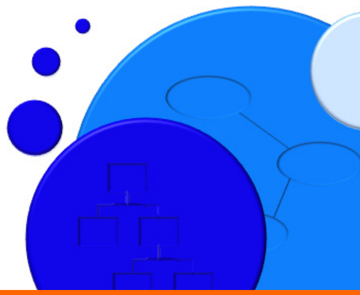
Cradle-7
From concept to creation...

3SL

Role and Representation of User Requirements in Systems Engineering Using Cradle

RA007/02 July 2016



© July 2016 3SL. All rights reserved.
Cradle is a registered trademark of 3SL in the UK and other countries. All rights reserved.
All other trademarks are the property of their respective owners.

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

Copyright © July 2016 Structured Software Systems Ltd
Cradle is a registered trademark of Structured Software Systems Limited. All other products or services in this document are identified by the trademarks or service marks of their respective organisations.

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: user requirements

Subject

The context for any project must be known and its boundaries clearly defined. This information can only come from the project's *stakeholders*.

Should stakeholders' statements be used directly as user requirements, or should the user requirements be created as a separate set of information?

Solution

This paper describes how *user requirements* should be derived from stakeholders' statements, 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

User requirements are statements of the characteristics to be possessed by a product that are visible externally to the product.

User requirements are, by definition, the primary source of product information and so are the starting point for all product development. However, because the language needed in user requirements is not the natural language of you or your customers, we recommend that the information you collect directly from stakeholders is used as *needs*, and that the user requirements are written from these needs.

There will be multiple sets of needs:

- A set generated internally in your business
- A set from each request from each customer

but only a single set of user requirements. Therefore, the transition from needs to user requirements is where you will resolve:

- Problems of ambiguity, contradiction, imprecision and duplication in each stakeholder's needs
- Complementary or contradictory wishes from all the stakeholders

This separates user requirements from each customer's statements. This is also valuable. It helps you make fundamental decisions of how to resolve complementary or conflicting needs of multiple customers:

- As a single user requirement that encompasses all needs
- As several requirements when needs are not compatible

where individual customer needs can be:

- *Accepted*, and linked to user requirements
- *Rejected*, when not linked to user requirements
- *Combined*, when several needs are linked to one user requirement
- *Separated*, where a need is linked to several user requirements to allow more precision in the transition between the two sets of data

This distinction is also a simple way to categorise and report accepted and rejected needs, and derive metrics and KPIs from them.

User requirements fulfil the most important role of all database items. They are a re-expression of needs into a form from which system requirements can be created. User requirements specify characteristics that are desired from a product by someone who can only see, and is only concerned with, the external product, not any of its internals.

Content and Creation 2

These will be the highest level requirements in the database. They will be written by you, not the stakeholders, and will probably be created manually, most likely by copying and rewording needs, and least likely by being captured from documents. The exception will be when existing or legacy data exists, when it should be loaded directly from the external documents into the database.

The purpose of the user requirements is to translate the language of needs (the only source material available for the database) into the language of the functional and non-functional characteristics of your current and new products.

The authors of user requirements must re-express needs into the externally-observable characteristics required of your new and existing products. This is a highly skilled task and, given that the user requirements are the basis for all work on all products, this work must be carefully reviewed.

Most user requirements express the functional and non-functional characteristics required by the users of a product, such as:

- *"When the product is powered-on, pressing the emergency button will stop all current operation within 50msec."*
- *"All exterior surfaces must be painted blue, using paint reference ABC123, to a maximum thickness of 50 micron."*
- *"The product's mass in its operating condition must be no more than 350g."*
- *"The product must have an angular positioning accuracy of not greater than 0.5°."*

but user requirements may also be written from the perspective of other stakeholders, including:

- Purchaser
- Maintainer
- Disposer / recycler

It is often helpful to attempt to specify the *scope*, or *level*, of a user requirement. This represents the *largest proportion* of the product whose externally-observable characteristic is being specified, such as:

