



[www.telelogic.com](http://www.telelogic.com)

# The DOORS Dynamic Object Oriented Requirements System

Dr. Richard Stevens, QSS Ltd.  
Tel: (+44) 1865 784285  
Fax: (+44) 1865 784286

## Summary

This paper covers DOORS, a new generation requirements tool. DOORS (dynamic object oriented requirements system) has been designed to meet the "requirements for handling requirements" - i.e. helping users do what they want to do with requirements. Other needs come from the underlying principles for organizing information or presenting it well. For example, configuration management of requirements requires a detailed and specific approach. Requirements information is particularly complex and inter-related and object oriented techniques turn out to be particularly appropriate for handling requirements information. DOORS starts from a hierarchical object oriented data base, and uses modern WSIWYG techniques to display information as if it were a requirements document.

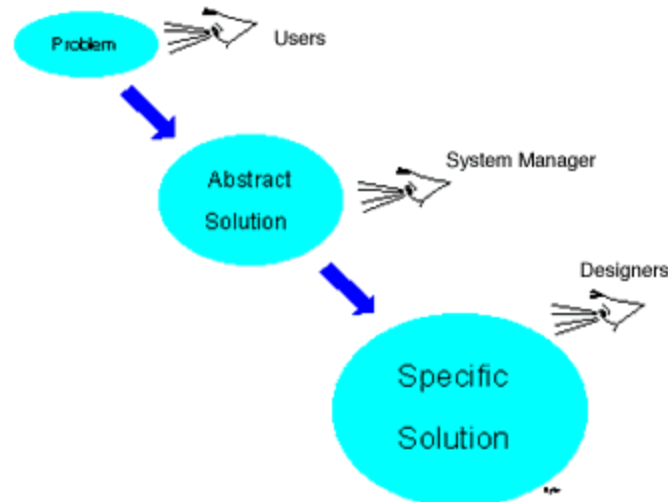
## Background

### Requirements as the Mechanism for Customer Control

Getting requirements right and maintaining them is therefore economically very important, if the customer doesn't know what is wanted and doesn't check what is delivered against the requirements, the project is unlikely to be a success.

Requirements are the ideal means of management, specially suited for handling for large or geographically dispersed systems. They are compact in size, stable, and not highly technical. They state everything we want, early in the project, and let us check we are getting it without needing to understand all details. Carefully organized, requirements allow control at any level of detail at every stage of the project. Requirements define what the user wants, not the system to satisfy those needs.

The potential power of requirements is often wasted, because of the lack of theory, methods, tools, and training for engineers. The methods discussed here are an evolution of traditional approaches, and the discipline of software engineering.



### Three models used during development

Requirements work within the framework of the systems life cycle, of which they form an important part. Requirements define the nature of the problem, and then the requirements imposed on the product. There are two fundamentally different types of requirements involved, and these are best handled as separate models.

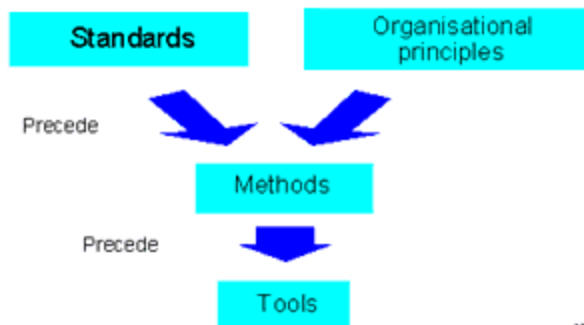
The first type - **user requirements** - cover the problem, i.e. defining the system in terms of what it will provide for the users. The user states the results required plus any constraints needed from a user perspective.

The second model - **systems requirements** - is centered around functionality, but also including internal interfaces, constraints, performance and external systems requirements. This model is very different from the user requirements, but should be traceable to them. The systems requirements provide an abstract, stable definition of the system with a minimum of implementation decisions.

Some requirements apply to the **development** rather than the product to be developed. These requirements need to be organized and managed as separate but related sets. They need to be checked during the development itself. Others apply to the verification system, way the product is to be verified. The verification system should be treated as a separate system built with the objective of ensuring that the product conforms to its requirements and specifications.

