

A WHITE PAPER *from* ARTISAN Software Tools



**SUCCESSFULLY MANAGING AN
INCREMENTAL REAL-TIME PROJECT**

Part Three: Requirements Management

Successfully Managing An Incremental Real-time Project is published by ARTiSAN Software Tools.

Author: Matthew Hause

Copyright 1999, ARTiSAN Software Tools

**North America, Latin America,
Asia/Pacific**

ARTiSAN Software Tools Inc
Two Lincoln Center, Suite 370
10220 S.W. Greenburg Road
Portland OR 97223
USA

Tel: +001 503 245 6200
Fax: +001 503 244 1443
e-mail: info.us@artisansw.com

Europe

ARTiSAN Software Tools Ltd
Stamford House
Regent Street
Cheltenham, Glos, GL50 1HN
UK

Tel: 01242 229300
Fax: 01242 229301
e-mail: info.uk@artisansw.com

www.artisansw.com

PART THREE: REQUIREMENTS MANAGEMENT

Introduction

For large systems, or those systems where conformance and traceability to requirements is essential, requirements management is a key aspect to building a quality system. Requirements management starts with the definition of requirements and continues through the project, culminating in the acceptance of the product against the requirements. In this paper, we show how ARTiSAN Real-time Studio (RtS) is used to analyze and design the system based on those requirements. We also show the DOORS Requirements Management Tool from Quality Systems and Software Inc. (QSS) can help manage the requirements, and how the ARTiSAN DOORS Synchronizer helps to show traceability, conformance, impact analysis, and completeness for the project.

The paper will cover a project lifecycle and show how DOORS and RtS should be used at each stage of the lifecycle. Additionally, it will describe best practice to show at which stages of modeling and coverage verification, synchronization should take place to make sure that all the requirements are being met.

Extracts have been taken from some of the papers on the QSS web site in order to provide the most consistent and coherent description of the capabilities of DOORS. References to these have been included and full descriptions of these are included at the end of this paper. For further information, visit their web site at <http://www.qssinc.com/>.

What is Requirements Management?

According to the Software Engineering Institute [FLORAC], the purpose of Requirements Management is "to establish and maintain a common agreement between the customer and the software project regarding the customer's requirements that will be addressed by the software project."

Requirements Management involves:

- Establishing and maintaining the baseline of allocated requirements for the software project;
- Reviewing the software project's plans, activities, and work packages to ensure that they are consistent with the allocated requirements.

In short, Requirements Management consists of both administering the requirements and validating that the end product satisfies these requirements. Crosby's definition of quality is "conformance to requirements". [CROSBY] Therefore, the tasks involved in Requirements Management (RM) are essential in the creation of a quality system. RM starts at the beginning of the project when the requirements are being collected. It is essential that RM is an active component of all phases of the project lifecycle.

Real-time Studio Requirements Management Capability

Establishing a consistent and coherent set of requirements early in a project reduces risk, even when delivery is incremental. When requirements are explored with a working model or prototype, sponsors can gain a clear idea of all their implications. The sponsor may take many forms. This could be systems engineering, marketing, or an external customer. On many real-time projects it is important, and often mandatory, to be able to prove that a requirement has been met. The proof depends upon the ability to trace forward from a requirement to model elements and code.

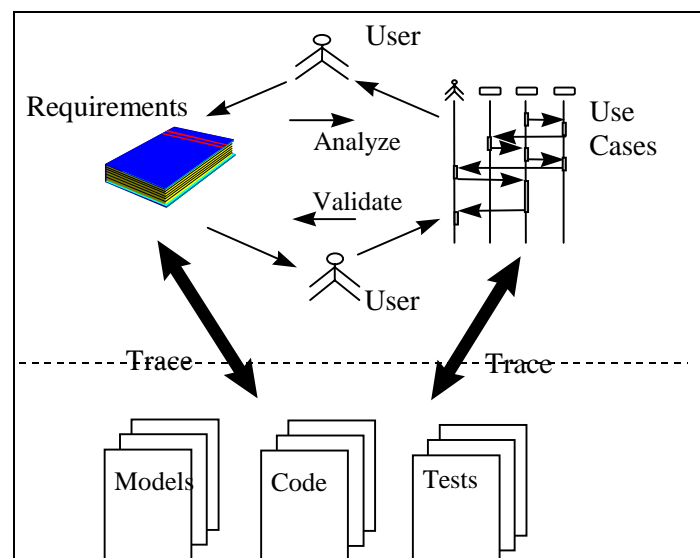
Once a mapping between artifact and requirement is in place, affected components can be easily identified if a requirement is changed. This is vital if the cost of that change is to be assessed up front. Conversely, if an artifact changes, then regression tests can be set up from this mapping to ensure that the original requirements are still being met.

Validating Requirements

Use Cases are one example of how RtS can be used to effectively capture functional requirements. Each Use Case describes one usage of the system, a set of scenarios showing the interaction between the system and its external devices and users. Having captured the functionality of the system from requirements documents and users in this way, it is important to validate the result.

To do this the analyst walks through each Use Case with a Sequence Diagram describing system drivers and responses to the users. The outcome, after several iterations, should be agreement that the Use Cases are valid and complete. Animation or a more detailed prototype may be of benefit in this process. Figure 1 shows how functional requirements can be traced to the modeling elements in RtS.

Figure 1
Tracing and Validating
Requirements



Tracing Back to Requirements

Systems development can be seen as a process of transforming a set of requirements into a real system that provides a solution to a known problem. To carry out this process in the most effective and efficient way, an organization needs to ensure that anything built as part of the system, in either hardware or software, directly relates to a defined requirement. Requirements traceability deals with controlling this process, and ensuring that a system is not 'over-engineered' above its requirement needs or 'under-engineered' by missing requirements.

