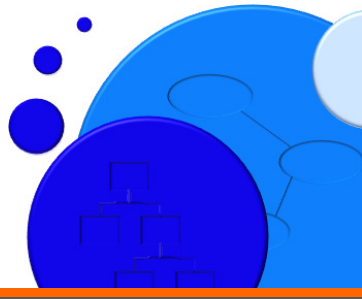
Cradle-7
From concept to creation...

3SL

Role and Representation of System Requirements in Systems Engineering Using Cradle

RA008/02 March 2017



http://www.threesl.com
salesdetails@threesl.com
support@threesl.com

Structured Software Systems Ltd (3SL)
Suite 2, 22a Duke Street
Barrow-in-Furness
Cumbria LA14 1HH, UK
Tel: +44 (0) 1229 838867
Fax: +44 (0) 1229 870096
Regd: 2153654 VAT: GB 473 2757 28

Contents

List of Figures

List of Tables

Introduction

This is one in a series of white papers that discuss the role and representation of different types of systems engineering information in any agile or phase-based process in organisations that produce products.

This paper is concerned with: system requirements

Subject

Once a project's context is known from its *stakeholders'* needs, and these needs have been refined into a set of user requirements, there will be a range of designs that could achieve what is wanted.

System requirements must be defined to express what is wanted in a form that allows the suitability of a design to be determined before it is built.

Solution

This paper describes how *system requirements* should be derived from user requirements, structured in a Cradle database, and used to engineer products.

Terminology

Table 1: Terminology

Term	Definition
Product	A physical or logical object that exhibits behaviour to change its environment
Need	Something that can be achieved by the realisation of requirements in products
User Requirement	An externally visible characteristic to be possessed by a product
System Requirement	A characteristic that a component of a product must possess for the product to be able to satisfy its user requirements
Validation	A check that an externally visible characteristic of a product is as required
Verification	A check that an internal characteristic of a component of a product is as required
Test	A check to exercise a product component to confirm its behaviour or structure
Project	A set of activities that create or update products and their related information

Concept and Purpose 1

System requirements define the characteristics of products' internal components, or more generally, of the system to be developed. They act as constraints on the design of these components. In simple terms:

- User requirements define ***what*** is to be provided
- System requirements define ***how*** it is to be provided

System requirements are the most common and most numerous of the requirements managed by any project. They are often labelled with names that reflect the nature of the system or products being defined, or the process being used. For example:

- Product requirements
- Common requirements
- Module requirements
- Technical requirements
- Domain requirements

are all system requirements, see section "Terminology" on page 1. We are concerned with the information for product-orientated processes. For simplicity, we refer to this information as system requirements.

Creating system requirements is often the most difficult of all systems engineering tasks, because it requires engineers to:

- Imagine the components that will be needed in the product
- Imagine these components' possible designs or implementations
- Define constraints on these components' characteristics (such as their behaviour, physical limits, accuracies and tolerances) in a way that will satisfy the user requirements

Specifying user requirements is relatively straightforward. They express what a *stakeholder* (a user, maintainer, or purchaser and so on) of a product wants to perceive about the product externally, such as what it will do, what it will cost, what it will weigh, and what its electrical interfaces will be.

Writing system requirements is much more difficult, since:

- They are statements about the characteristics of components that do not yet exist
- That will be combined into a design or architecture that also does not yet exist
- Of a product that does not yet exist

and then asserting that:

- ***If*** the components are built into a product
- ***And if*** these real components achieve their characteristics
- ***Then*** the stakeholders will have a product that meets their needs

System requirements are almost retrospective, in that if the product already exists and meets users' needs, then its components' characteristics could be measured and these measured values would be written as the system requirements, which will then be known to definitely satisfy the user requirements and hence the users' original needs. But the product does not exist and its components' characteristics cannot be measured. Nonetheless, values for these characteristics must be defined

and used to constrain the design of components that do not exist for a product that does not exist.

That these statements are circular explains why system requirements are so difficult to write, Hence the transition from user requirements to system requirements is both a large and very complex step.

To reflect the difference in language, subject and emphasis of system requirements:

- User requirements are ***validated***, to ensure that the product has the characteristics required
- System requirements are ***verified***, to ensure that the products' characteristics have been provided correctly
- Product components are ***tested***, to ensure that they behave correctly and have been manufactured correctly

Content and Creation 2

System requirements are constraints on the possible design of your products' components. They do not specify how the components will be designed, but they do specify the constraints to which these designs must conform, for example by specifying:

- Materials and manufacturing tolerances to be used for mechanical components
- Voltage and current ranges and tolerances, or RF interference levels, or voltage transition times, for electrical components and interfaces
- Functionality, behaviour, data formats, interface protocols or response times for software components

System requirements are critical to the successful design and build of any system. They are a ***bridge*** between the desires of stakeholders and the design that realises these desires. They create this bridge by:

- Translating the (often technical) languages of stakeholders into the (very different) languages of the designers, builders and testers
- Specifying technologies and technical details
- Being very precise, which typically means being very numerical

For example, where a user requirement may say:

- ***The emergency air supply will allow 5 people to breath normally for not less than 4 hours.***

then the system requirements that satisfy this statement could be:

- ***Emergency air will be delivered at 1 bar +/- at most 5%.***
- ***The oxygen concentration in emergency air will be 18-25%.***
- ***The reservoir of emergency air will be not less than 19,200 litres, including a 100% safety margin.***

As shown in this example, there are often more system requirements than user requirements, and we must justify or explain how the system requirements satisfy their user requirements. In this case, a person breathes 8 litres of air per minute, so 5 people breath 5 x 8 x 60 litres in an hour and adding a 100% margin = 2 x 5 x 8 x 60 x 4 = 19,200.

System requirements will always be written using language that will:

- Specify characteristics of the internal components of a product component
- Be either functional or non-functional
- Be singular (often termed ***atomic***), so words such as **"and"**, **"both"** and **"also"** are typically not appropriate
- Be precise
- Be measurable and testable

Although atomic statements are generally helpful in all information, it is particularly important for system requirements since they appear in the majority of traceability and coverage analyses, typically presented in ***compliance tables*** (such as system requirements to needs, or system requirements to test cases) or matrices. These analyses use the cross references to and from each system requirement. If a system requirement contains a list of paragraphs, then cross references for each system requirement will refer to all of the statements in the list inside it, and not to an individual statement in this list. This will mean that the analyses have more links to and from each item, making the analysis more complex and more difficult to understand and to check. When requirements are atomic, there will be fewer links to and from each requirement which means that the analyses of these links will be clearer and easier to complete and review.

It is essential to specify the ***scope***, or ***level***, of a system requirement. This represents the ***largest product component*** whose characteristic is being specified, such as:

1. A system requirement that states:
   "The maximum RF emission in the 0.3 to 3.0 MHz band will be less than 100 mW/cm2 averaged over any 30 minute period."
   relates to the entire product, so its scope would be: **Product**
2. A system requirement that states:
   "All parts of the product casing will be fabricated from 1.3 mm (16 gauge) 5052 aluminium."
   relates only to the product casing, so its scope could be either **Part** or **Assembly**, hence the scope would be set to the largest of these values: **Assembly**
3. A system requirement that states:
   "All user documentation must be supplied on 80gsm A4 white paper, 3-folded along the long side (landscape)."
   relates to each document supplied with the product, so its scope would be: **Part**

Specifying the scope of a system requirement is intrinsically valuable, adding extra information that aids the understanding of its content, meaning and intent. It also helps when the traceability of user to system requirements is checked, and when system requirements are allocated to the system breakdown structure. Specifying scopes allows engineers to divide the system requirements into groups with the same scope, and then to consider each group, starting at the highest scope and progressing to the lowest scope.

The scope of a user requirement does not represent its position in the system requirement hierarchy. The system requirement hierarchy is likely to contain many levels of requirement, structured as described in the next section, in which the higher levels are used to provide grouping in a manner that is logical and easy to understand. It is likely that only the lowest levels and the ***leaves*** in this hierarchy will contain requirements, and it is only in these lower-level requirements that the scope values will be valid and meaningful.

The system requirement statement that is finally agreed is, of course, of paramount importance. However, as time passes, it becomes equally important to be able to understand why the requirement ***is as it is***.

This does not mean to simply understand the evolution of the system requirement's statement into its then current form. A detailed record of that evolution is always immediately available through the ***change history*** associated with the item. Rather, it is important to understand ***why*** the statement is as it is and why it is not ***something else***.

This means that we strongly recommend that a ***rationale*** is recorded for each system requirement. This rationale could include many things, but we consider the most important guidelines to be that it should contain:

- All of the ***alternatives*** that were considered
- ***Why*** a particular alternative was chosen
- The ***assumptions*** that were used to make this decision
- For numerical values, sometimes termed ***measures of performance*** (MoPs), why that particular value, or range, have been specified and why any tolerances applied to these value(s) are as they are

The assumptions are a particularly important part of the rationale. They are often associated with then current technology or techniques. Over time, these assumptions can easily become invalid, and a review of system requirements must include a re-appraisal of the assumptions to re-confirm their validity. Assumptions that have become invalid with advances in technology or techniques could easily change which of the alternatives should become the system requirement, or indeed introduce new alternatives for the system requirement's statement.

System requirements are said to ***satisfy*** user requirements. This means that if a system requirement is met, then the user requirements that it satisfies will also be met. Therefore, cross references are created between a system requirement and the user requirements that it satisfies. Sometimes these cross references are ***parametised***, for example to show that a system requirement ***fully*** satisfies one or more user requirements, or only ***partially*** satisfies these user requirement(s).

