

Creating and Managing a Resilient DOORS Schema

Paul Miller,
Manager - Quality & Systems Administration Department
Fujitsu Australia Limited,
5 Lakeside Drive,
East Burwood, Victoria, 3151,
AUSTRALIA.

Abstract. So you've purchased DOORS, it's been installed, the pilot project's been selected, the boss is waiting anxiously for signs of a return on investment and you're blankly staring at the PC screen wondering what happens next. You've seen the demo's, perhaps visited a reference site or witnessed DOORS in action at a user group meeting, but how did they take DOORS out of the box and get it to do all that great stuff. The beauty of DOORS is that not only is it's base architecture designed for Requirements Management, it's power lie's in it's ability to be customized to meet your specific Requirements Management needs. This paper will step through the process that Fujitsu Australia used to take DOORS out-of-the box and architect a module, attribute and view schema to meet the Requirements and Test Management needs of it's System Development Life-cycle (SDLC) model. This paper will also walk through some process management examples including how DOORS has been customized to strengthen it's baseline management features and how to manage suspect incoming links that come from source objects that have a modification time-stamp that's later than that of the current target object being viewed.

INTRODUCTION

Having a well designed and resilient schema in place is essential for achieving acceptance by users and retaining the sanity of tool administrators! A poorly designed schema can hinder the tools potential performance, make it very difficult to introduce improvements and potentially attract negative user perceptions which can relegate the tool to spending a lot of time sitting on the shelf.

Users are increasingly being required to become proficient with many software based

tools that all come with their various enchanting nuances. Making their life easier with a schema that reflects consistency and allows them to quickly build up a degree of anticipation and familiarity will go a long way towards achieving positive acceptance. If schema's are overly complex, require excessive administration and are being improvised and maintained in an ad-hoc manner, the comfort zone and patience of users will be tested and this can feed and propagate false perceptions of the capability of the tool.

You need to aim for a schema that meets users needs, is easy to administer, resilient to change and basically operates as a system in it's own right.

DEVELOPING A DOORS SCHEMA

If you want your schema to have a sense of system about it, you'll need to adopt some of the basic fundamentals associated with the practices of the systems engineering discipline. A good fundamental to start with is that "form follows function". That is, the form of the solution can only be fit for purpose when it is derived and traceable to the problem space to which the solution is designed to resolve. Put another way, don't create a solution that's looking for a problem. So what has this got to do with the DOORS tool? In short, your schema solution will benefit if there's a well defined and understood process in place to act as a specification for the tool to manage. If you already have some procedures that describe your specification life-cycles, then these will be a valuable source of base requirements for customizing a DOORS schema to meet the needs of your process.

The schema described in this paper is based on meeting and adding value to the Specification and Test life-cycle needs of our Systems Development Life-Cycle (SDLC) model.