Tracing requirements sounds like a simple and obvious element of the software engineering process, but in practice, in dynamic, large scale and complex systems, it becomes a major problem. The amount of design information and deliverables created within the project can be massive, and requirements can be lost in the morass of information available. Requirements also change, and the effects of these changes need to be monitored and controlled. What is needed is a process that automatically tracks requirements throughout a system development lifecycle. The process should have the following characteristics:

- **Capture and trace requirements throughout the development lifecycle.**
Full lifecycle traceability is crucial because if compliance checking is left until acceptance testing, problems will be spotted too late. Over, or under engineering needs to be identified early, so that any rectification can take place early. Within the constraints imposed on the project, appropriate steps should be taken to ensure that the allocated requirements are documented and controlled.
- **Minimizes risk.**
Any process should contribute to the management and control of risk by enabling errors to be trapped at an early stage. This means that traceability must occur at, and within, each stage of the lifecycle. This ensures that the evolving system will meet the requirements, and that all requirements are being met.
- **Manages change.**
In a dynamic development approach such as Real-time Perspective [MOORE], the management of change to the requirements is critical. The process should enable the project management to easily assess the impact of potential changes, as well as control 'requirements creep.' Tying everything in the design to an approved source requirement, and allocating all requirements to a design solution can achieve this.
- **Supports testing.**
All requirements should include the acceptance criteria against which their completion will be assessed. This is an important source for the creation and control of acceptance tests. However, ensuring problems are combated prior to acceptance testing is a key to requirements traceability, and a good system builds this ability in.
- **Configurability.**
The process should be able to support a wide range of requirements captured from diverse sources.

- **Managing and reporting.**

The process should enable management, and have a set of built in reports to allow the whole process to be effectively controlled. This means that effective browse, search, and change facilities must exist for all requirements, and tailorable and configurable reporting must be possible.

The Importance of Requirements

Clearly stating the problem allows all those who will be involved in the project to understand the basis of what they are doing. The documents that describe the problem will be the start of an agreement with the customer as to the nature of the completed product. The agreement with the customer may be described in a document such as a statement of work, product specification, contract, or requirements document, or may be described in a combination of documents. This agreement covers the technical requirements (for example, functional requirements), the non-technical requirements (for example, delivery dates), and the acceptance criteria that will be used to validate the software products. These requirements form the basis for planning, performing, and tracking the software project's plans and activities throughout the software life cycle.

Requirements communicate what is wanted

Requirements define what is wanted. Many projects fail although the solution is technically high quality. If a product does not do what the customer or the users want, then the build quality is irrelevant. The key role of requirements is to show developers and users what is needed from a system. Provided the requirements are a language that everyone can understand, all people involved, including customers, can see the whole project and their role within that project. The first and most basic role of requirements is therefore communication. An essential part of requirements management is the validation of the requirements. In part two of this series, Managing Change and Monitoring Progress, we showed how prototyping requirements help to ensure that the requirement is what the customer really wants.

What are Requirements?

The number of requirements models that are built should be determined by the systems engineering standard used. Traditionally, there are two models for requirements - user and systems requirements - followed by a design.

The "user requirements" are a statement of the customer's needs and expectations for the project. The customer requirements are stated from the customer and end user perspectives, are intended to achieve a shared understanding between the customer and the project, and provide the criteria to determine whether the products satisfy the customer's needs and expectations. Whoever produces the user requirements, the users must own them.

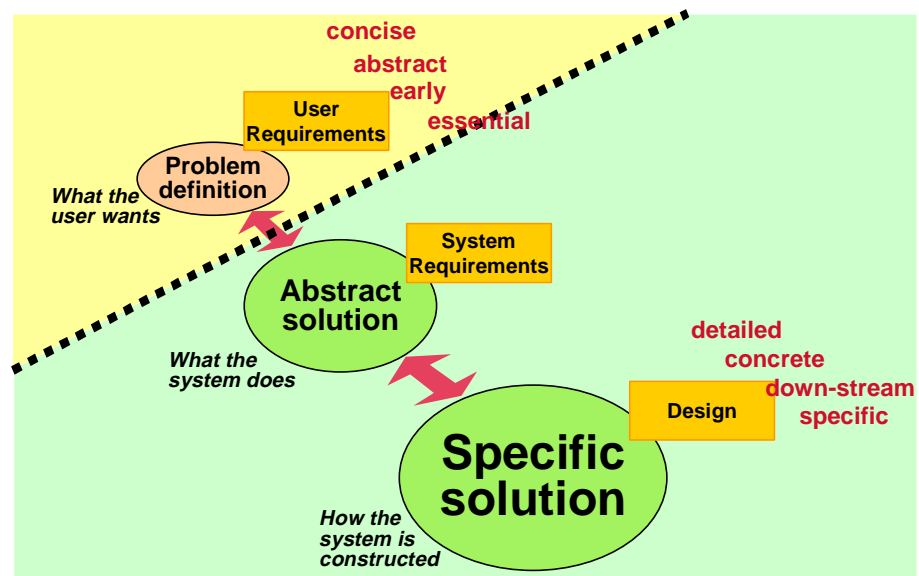
The "system requirements" are an elaboration of the customer requirements to a level of detail needed to plan the project's activities and work products and should be objective and verifiable. The subset of the system

requirements that will be implemented by the software project are referred to as the "system requirements allocated to software," or simply as the "allocated requirements." The systems requirements model represents the solution in abstract, often through a functional or behavioral breakdown. Typically this model is organized by functionality or behavior.

The design itself is a third (non-requirements) model. Clearly this is highly representative of the final solution. Many different designs can meet the requirements, and a change of solution may be possible without changing the requirements. Figure 2 shows the three different types of requirements in a more graphic representation. [STEVENS]

In addition to all of these types of requirements, there is a separate set of corporate requirements. These will include: reuse, marketing requirements, 'productization' requirements, coding, safety and industry standards, and many others. Many of these will be modeled as constraints in RtS to ensure their compliance in the final product. However, it is essential that these requirements are identified, and plans made to ensure that the final product satisfies these requirements.

Figure 2
Different Types of Requirements



What is Requirements Traceability?