### Example Requirements

- Product - "The system shall be able to send scanned images within one minute"
- Development - "The software development shall use the USAF set of CASE tools"
- Verification - "All safety-critical parts shall be independently verified"



**Standards precede methods which precede tools**

### Requirements for Requirements

Once the systems lifecycle has been defined, requirements for handling requirements are derived from:

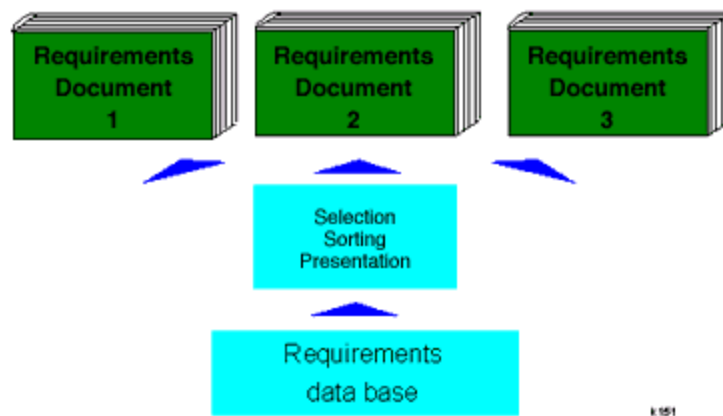
- Uses of requirements during the life cycle;
- Major functions for handling requirements;
- Principles for handling information.

Problem definition and acceptance are the basic uses of requirements. During the life cycle, requirements status also monitors the project progress providing that there are clear relationships between requirements and design. The customer can also use requirements for cost-benefit work or for impact analysis at any time during the project life cycle.

Requirements are not to be gathered and then left unchanged. If we could gather the requirements in a single-shot, then a stable set of requirements could be used as a basis throughout the life cycle. There would really be a "requirements phase", and then we could move over to design. In practice, during the course of the project, requirements are added and deleted. Design options turn out to be too expensive, or easy to implement than imagined. The only rational basis for determining whether or not to make change is by analyzing the costs and benefits of the decision. The benefits (however approximate) are the value attached to the requirements. Unless there is traceability between the design and the requirements, these decisions cannot be made sensibly. Even the simplest of requirements tools manage traceability between requirements and design.

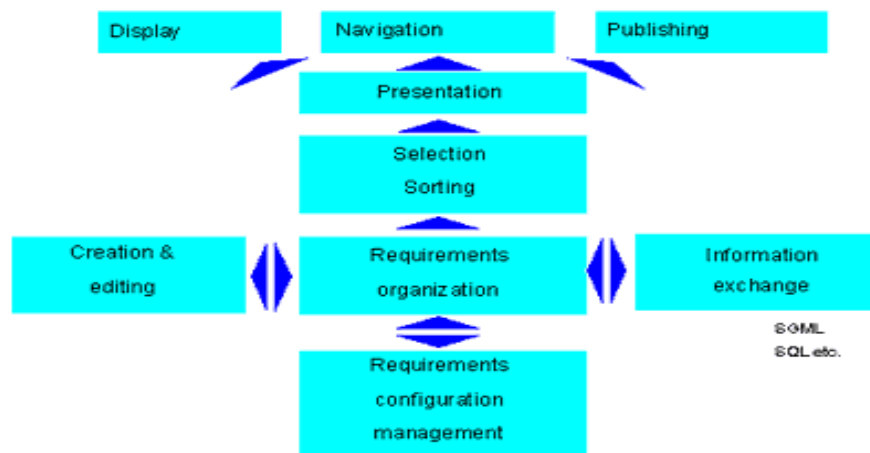
The net effect is that requirements information is highly structured and complex; it must be handled within a data base. Documents are then treated as a partial view of the requirements database. Individuals produce their own document from single, consistent source of information.

