Cradle-7

From concept to creation...

3SL

Role and Representation of Needs in Systems Engineering Using Cradle

RA006/02 July 2016

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: needs

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, or should they be linked to a separate set of items that form user stories or requirements?

Solution

This paper describes how stakeholders' statements can be captured as *needs*, structured in a Cradle database, and used as the basis for later engineering.

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

We propose that the starting point for any project is ***needs***, not requirements. Needs are expressions of outcomes to be achieved by new or existing products. The term ***objective*** is often used in business analysis to mean something achieved by following one or more steps. In a systems engineering context, objectives and needs can be considered equivalent. A ***goal*** is more intangible, a situation which an organisation seeks to achieve by completing a series of objectives, that is, by satisfying a set of needs.

There are multiple sets of needs:

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

We propose that these are not ***requirements*** as they are not in the language normally used for requirements, particularly for statements from customers. Customers will state their wishes in their terms, and not as desired changes in your products, and not as requirements for new products that you might produce.

Content and Creation 2

Needs are not written in the language of your products, nor in the language of the technologies used in your products. Rather, they will be written in the language of the customers' products and business, or (for internal needs), written in the language of business or commercial objectives.

Examples of internal needs are:

- *"Reduce the number of types of bought-in component, to increase order volumes and purchase discounts."*
- *"Reduce software maintenance by standardising drivers across all products, disabling features where needed."*
- *"Achieve compliance with standard IEC 12345."*

Needs will be created by:

- Capturing from documents or spreadsheets
- Structured interviews, normally only for clarifications

It is important that needs are expressed in the natural language of the group that provides them, so these providers (customers and internal) will have confidence that their needs have been properly documented in the database.

Therefore, the validity of the database is primarily determined by the providers' acceptance of the needs inside it.

Using the providers' statements will inevitably mean that needs will be ambiguous and duplicated. It may mean that some needs are expressed in high-level terms whilst others are extremely detailed. There may be a need to decompose some needs into simpler statements, to achieve a consistency in the level of detail amongst the lowest-level needs. Since this means changing the information originally provided, each decomposition must be formally accepted by its provider.

We recommend that decomposition is the only manipulation of the needs that will occur. Identifying and resolving the other problems, such as ambiguity, imprecision and duplication will occur in the user requirements.

Structure and Organisation 3

Needs are hierarchical. As their main characteristic is the group that produces them, we propose a hierarchy of internal needs, and one hierarchy for each customer. Further, since needs are the basis for projects, we propose a sub-hierarchy of needs for each project. Hence the overall structure of needs will be:

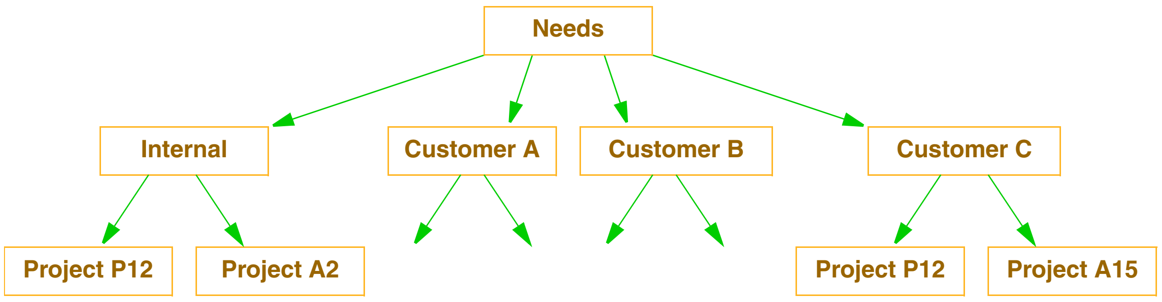

Figure 1: Overall Structure of Needs

If a project has needs from several groups, such as **Project P12** above, it will have several needs hierarchies, all linked from the project, as described in section 4 “Traceability” on page 4.

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

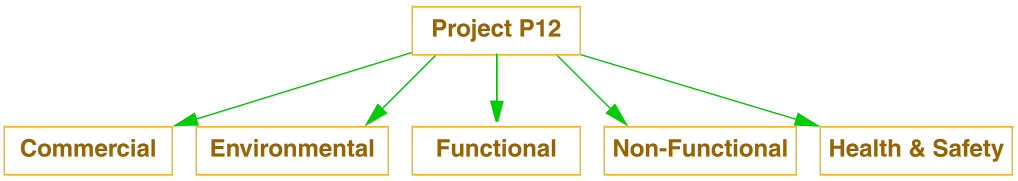

Figure 2: Organise Needs by Type

or they could be organised by engineering discipline, such as:

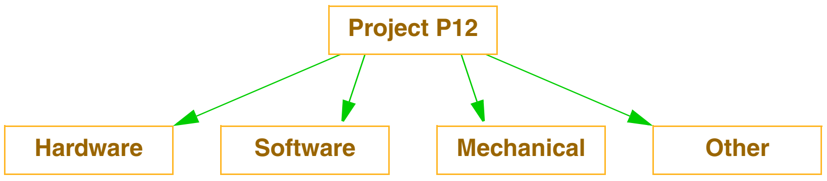

Figure 3: Organise Needs by Discipline

We recommend that needs are organised by type since we expect your customers and internal groups will naturally express needs in groups based on these types.

