

Requirements Management with DOORS

A Success Story

by Ian Alexander

<http://easyweb.easynet.co.uk/~iany/index.htm>

WHAT it is

DOORS is a requirements management tool. Requirements are conceived to be pieces of text with attributes – each one is owned by somebody, has a certain priority, has or has not been checked and approved, and so on.

When a user logs on to DOORS, a list (now a navigable tree) of ‘projects’ is displayed. Projects contain document-like Modules (see below) which can contain any combination of headings, text, and graphics. Much of the text is expected to be requirements, but the tool does not enforce this in any way. The user (depending on access rights) is free to edit the Modules, create new ones, and to import material from documents or other databases. The tool maintains a history of changes made to modules and to individual Objects (roughly equivalent to paragraphs, requirements, or database records) within them.

Up to this point the tool appears much like a somewhat elaborate word processor. The special character of the tool comes into view when the user starts to document attributes of requirements, for instance classifying them by type such as Safety, Reliability, and so on. The full power of the tool, however, is revealed only when the user creates traceability links between one requirement and another. Typically these are user and system requirements, but the tool can equally be applied to manage traces between system and subsystem requirements, or for that matter design elements, tests, dictionary entries and other items of project information. The user is free to create links of different types (see below) to indicate different logical relationships.

Users can then exploit the structure of requirement objects related by different types of link to select any desired subset of the project: for example, all system requirements that relate to version 1.2 of a product, but which have no verification links. Those specific requirements can then be printed, edited as a set (in a single operation), or exported in a wide range of formats. Requirements Management tools like DOORS permit such database queries, but at the same time provide a convenient document-like interface for viewing and editing requirements.

Earlier tools with the same general purpose did exist, but DOORS was both the first in the current wave of interest in requirements – starting in the early 1990s, and the first to achieve widespread success in the marketplace. It is said to be the market leader, at least in its class of relatively ‘heavyweight’ industrial-strength tools.

Among the questions that this success raises are what factors contributed, and to what extent knowledge gained through academic research may have contributed. These questions are addressed in the 'Where...' section below.

WHAT Technology was used

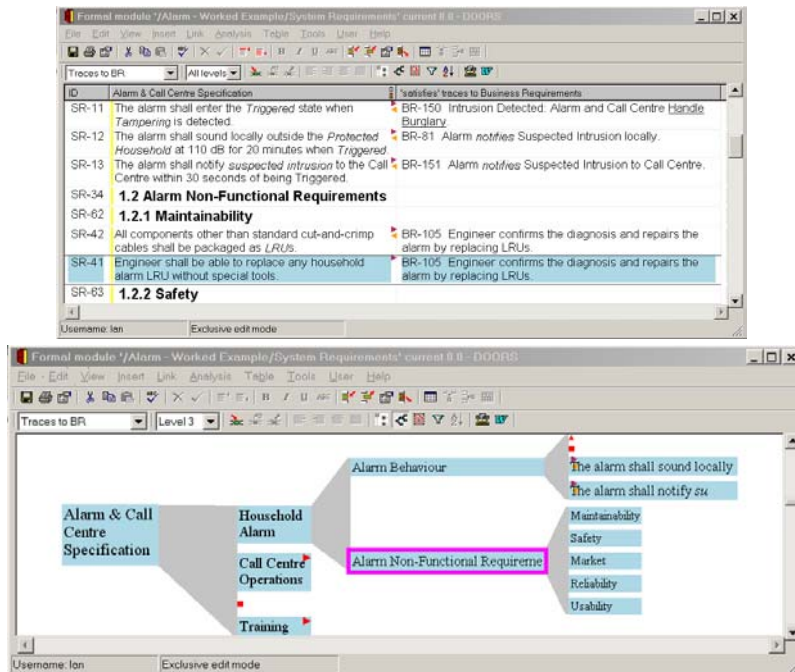
The tool (in its earliest form, and as conceived by Stevens) was based around a hierarchical data store, to consist of separate Projects. Each project consisted of Modules, which were to be of 3 types:

- "Descriptive", containing unstructured text, intended to capture notes, memos, and other raw sources of requirements;
- "Formal", containing structured text organized into a hierarchy of numbered headings (branch nodes) and requirement objects, expected to be leaf nodes; and
- "Link", containing sets of links between modules of the other two types; individual links are always between two objects, usually in different modules.