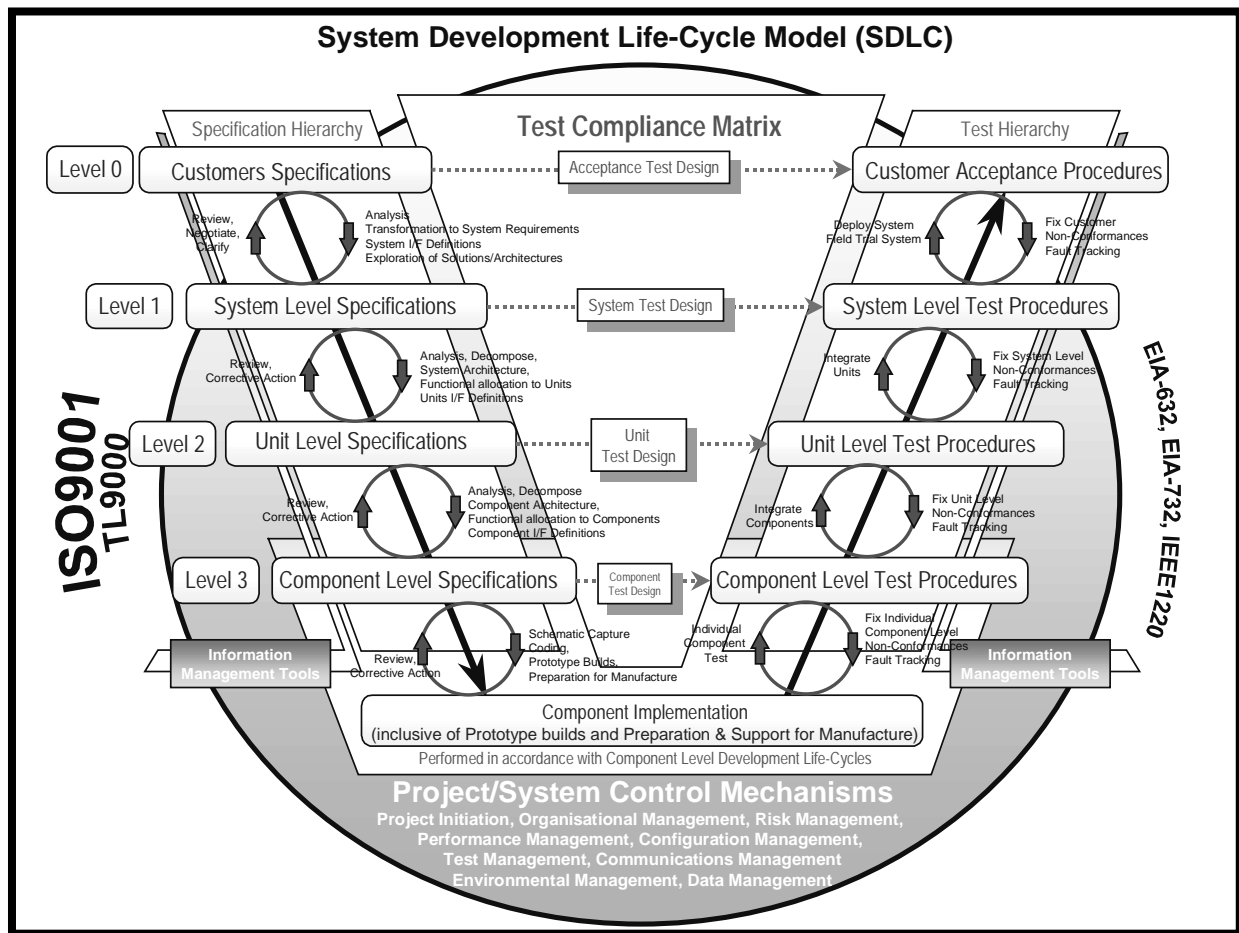


Figure 1.0 System Development Life-cycle Model (SDLC).

SYSTEMS DEVELOPMENT LIFE-CYCLE (SDLC)

Figure 1.0 depicts the SDLC model adopted by Fujitsu Australia’s Telecommunications R&D centre. The model is a functional representation of the structure and relationships between critical elements of information that define (Specification Hierarchy), produce (Component Implementation), verify (Test Hierarchy) and manage (Project/System Control Mechanisms) the development of a complex system.

A presentation of the functional definitions for every element in this model would be outside of the scope of this paper. Descriptions of the major structural aspects can be found in a previous paper [Miller.P]. For the purposes of this paper, only the following points need to be understood:

i) The Specification Hierarchy represents the progressive and traceable decomposition of parent requirements in one level, to child requirements in the next level below. This promotes the concept of validation where child requirements in the lowest level component specification can be traced to a

higher level customer needs and business requirements.

ii) The Test Compliance Matrix & Test Hierarchy represent the traceability of requirements from specifications to test procedures. This promotes the concept of verification where all requirements must be verifiable and tested at least once. The Test Hierarchy tracks the same levels as the Specification Hierarchy which promotes the concept that tests on requirements should be applied at the same level as they appear in the Specification Hierarchy.

The DOORS schema devised for Fujitsu Australia has been designed to support the relationship, data and functional needs of the Specification Hierarchy, Test Hierarchy and Test Compliance Matrix represented in the Systems Engineering model of Figure 1.0.