Traceability 4

Each need will be linked back to its project and will link to the user requirements created from it. For example:

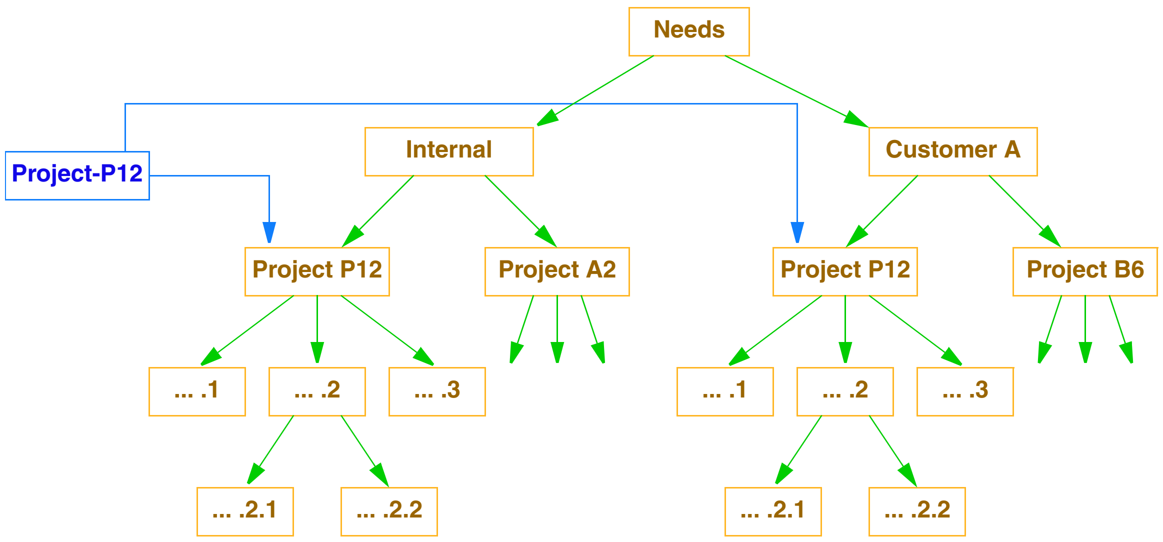

Figure 4: Projects and Needs

We recommend traceability matrices between projects and needs.

Database Representation 5

The same structure can be used for all internal and customers' needs, so all needs can be represented by one ***item type*** called **NEED**. This item type will be hierarchical and auto-numbered to allow reordering of the hierarchies. The hierarchical numbers will be stored in the **NEED** items' **Key** attribute, and will use a prefix **Internal** for internal needs and a suitable abbreviation for each customer's needs. The **Key** values' format is therefore:

organisation.project.hierarchical-number

such as:

Internal.Project-P12.1.2
CustB.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 every hierarchy of needs will be **Commercial**, **Environmental** and so on.

Needs will require attributes to:

1. Name the need, as a convenient summary or shorthand
2. Specify the need, 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 notes for the need
4. Characterise the need:

- **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**

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

The method chosen to structure needs replaces an attribute that would otherwise exist inside them. Since we recommend that needs are structured using type, they have no **Type** attribute.

Database Schema 6

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

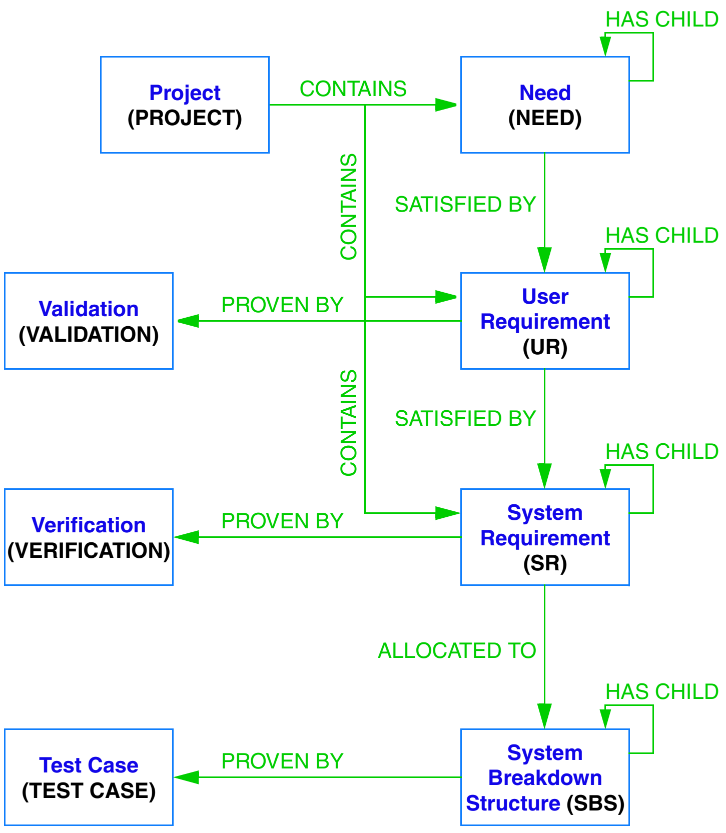

Figure 5: Database Schema

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 need will be an item of type **NEED**, defined as:

Table 2: Item Type Definition

Name	Categories	Frames		Auto-Numbering	Hierarchical	Adaptations	Change History
		Name	Type				
NEED	Discipline Maturity Priority	CALCS	EXCEL (XLSX)	Yes N-*n* *n*=1,2,3...	Yes, in the **Key**	No	Yes
		DIAGRAM	VISIO				
		DOCUMENT	WORD (DOCX)				
		FIGURE	JPEG				
		NOTES	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***

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 5 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
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 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 5

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**