Both the context and content of user requirements (what external stakeholders can perceive) and system requirements (constraints on characteristics of design components) are so radically different that it is, in general, impossible for stakeholders (users, buyers, maintainers and so on) to review the cross references between user and product requirements and to agree that these links are correct.

Nonetheless, there are projects where attempts are made to seek

stakeholders' approval of the traceability between user and system requirements. We suggest that this is almost always impossible. We recommend that it is only your engineers who review and approve the:

- Content of system requirements
- Cross references between user and system requirements

Creating system requirements is a highly skilled task. Given that system requirements are the basis for all design, implementation and testing work on all products, which in turn are the majority of your product development costs, system requirements and their cross references must be reviewed very carefully before being used as the basis for design and implementation work.

Levels 3

Since writing system requirements is so difficult, we recommend that they are written in *levels* corresponding to the assembly or packaging of logical (such as software) and physical (such as mechanical and electronic) components in a product. Starting with user requirements that specify externally-observable characteristics at product level, a corresponding level of system requirements is created and then reviewed. Once agreed, this level can be decomposed.

Typical levels are: **Product**, **Subsystem**, **Assembly** and **Part**.

The levels are used in system requirements, product components and potentially test details, verifications and validations.

Label each level with a name that is meaningful for all engineering disciplines and define what that level means for each discipline.

For example:

Table 2: Definition of Levels

Level	Description	Use by Engineering Disciplines			
		Hardware	Software	Mechanical	Other
Product	The entire product	The entire product	Full software build for all processors in the product	The product as shipped to customers	Shipped materials, documentation and shipping container
Subsystem	The highest level product components	Major components, such as power and data exchange, often a single board or ECU	Complete build of each executable, process or thread for one processor	Principal physical components, such as a casing, drive train, or user controls	Items supplied with the product, such as user manuals, shipping materials, and product labelling and certifications
Assembly	Combinations of basic parts, the subject of *make or buy* decisions	Minor component, such as a daughter board, connection block, or wiring loom	Set of source files that are built into an executable, process or thread	Elements of the major assemblies, often the FRUs (field replaceable units)	Elements of other items, such as document text files templates for labels and other markings
Part	Lowest level at which product components are managed. Identified by an in-house or supplier part number and version	Individual chip, connector, resistor and so on	A source file	Physical things made in-house or bought-in such as a bearing, shaft, rotor or panel	Basic pieces of ancillary item, such as boxes, labels, cardboard

Creation Process 4

A recommended process to create system requirements is:

1. Define a set of levels to organise requirements.
2. For each of these levels, from highest to lowest:
   a) Identify those user requirements that apply at this level. We recommend that all user requirements have a scope to indicate how much of the product is being defined by the user requirement. These scopes also identify the level of the corresponding system requirements.
   b) Imagine the components that would need to exist to meet these requirements, with reference to:
      i) Any higher-level components that may have already been imagined
      ii) Previous experience
      iii) Previous products
      iv) Domain knowledge within your organisation
   c) Imagine the characteristics that these components would have, such as performance, power consumption, resolution, noise levels, and so on
   d) Imagine the constraints to be defined for these characteristics for it to be possible to design these components
   e) Write system requirements for each characteristic and constraint
   f) Link the new system requirements to the user requirements
   g) Review the system requirements and cross references
   h) Expand the set of system requirements to the next level by repeating from step a) above

Structure and Organisation 5

Since system requirements will be derived from the user requirements and needs, which in turn are organised by projects, we propose that there will be one hierarchy of system requirements for each project. Each system requirement hierarchy will contain a mixture of different types of system requirements.

All hierarchies will have the same organisation, because it will be confusing to users if each hierarchy is organised differently to the others.

It is your decision how to organise the system requirements in each hierarchy. System requirements could be organised by types, such as:

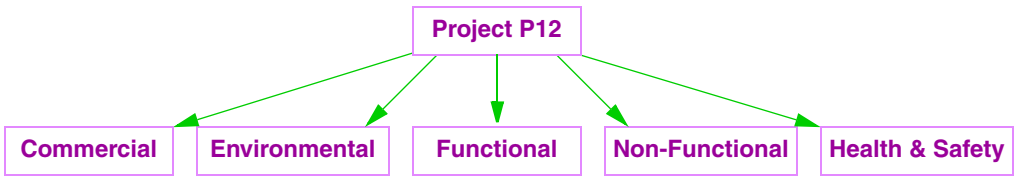


Figure 1: System Requirements by Type

or they could be organised by discipline, so that the names in the top two levels of each hierarchy would be:

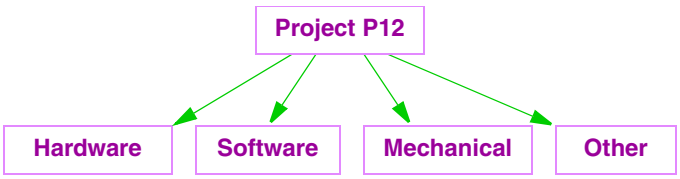


Figure 2: System Requirements by Subject

or they could be organised using the products' lifecycle, how products will be used, such as:

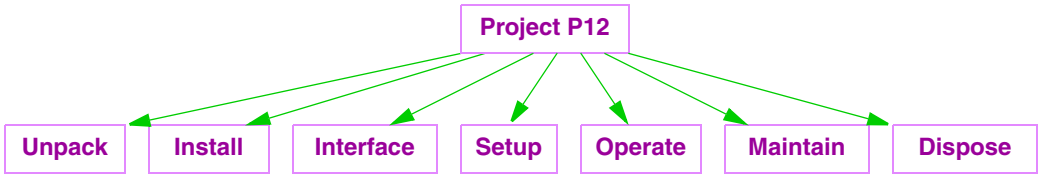


Figure 3: System Requirements by Lifecycle

We recommend the first of these examples, that system requirements are structured into hierarchies by their type. We believe that this is the easiest approach, because:

- It is easier for engineers to consider the functional, health & safety and so on, aspects of product components than to think of product components, and their characteristics, in any other way.
- So the system requirements will be easier to write and review

This is a different convention to that used for user requirements, but we believe that this is outweighed by the advantages listed above.

Traceability 6

Each system requirement will be linked to the user requirements from which it has been derived. We propose that only the bottom-level user requirements are linked to system requirements, which will have further decomposition. So the links will be from bottom-level user requirements to middle-level system requirements.

This will apply at each of the system requirement **Levels**, such as **Product**, **Subsystem** and so on. There will be links between the bottom-level requirements at one requirement **Level** and the top-level requirements at the next requirement level, that is:

- From bottom-level **Product**-level requirements to top-level **Subsystem**-level requirements
- From bottom-level **Subsystem**-level requirements to top-level **Assembly**-level requirements
- From bottom-level **Assembly**-level requirements to top-level **Part**-level requirements

The top of each project's system requirement hierarchy will be linked from the **PROJECT** item for that project:

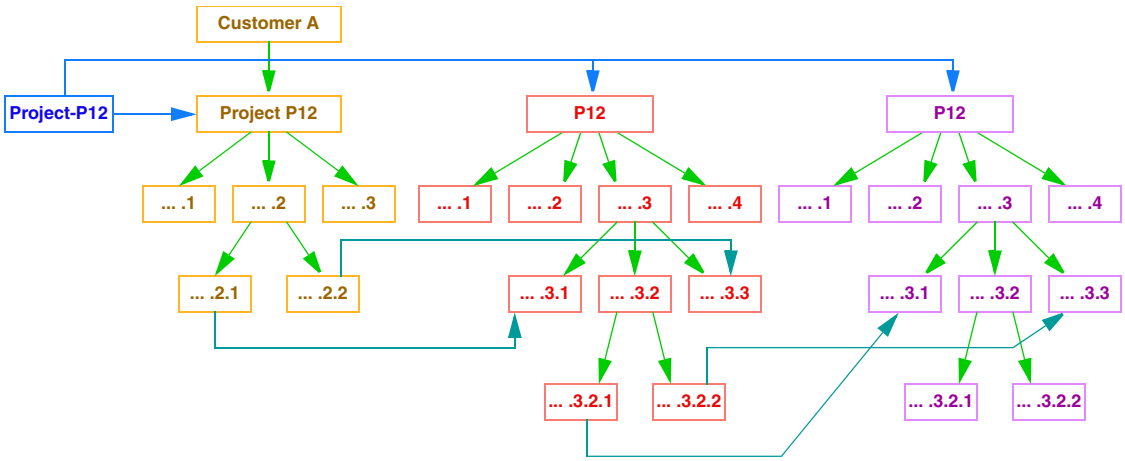


Figure 4: Projects, Needs and All Requirements

We recommend that traceability matrices are produced between system requirements and user requirements.

Database Representation7

The same structure can be used for all system requirements, so they can be represented by one item type called **SR**. This item type will be hierarchical. The **SR** items will be auto-numbered to allow reordering of the hierarchies. The hierarchical numbers in these hierarchies will be stored in the **SR** items' **Key** attribute.

Since there will be one system requirement hierarchy for each project, the format of the **Key** attributes in the user requirements will be:

project.hierarchical-number

such as:

Project-P12.1.2
P4-2016.2.3.4

where the names of the **.1**, **.2**, **.3** ... items in the hierarchy will be based on the structuring criterion you choose. If this is type, the names of the **.1**, **.2**, **.3** ... items in each hierarchy will be such as **Commercial**, **Environmental**, **Functional** and so on.