Traceability is the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another. [IEEE]

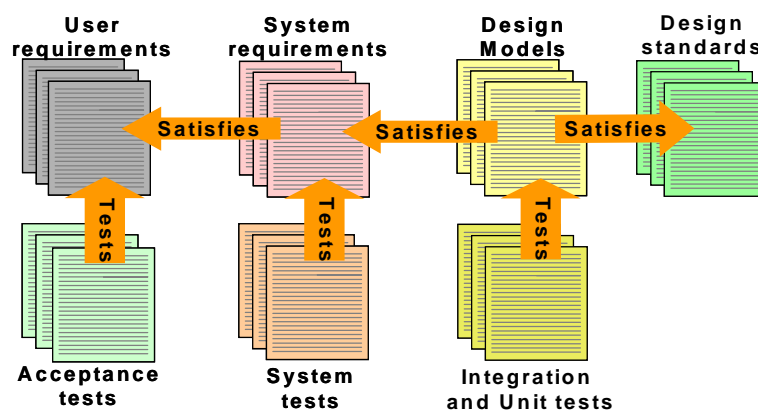
Requirements traceability is the ability to describe and follow the life of a requirement, in both a forward and backward direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases). [SEI]

Requirements tracing techniques were developed in order to influence the completeness, consistency, and traceability of the requirements of a system. They provide an answer to the following questions:

- What system need does a requirement address?
- What model elements implement a requirement?
- What requirements are satisfied by a model element?
- Is a requirement useful?
- Is a model element useful?
- What design decisions affect the implementation of a requirement?
- Have all requirements been allocated to either model elements or more detailed requirements?
- Does the design implement the requirements?
- What acceptance test will be used to validate a requirement?
- What will be the impact if a requirement is changed?
- What will be the impact if a model element is changed?
- Have all milestones been completed?

Traditionally, requirements tracing has been achieved by: cross-referencing, (embedding phrases like "see section x" throughout the project documentation and using a spreadsheet to keep track of them), specialized templates which are used to store links between documents, and restructuring, where the documentation is restructured into a network to keep track of changes. Figure 3 shows the different requirements models, and the relationships between them, and the verification tests.

Figure 3
Example Information Model



Defining the Rules

Before starting the project, the project manager and the team must determine and agree upon requirements traceability details. Three important questions need to be addressed:

1. **What needs to be traceable?**

This should address which documents should be included in the database, which should simply be referenced, how should they be organized, etc.

2. **What linkages need to be made and at what level?**

The design model will contain many distinct objects and relationships. The linking of a requirement to a model element should be done so as to have direct traceability, but to allow the model element to be modified without the need for modifying the link. This is covered in more detail in the Mapping of Requirements to Model Elements section.

3. **How, when, and who should establish and maintain the resulting database?**

If too many people have the ability to update the requirements linkages, then it will be difficult to maintain a consistent approach. Conversely, if the project manager does it, he may not have the necessary detailed knowledge of the design. This is covered in more detail in the Management of Links section.

Mapping of Requirements to Model Elements

Requirements can express many different things that will be expressed as model elements. They are described in the following sections.

Functional

Define how the system's users interact with the system, including typical scenarios and contingencies. These should be modeled as Use Cases. The advantage of modeling them as Use Cases is that they are at a sufficiently high level that the implementation of the Use Case as the Sequence Diagram will not affect the requirement. Additionally, because the model contains these relationships, the Sequence Diagram, its classes, and the operations used will also be linked to the requirement. Therefore if an impact analysis is performed on a requirement, all of these elements will also be listed. Since Use Cases describe functional requirements or scenarios, they will usually map to one or more system test scripts. Use Cases therefore provide the linkage between the requirement and the acceptance tests. If appropriate, timing constraints on the functionality may also be included, or may be defined on a Constraints Diagram. Detailed calculations or specific interactions can also be modeled as operations.

Non-functional or Operational Constraint

Define general constraints that apply to the system as a whole, or specific functionality, such as timing, reliability, graceful degradation etc. These will

be defined on Constraints Diagrams. The linkage between Constraints and the Use Cases to which they apply should be performed in order to further specify the acceptance criteria of the requirement.

Users or External Systems

People, other systems, or hardware devices that communicate with the system's software via interface devices or subsystems. These will be external to the system, and either influence, or are influenced by the system. These should be modeled as actors. Linking them to Use Cases documents the functionality available to each actor, or which an actor will trigger. Acceptance tests should verify that this functionality is available to appropriate actors. It should also check that functionality is not available to actors who are not linked to the Use Cases.

Technical Specification of a Component

These will include physical interfaces, for example displays, sensors, actuators, etc. These will also include the physical distribution of the system, showing all proposed or actual processing nodes, storage and interface devices, and the connections between nodes and devices. These requirements will be modeled as the elements found on the System Architecture Diagram such as boards, disks, buses, interface devices, etc.

Operational Modes

The valid modes (or states) that the system can be in, e.g. stopped, running, failed, training, and how the system transitions between the states. This may also include what functionality will be available in each mode. The mode information will be modeled to the modes on the System Modes Diagram, and the availability of functionality will be modeled as relationships between the modes and Use Cases. Acceptance tests should verify that this functionality is available in the appropriate modes. It should also check that functionality is not available in unlinked modes.

Need to be Remembered

These requirements describe persistent data or information that must be available during the execution of the system. More complex data structures will model to classes and tables, and simpler data will map to attributes.

Subsystems or Logical, Physical or Functional Groupings

In any non-trivial application some form of technique is needed to help manage the size and complexity of the system and object architecture for a domain. To accomplish this, classes can be organized into packages, each representing closely related functionality. Additionally, it may be necessary to group the physical elements of a node together to define a node; for example, a description of the computer and its disks and interface devices. These will be modeled as subsystems, which model physical groupings, and packages, which model functional and logical groupings.

Relationships

These requirements specify the connections between nodes and devices, and the connection between nodes, that is the physical relationships, as well as logical relationships between data. These will be modeled as associations between classes, and connections between physical devices in the system architecture model.

Increments and Milestones

RtS provides for the definition of increments that will be implemented. The first part of any planning activity should be to define the goals of each increment. These will generally map to a set of Use Cases that will be implemented, a set of constraints that will be met, or both. RtS provides the capability for increments to be defined, and Use Cases and constraints to be linked to the increments. The RtS reporting capabilities allow the user to view the individual elements such as the classes and interface devices that will need to be implemented for each increment. This is possible since the Use Cases are further refined as Sequence Diagrams.

Definition of DOORS and its Capabilities

DOORS, or **D**ynamic **O**bject-**O**riented **R**equirements **S**ystem, is an Object Oriented, multi-user information management and linking tool specifically designed to Manage, Search, Retrieve and Link textual and graphical data. Spreadsheets and/or databases can be configured to support requirements tracing. However, by using DOORS, which has been engineered for this purpose, the following capabilities are available:

- Bi-directional requirement linking to system elements
- Capture of allocation rationale, accountability, and test/validation
- Identification of inconsistencies
- Capabilities to view/trace links
- Verification of requirements
- History of requirements changes.