1. A user requirement that states:

   "When the product is powered-on, pressing the emergency button will stop and all current operation within 50msec."

   relates to the entire product, so its scope would be: **Product**
2. A user requirement that states:

   "All parts of the product casing that provide access to potentially hazardous electric currents (voltage > 12V or current > 250 mA) must be secured with Torx headed bolts."

   relates only to the product casing, so its scope could be either **Part** or **Assembly**, hence the scope would be set to: **Assembly**
3. A user requirement that states:

   "All user documentation must be provided in all official languages of the European Union, Simplified Chinese and Korean."

   relates to each document supplied with the product, so its scope would be: **Part**

Specifying the scope of a user requirement is intrinsically valuable, adding extra information that aids the understanding of its content, meaning and intent. It also helps when user requirements are satisfied by system requirements. Specifying scopes allows engineers to divide the user 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 user requirement hierarchy. The user requirement hierarchy is likely to contain several 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 *leaves* in this hierarchy will contain requirements, and it is only in these lowest-level requirements that the scope values will be valid and meaningful.

User requirements will be written using language that will:

- Only describe characteristics visible externally to the product
- 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 user requirements. More than any other set of information, user requirements are the subject of traceability and coverage analyses, typically presented in ***compliance tables*** (such as user requirements to system requirements, or user requirements to validations) or matrices. These analyses use the cross references to and from each user requirement. If a user requirement contains a list of paragraphs, then cross references for each user 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 them will be clearer and easier

The user 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 user 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* for the user requirement is also recorded. 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 effectiveness* (MoEs), 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 user requirements must include a re-appraisal of the assumptions to re-confirm their validity. Assumptions that have become invalid with changes in technology or techniques could easily change which of the alternatives should become the user requirement, or indeed introduce new possible alternatives for the user requirement's statement.

We recommend that, wherever possible, the providers of the needs are consulted to approve the re-expression of their needs in the user requirements. We understand that this may not always be possible and that a careful decision will need to be taken whether it is commercially helpful to expose customers to this information.

Structure and Organisation 3

Since the user requirements will be derived from the needs, and the needs are organised by project, there will be one hierarchy of user requirements for each project. Each hierarchy will contain a mixture of different types of user requirement.

It is your decision how to organise the user requirements in each hierarchy. User requirements could be organised by type, such as:

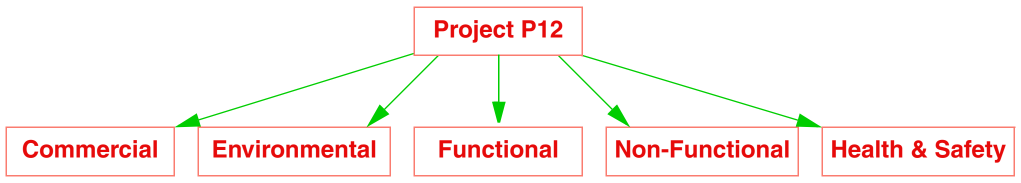


Figure 1: Organise User Requirements by Type

or they could be organised by discipline, such as:

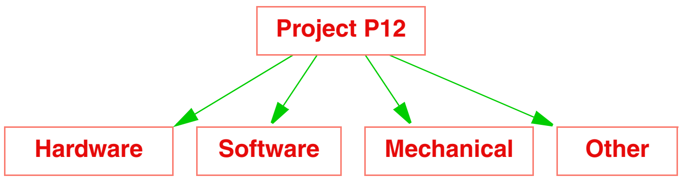


Figure 2: Organise User Requirements by Subject

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

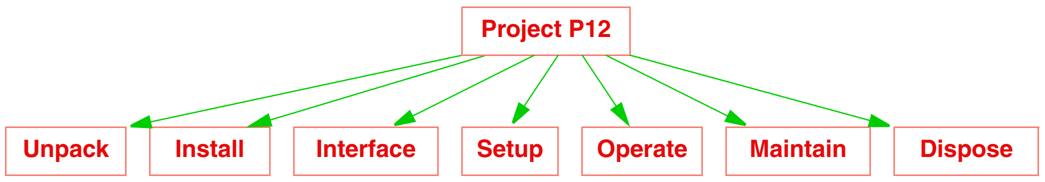


Figure 3: Organise User Requirements by Lifecycle

We recommend the last of these examples, that user requirements are structured into hierarchies based on the products' lifecycle. We believe that this is the easiest approach, because:

- It encourages user requirements to be created from the perspectives of all stakeholders
- These perspectives are closer to the perspective of the people who create the needs
- There will be a smaller step between needs and user requirements than if the user requirements were structured in any other way
- The user requirements will be easier to write and review

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

Traceability 4

Each user requirement will be linked to the need(s) from which it has been derived. We propose that only the bottom-level needs are linked to the user requirements. It is likely that the user requirements that are linked to needs will have further decomposition, so that the links between the needs and user requirements will be from the bottom-level needs to middle-level user requirements.

This approach is also a simple way to categorise and report the completeness of the satisfaction of needs by user requirements, and hence to derive metrics and KPIs from them. Finding all bottom-level needs is both simple to do, and, by this approach, identifies the complete set of needs for which traceability to user requirements must be defined.

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

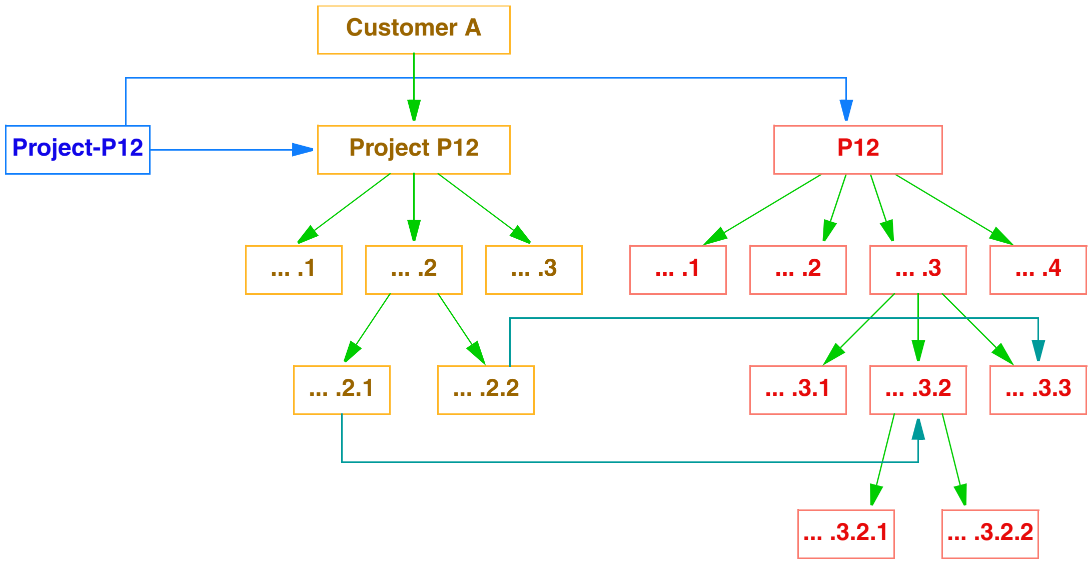


Figure 4: Projects, Needs and User Requirements

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

Database Representation 5

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

Since there will be one user 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 lifecycle, the names of the **.1**, **.2**, **.3** ... items in each hierarchy will be such as **Unpack**, **Install**, **Interface** and so on.