System requirements will require attributes to:

1. Name the system requirements, as a summary or shorthand, and potentially only used in system requirements that have children, as a convenient name for that group of related requirements
2. Specify the system requirement, as:
   a) Plain text, and/or rich text, with also
   b) An optional figure (JPEG) and table (RTF)
   c) Optional Excel, Word, Visio or PDF documents
3. Provide the rationale for the system requirement being as it is
4. Provide notes for the system requirement
5. Characterise the system requirement:

- **Discipline**, as one of: **Hardware**, **Software**, **Mechanical**, **Sales**, **Marketing**, **Other**
- **Maturity**, as one of: **Accepted**, **New**, **Rejected**, **Pending**
- **Priority**, as one of: **High**, **Medium**, **Low**
- **Scope**, as one of: **Product**, **Subsystem**, **Assembly**, **Part**
- **Type**, as one of: **Commercial**, **Environmental**, **Functional**, **Non-Functional**, **Health & Safety**
- **Verification**, one of: **Inspection**, **Analysis**, **Demonstration**, **Test**

which could be set for system requirements at all levels, but must be set for bottom-level system requirements

The method chosen to structure system requirements could replace the **Type** attribute. Despite an overhead to maintain it, we keep the attribute in the item type to simplify searches.

Database Schema 8

System requirements are one set of database items. A simple schema for product development processes is (item type names in parentheses):

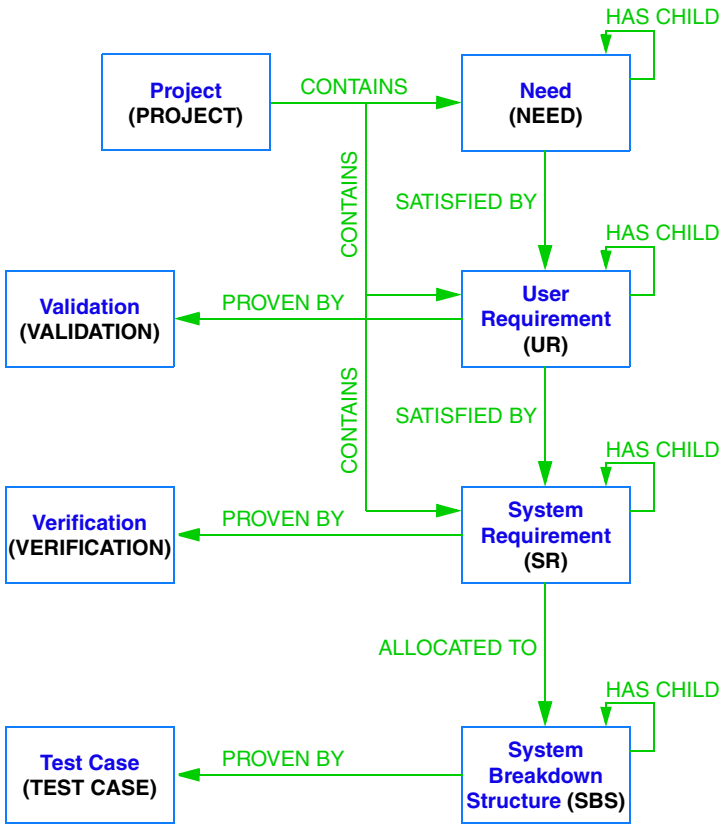


Figure 5: Database Schema

Verification 9

Validation means to check that "we built the correct product" and verification means to check that "we built the product correctly".

There are two distinct verification activities, one related to the product being built and described by the database, and one that confirms the correctness of the database itself:

- Verification items that describe the checks to be applied to the product components to ensure that system requirements have been met
- The act of checking all sets of **SATISFIED BY** cross references to ensure that the items at the to end of these cross references are indeed a correct evolution of the items at the from end of the cross references

This second activity is a manual check by users using traceability and coverage views produced by Cradle and their engineering knowledge and experience.

A common use for cross reference attributes is to record the result of reviewing cross references. To do this, a link attribute such as **Status** could be defined with values **Approved**, **Rejected** and **TBD**. An approved cross reference is a cross reference whose **Status** attribute has the value **Approved**. One or more navigations would be defined that either only follow, or specifically exclude, cross references with this value in their **Status** attribute. These navigations could be used to produce all the reports and documents that are part of the project's formal deliverables.

In this way, not do such deliverables only contain reviewed and approved items, but they will only contain links that have also been reviewed and approved.

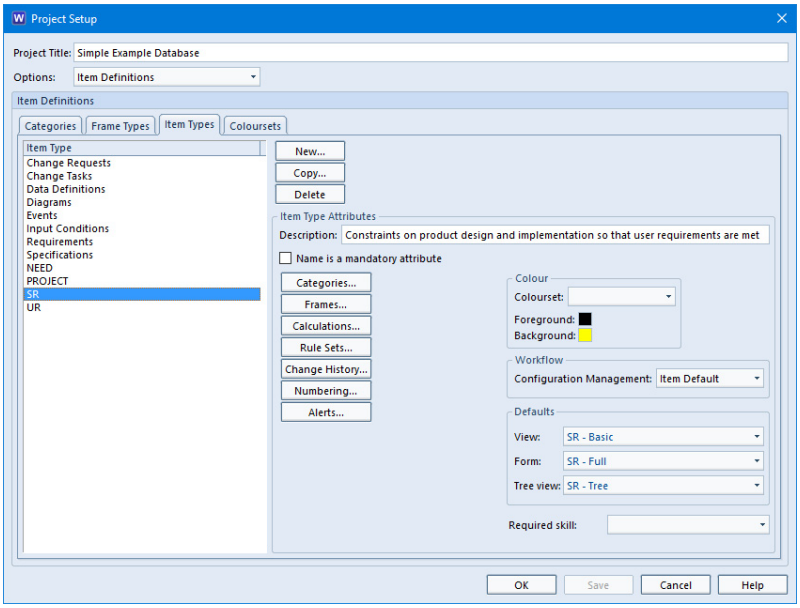
Although we recommend this verification technique, it is not often used in practice and so has not been included in the schema definition described in the following sections.