Overview of DOORS

DOORS has been designed to support Engineering Information Management and Traceability. One of the life cycle support features in DOORS is the ability to maintain links in separate Link Modules. This object-oriented feature allows different categories of links to be identified. As such, DOORS can manage and link any type of data. It is NOT restricted to methodology-specific information; rather, it is used *in conjunction* with tools that implement specific methodologies to fully manage the development lifecycle. Therefore RtS is used alongside DOORS providing a specific methodology while DOORS provides the requirements management and traceability on the project.

Characteristics of a Requirements Model

An individual requirement is meaningless in itself - it is a single element in a complete model. Some properties e.g., completeness or consistency are properties of the model as a whole. A requirements model must be a complete representation of what is known about the system at that point. The solution must meet some acceptable sub-set of the totality of requirements (perhaps the whole set). A set of requirements forms a complete model, representing the system completely at some level of abstraction. The user requirements show the user the complete set of results to be obtained by using the system. The system requirement might show everything the system must do, plus all the constraints on that functionality. DOORS is organized to encapsulate the project data into *Modules*. Modules represent any collection of similar data, and often represent documents, specifications, hardware or software lists, etc. [SMITH]

Linking Requirements

A substantial part of requirements work involves making links to other parts of the project. If we define a requirements model correctly, and check that the design meets the requirements, then we have traceability. One of the main roles of a good system standard is to limit the links required. By using the strategy outlined in the Types of Requirements and How to Model them, links to requirements can be kept to a minimum. Additionally, links may be used to show:

- Tests that will check the appropriate design or requirements;
- Relationships between development information structures (e.g. between configuration item structure, schedule, and work breakdown structure);
- The cloning of information across the project;
- That each systems requirement will be tested;
- That the rationale for critical requirements is documented.

Management of Links

Procedures and protocols for management of links will largely depend on the project structure, number of models, division of responsibility, size of teams, required formalism, etc. specified for the project. It will also depend on who is responsible for the requirements. Smith defines different levels of DOORS users: Repository Creators, Data Owners and Reviewers. [SMITH] The job title will vary from organization to organization and project to project. They are described in the following paragraphs.

Reviewers can examine the repository using pre-defined views or by viewing data exported to the Internet. They enter data or comments using pre-defined data-entry windows. Reviewers use the repository for their day-to-day jobs reviewing documents and populating specific attributes they might be responsible for, etc. Reviewers don't have full access to the data, and in general don't make un-approved changes to the requirements. Highlighting a requirement and submitting an electronic change request is one example of the Reviewers use of the tool.

Data Owners have more responsibility and training. They will populate DOORS within their area of responsibility. A requirements engineer or test engineer would develop requirements or tests directly in DOORS. They would be responsible for creating and populating attributes, making links, and so forth. As the name suggests, Data Owners have full responsibility for the integrity of the engineering data under their control. Software engineers, team leaders, and hardware engineers would be Data Owners.

Repository Creators are DOORS tool specialists. Creating and maintaining the data architecture, extracting performance metrics, specialized importing, and template creation all fall within their remit. In addition, they create any pre-defined data entry windows needed by the Reviewers using DOORS customization utilities. System engineers and software architects would be repository creators. [SMITH]

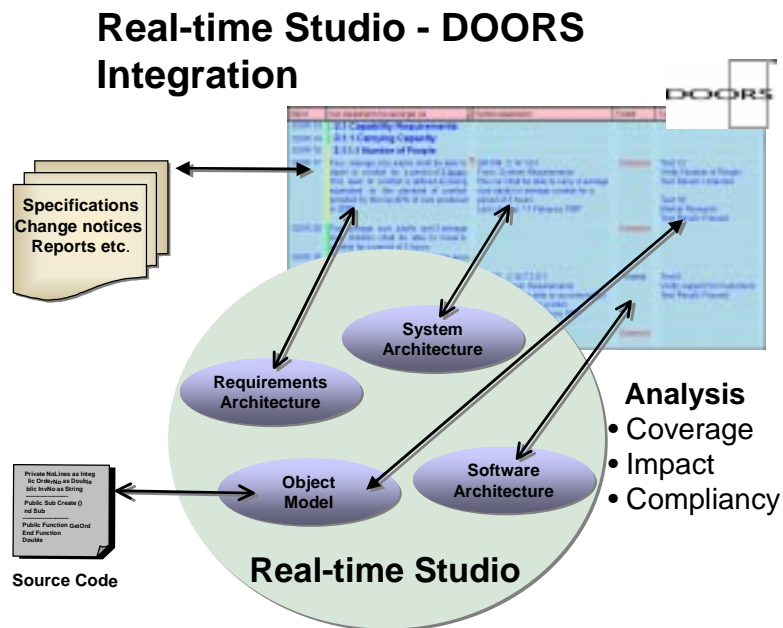
The ARTiSAN DOORS Synchronizer

The linking of a DOORS requirement to a Real-time Studio model element is achieved through the creation and maintenance of a 'surrogate' module and surrogate objects in DOORS. A surrogate module represents a Real-time Studio model. ARTiSAN DOORS Synchronizer creates a surrogate module the first time a DOORS project is synchronized to a Real-time Studio model. A DOORS project can contain many 'surrogate' modules, that is, one surrogate module for each Real-time Studio model the DOORS project is synchronized to. A surrogate object represents a Real-time Studio model element, that is, a dictionary item or a diagram. A surrogate object represents each model element in that model. The name of each surrogate object is set to that of the model element it represents. Each surrogate object has attributes that store information about its associated model element. These attributes can be included in views of the surrogate module. Each surrogate object serves two purposes:

- It represents a model element. When you link a requirement to a surrogate object, you are effectively creating a traceable link between a requirement and the model element the surrogate object represents.
- It can be used to view its associated model element in Real-time Studio.
- For classes, it can be used to navigate to the code files associated the class.

ARTiSAN DOORS Synchronizer creates and maintains surrogate modules and the surrogate objects they contain. In addition, ARTiSAN DOORS Synchronizer creates and maintains links between surrogate objects. These links represent relationships that exist between the model elements in a Real-time Studio model. To ensure that a history of deleted model elements is maintained, ARTiSAN DOORS Synchronizer does not delete surrogate objects. Figure 4 shows the links between DOORS, the documents, tests, code, RtS modeling elements, source code, and other project information.

