

Requirements management as a matter of communication

Ingmar Ogren

Tofs corporation, Fridhem 2, SE-76040 Veddoe, Sweden

URL: <http://www.toolforsystems.com>, E-mail: iog@toolforsystems.com

Published in INCOSE Brighton. Accepted for publishing by Crosstalk (US Air Force) and Militärteknisk tidskrift.

Abstract. Requirements work all started as a dialogue between a vendor and an end user. Nowadays this dialogue has been complicated through introduction of marketing and acquisition personnel. This has led to introduction of requirements specifications, but the need for a basic dialogue is still there since a specification cannot capture the increase of knowledge that will always take place during development.

An efficient dialogue must not only include requirements, but also stated missions, problem management and understandable formalism for the system's structure and behavior.

Further study of the dialogue shows that requirements management must be managed as a process in parallel with processes for development and verification.

INTRODUCTION

Once there was a buyer (end-user) who asked a producer: "Can you sell me this and that and what is your price?"

Alternatively the producer could ask a buyer: "Do you want to buy this and that at this price?"

This was a long time ago, before the days of complex systems, when both buyers and producers understood the meaning and the use of "this and that".

To-day, with complex systems, the situation has become somewhat more complicated:

- The buyer is not always the end-user since the end-user needs to take help from acquirers and contract people, who may understand more about bargaining and contract issues than about the end-user's real needs
- The seller is not always the producer since the producer needs to take help from marketing and sales people, who may understand more about marketing than about the product
- Nobody involved really understands "this and that", because the product sold is a new complex system, which was never seen until it is delivered.

Still the basic questions, cited above, are still asked but the complicated situation necessarily makes the answers somewhat misty, leading to a situation where we need requirements specifications and requirements management.

TODAY'S SITUATION

Today we have gotten into a situation where all too often a system project may run like this:

- The end-user and the acquirer put together a requirement specification. They may take help from one or several consultants to make it really complete and correct. The result is a specification, which requires considerable work just to be read and understood.
- The acquirer asks for proposals, based on the specification.
- Several vendors look into the specifications, with various reactions, such as:
 - We can do it, but we do not have the time to analyze these requirements until we get the contract, propose 40 M\$.
 - This looks interesting, but these requirements need to be analyzed, propose 3 M\$ to analyze the requirements and to build a first model of the system.
 - We do not understand these requirements, but we desperately need business, propose 20M\$, with 30% in advance.
- The acquirer interprets the acquisition regulations and gives the contract to the third vendor since this was the one with the best price.
- A couple of years pass and the vendor tries to understand the requirements and to build a system in accordance with this understanding. During the process of building the understanding, the vendor will find a number of inconsistencies, contradictions and gaps in the requirements.
- The vendor runs out of money and comes back to the acquirer saying: There are some problems with this contract, which need to be sorted out. We need an additional 20M\$ or we cannot manage to complete the contract.
- The acquirer is then left with two alternatives:
 - Not paying extra, not getting any system and possibly causing the vendor to go bankrupt.
 - Paying extra and getting a system, which is probably late and not up to the "real needs"

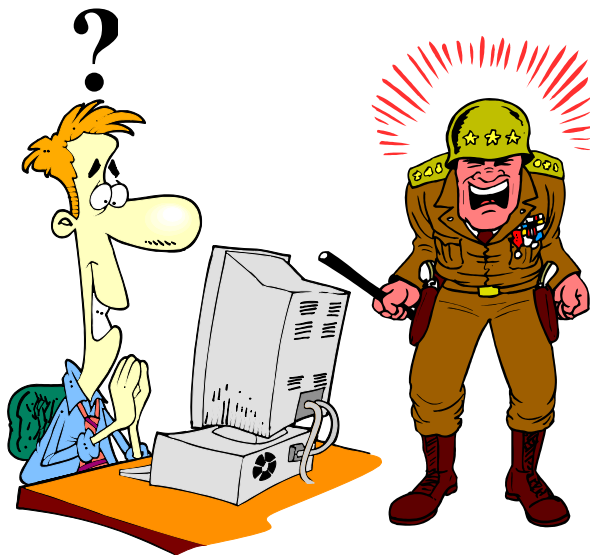
since the end-user's understanding of needs has grown during system development.

