N95- 17561

SEQ_REVIEW: A Tool for Reviewing and Checking Spacecraft Sequences

Pierre F. Maldague
Mekki El-Boushi
Thomas J. Starbird
Steven J. Zawacki

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099

ABSTRACT

A key component of JPL's strategy to make space missions faster, better and cheaper is the Advanced Multi-Mission Operations System (AMMOS), a ground software intensive system currently in use and in further development. AMMOS intends to eliminate the cost of re-engineering a ground system for each new JPL mission. This paper discusses SEQ_REVIEW, a component of AMMOS that was designed to facilitate and automate the task of reviewing and checking spacecraft sequences.

SEQ_REVIEW is a smart browser for inspecting files created by other sequence generation tools in the AMMOS system. It can parse sequence-related files according to a computer-readable version of a "Software Interface Specification" (SIS), which is a standard document for defining file formats. It lets users display one or several linked files and check simple constraints using a Basic-like "Little Language".

SEQ_REVIEW represents the first application of the Quality Function Deployment (QFD) method to sequence software development at JPL. The paper will show how the requirements for SEQ_REVIEW were defined and converted into a design based on object-oriented principles. The process starts with interviews of potential users, a small but diverse group that spans multiple disciplines and "cultures". It continues with the development of QFD matrices that relate product functions and characteristics to user-demanded qualities. These matrices are then turned into a formal Software Requirements Document (SRD). The process concludes with the design phase, in which the CRC (Class, Responsibility, Collaboration) approach was used to convert requirements into a blueprint for the final product.

THE UPLINK PROCESS

The multi-mission environment in which SEQ_REVIEW is intended to operate is fairly complex. This Section introduces the basic elements of the uplink process and explains where SEQ_REVIEW fits in that process.

Sequence Generation

The ultimate goal of the uplink process is to allow ground operations personnel to control the spacecraft by sending it radio signals that the spacecraft can receive, decode and store in its memory. The decoded information usually consists of commands that are to be executed in a precise sequence at specified times. We will refer to these commands as "spacecraft commands", and to a set of such commands sent to the spacecraft as a whole as an "on-board sequence".

Much of the uplink process is concerned with the planning, generation and verification of on-board sequences. This process can involve many people: mission scientists interested in planetary data request new observations;

engineers concerned about the capability, health and safety of the spacecraft issue maintenance requests; mission planners try to accommodate requests into a realistic schedule; sequence engineers translate high-level requests into detailed instructions that will cause the spacecraft to perform the required tasks; and finally, the flight team must check the detailed sequence against all flight rules, possibly including rules that were added at the last minute to compensate for equipment not operating at specification or software bugs aboard the spacecraft.

Analogy with Programming

The process just described resembles that of generating executable code for an ordinary computer, an analogy that will be used extensively in this paper. The spacecraft and its sequence are analogous to a microprocessor and its machine instructions. The process of planning and generating a sequence is similar to the task of designing and implementing software. Just as software engineers would find it impossible to do their job using machine code, sequence engineers find it useful to work not with the on-board sequence itself, but with a human-readable version of it that is similar to an assembly language program.

Of course our analogy between a spacecraft and a microprocessor is not perfect. Modern spacecraft have considerable processing power at their disposal, so that spacecraft commands are usually much more complex than typical microprocessor instructions. This complexity is reflected in the large number of arguments required by many commands. In spite of this, the analogy between spacecraft commands and assembly code remains valid in the sense that spacecraft commands are expressed in a special-purpose language that is hard to understand unless one is familiar with the architecture of the spacecraft.

Translating Requests into Commands: SEQ_GEN

Programming efficiency can be increased dramatically when using a high-level language instead of assembly code. The tool that makes this possible is the compiler, which translates high-level code into assembly code. Sequence engineers also find that programming sequences directly is prohibitively difficult, and that time can be saved by expressing commands as high-level "Requests" instead of low-level "Commands". Something similar to a compiler is now needed to translate the former into the latter. In the AMMOS system, this role is assumed by SEQ_GEN, a program that expands requests into sequences of commands. The figure on the following page shows the similarities between the conventional code development process and the uplink process.