Figure 4
Real-time Studio – DOORS
Integration



Synchronization Process

The synchronization process keeps the DOORS surrogate modules and surrogate objects inline with the Real-time Studio models they represent. The synchronization process can be started from either Real-time Studio or DOORS. It can be performed at two levels: DOORS surrogate module to Real-time Studio model, and DOORS surrogate object to Real-time Studio model element.

DOORS Surrogate Module to a Real-time Studio Model

This should be performed when a Real-time Studio model is synchronized with a DOORS project, and when a surrogate module needs to be brought inline with its associated model. The first time a DOORS project is synchronized with a Real-time Studio model, ARTiSAN DOORS Synchronizer:

- Creates a surrogate module in the active DOORS project.
- Creates in that surrogate module, a surrogate object for each model element in the active model.
- Creates links between the surrogate objects, where each link represents a relationship defined in the RtS model.

After the synchronization is complete the surrogate module and link module that are created in DOORS must be saved. Thereafter, when a surrogate module is synchronized with a Real-time Studio model, ARTiSAN DOORS Synchronizer performs an update creating new elements, marking removed elements as deleted, renaming elements, and redefining links to other DOORS modules.

It is important that the synchronization process is performed regularly, so that the surrogate modules and their surrogate objects accurately reflect the models they represent. However, synchronizing a surrogate module to a model is a complex process that may take a significant amount of time, and while the synchronization process is in progress DOORS cannot be used. The following strategy for using ARTiSAN DOORS Synchronizer is recommended.

- As an overnight administration task, synchronize a DOORS project with its Real-time Studio models, so that the project is up to date at the beginning of each working day.
- If a model element is created, synchronize the surrogate object to the model element from Real-time Studio.
- If a model element is deleted, synchronize the surrogate object to the deleted model element from DOORS.
- If a model element is changed, synchronize the surrogate object to the model element from either Real-time Studio or DOORS.

By saving a surrogate module before starting the synchronization process, the change bars can be used to review which surrogate objects were created or updated by the process.

Model Element Relationships

The model elements in Real-time Studio have relationships with each other. These relationships provide useful information when determining the impact of any changes that are made to a model element. For this reason, ARTiSAN DOORS Synchronizer represents these relationships in DOORS. To represent the relationships between model elements, ARTiSAN DOORS Synchronizer creates links between the related surrogate objects. Some relationships have an implied direction, for example, a Disk is based on a Disk Type. When a relationship has a direction, this direction is represented through the direction of the associated link in DOORS.

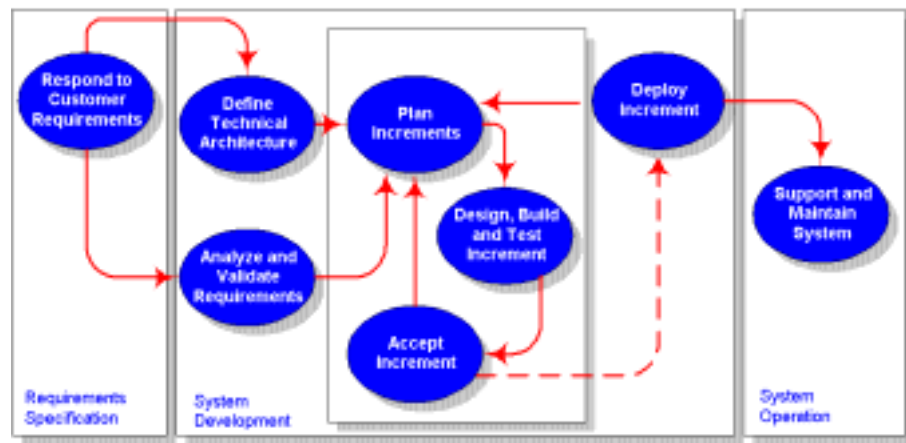
In the next section on Project Lifecycle, we show how links and their use will vary during the project lifecycle.

Project Lifecycle

Links and their use will vary during the different phases of the project lifecycle. It is important that the person performing the linkages has a clear understanding of the elements that are being linked. By linking a requirement with a model element that satisfies the requirement, a specific relationship is being created. If the model element does not in fact satisfy the requirement, this could lead to problems later on in the lifecycle. Figure 3 shows the relationships implied by links between the various modules. During the initial phases of requirements analysis, DOORS reporting procedures should be used to determine the extent of unsatisfied requirements, and those requirements for which verification tests have not been defined. As a result, dangling, un-implemented requirements can be seen using DOORS filtering features. "Orphaned" model elements, unlinked to requirements, can be

found quickly preventing unnecessary complexity. It is important that baselines of both the DOORS and RtS models are created at appropriate milestones in the project. This ensures that when the product is released to the customer or to the QA department, the requirements and models that define the product are in step with the release. Figure 2 shows the different types of requirements and their uses. Figure 5 shows the lifecycle stages of the incremental development process.

Figure 5
The Real-time Perspective
Incremental Process



Respond to Customer Requirements

At this stage, the user requirements must be captured and signed off. If user requirements have been specified in the form of a document, it should be parsed into DOORS, and change procedures should be established. At this stage, test scripts can be written and linked to the requirements. System requirements can then be derived.

If the user requirements are unknown or non-existent, these can be individually created in DOORS as they are discovered. Joint Application Development, (JAD) sessions and rapid prototyping are two good ways of discovering requirements. Using Altia Faceplate can help to validate user requirements by allowing the easy creation of functional user interfaces. Regardless of whether JAD sessions or rapid prototyping, (or both) are used, requirements must be documented and formally agreed. DOORS reporting facilities can be used at any time to generate a user requirements document.

Analyze and Validate Requirements

At this stage, user requirements will be translated into system requirements in the form of Use Cases and constraints. To be valid, they must support the user requirements. It will therefore be essential that the individual analysis elements be created in reference to the user requirements. During this stage, it will be essential to avoid constant update of individual model elements, or the update of the entire system where changes and the reason for the changes gets lost. Because of the model links which are included in the DOORS / RtS link, many of these problems can be alleviated. Upon the creation of a Use Case that satisfies a user requirement, the Use Case can be added to the surrogate model via the Synchronize Model Element tool command in RtS, and linked to the requirement. During the subsequent stages, Sequence Diagrams containing the interface devices and classes will be added to the model, which describe the Use Case. By updating the Use

Case model element in DOORS, these items are automatically added to the surrogate model along with their relationships. This approach should also be taken for other grouping objects such as subsystems where functionality is specified, but not internal composition. Some types of packages will also fall into this category such as operating systems.