CREATING THE DOORS SCHEMA

Here’s tip N^o 1, "Less is more", that is, try and keep your schema's simple. Each and every customization that you add represents work now and work later. When modifications are required, you will most likely need to perform this

modification across many projects and modules. So save yourself some time, be sure that each and every customization is value added and is going to be used.

When devising a schema take into consideration some of the following:

- i) Identify the suite of module types needed eg: Specifications, Test Procedures, Architecture etc.
- ii) Identify the types of views required and an outline description of the content of each column making up a view.
- iii) Identify relationships between modules in terms of links and any special attributes required to assist with managing links.
- iv) Identify user access rights to projects, modules, objects, views, attributes etc.
- v) Identify DXL scripts needed to provide custom functionality.
- vi) Work out a module naming syntax to assist users with locating modules within the DOORS Project Explorer Window.
- vii) Work out standard printing formats and layouts for hard copies.
- viii) Think about the general look and feel of each module and across modules. Users appreciate modules with a common look and feel as it promotes a sense of comfort and familiarity [Pajerek L].
- ix) Take into consideration any interfacing with other external information management tools, component design tools or manual (personware) processes.

STEP 1

IDENTIFY YOUR MODULE TYPES

The first port of call is selecting what common module types are needed. If you understand the process of creating an Information Architecture, then this will help to identify module types. In our case, the SDLC model acts to present the architecture of our main sources of critical information.

Analysis of our SDLC model determined that we needed at least two basic base types of modules, a Specification type and a Test Procedure type. From these base types, variants can be created, but because each variant will be created from a base type, there will always be a common look and feel.

Starting from a freshly created formal module within DOORS, each base type of module was customized with the appropriate object attributes, views, heading layouts and base

user access rights. These modules have been archived as restorable module templates. Something akin to the .dot template files used in Microsoft Word . The steps for devising views, object attributes, access rights and DXL scripts for inclusion in the base type template modules are handled in the subsequent sections.

Each base module type includes special module level attributes. These assist with general schema management by categorization of each module instance. It also improves the performance of DXL scripts that will need to scope, locate and operate on particular modules quickly. Examples of module level attributes we have used are shown below.

Base Type (enum): Specification Module, Test Module.

Base Type Version (string): Base type version control, updated when the base type module is modified and when instances are upgraded to be in synch with the base type.

SDLC Hierarchical Position (enum): Customer Level, System Level, Unit Level, Component Level

Model Element (enum): Specification Hierarchy, Test Hierarchy, Project Management, Support.

These module level attributes assist DXL scripts to focus their operations on particular groups of modules eg: collecting data on System level specifications and Test Procedure modules only, modifying specific data in all Unit Level specification modules, performing an audit on the versions of base types in use.

Every time the schema of a base module type needs to be modified, it's applied to the module template first. When ever a change is made, the Base Type Version attribute in the template is changed accordingly. Existing module instances using the same base type are then manually upgraded with these changes and their Base Type Version attribute modified accordingly to match the template. It is for this reason that schema's should be kept simple because each change can amount to a significant amount of work to modify all instances of that base type in each project. DXL scripts can assist with making modifications but if your schema is too complex and susceptible to regular modification, you will increase the chances of an error creeping in.

Every time a new specification or test procedure module is required, the appropriate archived base template is restored into the

required project and the module level attributes modified accordingly.

STEP 2 DEVISE YOUR VIEWS

Next is the determination of what sort of base information needs to be viewed by the various users who will be authoring and reading the module. It's highly recommended that users are consulted on what they need to see, this will go a long way towards them becoming positive advocates of the tool.

Each view should be defined as a set of columns. These columns will of course resolve into DOORS attributes which will require further definition (covered in the next section). Tools like Microsoft Excel can be used to quickly knock up a prototype view in front of a user and get their feedback.

With respect to views for a specification type module, everyone will want to see the main requirements text and unique identifier so these should appear in each and every view. Managers will want to be able to approve, set priorities, assign responsibilities and check on any attached change proposals so they will need their own specific view. Test Engineers will want to be able to assess the verifiability of requirements and start mapping requirements to Test Procedures so they will need their own view. Authors and readers will need a view that will display details about the source of incoming links and the targets of outgoing links.

