheaper", QFD will play a major roll in achieving these goals. New and existing missions will require that software be more efficient and provide the correct capabilities. More missions will require that we come up with a process of reusable software. In the future, QFD will be expanded to not only address the VOC, but provide a more efficient way at building object oriented programs. Also, we are thinking of using QFD in the testing phase of the REVIEW package. In all, we hope to streamline the whole process of going from requirements to implementation and maintenance of a software project using QFD.

Acknowledgment

The work described in this paper was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration.

We wish to thank Dr. Pierre Maldague for his many efforts in completing this manuscript

References

1. "Designing Object-Oriented Software", Rebecca Wirfs-Brock, Brian Wilkerson and Lauren Wiener, Prentice Hall (1990)

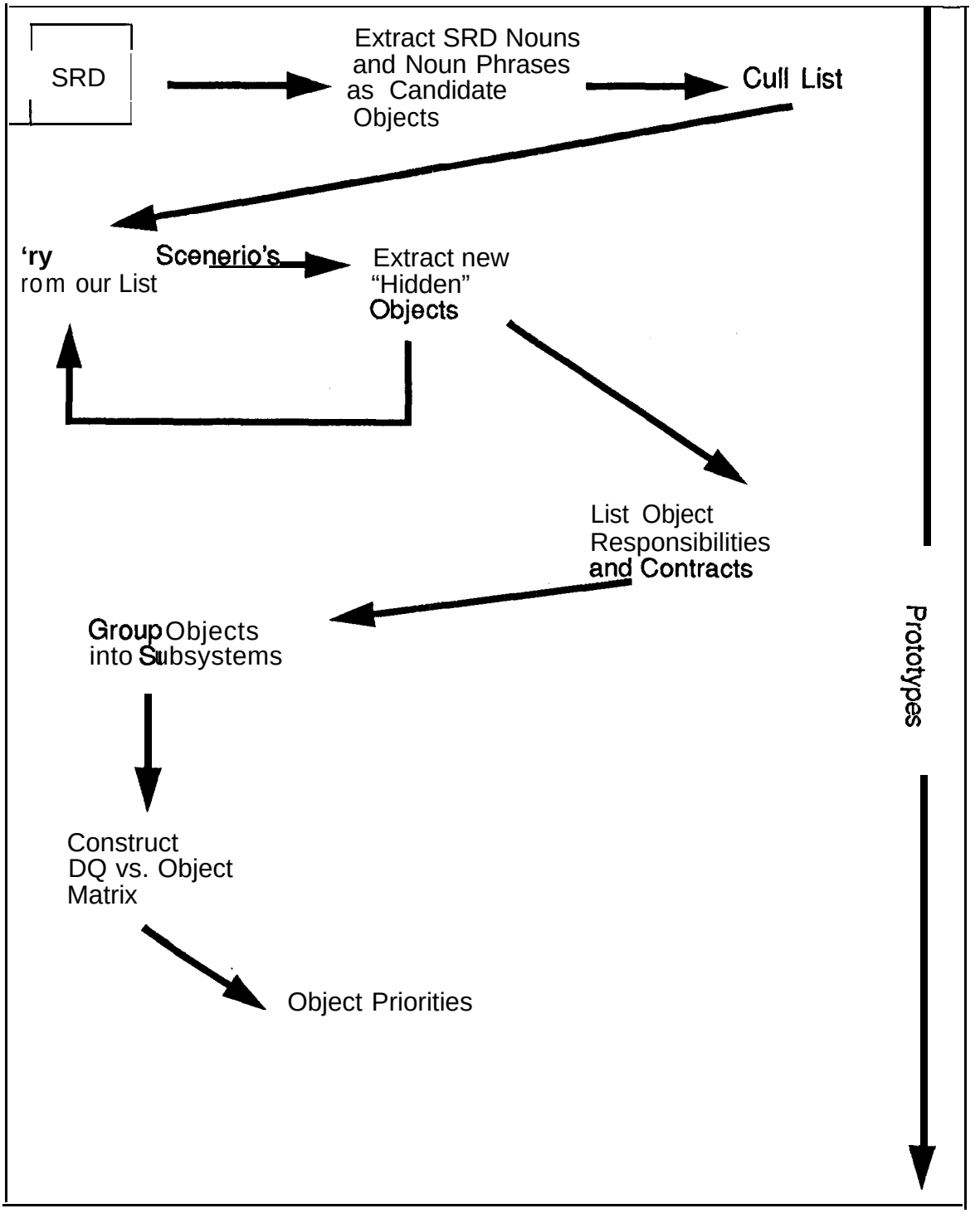


Figure 8: Object Oriented Design

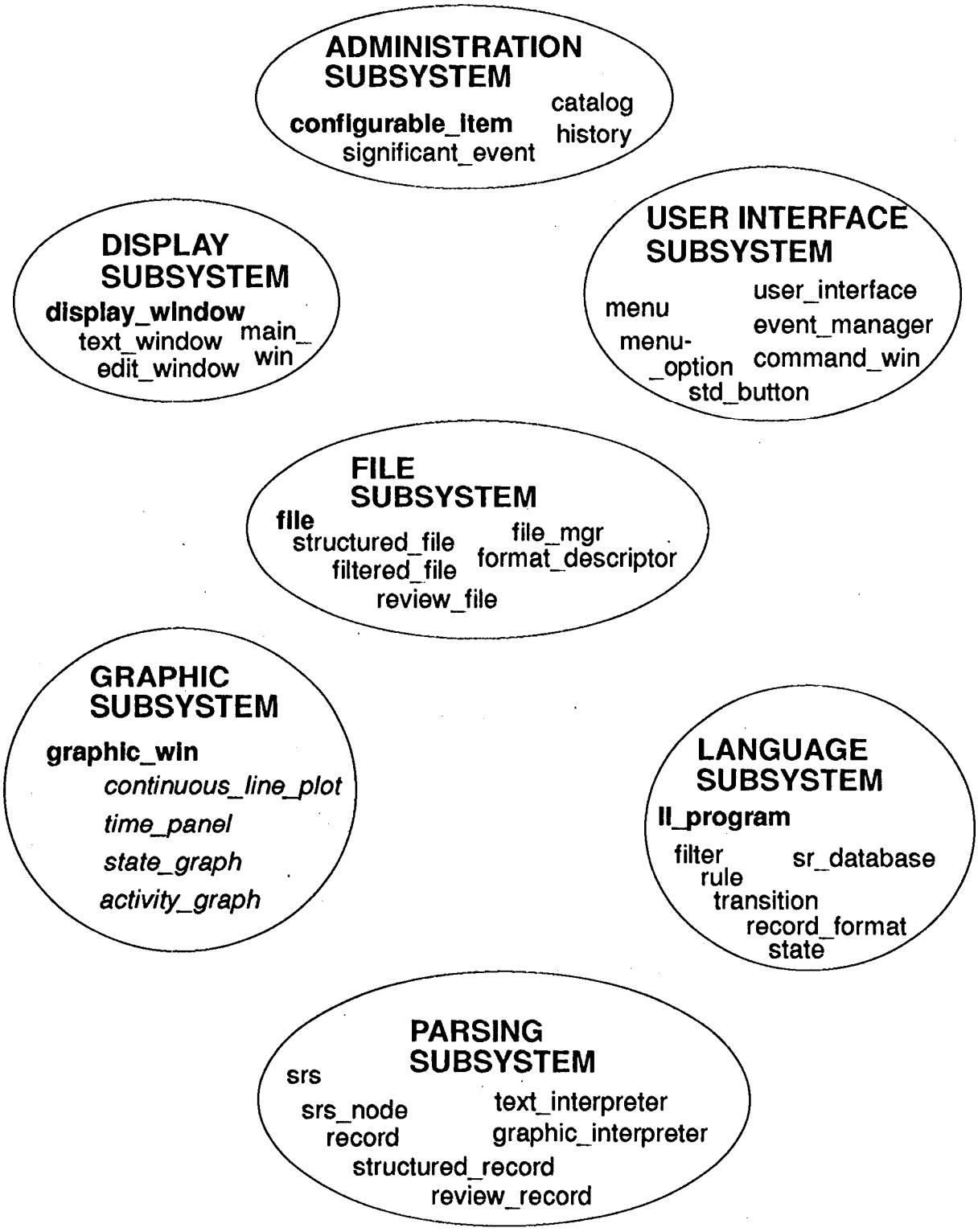


Figure 9: Subsystems and Objects

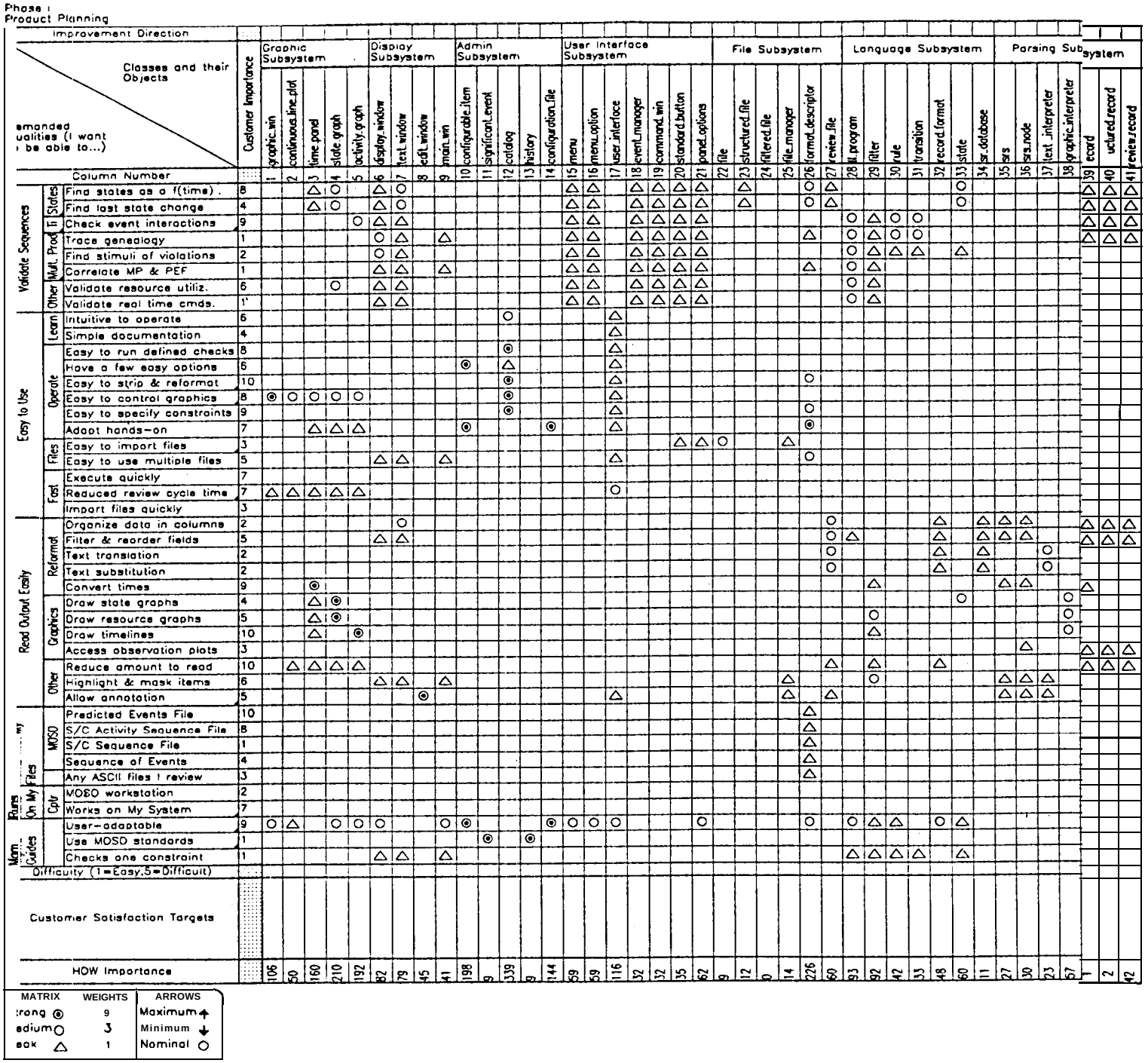


Figure 10 DQ vs. Classes Matrix