Define Technical Architecture

During this stage, the initial Class Diagrams will be created, objects will be allocated to packages, Sequence Diagrams will be updated to encompass the newly defined classes, the System Architecture Diagrams will be further specified and State Diagrams will be created for dynamic classes. The purposes of this stage are, to provide a technology infrastructure, to serve as the basis for incremental development, and to prove to project sponsors that the proposed technology solution is valid. This stage provides the definition of the system requirements. At this stage, it is critical that links do not start to proliferate. A good rule of thumb to adopt is that links should be made only to elements in the requirements architecture unless it is absolutely necessary. This will not only cut down on the number of links which are made, it will allow for the design to evolve without constant deletion and creation of links.

Plan Increments

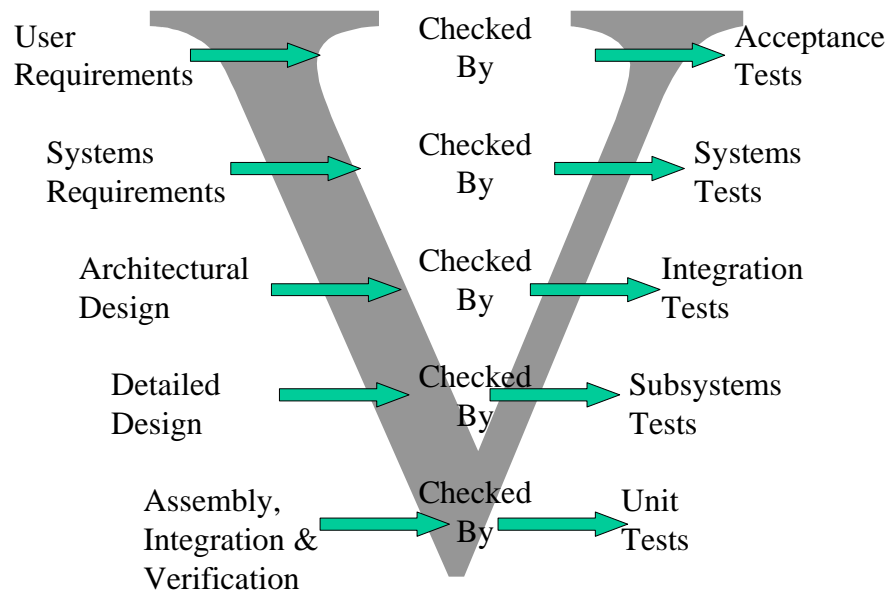
In some quarters, requirements management is seen as a technical issue, i.e. concerned with the product, but not with project management issues such as schedule or resources. However, project management actually manages interdependent issues - schedule, resources, and quality. Requirements must be the basis of project management and financial control. For example, "finishing" a design does not constitute a milestone. But supplying a design that efficiently meets the requirements is a real milestone. Similarly, when a product meets the requirements under test or during operations, and the item is ready for input to the next stage a meaningful milestone has been achieved.

Links should be made between project phases defined in any project documents and the increments defined in RtS. Because the surrogate model will contain the Use Case and constraint linkages, the available functionality and constraints will also be included. Additional requirements can also be linked to the increments to ensure that a full definition of the increment goals has been achieved.

Design, Build and Test Increment

Requirements are the basis on which systems are finally accepted (or rejected). The final product is tested against the requirements. The V-Model of verification in Figure 6 shows how the different models used to define the product are used sequentially for verification. It also shows the relationship between product, test system and the requirements.

Figure 6
‘V’ Model



Requirements to Manage Change of Iterative Systems

Change management will take place during the entire project lifecycle. If a change management regime is not in place, it will be difficult to track changes, to ensure that necessary changes are made and ensure that unnecessary changes are not made. Additionally, it will be impossible to adequately determine the impact of a change in either the implementation or the requirements. This can mean that the implementation of “One Small Change...” could have disastrous consequences.

When a change is proposed, the following must be performed at a minimum:

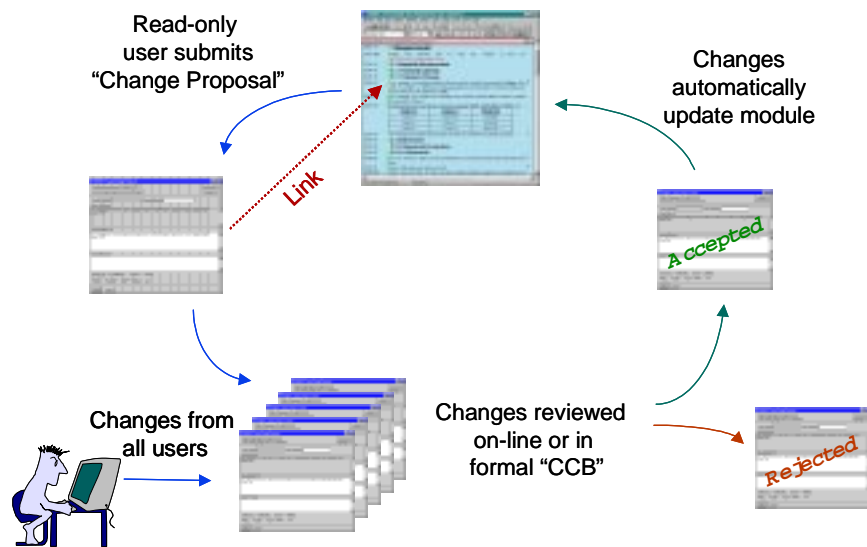
- Changes to the requirements should be reviewed with the software project team and the customer for correctness.
- The impact to milestones and changes to milestones must be determined and costed.
- If the change is to be made to the product, the impact on the requirements and the verification procedures that must be performed should be quantified.
- Changes to the requirements should be incorporated into the project.
- The products should be re-evaluated for consistency with the requirements

During development, changes to the requirements can come from both the customer and the developers. The developers may find better and easier ways to implement the system: different interface devices, different platforms, etc. Additionally, problems may be found such as hardware interface

mismatches, missing software functionality depended upon by another software component, or inadequate processing of data coming across a real-time interface.

After the product has been released, suggestions for change and problem reports become the basis for driving future evolution. These should be collated from the various sources. These include: customer help lines, acceptance test feedback, customer trials, internal QA procedures and competitor information. Figure 7 shows the various elements of a change proposal system.