Since SEQ_GEN is a multi-mission tool, it must obtain mission-specific information from external files. This is unlike most compilers, which are hard-coded around the syntax of a specific language. A second difference with compilers is that SEQ_GEN defines and maintains an internal model of the spacecraft. The mission-specific files required by SEQ_GEN therefore need to describe the spacecraft model as well as the basic commands and their effect on the model. Other mission-specific files used by SEQ_GEN define high-level "activity types", which are analogous to subroutines, and flight rules, which are formulated in terms of the spacecraft model (see Ref. 1 for more details on the operation of SEQ_GEN).

SEQ_GEN generates two basic output files. The first file is the Spacecraft Sequence File, which is an ASCII representation of the actual on-board sequence. This file is an input to another program, SEQ_TRAN, which converts ASCII mnemonics into binary code, links the program, and performs necessary memory management and packetization tasks. The second file is the Predicted Event File (PEF), which shows in time-ordered fashion the complete sequence of commands, ground events, and optionally the status of the internal spacecraft model that is predicted to result from the Request File. In the following, we focus on the PEF.

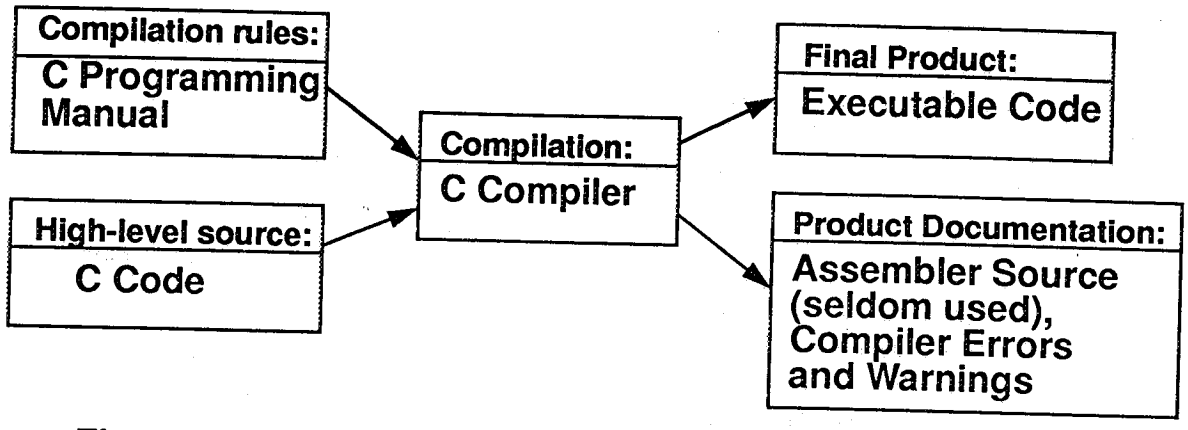

Fig. 1: SOFTWARE DEVELOPMENT PROCESS

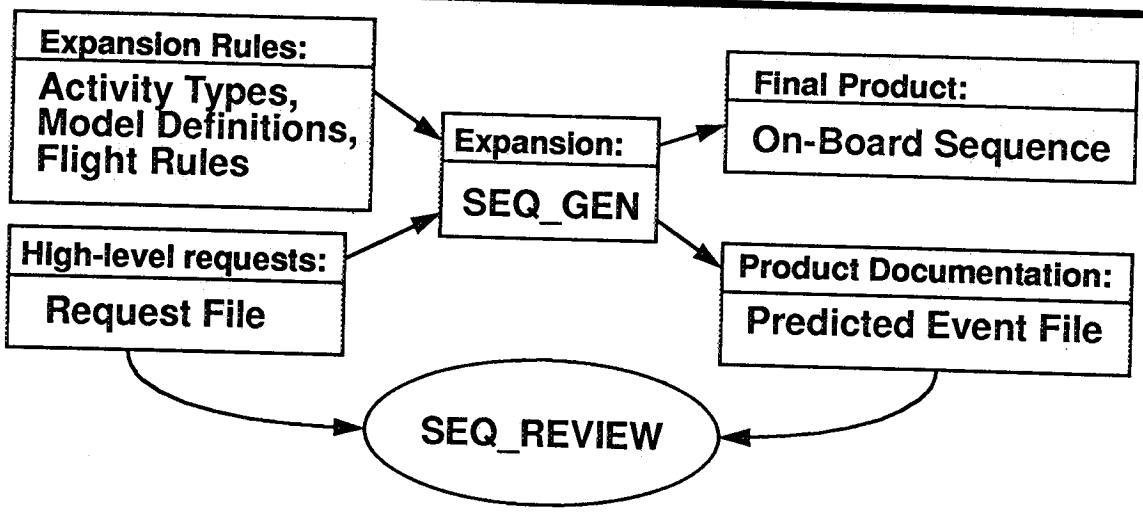

Fig. 2: SEQUENCE GENERATION PROCESS

Checking the Sequence: SEQ_REVIEW

Testing conventional software is a straightforward procedure: the worst that can happen is that the program "crashes" under the benevolent supervision of the operating system. In space exploration, however, sequence engineers do not have the luxury of trying again: the sequence HAS to work the first time. Simulation tools such as those incorporated into SEQ_GEN provide valuable help in validating sequences. However, the final arbiter of a sequence's validity is the sequence engineer and other flight team members who review it.

The main difficulty in checking a sequence is to zero in on the information that is pertinent to a single flight rule or constraint. The documentation provided by SEQ_GEN in the form of an event file is quite extensive, but that makes it hard to read. Traditionally, sequence checkers have used a variety of ad hoc methods to deal with this complexity:

- manual inspection of computer printouts
- BASIC and C programs that "strip" event files of unwanted information
- UNIX "awk" scripts for reformatting event files

The purpose of SEQ_REVIEW is to offer sequence engineers and other sequence reviewers an alternative, multi-mission package that is easy to use, adapt, port and maintain.