Item Type 10

The *item* is the basic unit in a Cradle database. Each system requirement will be an item of type **SR**, defined as:

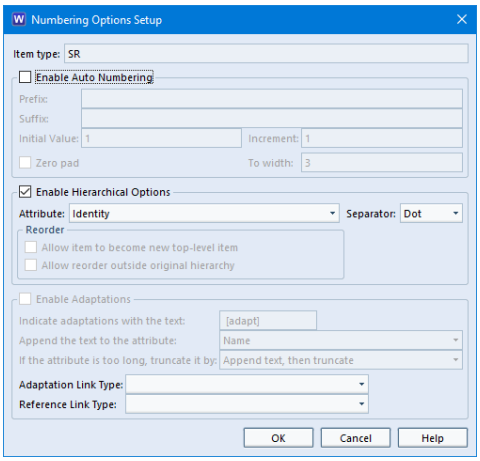
Table 3: Item Type Definition

Name	Categories	Frames		Auto-Numbering	Hierarchical	Adaptations	Change History
		Name	Type				
SR	Discipline Maturity Priority Level Verification	CALCS	EXCEL (XLSX)	Yes SR-*n* *n*=1,2,3...	Yes, in the Key	No	Yes
		DIAGRAM	VISIO				
		DOCUMENT	WORD (DOCX)				
		FIGURE	JPEG				
		NOTES	Plain text				
		RATIONALE	Plain text				
		RICH TEXT	RTF				
		SLIDES	POWERPOINT (PPTX)				
		TABLE	RTF				
		TEXT	Plain text				



In this item type definition:

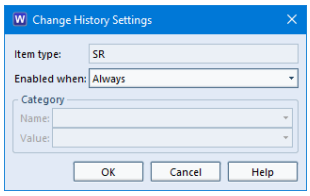
1. *Auto-numbering* is enabled, so all system requirements have a unique identity that will not change even if the system requirements are reordered in the hierarchies. We recommend that items are auto-numbered.
2. *Hierarchical* is enabled, with the hierarchical value stored in the **Key** attribute and with a period or dot (.) as the level separator in these hierarchical values. Enabling this feature means that Cradle will provide the **New Child**, **New Sibling**, **New Hierarchy** and **Reorder** operations for system requirements, and other capabilities.



3. *Adaptations* is disabled. This feature is not necessary for system requirements.
4. *Change histories* are enabled, and set to **Always** so that all changes are always recorded in system requirements. Other options are available for recording change histories.
5. *Categories* are the attributes used to characterise the system requirements. They are discussed in the next section.
6. *Frames* are large attributes that contain the data inside the item. They are discussed in a later section.

There are other options for each item type, such as:

- The colours used to show items in trees and Hierarchy Diagrams (graphical views of traceability and coverage)
- The *form* used to show items individually
- The *views* used to display system requirements by default (unless a specific view is chosen), and to show the items in trees

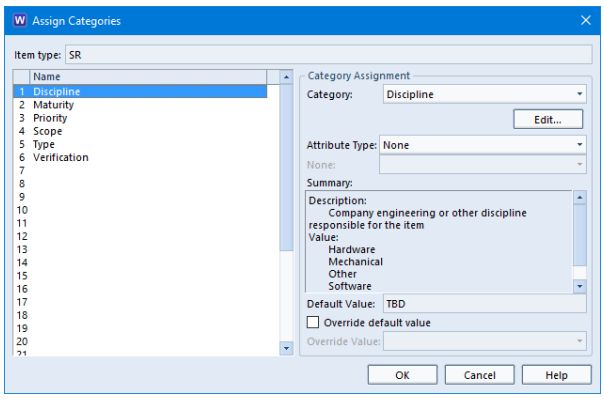


You can control users' access to items of the type with a *skill*. We recommend that such access controls are ***only*** added when you know that your process and project *organisation structure* (*users* and *teams*) require this facility.

Category Codes 11

Category codes are small data values used to ***characterise*** database items. You can define any number of category codes and assign up to 32 of them to each item type. Category codes are defined once and used in many item types. Cradle is optimised to find items based on category code values, and stores all items pre-sorted by all categories so filtering and sorting by category value is very efficient.

