Requirements Management as a Matter of Communication

Requirements work started as a dialogue between a vendor and an end user. Today 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.

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.*

Today, with complex systems, the situation has become more complicated:

- The buyer is not always the end user since the end user needs help from acquirers and contractors who understand more about bargaining and contract issues than end users' real needs.
- The seller is not always the producer since the producer needs to take help from marketing and sales people, who 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 not seen until delivery.

The basic questions above are still asked, but the complicated situation makes the answers somewhat hazy, leading to a situation where requirements specifications and management are needed.

Today's Situation

We are in a situation where too often a system project runs like this:

- The end user and the acquirer put together a requirement specification. They get help from one or more consultants to make it 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 proposal of $40 million.
  - This looks interesting, but these requirements need to be analyzed. Advance us $3 million to analyze the requirements and build a first model of the system.
  - We do not understand these require ments, but we desperately need business. We will accept a proposal of $20 million, with 30 percent in advance.
- The acquirer interprets the acquisition regulations and awards the contract to the third vendor as it had the best price.
- A couple of years pass, and the vendor tries to understand the requirements and build a system in accordance with this understanding. During the process of understanding, the vendor will find a number of inconsistencies, contradictions and gaps in the requirements.
- The vendor runs out of money and returns to the acquirer and says, "There are some problems with this contract that need to be sorted out. We need an additional $20 million or we cannot complete it."
- The acquirer is left with two alternatives:
  1. Not paying extra, not getting a system, and possibly causing the vendor to go bankrupt.
  2. Paying extra and getting a system that is probably late and not equal to the true needs since the end users' understanding of needs has grown during system development.

The Need for Dialogue

As previously discussed, there are obvious problems. If you talk to people involved in building military software systems, for example, you will get some clear opinions about what causes 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 backward on an A32!"

The programmer: "These brass hats do not understand a thing about their own systems. They try to express accuracy percentage in inches."

This may seem humorous, 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 . . .

- Understanding—in a language, which is common to everyone involved.
- Respect—for others' professional competence, with no name calling.
- Courage—to speak up whenever someone says or writes something you do not understand.

Take Care of Knowledge Growth

If you start a dialogue between end users and producers in a complex system development project, it is inevitable that knowledge grows among those involved. Knowledge growth may reveal fundamental glitches in the original specification and/or the original proposed solution.

If you have a fixed contract with negotiated deliveries and payments, the best thing about the situation is that it is a real challenge to the contractors, who may not understand there is a problem.

What you need is a work principle, which anticipates that knowledge will grow during system development and that this new knowledge will change the

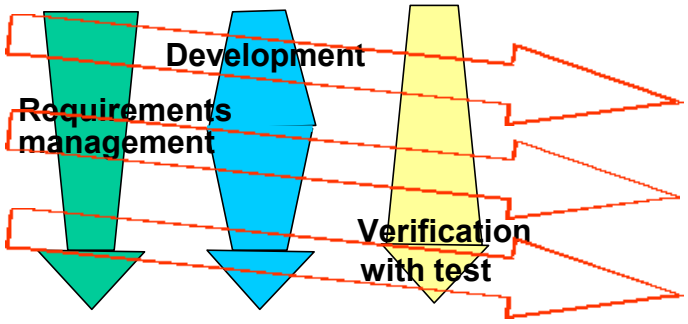
direction and content of the development effort. To find a contract form that considers growth of knowledge is the real challenge for contractors.

Requirements on the Dialogue, Its Language

Besides the obvious need for mutual respect, the dialogue and its language require that:

- The dialogue must use a language that is readily understandable by end users and developers without much training in the language used.
- The language used must be formal in order to avoid misunderstandings.
- The language must be able to describe objects to be managed by the system under development.
- The language must describe system performance precisely.
- For critical systems, the dialogue must support discussion of fault-tolerance issues, including unexpected operator behavior.
- The dialogue must include definition, discussion, and decision on problems, which will surface during development of any nontrivial 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 that requirements management, development, and verification are three parallel processes during a systems engineering effort.

Figure 1: *Successive Deliveries with Parallel Processes*



Elements of a Solution Alternative

As understood from the aforementioned reasoning, there is more to requirements management than writing and accepting a requirements specification. The following are some aspects to be used as the basis for necessary dialogue.

Missions and Scenarios

To the end user, a system's missions are often so obvious that he does not even state them. Instead, he defines a set of scenarios that 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 may or may not cover the mission space.

To create understanding, it is necessary that the end user state the missions clearly as they constitute the foundation of requirements and design. Scenarios can be added to clarify the missions' meaning. 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.

Original requirements are often stated in a requirements specification. They may also be found in published standards and regulations. Railway systems, for example, 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 insert the requirements by copying and pasting from a document or even having the original document parsed automatically for requirements. One can also use common office tools like Word or Excel to get a low-cost requirements database.

However, doing requirements insertion manually is strongly recommended in order to check uniqueness, testability, contradictions, etc. These checks give valuable understanding of any problems pertaining to the original requirements.

Derived Requirements.

