COMPUTING SCIENCE AND TECHNOLOGY INTERNATIONAL JOURNAL, VOL. 1, NO. 2, DECEMBER 2011

Experiences and Best Practice Requirements Engineering for Small Satellites

Sergio Montenegro, Frank Dannemann

I.

Abstract—**The design and implementation of a satellite mission is divided into several different phases. Parallel to these phases an evolution of requirements will take place. Because so many people in different locations and from different background have to work in different subsystems concurrently the ideas and concepts of different subsystems and different locations will diverge. We have to bring them together again. To do this we introduce synchronization points. We bring representatives from all subsystems and all location in a Concurrent Engineering Facility (CEF) room together. Between CEF sessions the different subsystems will diverge again, but each time the diversion will be smaller. Our subjective experience from test projects says this CEF sessions are most effective in the first phases of the development, from Requirements engineering until first coarse design. After Design and the concepts are fix, the developers are going to implementation and the concept divergences will be much smaller, therefore the CEF sessions are not a very big help any more.**

Index Terms—**CEF, Concurrent Design Facility, Requirements management, Space Missions phases.**

II. Introduction

THE design and implementation of a satellite mission is divided into several different phases, normally ranging from phase A (feasibility) to phase E (utilization). Parallel to these phases an evolution of requirements will take place starting with the user requirements, going to the system requirements (whole satellite) and deriving the sub system requirements from the system requirements. The whole development process has to be traceable back to requirements until the system engineers can achieve the final state, which resides in the actual specification of the system. Therefore, the implementation of a satellite can be defined as a continuous process with the requirements at its center.

At one point in the development the system requirements will be decomposed in requirements for the different subsystems, like for example Payload requirements, Software requirements (which are decomposed in requirements for about 10 different applications), hardware requirements, power, communications (downlink and uplink), thermal, structural, accommodation, commanding, telemetry, FDIR (Fault detection Isolation and Recovery) and ground segment. Then each subsystem will manage its requirements almost independently from each other. To ensure all subsystems will be compatible with each other, like the subsystem requirements and subsystems with system we have to add synchronization points in the development in order to converge requirements. Between this synchronization points, the subsystems will begin to diverge from each other. In the first steps of design this diversion may be very big (experience value), but in the development process it shall become smaller and smaller until no synchronization points are required any more (theoretical, in reality we need them until the end of the development).

Another problems faced is the handling of the requirements, which are edited mainly with WYSIWYG word processors (such as MS Word or OpenOffice). During the process of manually copying the actual requirements (or references to them) between documents (e.g. from the functional to the technical specification) a lot of errors may arise while attempting to achieve backwards traceability, if not in the initial stages of requirement creation or tracing, then certainly in the later project phases when the possibility exists of and increase in the number of inconsistencies. Such critical points in time can occur after a review process like a detailed/critical/design review, or after major changes in one of the subsystems of the satellite.

In this paper we present the approach which we are using for both the TET [1] and AsteroidFinder [2] missions, where the design of the satellite missions is driven by a tool-based requirements management process. Using DOORS [3] as the tool of choice, all requirements will be stored in a central database which is accessible by all project partners regardless of their location. Using features such as baseline management and change requests the software supports the typical development phases of a satellite in an ideal way, also making the time-consuming process of requirements reviewing and RID-processing a lot easier. The usage of the web interface and e-mail notification completes the projects' goal of allowing internal and external users to stay informed of all changes in the satellite's database.

III. Requirements Network

Manuscript received December 2, 2011. This publication was funded by the German Research Foundation (DFG) and the University of Wuerzburg in the funding programme Open Access Publishing.

S. Montenegro is Professor for aerospace informatics at the university of Würzburg, Germany. Email: sergio.montenegro@uni-wuerzburg.de

F. Dannemman is Team leader at the DLR (German aerospace Center) in Bremen, Germany. Email: frank.dannemann@dlr.de

The requirements are structured and linked into a network and are intended to be more than just a list of to-do items. This structure clarifies any decisions taken and can be considered as a part of the design justification folder. Following the links the reader can get answers to the hows and whys (e.g. why X-band? How to transmit 80 gigabits per day?), so increasing the understand ability of the requirements and providing a mean for traceability beginning with the most abstract requirements through to the implementation requirements and down to the implementation itself. The technical requirements are linked from goals/abstract/concern requirements, going through some intermediate steps, finishing with the implementation requirements (what the system designer has to consider), like it is shown in the following Figure 1:

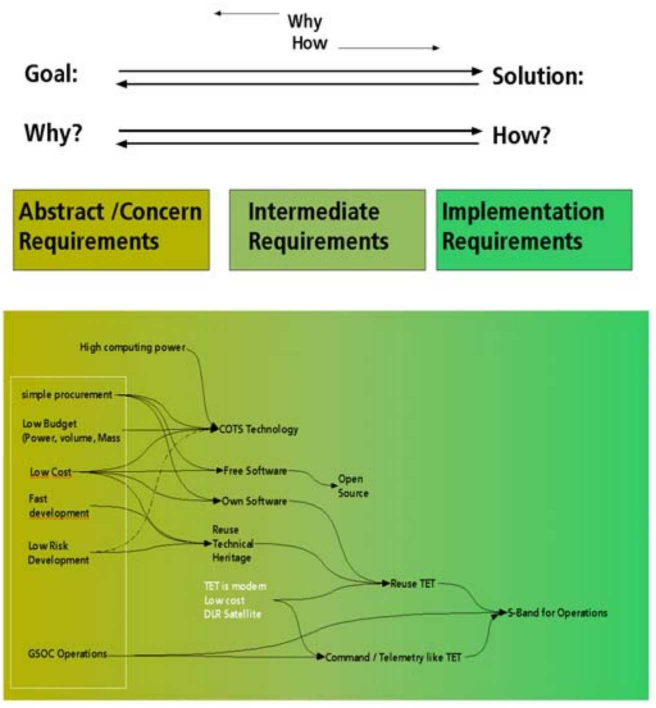

Fig. 1. Network arrangement of requirements (hypothetical example)

The abstract requirements are mainly functional requirements (what to do) and the implementation requirements are mainly structural ones (how to do it). This produces a graphical network of goals and sub-goals. Following the arrows (mostly from left to right) in the network we answer the question how? The opposite direction (right to left) answers the question why?

The overall network of requirements does not have a graphical representation. The picture above is solely to clarify the concept. Nevertheless DOORS provides the possibility to visualize all links between two formal requirements modules. The picture below gives an example of such link graph, showing some of the connections between functional and technical requirements for the AsteroidFinder satellite:

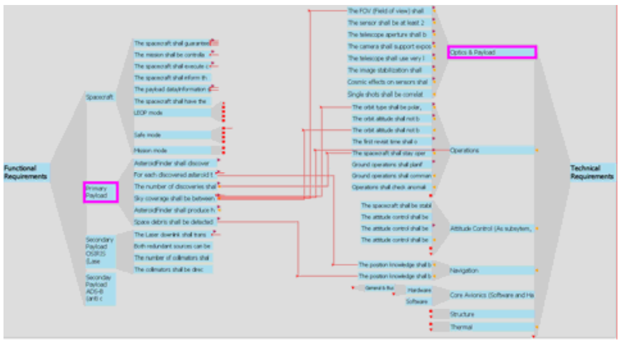

Fig. 2. Link graph between two requirement modules.

A requirement without an output arrow implies that it has no further consequence and it can be considered as a final requirement. Many output arrows mean this requirement has many implications to the system. Many input arrows mean this requirement is a solution for many other implications. Only one input arrow would be enough to justify this decision.

A similar linkage of requirements can be done using public domain tools, like (tables from) OpenOffice to. From OpenOffice we cannot get this graphical structure, but in the practice it is not required because the navigation is done following the (hyper-)links.

IV. Requirement Types

Not all requirements are the same. They are classified into different groups which are interconnected into a network as shown in the following figure:

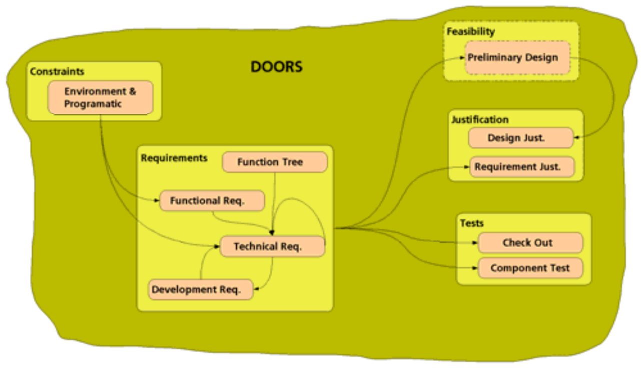

Fig. 3. Types of Requirements

As shown in Figure 3, we can see that not only requirement modules influence the requirement graph. Modules containing constraints also have an important impact, like for example: “Space is cool” or “Reuse TET”. These constraints are facts we cannot change and have to accept. Constraints are grouped into environment constraints (e.g. “Space is cool”) and programmatic constraints (e.g. “Reuse TET”). Constraints influence directly the functional requirements (e.g. “Find at least 10 asteroids!") and the technical requirements (e.g. how to survive in space).