System requirements can be characterised in many ways. We suggest you use categories for each of the unused structuring methods (see



section 5 “Structure and Organisation” on page 7), in this case type and discipline, and to represent maturity, importance and validation method.

The categories defined in the schema for system requirements are:

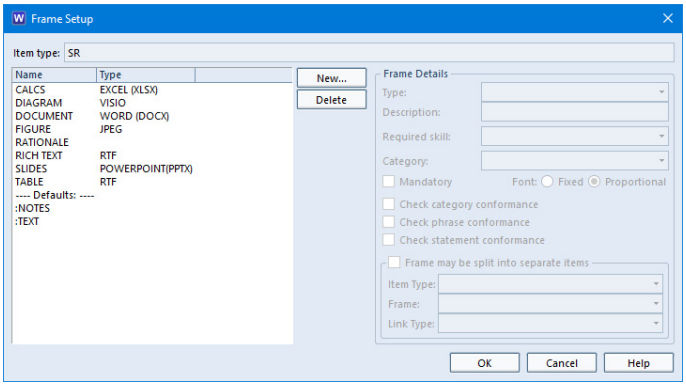
Table 4: Category Code Definitions

Name	Type	Values (* = default)	Description and Purpose
Discipline	Single-value pick-list	**Hardware** **Software** **Mechanical** **Sales** **Marketing** **Other** **TBD***	Which group or engineering discipline is responsible for creating the system requirements
Maturity	Single-value pick-list	**Accepted** **New** **Rejected** **Pending** **TBD***	How mature or complete is the system requirement. This is also used to avoid deleting system requirements when their authors have removed them. Such system requirements may be re-introduced, so keeping them available saves time. Such system requirements may be linked to other items, so keeping them makes it easy to find items that link to redundant system requirements, so these items can be considered for removal. It also allows the justification for everything in the database to be traced back to system requirements that are accepted.
Priority	Single-value pick-list	**High** **Medium** **Low** **TBD***	The importance of the system requirement, which will be based on the priority of the user requirements that the system requirement satisfies. In agile projects, this is a determinant of the ordering of the feature backlog, so that the earliest sprints or iterations will implement those user stories that satisfy the highest priority system requirements.
Scope	Single-value pick-list	**Product** **Subsystem** **Assembly** **Part** **TBD***	The level or scope of the system requirement. This helps to group the system requirements when design components or verifications are being linked to them.
Type	Single-value pick-list	**Commercial** **Environmental** **Functional** **Non-Functional** **Health & Safety** **TBD***	The type of the system requirement. This is the type of information in the system requirement. It helps to group user requirements that refer to the same type of product characteristic, and helps to determine how the user requirement will be satisfied, and by whom.
Verification	Single-value pick-list	**Inspection** **Analysis** **Demonstration** **Test** **TBD***	The verification method that will be applied to the product to demonstrate its compliance with the system requirement. The verification method is closely tied to the **Type**.

Frames 12

Frames are large attributes (up to 1 TByte each) that contain the data inside the item.

Each frame has an underlying data type, called a ***frame type***, that



defines the type of data, how and where it is stored (in Cradle, in a file, in another tool or at a URL), and how to operate on the data (view, edit, check and so on).

We suggest a set of frames that should be sufficient to manage any information in your system requirements. You can ignore any frames that are not relevant to you.

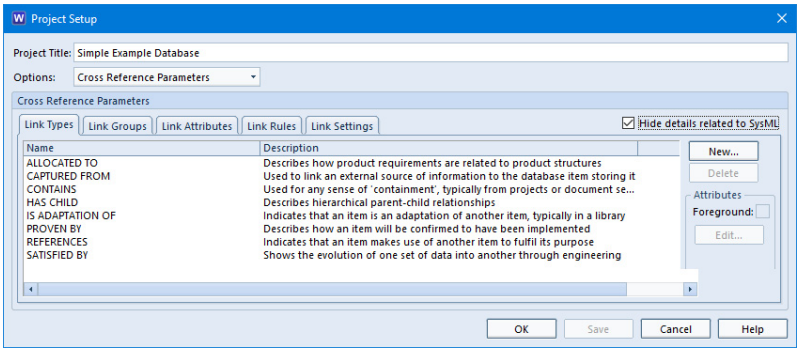
The frames defined in the schema are:

Table 5: Frame Definitions

Name	Type	Description and Purpose
CALCS	**EXCEL (XLSX)**	Store calculations or any other information in a spreadsheet
DIAGRAM	**VISIO**	Store any diagram or sketch that is a Visio drawing
DOCUMENT	**WORD (DOCX)**	Store any document held in Word
FIGURE	**JPEG**	Store any image
NOTES	**Plain text**	Provide explanatory notes for the user requirement
RATIONALE	**Plain text**	Why the user requirement is as it is, and why it is not something else. This could include many things, but we recommend that, as a minimum, it contains: • All of the ***alternatives*** that were considered • ***Why*** a particular alternative was chosen • The ***assumptions*** that were used to make this decision
RICH TEXT	**RTF**	The user requirement statement, if it must be represented in rich text, using fonts, colours, formulae and so on
SLIDES	**POWERPOINT (PPTX)**	Store any information that is a presentation
TABLE	**RTF**	Store information held in a table for edit by WordPad, Write, Word
TEXT	**Plain text**	The user requirement statement, stored here except if it must be rich text