### Major Functions for Requirements Management



### Selection, Sorting, and Presentation of Requirements

After the key user requirements have been considered, we can move on to define the major functions (see Figure). These include the organization and configuration management of the data base, the presentation selection and sorting capabilities, the need to publish requirements, to import and export them into other formats, as well as the normal word-processing like functions to create requirements and navigate around the document/data base.



### Major functions for handling requirements

©OML  
©QL etc.

842

## Organizing Information

Requirements information is complex in structure with many different inter-relations. The information objects have many different attributes, as do the links between them. Requirements have to link to all aspects of the project, include completely separate areas such as CAD/CAM. Relational data bases force a complex problem into an unnatural structure and with difficult compromises on the data base. The result is slow and inefficient, and difficult to make a good presentation.

An object oriented data base has significant advantages over a relational database in handling requirements. The information is in the form of a mass of small inter-related requirements objects, where the relationships actually have a status and meaning. The granularity of requirements is inherently more suited to an OO approach, and the complexity of links between objects is impractical in a relational data base. Some key principles of OO rely on human discipline rather than software tools - but the tool must support that discipline.

However, OO needs to be extended in two major ways for requirements data bases. The first is the provision of hierarchical OO systems, to allow users to structure objects to keep them manageable. The second is the provision of multiple hierarchies with the ability to link between them. These two factors allow the user to make a good model of the complete requirements system. Traceability of requirement hierarchies with the ability to link between them. These two factors allow the user to make a good model of the complete requirements system.

## Traceability of requirements and other project information

Requirements are only a small part of project information, but they are related directly or indirectly to all aspects of the project. They need to be linked to other project information for a variety of reasons, e.g.

- **Traceability** to ensure that, for example, the design conforms to the systems requirements.
- To convey **status** such as the current test status against a requirement.
- **Applicability** where a constraint (e.g. safety-criticality) is applied to a function.
- To **replicate** information and thus avoid the need to repeat it.

The links have to have attributes and meaning of their own, and simple hypertext-type links are inadequate. Users will need to make their own links and link attributes.

## Presentation of Requirements

Requirements are a managerial tool, and requirements tools must therefore be usable by managers, not just technicians. This means providing tools as easy to use as a word processor. Early requirements tools, home-made or commercial, were based on relational databases with poor, slow interfaces. In presenting requirements information, we need a good physical metaphor, allied to modern presentation techniques.

The on-screen view used by DOORS is the traditional requirements document. The user sees a document, with attribute columns to show factors such as the absolute number of the requirement, or the method of verification. Interactive navigation, outlining and compression functions allow the user to manipulate the text on screen.

The on-screen presentation must be programmable by the user, who can use selection, sorting, and outlining to minimize the amount of information. Requirements are used by many different people, and often they are interested only in a small subset of the information. The tools should allow them to sort and select requirements and then store the "view" permanently, so that it be re-constituted when the underlying information changes.

## Graphics Views of Requirements

For showing structure and inter-relationships, a graphical view is often better. The presentation needs to be able to switch seamlessly between a graphical and textual view of the same information. Notation for the graphical view should be programmable, allowing engineers to view information in IDEF or any other box-drawing notation with which they are familiar. Users should be able to display complete structures as well as just brother nodes within a structure. Cross-reference matrices have long been used in requirements documents - DOORS replicates these.

## Configuration Management Needs for Requirements

For the requirements set as a whole, the traditional configuration management discipline must apply. Every version and every issue of requirements documents must be recallable and displayable on-screen. But this is insufficient - requirements are the soul of a project. We may need to examine the history of every requirement, who changed it, when and why it was changed. Obviously every requirements must have a unique number which can never be changed or re-used. Deleted requirements are retained for reference. These facilities must be available on-screen interactively with needing to write complex commands.

## Principles for Organizing Information

Some basic principles of information organization have to be applied to requirements to keep them under control. Requirements have always been treated as individual entities to be stated and verified individually. Viewed in abstract, requirements are complex sets of intensely inter-related objects. As the system is developed, the requirements first expand, then their relationships to project "objects" grow more and more intense. The cost of change increases dramatically.

