Cradle-7
From concept to creation...

3SL

Role and Representation of System Requirements in Systems Engineering Using Cradle

RA008/07 February 2023

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

ISOQAR REGISTERED
UKAS PRODUCT CERTIFICATION 0026
Certificate Number 16926
ISO 9001

Contents

List of Figures

List of Tables

Introduction

This is one in a series of white papers that discuss the role and representation of types of systems engineering information in agile and phase-based processes in organisations that create products.

This paper discusses: ***system requirements***

Subject

A project context is defined by ***stakeholders***' needs. These needs are refined into a set of user requirements. There will be a range of designs that could achieve what is wanted.

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

Solution

We describe deriving ***system requirements*** from user requirements and how to structure them in Cradle for further engineering.

Terminology

Table 1: Terminology

Term	Definition
Product	A physical / 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	Characteristic of a product component for the product to be able to satisfy its user requirements
Validation	Check that an externally visible product characteristic is as required
Verification	Check that an internal characteristic of a product component is as required
Test	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 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 "Terminology" on page 1. We are concerned with information for product-related processes. For simplicity, we refer to this information as system requirements.

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

- Imagine what components are needed in the product
- Imagine the possible designs for these components
- Define constraints on the components' characteristics (such as their behaviour, physical limits, accuracies and tolerances) so as to 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 define 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 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 system requirements, which will then be known to satisfy the user requirements and hence the users' original needs. But the product does not exist and its components' characteristics cannot be measured. Nonetheless, 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 the language, subject, content 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 their manufacture and behaviour are correct

Content and Creation 2

System requirements are constraints on the design of products' components. They do not specify the design of the components, but they do specify the constraints to which the designs must conform, for example 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 (often technical) language of stakeholders into the languages of designers, builders and testers
- Specifying technologies and technical details
- Being very precise, typically 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 it could be:

- ***Emergency air will be supplied at 1 bar +/- at most 5%.***
- ***Oxygen concentration in emergency air will be 18-25%.***

- ***The reservoir of emergency air will be not less than 19,200 litres at 1 bar, 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.

It is important to set a ***scope***, or ***level***, for each system requirement. This is 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 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."
   refers only to the casing, which will be an assembly or a part, so its scope would be: **Assembly**
3. A system requirement that states:
   "All user documentation will be supplied on 80gsm A4 white paper, 3-folded along its long side (landscape)."
   relates to each document, 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.

System requirements will be written in 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 for all information, it is vital for system requirements since they appear in most 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 cross references to and from each system requirement. If a requirement contains a list of paragraphs, then cross references for that 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, so the analyses of these links will be clearer and easier to complete and review.

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 understand the evolution of the system requirement into its current form. A record of that evolution is always available in the item's ***change history***. Rather, it is important to understand ***why*** the statement is as it is and why it is not ***something else***.

We strongly recommend that a ***rationale*** is recorded for each system requirement. This could include many things, but, as a minimum, we believe that it should record:

- 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 based on current technology or techniques. Over time, these assumptions can become invalid, and a review of system requirements must include an appraisal of the assumptions to confirm their validity. Assumptions that have become invalid with advances in technology or techniques could easily change which of the alternatives should be the system requirement, or indeed

introduce new alternatives for the system requirement.

System requirements are said to ***satisfy*** one or more 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 links are ***parametised***, for example to show that a system requirement ***fully*** satisfies, or only ***partially*** satisfies, each of the user requirement(s).

Both the context and content of user requirements (what external stakeholders 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
- Links between user and system requirements

Creating system requirements is a skilled task. Given that system requirements are the basis for all design, build and testing work, which in turn are the majority of the 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 (mechanical and electronic) components in a product. Starting with the user requirements that specify externally-observable characteristics at product level, the corresponding level of system requirements is built and reviewed. Once agreed, this level can be decomposed.

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

These levels are used in system requirements, product components and potentially test details, verifications and validations. We will use an attribute called **Scope** to store the level of an item, to avoid confusion with the term level in the context of a hierarchy.