THE REQUIREMENTS PHASE

The SEQ_REVIEW software requirements document (SRD) was based on the TQM tool Quality Function Deployment (QFD), which we briefly outline here. A more detailed account of our QFD approach will be found in Ref. 2.

The QFD Approach

The emphasis in the QFD approach is on customer requirements and on how to ensure that these requirements are reflected, i. e., "deployed", through the design process. The first step in the process as implemented here was to collect information from potential users of the software through interviews. Responses to the interviews were then analyzed by a Committee with representatives from user, developer and systems engineering groups. The primary goal of this first step was to come up with three basic lists:

- Demanded Qualities, which express what the user wants to be able to do with the program. Example: easy to strip and reformat a PEF. All of these Qualities were taken from user responses. The Committee organized them 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".
- Quality Characteristics, which express in a quantitative manner how users and developers would rate the SEQ_REVIEW product against other methods for achieving the same task. Examples: "check one constraint in at most 5 lines of SEQ_REVIEW 'Little Language' code"; "keep the program to 18,000 or fewer lines of code".
- Functions. These are program features which will allow the product to meet customer requirements. Most of these were requested by users directly ("Perform time conversion on request"); a few were provided by developers.

A questionnaire was then circulated, asking users to rank the Demanded Qualities in order of importance. The responses were used to compute an average score for each one of the Demanded Qualities. Listed at the top were:

- "Easy to Strip and Reformat a File"
- "Draw Timelines"
- "Reduce the Amount to Read"
- "Allow Annotations"
- "Work with Event Files"

Some of the least important Qualities were "Correlate Event and Request Files" and "Work with Spacecraft Sequence Files." Clearly, our users were mostly interested in making event files easier to read.

In the next step of our QFD implementation, these user-assigned priorities were propagated through a set of "correlation matrices" that relate the users' Demanded Qualities to factors that the developers can influence through their design, primarily Quality Characteristics and Functions. These matrices specify whether for any given Demanded Quality/ Quality Characteristic or Demanded Quality/ Function pair, the correlation between the two members of the pair is (i) nonexistent, (ii) weak, (iii) moderate or (iv) strong.