Figure 7
Change Proposal System

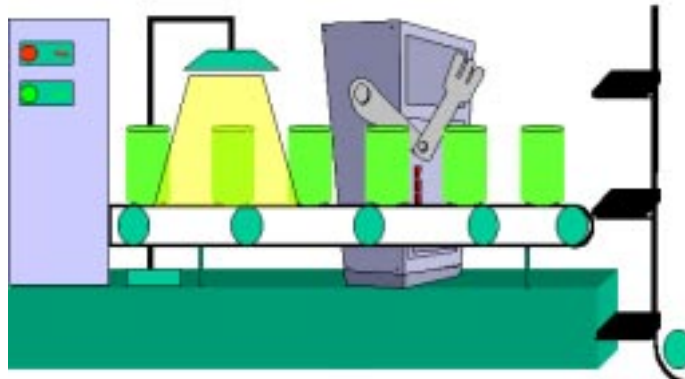


Example of Requirements Tracing

The following example describes an example system and how the different requirements can be modeled in RtS. For simplicity, we will just show the high level links defined by the models in the requirements architecture, namely, the System Scope Diagram, Use Case Diagram, Constraints Diagram and the System Modes Diagram. The following is a brief description of the system:

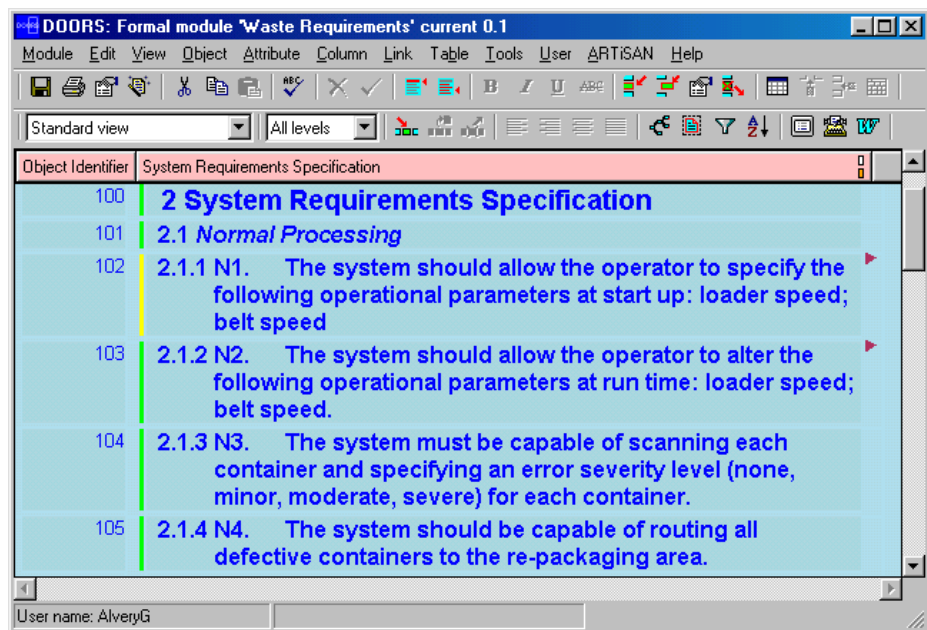
“The plant handles toxic containers. Containers are fed along a belt and are scanned for faults before routing. If the containers are badly damaged then they are routed off the belt by the robot arm for special handling; if not, then an elevator carries them off for normal processing. The plant is controlled remotely by an operator, although manual overrides are available.” The system is shown in Figure 8.

Figure 8
Waste Treatment Plant



The illustration shows the containers being scanned by the scanner as they are carried along on the belt. The robot arm removes defective containers, and correct containers continue along to the elevator. There is also a simple user control panel. A more formal description has been parsed into DOORS, and is shown in the Figure 9.

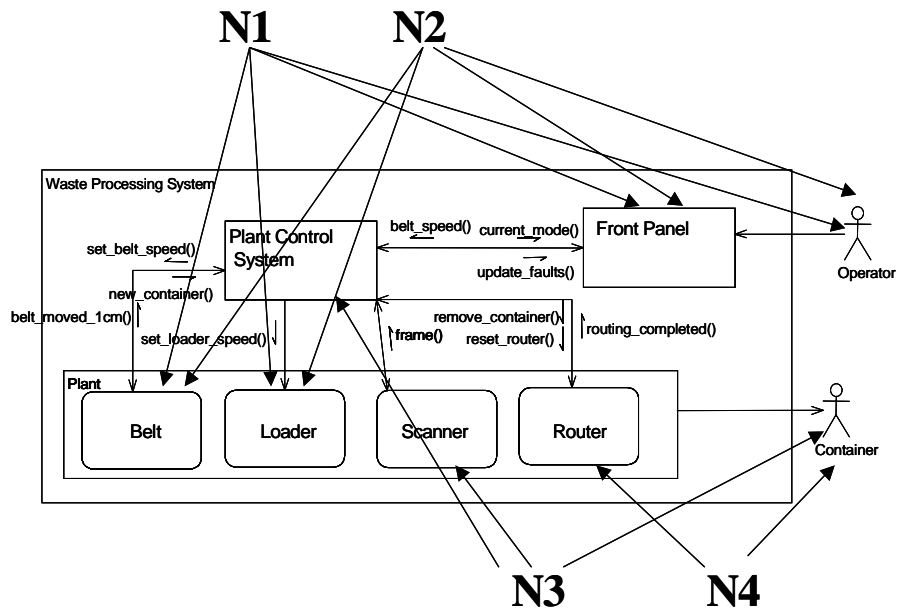
Figure 9
Requirements Document



Scope Diagram

Figure 10 shows the Scope Diagram that has been derived from requirements N1 through N4. Elements include the operator, container, belt, scanner, loader, router, user control panel, (shown as the front panel), and plant control system. To define the interfaces, the messages that will be generated by the interface devices, and received by them are also included.