The data in modules was further organized into an unlimited number of attributes of any type (a few built-in, others to be added by users), which could be of individual objects or of modules as a whole. For instance, a module's Name is a module attribute, while a requirement's Object Text is an object attribute.

It can be seen that this design creates a need for an unusual kind of database, which must be capable both of representing types, attributes and arbitrary links between any pair of objects (like a relational database); and of representing a hierarchy of modules and heading objects (like a hierarchical database, or a set of word-processed documents). This allows people to think of requirement documents, while managing requirements as linked database objects. To achieve this, DOORS was designed and implemented initially on UNIX, and in its earliest form relied on that operating system to store and organise the data in an elaborate hierarchy of folders and files. These were initially unencrypted, so security and access control were at first rudimentary; clearly requirements for such facilities were secondary.

The active parts of the tool include a requirement browser/editor, which presents the data in any of several views: document-like; table-like; and tree-like, and permits elaborate filtering and sorting.



DOORS views: on the left, linked requirements viewed in a column; on the right, a tree view

In addition, the tool possessed from the beginning a powerful interpreter which was immediately tailored to provide access functions for the DOORS data structures – Module, Object, Link, and so on. The {C + access functions} language provided by this interpreter is called DXL, for the DOORS eXtension Language. Much of DOORS’ power comes from DXL scripts, including those provided with the tool, and 3rd-party and custom software. The tool is in effect a generator of a constellation of customised requirements tools with varying capabilities, including metrics, graphics, interfaces to modelling tools, and so on.

WHERE does DOORS come from

ESA Experience

DOORS’ creator, Dr Richard Stevens, was a researcher through the 1970’s and ’80s at the European Space Agency’s Research Institute (ESRIN) at Frascati in Italy. The agency, under its charter agreed by treaty among the member nations, develops space technologies, relying on industry for the work of creating satellite systems, launchers, and ground control systems (operated at the agency’s Operations Centre (ESOC) at Darmstadt in Germany). These are software-intensive, and indeed software constitutes the essence of ground control systems. Stevens observed repeatedly that despite the agency’s best efforts at specifying the required functionality, the software was delivered late, was unreliable, and often failed to perform essential functions. Something seemed to be wrong with the way it was specifying its software needs.

The European Space Agency's (ESA's) efforts included a major push to define and standardize good practices in software development. This yielded the agency's PSS-05-0 Software Engineering Standards¹, first issued in 1984. Work continued after that, under the chairmanship of Carlo Mazza with Stevens participating, and a second edition was issued in 1991, and published by Prentice-Hall¹ in 1994. A separate set of guides to the activities and processes required by the standards was commissioned by the agency and published by Prentice-Hall in 1996. This constituted a practical transfer of knowledge from the agency and its researchers to industry.



The visible products of ESA's efforts to improve software quality

¹ <http://i.f.alexander.users.btopenworld.com/reviews/mazza.htm>

Despite the success of the Software Engineering Standards, Stevens however found that it was not possible within the framework of ESA's charter to develop a practical set of tools that industry could use to comply effectively with the systematic approach dictated by the standards. He therefore resigned from the agency and set up a company, QSS, to develop and market suitable software, which he named DOORS ('Dynamic Object-Oriented Requirements System', though the tool is not especially object-oriented).

Predecessor Projects and Tools

Integrated Project Support Environments (IPSE)

The idea of managing project documents and their traceability relationships in a customisable database was pioneered by Teichroew in the ISDOS project in the late 1970s. This gave rise to several research projects and prototypes such as IDA, PMDB, and ALMA in the 1980s. There was some excitement in the software industry about the concept of an Integrated Project Support Environment (IPSE) which would use a database to manage a linked set of documents of some kind. However these hopes came to nothing at that time.

Stevens may possibly also have been aware of the idea of presenting project data from multiple viewpoints, which was similarly researched and prototyped at that time.

'Fourth Generation Languages' (4GL)

The concept of having an open working environment in a programmable tool, using some kind of scripting language, was also pioneered in the '70s and '80s, both in research (as in MENTOR) and commercially.

Many simple commercial database management systems such as DataEase and FoxBase combined a relational database (based on the pioneering work of Codd and Date, among many other researchers) with a scripting language and what are now known as wizards to guide users through tricky processes such as setting up relationships and writing reports.

These so-called 4GLs ('fourth generation languages, presumably counting Assembler-Basic-C-Database) provided windowed graphical user interfaces (GUI) onto database tables, report editors, and similar integrated tools. DOORS, like many modern software tools, is plainly in that tradition.