Based on these matrices, we used a QFD software package to compute scores for each Quality Characteristic and for each Function. These scores were then used to prioritize the development process as well as the overall objectives for the product. The highest-priority items were

- provide users with a rule definition language (the "Little Language")
- provide a graphic interface that lets users specify a rule in under 5 minutes
- design the "Little Language" so that users can formulate a rule in 5 statements or less
- adapt existing timeline generation software from other programs (such as SEQ_GEN)

Some of the less important characteristics were "Ability to add a feature in one week or less", and "Keep the code to 18,000 lines or less". Clearly, the emphasis was on providing users with simple ways to express rules and on providing timeline capabilities without re-inventing the wheel.

Generating Requirements

Since the QFD methodology does not prescribe a specific method for generating requirement documents, we had to come up with our own. Our first attempt consisted in translating the correlation matrices for Functions and Quality Characteristics into plain English. The Functions were used primarily to explain the method used to meet the requirements, while the Quality Characteristics were used primarily to state testable objectives for the finished product.

This first approach was rejected because the resulting requirements document was 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 SEQ_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:

(i) design a tentative Graphical User Interface (GUI). This would give us a chance to organize user-demanded features in a logical manner. It was also decided to implement this preliminary design in Visual BASIC and make it available to potential users for feedback.

(ii) show a concrete example of a Little Language (LL) and explain how it relates to the desired functionality of SEQ_REVIEW. This step actually required little effort since a LL was already developed as part of the prototyping effort (see the next Section). While this LL didn't meet all the requirements, it is close enough to provide the reader with a sense of how the product would operate.

While these tasks delayed the SEQ_REVIEW SRD somewhat, we felt that the overall schedule would not be adversely impacted. First, additional up-front work would make design and implementation easier later on. Second, our tentative GUI could be turned very easily into the first Section of the SEQ_REVIEW User's Guide, again saving us time later on. Finally, we felt that making our tentative GUI available to users early on would contribute significantly to the ultimate success of SEQ_REVIEW.

The software requirements for SEQ_REVIEW were strongly influenced by two parallel efforts that took place in the summer of 1993.

Prototyping Activity

First, prototypes were built to demonstrate the feasibility of SEQ_REVIEW. These prototypes established a firm basis for the following concepts:

1. zero in on useful information by letting the user specify patterns and searches in a simple, intuitive way
2. translate sequence files into text files suitable for input into spreadsheet programs such as Lotus 1-2-3
3. express rules and constraints easily by writing simple programs in a Little Language designed to handle the type of information found in sequence files
4. reformat sequence files by letting users specify records of interest and fields of interest within these records, using either simple pattern definitions or the Little Language
5. build on previous experience by saving search patterns and simple algorithms so they can be reused in future review sessions
6. allow the program to read arbitrary (within reason) text files by specifying the file format on-line, as opposed to re-

compiling a new version of the software featuring new hard-coded file formats

Second, a Quality Function Deployment (QFD) Committee was formed. This Committee included representatives from potential users of SEQ_REVIEW as well as software developers. The Committee used the QFD methodology to identify desired features and qualities that the SEQ_REVIEW product should exhibit. How this work was used to establish the present requirements was described in the previous paragraphs.

User Interface

Since the primary purpose of SEQ_REVIEW is to display sequence file information to the user, it is anticipated that most users will want to interact with the program through a Graphical User Interface (GUI) similar to that used by many text editors. This should be qualified in two ways:

- a small but significant minority of potential SEQ_REVIEW users requested the ability to control the program through a command-line interface, as opposed to clicking on buttons and pull-down menus;
- SEQ_REVIEW needs to support "batch-mode" operation, in which a pre-defined set of commands is fed to the program from a command file. In this mode, SEQ_REVIEW acts as a "filter", e. g. to identify violations of rules not yet implemented in SEQ_GEN.

To accommodate these requirements, SEQ_REVIEW will be provided in two forms: interactive and batch. The interactive version will be GUI-based. In addition to the usual menu bar and push-button, the GUI will feature a special window for command-line input. Every SEQ_REVIEW function will be accessible as a command line as well as through menu selections. "Menu accelerators" will also be provided; these are short, user-definable keystroke combinations that can be used as a substitute for menu selections.