User requirements will require attributes to:

1. Name the user requirements, as a summary or shorthand, and potentially only used in user requirements that have children, as a convenient name for that group of related requirements
2. Specify the user 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 user requirement being as it is
4. Provide notes for the user requirement
5. Characterise the user 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**
   - **Validation**, as one of: **Inspection**, **Analysis**, **Demonstration**, **Test**

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

The method chosen to structure user requirements replaces an attribute that would otherwise exist inside them. Since we recommend that user requirements are structured using their role in the products' lifecycle, they have no **Lifecycle** attribute.

Database Schema 6

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

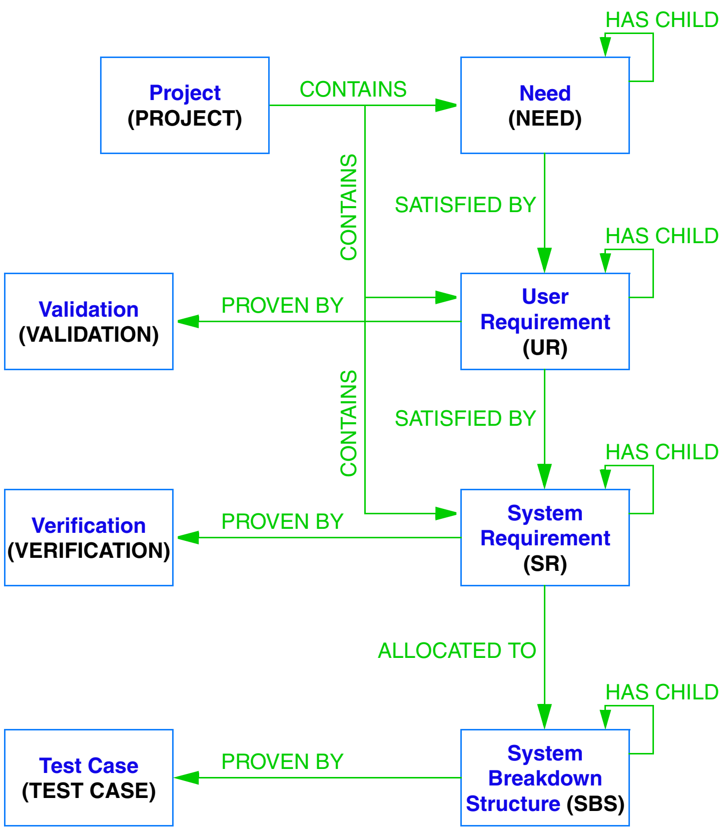


Figure 5: Database Schema

Verification 7

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 8

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

Table 2: Item Type Definition

Name	Categories	Frames		Auto-Numbering	Hierarchical	Adaptations	Change History
		Name	Type				
UR	Discipline Maturity Priority Scope Type Validation	CALCS	EXCEL (XLSX)	Yes UR-*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				

Category Codes 9

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 sorting by category value is efficient.

The categories defined in the schema are:

Table 3: Category Code Definitions

Name	Type	Values (* = default)
Discipline	Single-value pick-list	Hardware, Software, Mechanical, Sales, Marketing, Other, TBD*
Maturity	Single-value pick-list	Accepted, New, Rejected, Pending, TBD*
Priority	Single-value pick-list	High, Medium, Low, TBD*
Scope	Single-value pick-list	Product, Subsystem, Assembly, Part, TBD*
Type	Single-value pick-list	Commercial, Environmental, Functional, Non-Functional, Health & Safety, TBD*
Validation	Single-value pick-list	Inspection, Analysis, Demonstration, Test, TBD*

Link Types 10

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.

The types of cross reference used in the schema in Figure 5 "Database Schema" on page 8 are:

Table 4: Link Type Definitions

Name	Details
ALLOCATED TO	Describes how system requirements are related to product structures
HAS CHILD	Describes hierarchical parent-child relationships
IS ADAPTATION OF	Indicates that an item is an adaptation of another item, typically in a library
PROVEN BY	Describes how an item will be confirmed to have been implemented
REFERENCES	Indicates that an item makes use of another item to fulfil its purpose
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 11

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 8

Navigations 12

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 5: 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**