We 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 a product, such as 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	Basic component. Identified by an in-house or supplier part number and version.	Individual chip, connector, resistor and so on	A source file	Things made in-house or bought-in such as a shaft, rotor or panel	Basic pieces of ancillary item, such as boxes, labels, cardboard

Creation Process 4

A recommended process to write 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 in your organisation
   - **c)** Imagine the characteristics that these components will have, such as performance, resolution, power consumption, noise levels, and so on
   - **d)** For each characteristic, imagine the constraints needed to be able to design the components
   - **e)** Write system requirements for each characteristic and constraint
   - **f)** Link the new system requirements to the user requirements in step **a)** above
   - **g)** Review the system requirements and their links
   - **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 hierarchy will contain a mixture of 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 system requirements in each hierarchy. System requirements could be organised by types, such as:

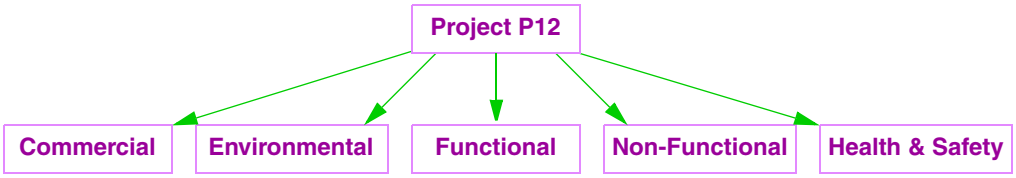


Figure 1: System Reqs by Type

or by discipline, such as:

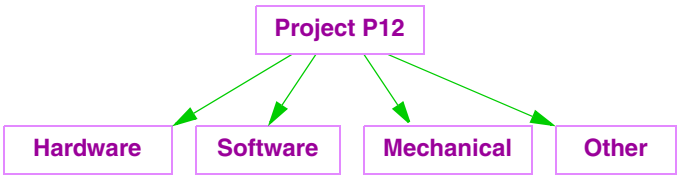


Figure 2: System Reqs by Subject

or by the products' lifecycle (how it will be used), such as:

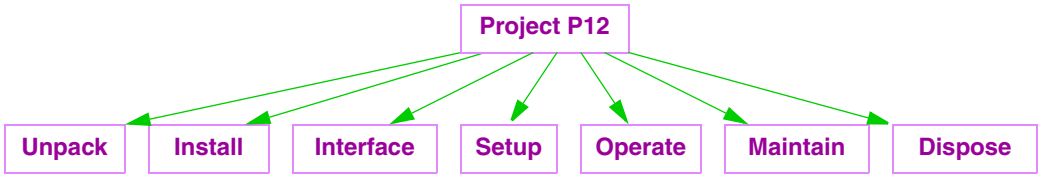


Figure 3: System Reqs by Lifecycle

We recommend the first option, 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 think about the functional, commercial (and so on) characteristics of components than to think about components in any other way
- Such requirements are easier to write and review

This is the same convention used for needs, but different to that recommended for user requirements. 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. This is mainly because higher-level user requirements are likely to group related information, so only the bottom-level requirements will have content. These bottom-level items will have different **Scope** values.

The **Scope** of a system requirement and the **Scope** of the user requirements that it is linked to (that it satisfies) will be the same. So a **Product** user requirement will be linked to one or more **Product** system requirements, and so on.

It is likely that the system requirements that are linked to user requirements will have further decomposition, so the links between these items will be from bottom-level user requirements to middle-level system requirements.

In turn, it is very likely that the decomposition of a system requirement at one **Scope** will be system requirements with a lower-level **Scope**. These system requirements may be linked to user requirements that also have that more detailed **Scope**. An artificial example with all levels of **Scope** in a hierarchy where each **Scope** appears in one level:

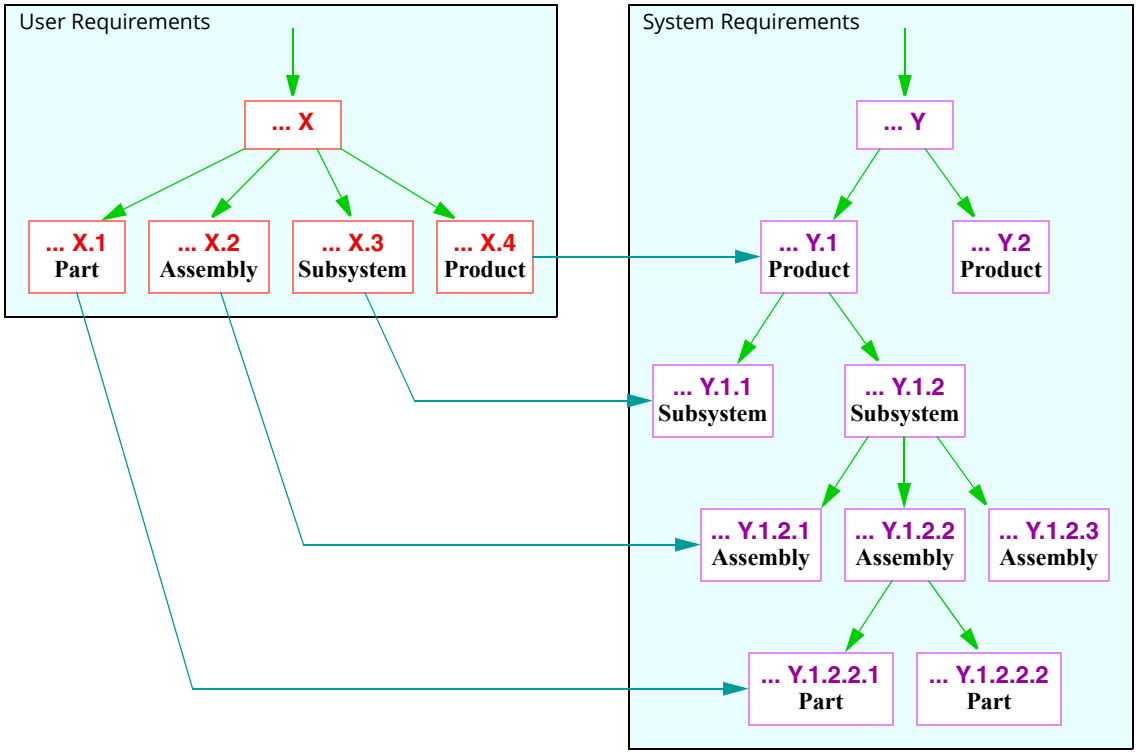


Figure 4: Scope of User and System Reqs

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

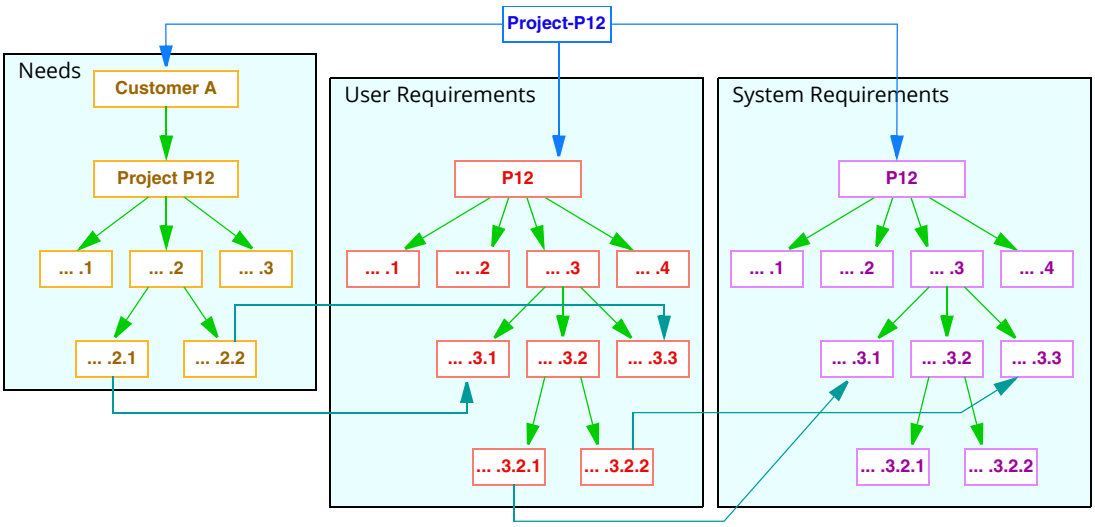


Figure 5: Projects, Needs and All Reqs

We recommend that traceability matrices are defined as views between system and user requirements.