As knowledge grows during development and design decisions are taken, new requirements will surface. These are called derived requirements, and should be put in the requirements database as derived.

Compositive Structure Allocation

Provided the system under development is modeled as a compositive structure, allocation of requirements is simple. A compositive structure is where the system is composed from objects and connected through their interfaces in such a way that it is always obvious which objects depend on other objects to complete their action. The compositive structure makes it possible to float requirements downward through the structure until you find the object, which the requirement should be tested with. The requirement becomes the fulfillment requirement of that object.

Problem Management

Management of problems or issues is a very important part of the dialogue during system evolution. Problems inevitably surface; most of them 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 alternatives.
- Solution decisions.

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

Understandable Formalism

An end user who has expressed himself in clear English, with diagrams, would normally believe he has made himself completely clear. That will often be the case, but it is amazing how such clear requirements are transformed into software and hardware that does unanticipated things.

It is doubtful that you would try to make the end user write formal specifications, since the necessary knowledge is normally not available when the specification is written.

You could, however, provide a formal representation as part

of the design effort with the objective to:

1. Get a reconfirmation from the end user that the specification is understood in an acceptable way.
2. Give a detailed and formal basis for the implementation work, be it hardware, software, or an operator role.

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

- Variables of defined types.
- Control structures.
- Comments.

Test Specifications

Requirements are not much good unless they can be tested. If they are distributed to design objects as fulfillment requirements, you can design test cases for each object to test that they are met.

To ensure a correct set of test cases has been written, they should be reviewed by the end user. It is helpful if test cases are presented together with the requirements they are intended to verify. One way to do this is to produce a requirements/test case matrix.

Conclusions

There are several aspects of requirements management. The most important issue is to establish a dialogue between end users and producers of a system.

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

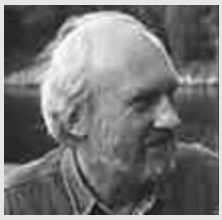
It is a necessity to make requirements documentation understandable both to the end user and to the developers concerned.

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

Furthermore, there is a need to create work situations where everyone involved communicates and respects the professional competence of others involved.

Finally, you cannot do the requirements in the beginning of a project, but you must establish a requirements management process in parallel with processes for development and verification as visualized in Figure 1.

About the Author



Ingmar Ogren graduated with an master's degree in science (electronics) from the Royal University of Technology in Stockholm in 1966; his final project was connecting a fighter aircraft training simulator with a sector operation center. He worked with the Swedish Defense Material Administration and various consulting companies until 1989 in systems engineering areas such as communications, aircraft, and command and control. Ogren chairs the board and holds majority ownership in two companies: Tofs, which produces and markets the Tool For Systems software, and Romet, which provides systems engineering consulting with the Objects For Systems method as its main product.

Ingmar Ogren
Romet Corp.
Fridhem2
SE-76040 Veddoe
Sweden
Internet: www.toolforsystems.com

Cost Estimation Web Sites

http://www.computer.org/tse/ts/s1997/e0485abs.htm

This site connects to a paper written by members of the Institute of Electrical and Electronics Engineers. The authors propose the use of a model they say considerably improves early prediction over the Putnam model An analytic proof of the model's improved performance also is demonstrated on simulated data.

http://www.jsc.nasa.gov/bu2/PCEHHTML/pcehfc.htm

This online version of NASA's second edition of the Parametric Cost Estimating Handbook contains seven chapters, and seven appendices, one of which is the parametric estimating system checklist.

http://www.eng.hmc.edu/courses/E180b/software.htm

This is a listing of professional societies and their links relating to software cost estimation. Among the links are the Software Technology Support Center, Defense Information Systems Agency, NASA Software Engineering Laboratory, and the Software Engineering Institute.

http://renoir.csc.ncsu.edu/SP/HTML/cost.html

The North Carolina State University Software Engineering Resource Center site has a tutorial on software cost estimation, and tools.

http://xanadu.bmth.ac.uk/staff/kphalp/students/bsi/predict/tsld002.h tm

This shows slides on software cost estimation, including newer approaches to Boehm's Constructive Cost Model (COCOMO), such as function points and Estimation by Analogy.

http://www.concentricmc.com/toolsreport/5-4-2tools1.html

This tools list on cost estimation has related links to commercial tools, tools used by government departments, and tool reviews and comparisons.

http://www.cpsc.ucalgary.ca/~adi/621/essasy_outl.htm

This site contains an essay on software size estimation, based on references that include the STSC Metrics Service, Watts Humphrey, Capers Jones, and the International Function Point Users Groups.

http://cse.usc.edu/TechRpts/Papers/usccse98-506/usccse98-506.html

This is a report from the Computer Science Department at the University of Southern California's Center for Software Engineering. The report appendices includes a COCOMO II Cost Estimation Questionnaire and COCOMO II Delphi Questionnaire, Defect Removal Model Behavioral Analysis.

http://irb.cs.tu-berlin.de/~zuse/metrids/History_00.html

This is a history of software measurement by Horst Zuse.

http://cse.usc.edu/COCOMOII/cost_bib.html

This is a general bibliography on cost estimation, updated in November.