THE NEED FOR DIALOGUE

As discussed above there are obviously problems and if you talk to people involved for example in building military software systems, you will get some clear opinions as to the causes of the problems:

The military end-user: "These keyboard monkeys do not understand a thing about military needs. They do not even understand that you cannot put an M317 backwards upon a A32!"

The programmer: "These brass-hats do not understand a thing about their own systems, look here where they try to express accuracy percentage in inches!"



You cannot have a dialogue without mutual understanding

This may be somewhat funny, but as long as the most important players in system production, the end-user and the producer talk about each other instead of to each other, the prognosis for the system to be produced is not very good.

What is needed is dialogue and still more:

- understanding in a language, which is common among everyone involved in a development
- respect for each other's professional competence, meaning that you stop calling other participants in the development funny names and start respecting their professional competence
- Courage in the sense that you dare to speak up whenever someone says or writes something you do not understand.

TAKE CARE OF THE KNOWLEDGE GROWTH

Provided you get the dialogue going between end-users and producers in a complex system development project, it is inevitable that knowledge will grow among those involved. The knowledge growth may reveal fundamental glitches in the original specification and/or in the original proposed solution.

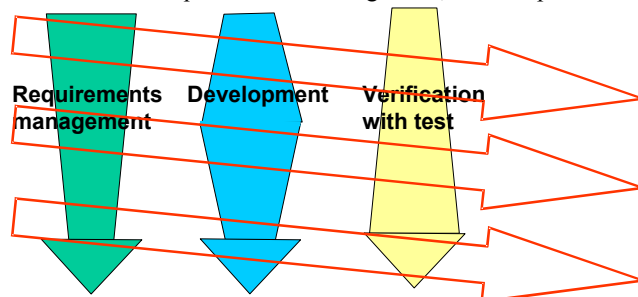
If you then have a fixed contract with negotiated deliveries and payments, the best that can be said about the situation is that it is a real challenge to the contract people, who may not even be able to understand that there is a problem.

What you need is a work principle, which anticipates that knowledge will grow during system development and allows this new knowledge to change the direction and content of the development effort. To find a contract form, which allows this consideration for growth of knowledge is the real challenge for the contract people.

REQUIREMENTS ON THE REQUIREMENTS DIALOGUE

Besides the obvious need for mutual respect, indicated above, some requirements on the dialogue and its language are:

- The dialogue must use a language, which is readily understandable by both end-users and developers after no more than a few hours' training.
- The language used must be sufficiently formal to avoid misunderstandings.
- The language must be able to describe objects to be managed by the system under development.
- The language must be able to describe system performance precisely.
- For critical systems, the dialogue must support discussion of fault-tolerance issues including possible unexpected operator behavior.
- The dialogue must include definition, discussion and decision on the problems, which will surface during development of any non-trivial system.
- The dialogue must anticipate that knowledge will grow during system development and allow this knowledge to influence requirements and design solutions.
- The dialogue must accept as a fact that requirements management, development and



Successive deliveries with parallel processes

verification are three processes which must be run in parallel during a systems engineering effort.

ELEMENTS OF A SOLUTION ALTERNATIVE

As understood from the reasoning above, there is more to requirements management than writing and accepting a requirements specification. In the following some aspects, which should be considered as a basis for the necessary dialogue, are discussed