For the purposes of gathering statistics about requirements and the removal of any ambiguity, make sure that at least one view contains a column that declares the information type of the object. For example, does the object contain a true requirement or just information only? Front matter such as introductions, scope, purpose, definitions and headings are not usually included in requirement metric gathering exercises. This will allow you to filter out objects that are not true requirements.

Here's some example views to consider.

Default View: The view that users see when the module is opened. Should show the most important information eg: Requirements text, unique identifier, Information Type.

Architecture Design View: Cross references to Use Cases or any other design information used to elicit more detailed child requirements for the next level below.

Management View: Approval, Priority, Responsibilities, Change Proposals.

Parent Child View: Display full text and unique identifier of incoming parent requirements and outgoing child requirements.

Edit History View: Display of history data such as Created by, Last Modified by etc.

Test Design View: Level to be tested at, planned test method (Test, Document, Inspection, Calculation), planned target test procedure.

When each view is created, protect them from inadvertent modification by setting the specific access rights so that only administrators can modify the view definition. Consider prefixing each view name with a number or an alphabetic letter so that the base schema views appear at the top of the view list and in the order that you want.

Pay attention to the advanced view properties, particularly the Sort, Filter, Current Object and Window Position properties. Users will become irritated if every time they select a new view, the window position jumps around the screen or the previously created filter or sort is unnecessarily wiped away.

Educate users to prefix their personal views with their login name so that when they are saved, they will be grouped together and not be mixed in amongst the base schema views.

STEP3 ATTRIBUTES

Once the different views have been determined, the attributes for each column of information can now be defined.

To assist with the management of your base schema attributes, prefix the names of each created attribute with an identifier. This will help to group all of your schema attributes together and separate them from the DOORS built in attributes.

It's recommended that the definition of your base schema attributes are protected by setting specific access rights to the definition of each attribute so that only administrators can modify them.

Here's some tips with respect to the available attribute properties settings.

Be careful with the "Inherit" property. If the "Affect Change Bars" property is also set, only the parent object will have its change bars modified even though the child objects have inherited the change. Also, once you have

modified the value of an attribute, it's inherit mode changes from acquired (ie: inherited) to specific (ie: no longer inherited). This can catch users out because there may be an assumption that when a parent is modified, all children will inherit the change. However, child objects with specific attribute values will remain unchanged.

Don't apply the "Affect Change Bars" and "Affect Change Dates" properties too liberally. Reserve these properties for critical attributes only. This will assist users if they have to track down critical modifications in the history log.

When using DXL Layout or DXL Attribute scripts, it's recommend that these scripts be version controlled outside of the DOORS tool with your configuration management tool of choice (See also Step 6 "*DXL Scripting*" below).

STEP 4 ACCESS RIGHTS

In the previous sections, some advice has already been given with respect to using specific access rights to protect your schema views and attributes from inadvertent modification.

With respect to the module level access rights for the base type modules, alter the default access rights to so that they don't inherit from the Parent, and are read only for "Everyone else". This means that when ever an instance of a template is restored, only the Administrator will initially have access. This can then be followed up with providing the author(s) with Read & Write access. DOORS V5 is very flexible with access rights, specific users can be given reduced administrator type rights so it's not always necessary for the Administrator to be involved.

When it comes to shared edit object access rights, keep it simple. Don't apply shared edit rights too deep into the hierarchy and if possible, apply shared edit object right to a common set of users.

STEP 5 LINK MODULES

Think carefully about your link modules. Don't try and cram every linkset into a single link module or this will prevent you from being able to take full advantage of the Impact and Trace utilities such as the Link Filter and Traceability wizard.

With respect to our SDLC model, separate link modules have been created for each of the important relationships between elements in the model. For example, consideration should be given to have all links between the System level

and Unit level specifications placed into their own link module, links from the System level to System Level test procedures placed into their own link module and so on. A modules properties can be set so that when users perform a drag-and-drop link, the link data is automatically stored in the correct Link Module, otherwise, users will need to be educated to make sure that they select the correct link module prior to the application of drag-n-drop linking.

This method of assigning key relationships to individual link modules will also support better control of linking access rights.