The batch version of SEQ_REVIEW will not display anything to the user and will accept commands from "standard input", which can be either the user's keyboard or a text file specified to UNIX as a source of redirected input. The only use of the batch mode version will be to create output (text) files that can be read by the user or scanned automatically to detect rule violations. It is anticipated that this version of SEQ_REVIEW will be used in highly automated, Operations-type throughput-critical environments.

The figure on the next page shows our preliminary design for a top-level menu of SEQ_REVIEW that satisfies user-demanded qualities and functions. When the user first activates the program, only the top (highlighted) line of each menu is visible; these lines form the "Menu Bar" at the top of the SEQ_REVIEW screen. The expanded menus shown in the figure appear when the user clicks on the corresponding menu title in the Menu Bar.

THE DESIGN PHASE

The method used to design SEQ_REVIEW is essentially the Class/Responsibility/Collaboration (CRC) approach described by Wirfs-Brock et al. (Ref. 3), with the following modifications/adaptations:

(M1)the starting point of the design is the SRD, which concentrates almost exclusively on the user's perspective of the program. The requirements do not address how the program is supposed to accomplish the various tasks.

(M2)SEQ_REVIEW will rely on the MOTIF toolkit for all graphics. Because MOTIF has its own class definitions, there is potential conflict with internal SEQ_REVIEW classes. This problem is not really discussed in Ref. 3.

(M3)a specific methodology was adopted early on to deal with the fact that SEQ_REVIEW needs to be delivered in two flavors, GUI and batch. The decision was that the two programs would share the same object structure, and that MOTIF, X Toolkit and X Window calls