Research on (interpreted) scripting languages certainly fed directly into DOORS; Stevens recruited George McCaskill whose PhD had been on scripting; his safe-subset-of-C interpreter formed the core of DOORS' invaluable DXL scripting subsystem.

Graphical User Interface (GUI)

All such tools rely absolutely on a windowed operating system with a GUI, such as UNIX with X-windows, or Microsoft Windows. GUIs were pioneered by research at

Xerox PARC, and commercialised by Apple with its Lisa and Macintosh personal computers.

IN WHAT CONTEXT was it used

DOORS was from the beginning understood to be a tool for managing not only requirements but development project information of all kinds, including specifications, designs, tests, standards, procedures and plans, as well as of relationships between these.

In addition, DOORS was quickly sold into a wide range of markets, including aerospace, telecommunications, electronics, automotive, railway, software, defence, and so on.

Early adopters of DOORS included Boeing, Hughes, BAE Systems and other aerospace companies; Nortel, the Swedish and Dutch telcos, Ericsson and other large telecommunication companies; and then many others. These organisations had invariably had difficulties with managing traceability in the face of requirements change.

On a short project, engineers can keep a picture of the desired system in their heads. But as projects become larger, fragmented over many geographical sites and across contractual boundaries, and especially longer (lasting many years) it becomes impossible for anyone to keep track of all the changes of status that occur without tool support. The same issues – information sharing; baselining; change control; reviewing and status tracking; impact and traceability analysis recurred time and again.

DOORS' visual simplicity – you wrote a document and linked its paragraphs to other paragraphs – seems to have suited engineers from many backgrounds, in contrast perhaps to tools aimed principally at software developers.

However, users soon clamoured for additional features, such as better handling of graphics and the ability to share data with other tools. Interfaces were quickly constructed to popular modelling and documentation tools, either by importing and exporting files, or by dynamic transfer using sockets or Microsoft OLE. Thus the context of DOORS use is often a cluster of tools with different purposes – including documentation, configuration management, modelling, testing, and planning.

HOW WELL did it work, and WHY

DOORS clearly fulfilled a basic market need, as QSS grew exponentially on the back of sales of the tool, as well as of consultancy and training to support its implementation within customer organisations. When Stevens retired, QSS was sold to Telelogic, where its success has continued to grow. DOORS is now marketed worldwide through many local offices.

Some specific technical reasons why the tool did so well can however be identified.

Both Document and Database

It still feels uncomfortable to hear DOORS described as "a database", when in truth it is both more and less than that term implies. If it is a database, it is strongly typed and as expressive as a programming language, but its language is as much about relationships as about data records. Many users, however, persist in thinking of DOORS modules as documents, and historically they have continued to demand more word-processor-like features such as text markup and formatting.

DOORS thrives by offering many of the advantages of both document and database: ease of editing and information sharing, along with permanence, baselining and change control. This surprising combination allows people to think of requirement documents, while managing requirements as linked database objects.