STEP 6 DXL SCRIPTING

DXL scripts are the power of DOORS. They're fun to write and there's a sense of achievement when a script provides a really valuable function that overcomes a previously error prone or difficult manual process.

However, don't become too relaxed about the process of writing DXL code just because they're scripts. Treat DXL scripts with the same respect as for any other software. They should be well documented, written in a consistent style and configuration managed. Your suite of scripts will most likely require effort to keep them maintained during their lifetime so keep the number of active scripts to a manageable level.

If you have multiple DXL authors, save yourself some trouble, make sure that you adopt a version control system of some sort that acts as a central managed repository for all DXL scripts. This will allow you to easily regress back to known working versions should there be a problem with a new version. When there are multiple authors, it's also recommended that you have some basic DXL style writing procedures to promote consistency in style which in turn will allow authors to more easily maintain each others code.

Try not to copy and paste common functions across DXL scripts. This is an invitation for inconsistent operation if modifications to a common function in one DXL script is not applied across all other copies. Store common functions in *.inc files so that there's a single instance of them.

BASELINE MANAGEMENT

As you're most likely aware, DOORS has an integrated baseline facility. Prior to a baseline being set, a process to review all implemented and proposed changes will most likely be

required. Value can be added to your review and baseline process by implementing a DXL Baseline Assistant facility that offers the user a list of selectable functions that can save time by collecting useful information and checking for problems that review participants may miss. This also allows the scope of review to concentrate more time on reviewing important requirement criteria that cannot be handled by the tool. Here's some examples of useful functions for the Baseline Assistant.

Pre Baseline Statistics: The total number of objects tagged as being requirements and not information-only, the total number of requirements marked for deletion, total number of new requirements added since the last baseline, total number of requirements modified since the last baseline. These are all handy data items to know prior to or during a review and prior to setting a baseline. They can also be used to determine the general stability of the specification.

Find Unresolved Change Proposals: Check and identify objects that are linked to change proposals that are still in the new state ie: they have not been resolved.

Find General Rule Violations: For example, we have a number of style rules to ensure consistency such as not allowing objects to contain a combination of Object Heading text and Object Text.

Perform Requirements Quality Check: Perform a search for words that are considered to be ambiguous and increase the chances of misinterpretation.

Create Parent Child Snapshot: Whilst a module can be baselined, the requirements located at the source of incoming links and targets of outgoing links should also be captured. Because these source and target objects will change in time, this snapshot will ensure that there's a captured record of their state at the time of setting the baseline of the current module. This snapshot facility can store the captured information in the form of a new view that displays the current modules baselined requirements along side adjacent columns that display a snapshot of linked source and target requirements.

In cases where checks are being performed, a report can be generated that lists the UID's of suspected objects. Such a report can be submitted

to a review meeting as evidence that these checks have been applied and meet acceptance criteria for the approval of the baseline.

Once a baseline is set, the same facility can be used to perform some post baseline functions. For example, immediately after setting a baseline, we purge all objects that are tagged for deletion as these have been recorded in the previous baseline and serve no real purpose for the next version.

SUSPECT LINK FACILITY

Being able to link and trace from one object to another is indeed a powerful feature of the DOORS tool. Of particular interest to an author is what's happening at the source end of an incoming link.

As a case example, Test Engineers usually have to cope with authoring Test Procedures whilst modifications are still being made to source requirements. Whilst some authors of requirements are mindful of the forward impact of any modifications they make, the pressures of any development environment can make it very difficult to ensure that all stakeholders are informed in a timely and accurate manner. By creating a DXL suspect link facility, you can flag objects that contain incoming links from source objects that have either been deleted or are from source objects that have a modification time-stamp that's later than that of the target object currently being viewed. DOORS V5.1 now has a built in suspect link facility, however, we have devised our own to include some extra features such as detecting objects tagged for deletion.

A suspect link facility can be entered as a real time facility in the form of a DXL Layout script. In our case, this script traces an incoming link to it's source and displays the requirements text and unique ID of the source object in a module column. It also tests if the source object has been tagged for deletion and also checks the modification date & time stamp check of the source object.

If the source object has been tagged for deletion, the script adds and displays the text "DELETED" along side the currently displayed requirements text and unique ID of the source object. It also changes the color of the displayed text to red to bring this to an authors attention.

