SOFTWARE FOR THE RESCUE OF NON-NOMINAL MISSIONS

Guy Janin

European Space Operations Centre
Darmstadt, Federal Republic of Germany

ABSTRACT

A software system for mission planning of emergency situations occuring during partial mission failures is presented here.

It will be shown that by properly designing software for routine mission analysis and preventing the build-up of mammoth programs, such software will also be ideal for contingency analysis.

RESUME

On présente un logiciel pour l'analyse de mission dans le cas d'une situation d'urgence se produisant lors d'un échec partiel de la mission.

On montre que, par une conception judicieuse du logiciel pour l'analyse de mission de routine et en prévenant la création de programmes-mammouth, un tel logiciel est aussi idéal pour l'analyse urgente de cas imprévus.

Keywords: Mission analysis, Software Design, Contingency situation, Emergency operations.

1. INTRODUCTION

Several times in the history of spaceflight a partial malfunction of one of the rocket stages sent the spacecraft into a non-nominal orbit. If subsequent stages were still to be ignited, the mission planning had to be revised in order to cope with this unforeseen situation.

A set of alternate final orbits had to be instantaneously estimated according to the available possibilities and taking into account, at least partially, the original mission requirements.

Such a set of possible final orbits were then proposed to the mission experts who made a choice and initiated the corresponding operations.

Such a mission analysis has to be done fast: in a few hours or in the best case in a few days.

How to be prepared for it?

First idea: to foresee all possible failures and to prepare corresponding software and procedures.

Unfortunately in todays situation of severe budgetary restrictions, there is nothing left in a project budget for such considerations. In addition it is difficult to foresee exhaustively all possible failures.

Alternate idea: nothing particular is foreseen but a general mission analysis software is available, which can be quickly adapted to any new situation.

The realization of such a software is feasible if the following software design concepts are applied:

1) the structure of the program closely follows the structure of the problem to be solved;

2) the structure of the program is such that a more general problem could be solved;

3) the program represents only the structure of the problem, it does not contain any functional modules, it is just a skeleton program calling dummy modules;

4) all functional modules are stored in a library of modules;

5) for each particular application, a particular version of the program is built by replacing dummy modules by appropriate functional modules in the skeleton program;

6) functional modules having the same function have the same name and calling sequence, they are interchangeable.

The advantage of applying these concepts, namely:

- simplicity and transparence of software structure,
- tailor-made program for each particular application,
- programs easy to modify,
- programs easy to extend,
- small core storage during execution,

Proc. Int. Symp. Spacecraft Flight Dynamics, Darmstadt, FRG, 18-22 May 1981 (ESA SP-160, Aug. 1981)

are particularly precious in the context of contingency mission planning.

Sections 2 to 5 will be dedicated to the problem of large software systems and how to prevent the development of mammoth programs. It will be shown that the software design concepts introduced here on a general level are particularly adequate for rescue missions. This will be discussed in section 6.

2. HOW A PROGRAM BECOMES A MAMMOTH PROGRAM

A program, once delivered to the user, is usually not kept unchanged. While using the program, errors and deficiencies are discovered which need corrections, adjustments and modifications. Later on, new tasks appear which require the extension of the program.

The users implementing the modifications and extensions of the program are usually not the same persons as the ones who designed and wrote the program. The original programming philosophy may not be understood and respected. Due to lack of time and often lack of scruples the modifications and extensions are often not documented.

After such a treatment, the program loses its initial structure and becomes more and more complicated and voluminous. Finally it becomes monstruous, a so-called mammoth program.

Everybody dealing with software has been confronted with such a situation. Can it be prevented?

The traditional way of dealing with such a situation is by introducing a tough management which strictly controls the work of the designers, programmers and users. However, such a heavy management frame applied to the working level makes work unpleasant and inefficient. It is a poor and expensive remedy which does not solve the actual problem.

3. A NEW TYPE OF SOFTWARE PACKAGE

A large program is usually run by several users for various applications. In order to prevent the situation described in the preceeding section, *the idea of having a large program has to be given up.*

A new concept is proposed here by means of a set of remarks:

1) The task which is associated with the software package has a logical structure. It has to be clearly isolated.

2) Can the structure be generalized in order to cover more general tasks? If yes it should be defined.

3) A computer program corresponding to the structure of the most general task is designed. This program is just a skeleton. It is only a succession of calls to subroutines but it does not contain any functional modules. However, this program could formally run by satisfying the missing entries by dummies.

4) Independently, a collection of functional modules is built. They are stored in a library. Their calling arguments are compatible with the parameters of the skeleton program.

5) If two or more functional modules have the same purpose, they have the same name and the same calling arguments. They are homonymes.