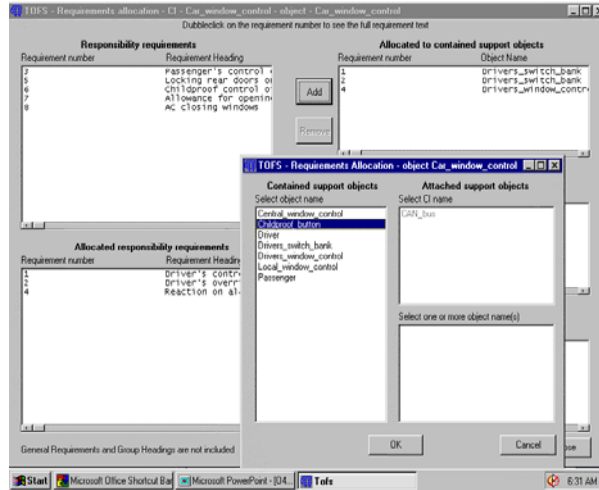
Missions and scenarios To the end-user the missions of a system are often so obvious that he does not even state them. Instead he defines a set of scenarios, which may or may not cover the complete mission space. On the other hand the developer will interpret the specification and create a number of “Use cases”, which also may or may not cover the mission space.

To create understanding it is necessary that the end-user states the missions clearly as these constitute the foundation of both requirements and design work. Scenarios can then be added to clarify the meaning of the missions, but the missions are fundamental.

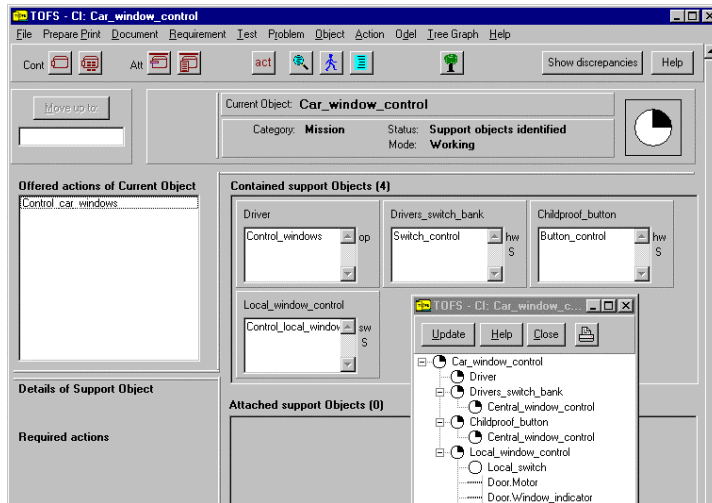
One way to express missions is as “Mission objects”, which are supported by other objects used to complete the mission at hand. See the figure below for the mission to control the windows of an automobile.

insert the requirements through copy/paste from a document or even have the original document automatically parsed for requirements. You can also use common office tools like MS Word or MS Excel to get a low cost requirements database.

It is however strongly recommended to do the requirements insertion manually as you will need to check uniqueness, testability, contradictions etc. These checks will give valuable understanding of any problems, which may pertain the original requirements.



Requirements allocation in a composite structure



The mission to control car windows is supported by support objects (operator, software and hardware)

Original requirements are often stated in a requirements specification. They may also be found in published standards and regulations. For example railway systems are heavily regulated.

To make it possible to work with the original requirements, you need a database. Many system engineering tools offer such a database where you can

possible to “float” requirements downwards through the structure until you find the object, which the requirement should be tested with. The requirement then becomes “fulfillment requirement” of that object.

One tool, that supports such composite structures is Tofs (Tool For Systems). The figure

Derived requirements. Since knowledge grows during development and since design decisions are taken, new requirements will surface. These are called “derived requirements” and they should be input into the requirements database, marked as “derived”.

Allocation through a composite structure. Provided that the system under development is modeled as a composite structure allocation of requirements is simple. A composite structure is a structure where the system is composed from objects, connected through their interfaces in such a way that it is always obvious which objects depend on which other objects to complete their action. The composite structure makes it

above shows the screen used for requirements allocation.

Problem management. Management of problems or issues is a very important part of the dialogue during system evolution, since problems will inevitably surface and since most of these require a combined effort from end-users and developers to find a feasible solution.

This means that the dialogue must include problem management, including:

- problem statements with category and priority
- problem headings and numbers
- solution alternative
- solution decisions.