Lifecycle 7

An organisation will need to define its own process for the development of system requirements, particularly if it is viable for customers and other stakeholders to be part of the user requirements' review. The end stage of such a process is that the system requirements are agreed by all parties involved, so the system requirements can be:

- Formally reviewed and baselined, if the project intends to use a formal configuration management (CM) approach, such as that provided by Cradle's Configuration Management System (CMS)
- Used as the basis for the development of the SBS

Several steps can be imagined in such a process. The path that each user requirement can take through these steps is called a ***workflow***. The position of each requirement in this workflow is recorded in an attribute in that item. We will use the name **Maturity** for this attribute.

Your organisation must decide what will happen if a system requirement is no longer required. There are two choices:

1. Delete the system requirement from the database
2. Keep the system requirement in the database for the historical record (it may reappear in the future) with a **Maturity** value that you use to exclude such system requirements from the sets of items that you use for the rest of the project

We recommend the second of these approaches.

Assuming that some stakeholders (probably not customers) are to be involved in the review of user requirements, we

suggest the following workflow (**Maturity** attribute values are shown in parentheses):

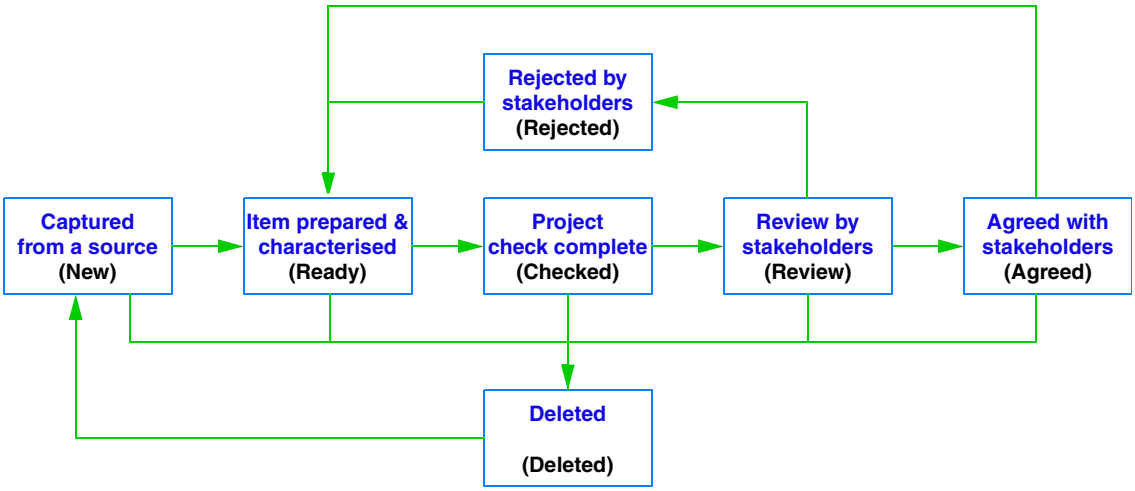


Figure 6: Workflow

Table 2: Workflow Stages	
Stage	**Description**
New	The item has been captured from an external document or entered manually
Ready	The item's statement has been cleaned of any compound or ambiguous statements and its attributes have been set to reflect its content and the structuring method used by the project
Checked	The item's content and attribute characterisation has been checked and is acceptable
Review	The item is being reviewed by stakeholders
Agreed	The item has been accepted by stakeholders
Rejected	The item has been rejected by stakeholders whose comments have been recorded in the item and it is to be reworked so as to become acceptable
Deleted	The item is no longer needed and will not be included in any further processing of items of this type

The workflow supports the idea that **Agreed** system requirements may be modified as the project progresses, perhaps after being baselined.

Database Representation 8

The same structure is 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 system requirements' **Key**

attributes 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 chose. If this is type, the **.1**, **.2**, **.3** ... items in each hierarchy will be named **Commercial**, **Environmental**, **Functional** and so on.

System requirements will require attributes to:

1. Name the system requirement, acting as a summary
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 requirement as it is
4. Provide notes for the system requirement
5. Characterise the system requirement:

   - **Discipline** : as one of: **Hardware**, **Software**, **Sales**, **Mechanical**, **Marketing**, **Other**
   - **Maturity** : as one of: **New**, **Ready**, **Checked**, **Review**, **Agreed**, **Rejected**, **Deleted**
   - **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**
   - **Validation**: as one of: **Inspection**, **Analysis**, **Demonstration**, **Test**

   which could be set in all levels of system requirement, 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 9