There is one additional requirement module which has no input, but a lot of output connections: the Function Tree is based on the experiences of past satellite missions and contains a list of typical functionality which has to be implemented in order to operate the satellite and to use the payload. This Functional Tree has implications on technical requirements which describe how to implement this functionality.

V. Synchronization Points

Because so many people in different locations and from different background (computer scientists, electronic engineers, mechanical engineers, space engineers, power, thermal, communication, operations and many other people and even a manager) have to work in different subsystems concurrently it is a natural process that the ideas and concepts of different subsystems and different locations will diverge. We have to bring them together again. To do this we introduce synchronization points like in following figure:

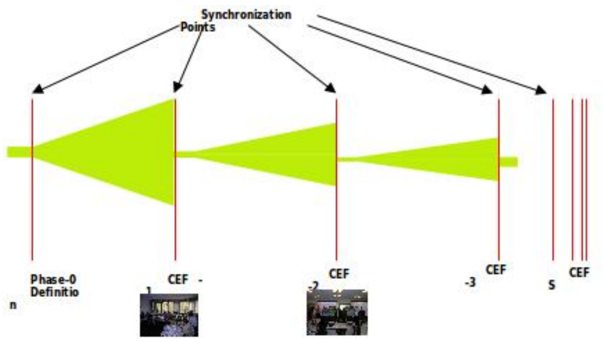

Fig. **4**: Synchronization points

For the synchronization points we bring representatives from all subsystems and all location in a Concurrent Engineering Facility (CEF) room together like in the following figure.

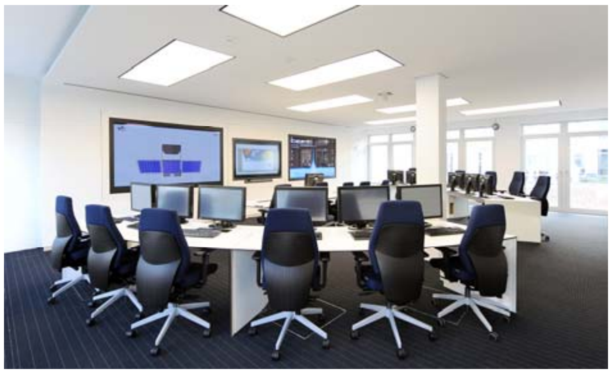

Fig. **5**: Concurrent Engineering Facility at DLR - Bremen

The CEF infrastructure allows and efficient and simple data interchange between different workstations which are occupied by representatives from different subsystems. For example when working out the power dimensioning of the system, each subsystem distributes its power requirements. Adding these values we can get an image of the system and all together search for better solutions and compromises in a very short time and effective way.

Between CEF sessions the different subsystems will diverge again, but each time the diversion will be smaller, as seen in figure 4. Our subjective experience from test projects says this CEF sessions are most effective in the first phases of the development, from Requirements engineering until first coarse design. After Design, going to implementation they are not a very big help any more. Therefore we recommend to have more meetings at the begin of the conception phase (every 4 weeks). In the same mass as the concept is being fixed, the distance between sessions (meetings) may be extended. After the concept is fix, the developers will "just" implement them and we expect only minor changes. CEF sessions will not be a big help any more.

References

[1] TET
http://www.dlr.de/rd/desktopdefault.aspx/tabid-2274//3396_read-5085/

[2] 2. Mottola, S., Börner, A., Grundmann, J.T. , Hahn, G., Kazeminejad B., Kührt, E., Michaelis, H., Montenegro, S., Schmitz, N., Spietz P.: AsteroidFinder: Unveiling the Population of Inner Earth Objects, 59th International aeronautical congress, 29th September to 3th October 2008, Glasgow, Scotland.

[3] Doors: http://www.telelogic.com/products/doors

[4] Bärwald W. and Montenegro S., BIRD-Spacecraft bus controller, Small Satellites for Earth Observation, Vol. 3, 371-373, 2001

[5] Sergio Montenegro, Jan-Thimo Grundmann, Bobby Kazeminejad, Peter Spietz, The new DLR Standard Satellite Bus series (SSB), Small Satellites Systems and Services - The 4S Symposium, German Aerospace Center, 2008

[6] BIRD (Bispectral InfraRed-Detector
http://www.dlr.de/rb/en/desktopdefault.aspx/tabid-2731/6724_read-6311