Visual Simplicity

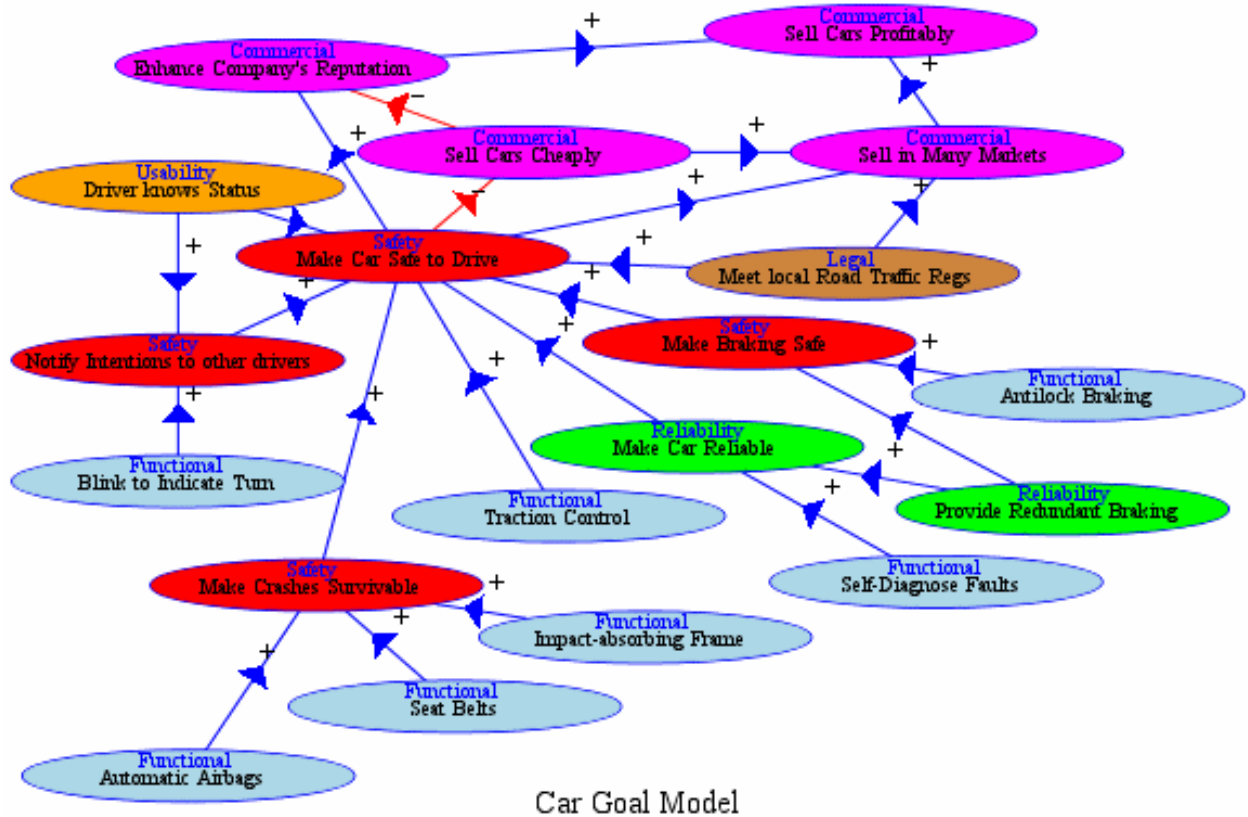
The powerful multiple-view user interface gives an immediate visual impression of the tool's unusual capabilities, which may have contributed to its success. Views can

- present modules as familiar-looking documents;
- show requirements fully-instrumented as database records, filtered by any chosen set of criteria;
- shrink documents to tree diagrams;
- or most strikingly, demonstrate the current state of all the traceability links between a document and all other documents, or any desired selection of them.

These complex capabilities can all be exploited fully without programming.

Customisability

DOORS is extremely customisable, and within limits it is fair to claim that it is easy to customise it, both with DXL (see e.g. the free tools at <http://www.scenarioplus.org.uk>) and through the built-in user interface, which allows users to create types, attributes, filters, sorts, views, linking rules and even (with a 'wizard') simple DXL scripts to display linked information. Customisation of a generic engine such as DOORS is effective because a small increment of work, e.g. to create a graphics editor, provides a large capability, as all the built-in functionality is at once available. For instance, adding a Goal Model editor at once enables goals to be linked to requirements and to tests – a capability that a stand-alone editor might well lack.



DOORS customisation: a Goal Modelling tool built with DXL

An attractive property of all such customisations is that the data objects, such as these goals, are full-valued DOORS objects which can be traced to requirements

Openness to 3rd Party Tools

Another reason for the popularity of DOORS may be the ease with which documents and models of all kinds can be shared with other tools, and indeed the ease with which new interfaces can be created. Certainly there are now many integrations with 3rd party tools.

Adaptability to Customer Processes

But perhaps the fundamental cause of its success is its flexibility in supporting the differing processes of customer organisations. This is achieved in several ways:

- by customisations of the kinds already mentioned
- by the inherent convenience of the built-in functionality, available through the user interface
- by modelling the information structures (projects, modules, attributes, relationships) to suit the problem at hand.

A Solution that Fits the Problem

Perhaps the truth is that DOORS, imperfect as it is, provides quite a good match to a general problem: represent a large body of pieces of (possibly hierarchically structured) information of different kinds, related to each other in several different ways. Perhaps few other tools attempt to do that; fewer still do it with the visual simplicity of DOORS; and none achieved it before DOORS created a new market for requirements engineering in the early 1990s. This success was based both on Stevens' direct personal experience of the requirements problem as a customer, and on the ESA's research work on the requirements process.