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

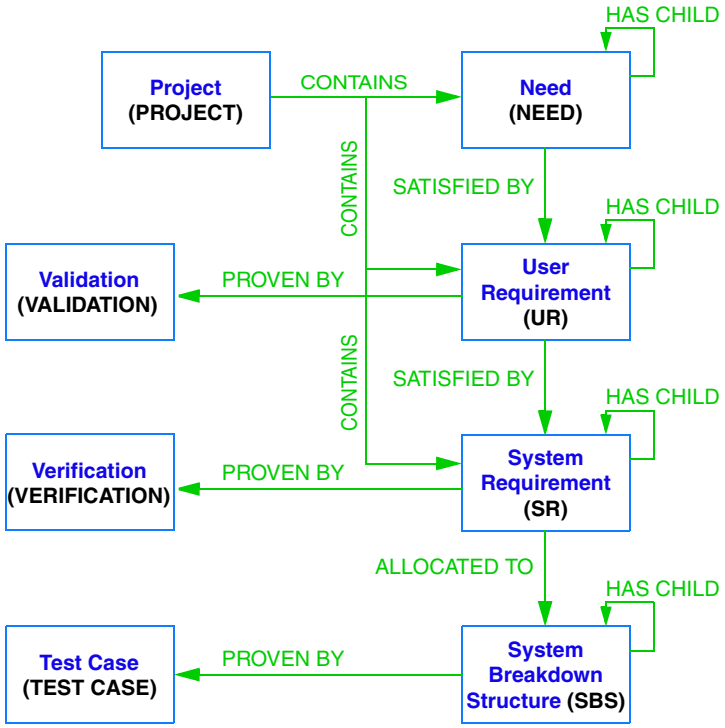


Figure 7: Database Schema

Verification 10

Validation is to check that "we built the correct product" while ***verification*** checks that "we built it correctly".

There are two distinct verification activities, one related to the product and described in 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 a correct evolution of the items at the **from** end of the links

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 11

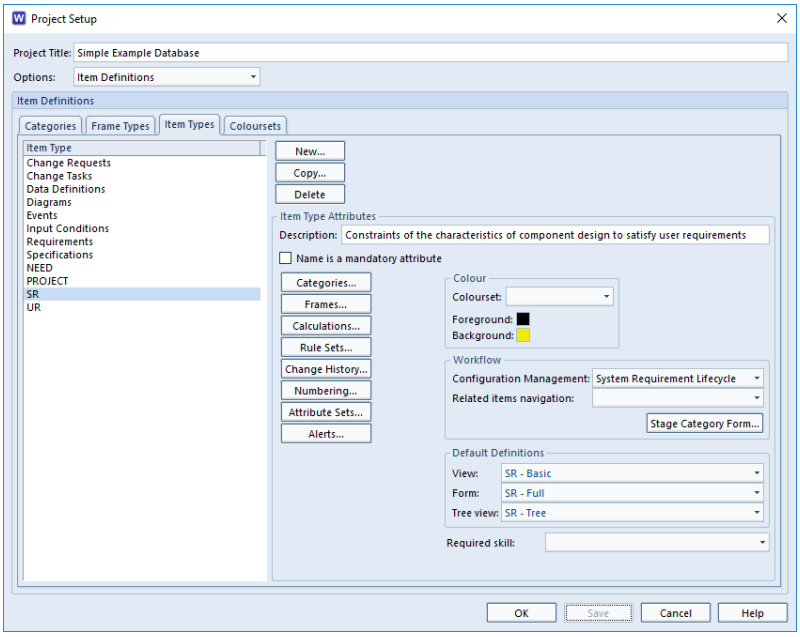
The ***item*** is the basic unit in a Cradle database. Each system requirement will be an item of type **SR**, defined as:

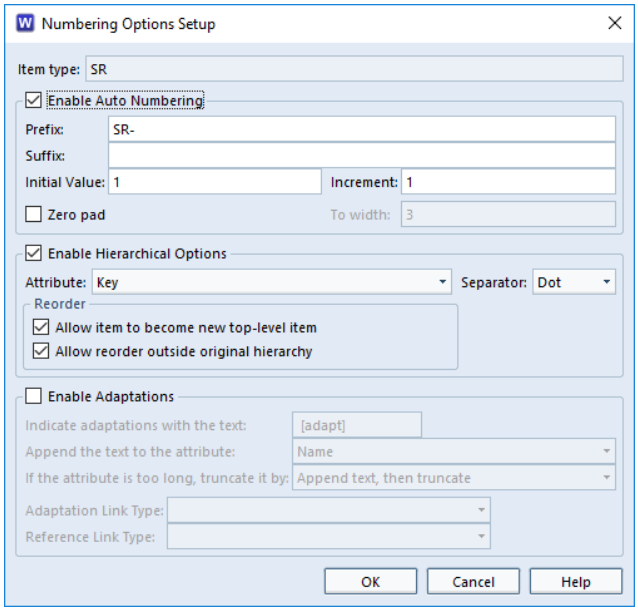
Table 3: Item Type Definition

Name	Categories	Frames			Hier-archical	Adap-tations	Change History
		Name	Type	Auto-Numbering			
SR	**Discipline** **Maturity** **Priority** **Scope** **Type** **Validation**	**CALCS**	**EXCEL (XLSX)**	Yes **SR-***n* *n*= 1,2,3...	Yes, in the **Key**	No	Yes
		COMMENT	Plain text				
		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 system requirements have a unique identity that is fixed even if they are reorganised in their hierarchies. We usually recommend that items are auto-numbered.
2. ***Hierarchical*** is enabled, the hierarchical value is the **Key** attribute with a level separator dot (.) in the 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.



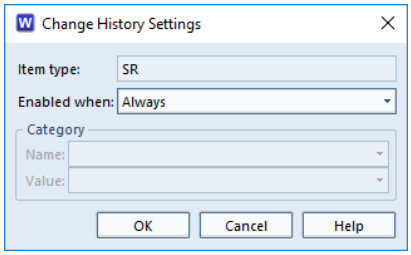


This feature is not needed for system requirements in this schema, but can be essential for system requirements in other contexts.

4. ***Change histories*** are enabled, and set to **Always** so 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 appear 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:

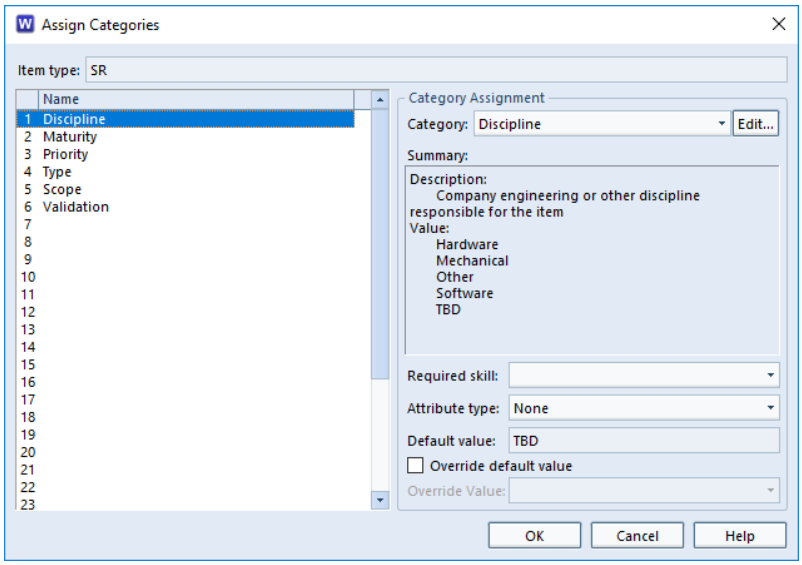
- The colours used to show items in trees and ***Hierarchy Diagrams*** (HIDs) for graphical views of traceability
- The ***form*** used to show items individually
- The ***views*** to show system requirements (defaults if a specific view is not chosen), and to show them in trees



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 it.

Category Codes 12

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 different ways. We suggest that you use categories for each of the structuring methods (see "Structure and Organisation" on page 8) that were not used to group the system requirements, in this case importance,

engineering discipline, maturity and validation method.