6) If two homonymes are to be used in the same run, they have to be part of different overlay segments in order to prevent ambiguous entries during linking.

7) The overlay structure is therefore not used for reducing core storage as traditionally, but rather the overlay structure represents a part of the original structure of the task. It contributes to a more transparent structure of the software.

For using such a software package for a given application the user does the following:

1) to take the skeleton program;
2) to take from the library of the available functional modules the modules which are relevant for the particular application, and to plug them into the skeleton;
3) to fill all non-used entries by dummies;
4) to run the program.

In this way, the user has a tailor-made program which fits exactly his needs.

More important: This personalized version belongs to one user. He can modify it without degrading the system. He can extend the capabilities of this program according to his needs. If the new extensions are of general interest, the person responsible of the software package may include them in the library so that they will be available to other users. As an additional feature, the personalized program is small in core storage as it contains only the modules necessary for the given task.

Such a concept is successful only if the skeleton program, common to all applications, is sufficiently general for coping with all possible new extensions. Therefore great attention has to be dedicated to the design of the skeleton program. Section 5 will emphasize the importance of respecting the proper hierarchy of the functions.

4. A PREPROCESSOR

The concept developed in the preceding section suffers from a considerable drawback: the user, in order to build his personalized version of the program, needs to be acquainted with the design of the system. This takes considerable effort and discourages some users.

An elegant solution to this problem is to set up a preprocessor. This is an interactive independent program which fills up the skeleton program with functional modules and builds the necessary data file. This is done interactively

by asking questions to the user and offering him the available options. The user chooses the options convenient for the foreseen application and enters the corresponding data.

After the preprocessor run, the user just needs to load the personalized program and to run it. At any moment he can modify the data or replace modules by homonymes.

If the user needs modifications or extensions, he has to pay attention only to a few modules. His effort will be limited to a minimum and his chances of making errors are small.

An extremely nice feature associated with the preprocessor is that it replaces the program documentation. The effort foreseen for documenting the system can be used for designing the preprocessor. In addition, every user prefers to run a preprocessor than to read a program documentation.

5. THE HIERARCHICAL STRUCTURE OF THE SKELETON PROGRAM

When designing the program structure, care should be taken of respecting the natural hierarchy of the various parts of the system. For instance, in all programs some standard algorithms are used: quadrature or root of a function, integration of an equation, etc. These algorithms are available in mathematical libraries. They are coded by mathematicians. In their point of view, they emphasized the algorithms itself. Of secondary importance is the function to which the algorithm is applied. In test samples, the chosen functions are rather simple. As a consequence, *the algorithm calls the function*. In the calling sequence only the interface arguments between the algorithm and the function are present.

But the point of view of the user is different: emphasized is the function, the algorithm being only a tool to be operated on the function. The function is usually the main part of the program and contains many modules (for instance the pertubations evaluation when integrating the differential equations of the motion of a spacecraft). A large number of parameters have to be communicated.

When programming in FORTRAN in the traditional way, these parameters can be transmitted from the main program to the function only via COMMON statements. But COMMON blocks are always a potential source of trouble.

If instead of the traditional way, it is not the algorithm which calls the function but *the function which calls the algorithm*, the relative importance of the modules is reestablished and the problem of transmitting additional parameters does not exist any more.

Such analysis of the proper hierarchy of the system is essential for designing a successful skeleton program.

6. APPLICATION TO A MISSION ANALYSIS SOFTWARE FOR RESCUE MISSION

The concepts proposed in the preceding sections have been applied to a Unified System for Orbit Computation (USOC). This system is used for reference trajectory estimation and in-orbit auxiliary calculations.

Such a system is precious for the rescue of non-nominal missions as it allows quickly verifying orbital stability, eclipse times, visibility from ground stations and other mission constraints for the proposed new orbit. The final choice of the orbit depends on the output of such an analysis. Therefore the software system should be easy to adapt to any particular situation.

If a non-standard situation occurs during a mission due to a failure resulting in a non-nominal behaviour of the satellite, a mammoth program for mission analysis is of little use. One is not able to use it as such and adapting the program to the non-standard situation involves a considerable effort.

USOC, due to its design, is friendly to modifications which can be done without effort in a short time.

This is why USOC, which has been initially designed as an efficient tool for routine mission analysis work, reveals to be not only most appropriate for rescue operations but the most successful approach for the design of a software package for emergency mission planning.

The design of USOC, its method and algorithms is described in Ref. 1 .

7. REFERENCE

1. Janin G 1979, Mission Analysis for Terrestrial Satellites and Planetary Orbiters: Software Design and Algorithm Description, *ESA STM-208*.