If the modification date value of the source is more recent than that of the currently viewed target, then the link is suspect because the source has been modified after the target was last

modified. In this case the script adds and displays the text "MODIFIED" along side the currently displayed requirements text and unique ID of the source object. It also changes the color of the displayed text to red to bring this to an authors attention.

If the source and target modification date values are the same, ie: changes performed within the same day, the script will interrogate the history records of the source object. It then identifies which critical source object attribute was modified and then extracts it's history record time stamp for comparison with the history record time stamp of the target. These time stamps record the hour, minute and second of the change for any given day.

When an author discovers a suspect link, the author can discuss the changes with the author of the suspect target object and determine if this requires an adjustment to the target module as well.

In the case of a "MODIFIED" suspect, when the author is satisfied that the suspect incoming link has been accounted for and understands what changes are required, the "MODIFIED" state is removed by editing changes to the target object requirements text. By doing this, the target objects Last Modified date and time is changed and the date & time stamp comparison with the source will indicate that the incoming links are no longer suspect. Even if no change is required, an innocent space entered into the object requirements text of the target object will reset the suspect state.

In the case of a "DELETED" suspect, the author will again discuss this with the author of the suspect target object. The only way in which the "DELETED" state can be reset is by either un-deleting the source object or if an agreement is reached to also delete the link connected to the source object.

In both suspect cases, the suspect link facility helps to bring a problem to the attention of authors and facilitate a dialog to fix the problem up. Additional scripts can be written to search for suspect links across an entire project and report on these daily.

CONCLUSION

In conclusion, spend the time up front to work out what it is that the DOORS tool must manage before you start implementing a schema. We used our SDLC model as the specification and framework for implementing a schema that

suits our needs. If you don't have an equivalent to the SDLC model, work out a flowchart or perform an Information Architecture analysis of the specification and test processes that are needed. The reward for a well planned schema is a resilient requirements management system that will keep users happy and administrators sane.

I'm quite happy to supply readers of this paper with a package consisting of the following:

- Sample archived project (DOORS version 5 only) that contains:
 - Sample Specification and Test Procedure modules
 - System & Unit level Base Type Template Modules.
 - Test Procedure Base Type Template Modules. (Contains DXL Layout "Suspect Link" script.).
- View & Attribute descriptions for each template.
- DXL Baseline Assistant script.
- Example coding style guidelines.

My contact details are shown below.

ACKNOWLEDGEMENTS

Fabian Musci BAppSc, GrdDip(CompSci), GrdDip(DigElec), Managing Director, EuroCyber Pty Ltd.

REFERENCES

- Miller P., "Taming the Expanding System Complexity of a Commercial Product: Embracing Systems Engineering Principles in a Commercial Development Environment" DOORS User Group Conference Proceedings, October 1998,
- Pajerek L. "Bought any Good Shelfware Lately?" "CrossTalk: The Journal of Defense Software Engineering" (Vol. 2, No. 12), December 1997.

BIOGRAPHY

Paul Miller is the Manager of the Quality and Systems Administration Department (QSA) at Fujitsu Australia's Telecommunications Research and Development centre. The QSA is responsible for providing subject matter expertise and administration of the processes and tools associated with Requirements, Change and Configuration Management. It's quality function also extends to responsibility for monitoring

compliance to quality standards such as ISO9001, TL9000 and conformance to corporate and customer specific standards. The QSA is also responsible for providing PC desktop support and the systems administration of the R&D centre's LAN and server environment.

He began his career working for NEC Australia modifying PABX and telephony products to meet Australian regulatory standards. He's also worked for Rockwell Systems Australia, where he acquired valuable training and a higher awareness of professional systems engineering practices.

Paul is a member of the Systems Engineering Society Of Australia (SESA) which is a chapter of the International Council of Systems Engineering (INCOSE).

Telephone: . +61 3 9845 4367 (Desk)
 +61 418 135 103 (Mobile)
 +61 3 9845 4300 (Reception)

Fax: +61 3 9845 4572

Email:..... paul.r.miller@fujitsu.com.au