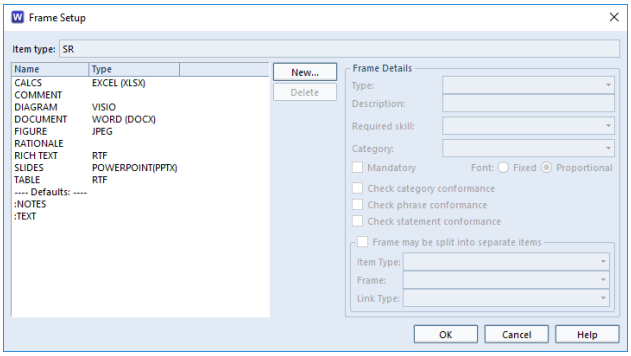
The categories in the schema for system requirements are:

Table 4: Category Code Definitions

Name	Type	Values (* is default)	Description and Purpose
Discipline	Single-value pick-list	**Hardware** **Software** **Mechanical** **Sales** **Marketing** **Other** **TBD***	Which group or engineering discipline is responsible for re-expressing each user requirement as system requirement(s)
Maturity	Single-value pick-list	**Agreed** **Checked** **Deleted** **New*** **Ready** **Rejected** **Review**	The values of the lifecycle for a system requirement. It also avoids deleting system requirements when their authors have removed them. Such requirements may be re-introduced, so having them available saves time. Such requirements may be linked to other items, keeping them makes it easy to find items that link to redundant requirements, so these items can be considered for removal. It also allows the justification for everything in the database to be traced back to requirements that are agreed.
Priority	Single-value pick-list	**High** **Medium** **Low** **TBD***	The importance of the system requirement to the user, based on the user requirements from which it was derived.
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 they are being allocated into the SBS.
Type	Single-value pick-list	**Commercial** **Environmental** **Functional** **Non-Functional** **Health & Safety** **TBD***	The type of the system requirement. This is the type of the externally-observable aspect of the product that the system requirement's linked user requirement(s) describe. It helps to group system requirements that refer to the same type of product characteristic, and helps to determine how the system requirements will be allocated, and by whom.
Validation	Single-value pick-list	**Inspection** **Analysis** **Demonstration** **Test** **TBD***	The validation method that will be applied to the product to demonstrate its compliance with the system requirement. The validation method is closely tied to the **Type**.

Frames 13

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 data in your system requirements. You can ignore any frames that are not relevant to you.

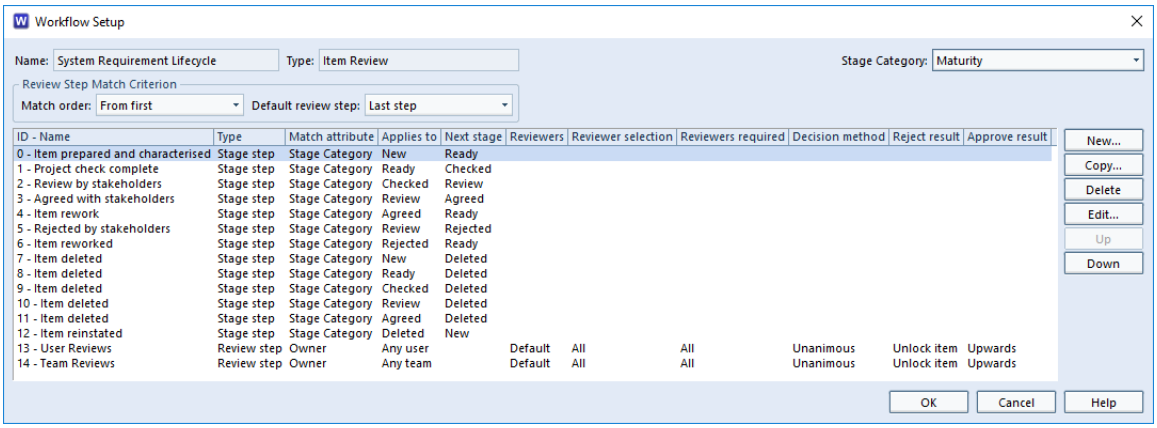
The frames defined in the schema are:

Table 5: Frame Definitions

Name	Type	Description and Purpose
CALCS	**EXCEL (XLSX)**	Store calculations or any other information in a spreadsheet
COMMENT	Plain text	For comments from external groups, such as stakeholders
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 need
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 need 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 need statement, stored here except if it must be rich text

Workflow 14

Each type of item in the database has a workflow set for it. This workflow can be one of the defaults that simply



specifies the CMS reviews to baseline information. A specific workflow is required for system requirements to implement the lifecycle defined in “Lifecycle” on page 10. This workflow also includes the steps that are necessary to review system requirements into baselines, if required by your project.