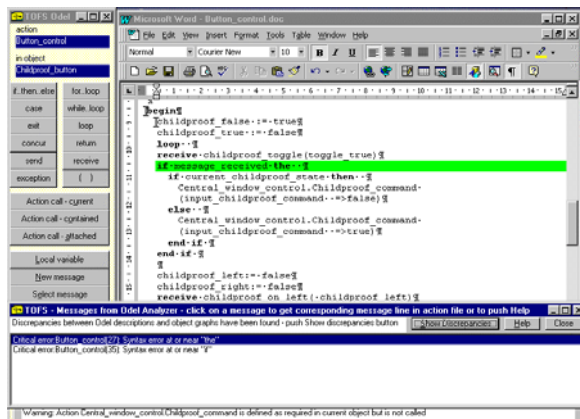
It is possible to manage this with an ordinary word processor, but a tool, which supports at least numbering of the problems, is preferable.

Understandable formalism. An end-user, who has expressed himself in clear English, with some explaining diagrams would normally believe he has made himself completely clear. That will also often be the case, but is amazing how such clear requirements can be transformed into software and hardware, which does quite unanticipated things.

It is doubtful if you should attempt to make the end-user write formal specifications since the necessary knowledge to do so is normally not available at specification writing time.

What could however be done is to provide a formal representation as part of the design effort with two objectives:

- to get a reconfirmation from the end-user that the specification is understood in an acceptable way
- to give a detailed and formal basis for the implementation work, be it hardware, software or an operator role.



Analysis of formal English (Tofs)

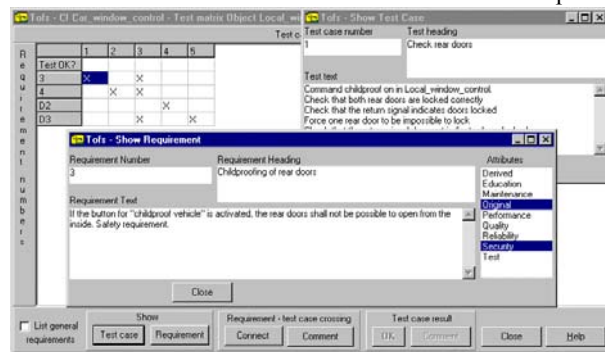
One way to get this formal description is to use “formalized English”, a simplified language containing:

- variables of defined types
- control structures
- comments.

As an example is shown above a description of part of the software in “Car window control”.

Test specifications. Requirements are not much good unless they can be tested. Provided that requirements are distributed to design objects as “fulfillment requirements”, you can design test cases for each object to test that these fulfillment requirements are met.

To ensure that a correct set of test cases has been written, they should be reviewed by the end-user. It will then help if the test cases are presented together with the requirements, which they are intended to verify. One way to do this is to produce a requirements/test case matrix. As an example is shown the matrix for the software object “central window control” in “Car Window control” example:



The test matrix connects requirements/test

CONCLUSIONS

Some aspects of requirements management have been discussed finding that the single most important issue is to establish a dialogue between the end-users and the producers of a system.

It has also been found that the original requirements specification is only a part of the information required for successful requirements management.

The issue of understanding has been discussed, stressing the necessity to make the requirements documentation understandable both to the end-user and to the developers concerned.

Since the dialogue will result in more information than can be comfortably managed manually, computer-based support tools will be helpful.

It has further been discussed the necessity to create a work situation, where everyone involved communicates and respects the professional competence of others in the process.

Another important conclusion is that you cannot "do the requirements" in the beginning of a project, but that you must establish a requirements management process in parallel with processes for development and verification.

THE AUTHOR

Ingmar Ogren was graduated with an M SC in Electronics from the Royal University of Technology in Stockholm in 1966.

He then worked with the Swedish Defense Material Administration and various consulting companies until 1989 with systems engineering tasks in areas such as Communications, Aircraft and Command & Control.

He now chairs the board and is owning partner in two companies:

- Tofs which produces and markets the Tofs (Tool For Systems) software.
- Romet, which consults in the area of systems engineering methods with the method O4S (Objects For Systems) as its main product.

Further information about Ingmar Ogren can be found on the web page <http://www.toolforsystems.com>