Link Types 13

Cross references describe relationships or dependencies between database items. They can optionally have a ***link type***, chosen to describe the relationship. Items can be simultaneously linked by any number of cross references with M:N ***cardinality*** provided that these cross references have different link types.



There are many link types in the schema. The majority of these are needed for Cradle's SysML implementation. You can use these link types or define your own. You can hide the SysML-related definitions in all parts of the schema, as shown in the figure.

The types of cross reference used in the schema in Figure 5

"Database Schema" on page 10 are:

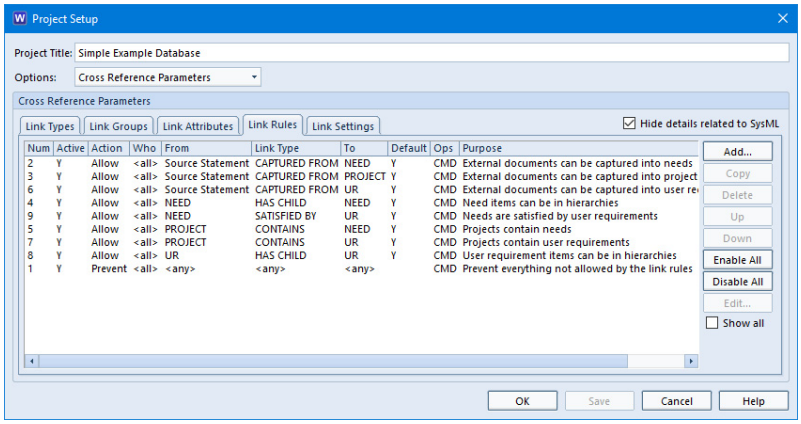
Table 6: Link Type Definitions	
Name	**Details**
ALLOCATED TO	Describes how product requirements are related to product structures
HAS CHILD	Describes hierarchical parent-child relationships
PROVEN BY	Describes how an item will be confirmed to have been implemented
SATISFIED BY	Shows the evolution of one set of data into another through engineering

Cross reference attributes have not been defined in this schema. These attributes can be free-format text or pick-lists of possible values. They can be used to select the cross references to use to find linked items in situations such as:

- Produce a view or report
- Find the expansion of an item in a tree node, or a table row
- Show a list of linked items in a Cradle UI
- Show a list of linked items in a form used to edit an item
- Show items in nested tables
- Display relationships graphically in a *Hierarchy Diagram*
- Publish a document

Link Rules 14

The *link rules* in the schema either allow or prevent operations on cross references. Initially, the list of rules is empty, so it has no effect. When a user does anything with cross references, Cradle checks the link rules, from first to last, to find a rule that matches the operation. If a match is found, the rule allows or prevents the operation. If there is no match, the operation is allowed.



Link rules can be generic, or very specific. They can refer to:

- One or all item types
- For items with stereotypes, part of Cradle's support for MBSE (model based systems engineering), one or all stereotypes, or a stereotype and its hierarchy
- All items of the type, or items matching a specific criterion
- All users, a group of users, or a single user
- One or more cross reference operations

To define the link rules for a schema, we recommend:

- Define a link rule that prevents all operations by all users on all cross references between all item types
- Precede this rule by one rule for each set of links shown in Figure 5 "Database Schema" on page 10

This approach is easy to understand, as the link rules specify the only cross reference operations that are to be allowed.

It is also helpful to add a purpose to each link rule. If a user performs an operation that violates a link rule, then the purpose of that link rule is shown to the user, to explain why the operation is not allowed.

Navigations 15

Navigations are usually created to filter cross references by link type. If two sets of items **A** and **B** are linked by more than one type of cross reference, create navigations that filter by each link type, so users can see which **B**s are linked to each **A** by the first or second link type, and vice versa for **A**s linked to each **B**.

There is only one type of cross reference between each pair of item types in the schema, so extra navigations are not needed and you will only need some of the navigations supplied by 3SL.

Since the needs will be auto-numbered, the most useful of these as-supplied navigations are those that sort linked items by their **Key** attribute. These navigations are:

Table 7: Useful As-Supplied Navigations

Name	Description
Bidirectional by Key	Follows all types of cross reference bi-directionally (upwards and downwards), sorting the linked items by their **Key**
Downwards by Key	Follows all types of cross reference downwards, sorting the linked items by their **Key**
Upwards by Key	Follows all types of cross reference upwards, sorting the linked items by their **Key**