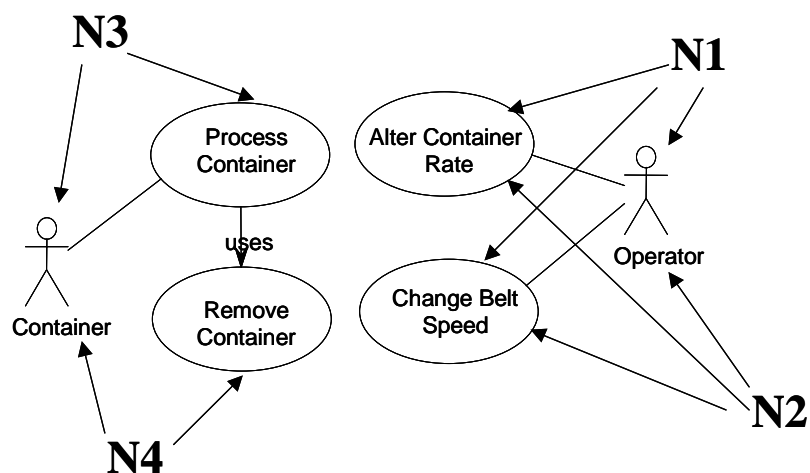
Figure 10
Scope Diagram



Use Case Diagram

Figure 11 shows the Use Case Diagram that has been derived from requirements N1 through N4. Elements include actors: Operator, and Container, and the Use Cases: Process Container, Remove Container, and Alter Container Rate and Change Belt Speed. These describe functional requirements. These should then be further specified as Sequence Diagrams to demonstrate how the system elements will interact in order to implement the requirements described in the Use Case.

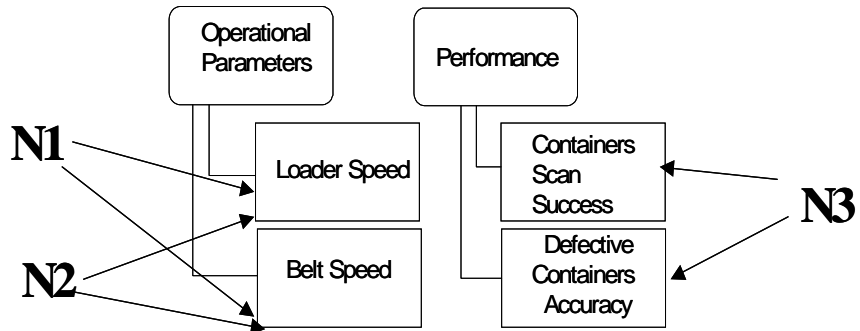
Figure 11
Use Case Diagram



Constraints Diagram

Figure 12 shows the Constraints Diagram that has been derived from requirements N1 through N4. Items fall into the categories of Operational Parameters and Performance.

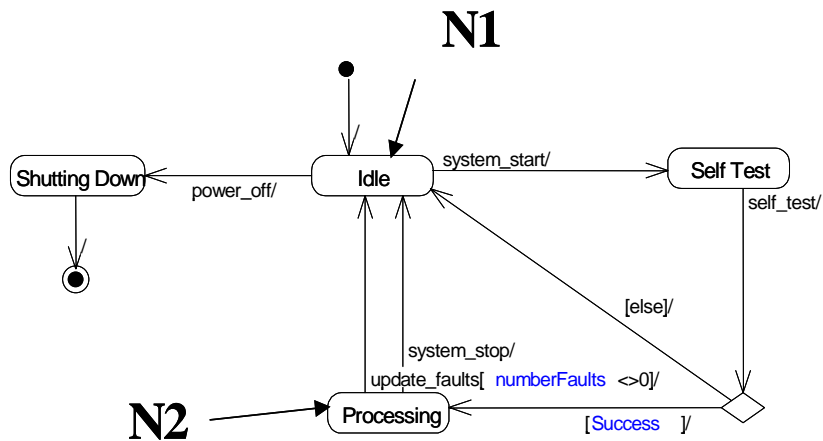
Figure 12
Constraints Diagram



System Modes Diagram

Figure 13 shows the System Modes Diagram that has been derived from requirements N1 through N4. Items include the Idle and Processing States. The additional states and transitions are derived from requirements that are described further on in the document.

Figure 13
System Modes Diagram



What Next?

The developer would continue through the requirements building up the model until all the requirements have been allocated to model elements. This would form the basis of the design that will implement the system.

Conclusion

Requirements management is a key activity for ensuring products are built to a high level of quality and conform to requirements. Requirements management will be employed at all stages in the development lifecycle. Using RtS, the requirements can be modeled to provide a closer mapping between the requirements and the implementation. By mapping requirements to modeling elements at the appropriate level, the consequential detailed design will not affect those relationships. DOORS can be used to provide additional requirements management facilities. These include change management, change history, links to test specifications and additional documents, and a host of others. Using the two together, developers can capture requirements for a real-time system in DOORS, and define and design the system in Real-time Studio, linking each requirement with its related elements at each stage of the life cycle. The result is that all members of the project team can ensure that the system delivered matches the original requirements.

Bibliography

- [BROOKS] BROOKS, Frederick "The Mythical Man-Month"
Addison-Wesley Publishing Company, 1995, ISBN 0-201-83595-9
- [CROSBY] CROSBY, Philip "Quality is Free, The Art of Making Quality Certain." New York: Mentor, New American Library, 1979
- [FLORAC] FLORAC William & CARLETON Anita "Measuring the Software Process: Statistical Process Control for Software Process Improvement"
Addison-Wesley Publishing Company, 1999 ISBN: 0-201-60444-2
- [GOLDBERG] GOLDBERG, Adele and RUBIN, Kenneth.S. "Succeeding with Objects"
Addison-Wesley Publishing Company, 1995, ISBN 0-201-62878-3
- [IEEE] Institute of Electrical and Electronics Engineers, IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.
New York, NY: 1990
- [MARSHALL] MARSHALL, Richard & CHEVALLIER Jean "Making Systems Engineering Accessible"
QSS Inc & CNES
- [MOORE] MOORE, A. and COOLING, N. "Real-time Perspective - Overview"
ARTISAN White Paper, 1997
- [SOMMERVILLE] SOMMERVILLE, Ian "Software Engineering"
Addison-Wesley Publishing Company, 1993, ISBN 0-201-56529-3
- [SMITH] SMITH, William B "Best Practices: Application of DOORS to system integration"
QSS Inc.
- [STEVENS] STEVENS, Richard & MARTIN, James "What is Requirements Management?"
QSS Inc.
- [SEI] Requirements Tracing -- An Overview
Software Engineering Institute
<http://www.sei.cmu.edu/activities/str/descriptions/reqtracing.html>