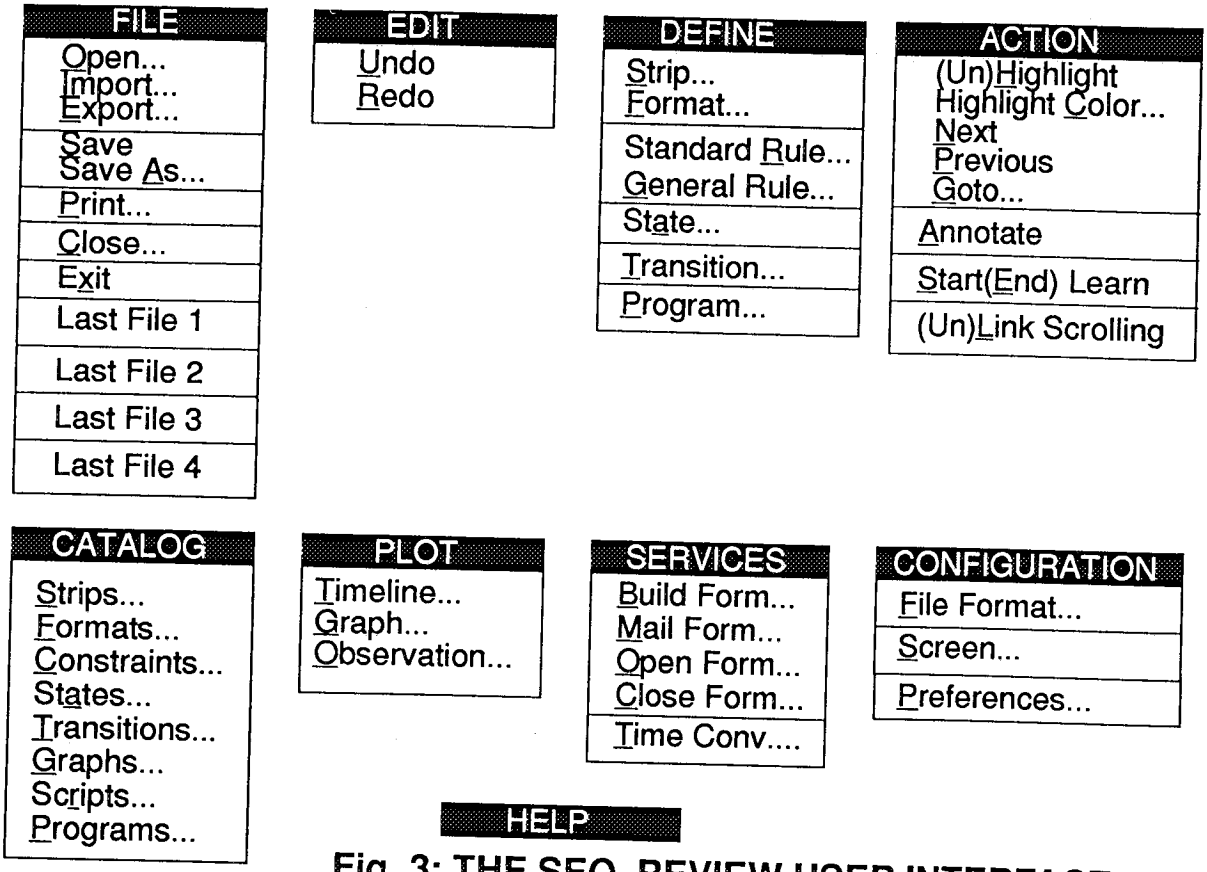

Fig. 3: THE SEQ_REVIEW USER INTERFACE

would be "dummied up" in the batch version.

(M4)we decided to use a fair amount of "vertical inheritance" in our design, as opposed to the Wirfs-Brock strategy which emphasizes "horizontal inheritance".

The starting point of our design was an index of keywords extracted from the SRD. The index was then edited into a table of "SRD objects", to be used as a first step towards designing the classes of SEQ_REVIEW.

As a result of (M1), however, 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 write scenarios that went beyond the user interface. We then decided to use the scenarios as a source of objects, rather than as 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 scenarios as simple, "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" program, featuring all these 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. We learned the following lessons:

- Our design is compatible with the C++ compilers we are using.
- 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. The need for this was made more pressing by the requirement phrased in (M2) and (M3) above: we need a clear description of how MOTIF is to be interfaced to the rest of the system.

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 was still only a shell, it was able to print in indented, scenario-like style what it was doing. It also provided a rudimentary user interface which demonstrated how the menu structure and the callback philosophy of the GUI version could be brought into the batch version of SEQ_REVIEW.

The second of these prototypes consisted of a MOTIF implementation of the "Define Strip" panel of the SEQ_REVIEW user interface. This is probably the most complex graphic object in the GUI. The prototype therefore demonstrated the feasibility of our approach and helped focus the discussion of how the GUI and batch versions of SEQ_REVIEW would coexist.

As a result of all this prototyping 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.

CONCLUSION

SEQ_REVIEW is a tool that will facilitate the task of reviewing the various text files associated with spacecraft sequences. The requirements for SEQ_REVIEW were derived from interviews of potential customers. These interviews were converted into a requirements document using the QFD approach. Requirements were then translated into a high-level design using an object-oriented methodology. The overall process was facilitated by the use of numerous prototypes. Multi-mission aspects were built into the requirements from the start.

ACKNOWLEDGMENTS

We extend our thanks to our many colleagues who contributed their time and insight, and in particular to the flight team members who helped phrase the requirements for SEQ_REVIEW. Special thanks to Todd Bayer, Vickere Blackwell, Carlos Carrion, Julia Henricks, Tim Kaufman, Bob Kerr, Chuck Klose, Bill Nelson, Brian Paczkowski, Steve Peters and Bruce Waggoner for their thoughtful comments, and to Jose Salcedo for sharing his knowledge of the sequence generation process.

SEQ_REVIEW is currently under development at the Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration.

REFERENCES

1. Salcedo, Jose, & Starbird, Thomas (1994, Nov.). *SEQ_GEN: A Comprehensive Multi-mission Sequencing System*, Space Ops '94 (These Proceedings).

2. ELBoushi, M. , Zawacki, S., & Domb., E. (1994, June). *Towards Better Object Oriented Software Designs With Quality Function Deployment*, Transactions from The Sixth Symposium on Quality Function Deployment, Novi, Michigan.

3. Wirfs-Brock, Rebecca, Wilkerson, Brian, & Wiener, Lauren (1990). *Designing Object-Oriented Software*, Prentice-Hall.