Link Types 15

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

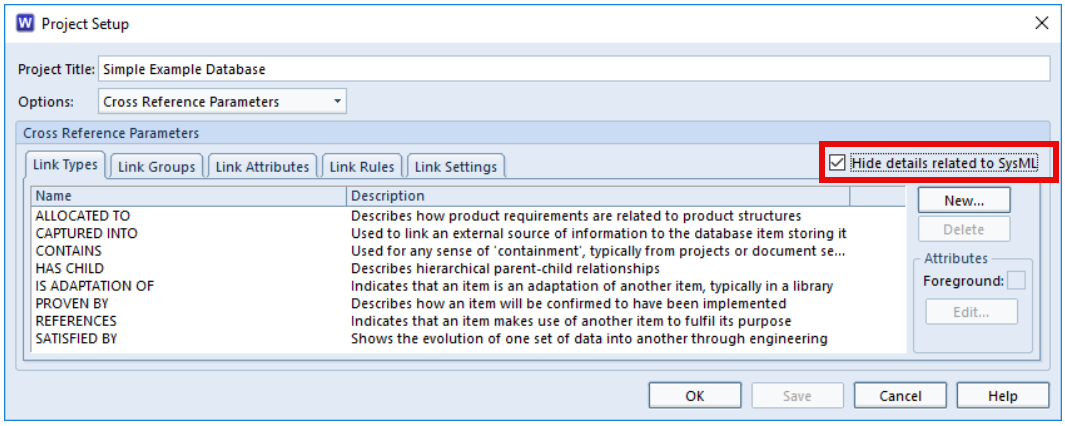
have different link types.

Cross references can store data in attributes inside the links themselves. These attributes are sometimes used to characterise, parametise, justify or simply explain the existence of a cross reference. 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
- Expand an item in a tree node, or a table row
- Show a list of linked items in the UI
- Show linked items in a form used to edit an item
- Show items in nested tables
- Display relationships graphically in a HID
- Publish a document

Cross reference attributes have not been defined in this schema.

There are many link types in the schema. The majority 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 7: "Database Schema" on page 13 are:

Table 6: Link Type Definitions

Name	Details
ALLOCATED TO	Describes how system 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

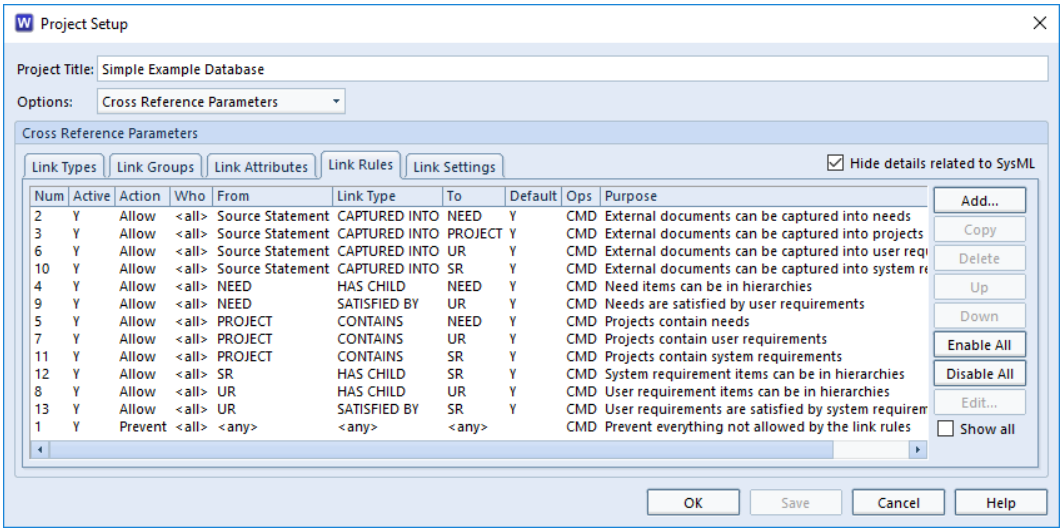
Link Rules 16

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 a type, or items matching a 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 7: "Database Schema" on page 13

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 17

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 system requirements will be auto-numbered, the most useful of these as-supplied navigations are those that sort linked items by their **Key**. 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**