This complexity demands a structured approach to handling the requirements information. The principles of object orientation [ref: 1] can be adapted wholesale. Standards information tends to be highly complex, with many glossary entries, applicable documents, definitions and acronyms. Some repetition is inevitable, and this leads to problems of simultaneous updating. To minimize the information involved, and to keep it updatable, one requires a set of information organization principles. The objective is to transform a cloudy uncontrolled mess of information into a small, controllable set that is easy to understand and change.

### The key principles for organizing information are:

- **Structure** - hierarchy, layering, abstraction, scale, inheritance, and a stable decomposition criterion.
- **Modularity** - the ability to change elements without complex side-effects, cross-references to other standards, glossaries.
- **Information minimization** - re-use, association, polymorphism, encapsulation.
- **Presentation** - outlining, navigation, retrieval, selection, sorting, publishing, typography, graphics metaphors, use of color.
- **Dynamics and history management** - ability to see the history of individual requirements and sets of requirements, to select and re-order different subsets.

Together these could be considered as an ***object oriented approach to requirements information***. The management of complex information requires a layered and structured approach. A well structured set with many small faults can be improved piece by piece. A layered approach allows the whole system to be viewed and reviewed at the top-level, letting errors be corrected early.

## Structure

Structure has many different aspects. In a correctly structured information set, small changes can be made without disturbing the structure. Similarly learning anything about a structured set tells us something about the rest of the set. Standards should be structured so that generic information is stated once and inherited by all lower level standards. In practice this calls for a layered approach. Abstraction implies that standards are viewable at different levels of detail, and levels of abstraction are not mixed within the same document.

## Modularity

Although requirements need to be baselined during the course of the project, improvements always need to be made. If the requirements set is modular, requirements can be altered with minimum side-effects. Achieving modularity requires an analysis of what is likely to change, and the definition of clear relationships between documents. A good systems life cycle is the basis of modularity, because it divides the project information into coherent independent sets, which are baselined at different project stages.

## Information Minimization

The total amount of information involved in requirements can be reduced by cross-references and a unified glossary to save repeating definitions. The requirements equivalent of polymorphism is to get the same process to act at different points of the life cycle. For example, review or change control procedures should be as similar as possible at different stages. Where concepts are the same - as in the many varieties of problem reporting - a single concept should be adopted. Links between requirements information are key to information minimization. They allow information to be maintained at one place and re-used elsewhere.

## Presentation

Good presentation techniques can reduce the information load on engineers who have to read and use the standards. The best publishing quality should be an aim - good typography, high quality graphics [ref 1], plus all the publishing knowledge to help the reader absorb complex information (e.g. live headings, color coding of pages).

These concepts can reduce the amount of information, make it easier to find, easier to change, and improve its quality. The cost is the extra understanding that is necessary to organize the information.

**Dynamics and history management, easier to change, and improve its quality. The cost is the extra understanding that is necessary to organize the information.**

## Dynamics and History Management

There is a trend away from the single-shot life cycle, to avoid the problems of long developments with no measurable outputs. These projects pose extra difficulty for handling requirements. During an iterative project, multiple versions of a single document may be relevant. For example, we may have implemented and be operating one set of software requirements, be testing the next version, and developing a third. The three sets are complex, overlapping versions of the same set of information. There is a continual need to select out complex sub-sets, to re-order the presentation of information for example, to find out which requirements were not implemented in the first iteration, or those which were not met under test.

## Summary

The need to handle complex, inter-related, dynamic information forces a very structured approach in systems development. Good requirements tools can help this difficult engineering task, as long as they are founded on systems engineering and information management principles. The basic principles must be methodless, so that they can be used in conjunction with all sensible lower-level methods. Users should be encouraged, but not forced, to use structured methods.

Up to now engineers have had simple tools based on word processors or relational databases. Hierarchical object oriented data bases and better presentation techniques will help with the problem of handling requirements information. With tools like DOORS, engineers can concentrate on the real intellectual problems of the project.

## References

1. Coad, P. and Yourdon, E. "*Object-oriented analysis*" ISBN 0-13-629981-4 (1991).
2. Tufte, E. R. "*Envisaging information*", Graphics Press, Connecticut 06410.