Luhmann's slip box – what can we learn from the device for knowledge representation in requirements engineering?

Andreas Faatz, Birgit Zimmermann, Eicke Godehardt
SAP Research, Darmstadt, Germany
{andreas.faatz, birgit.zimmermann, eicke.godehardt}@sap.com

Abstract

This paper is a thought experiment in finding structures for the elicitation of requirements on top of structures for retrieving requirements. We transfer from structures used by the German sociologist Niklas Luhmann, who applied a wooden slip box to collect notes on processed literature. Each slip contains bibliographic information and Luhmann's thoughts to the referenced content. In this paper we propose to use Luhmann's slip box approach for requirements engineering, by enhancing it to a multi user suited to sort requirements in a requirements repository.

1. Introduction

The sociologist Niklas Luhmann collected his thoughts of processed literature on slips, ordered in a wooden slip box. He considered this slip box as a kind of a communicative partner that, over time, evolves its own knowledge. To be able to achieve a reasonable amount of knowledge the slip box has to be fed over a long time. Luhmann collected thousands of slips during his whole life. In requirements analysis you often find the problem of not knowing if you considered all requirements needed to build a system. We propose to use the idea of Luhmann's slip box in requirements analysis to bring up additional requirements.

The paper is organized as follows: Section two introduces Luhmann's approach. Section three presents related work. Section four explains how the slip box model can be used to sort requirements. Section five concludes on the applicability of the slip box approach to requirements engineering.

2. The slip box model

There exists one original description of the slip box by Luhmann [1] from a communication perspective. The structural description can be found in [2]. Here we describe the structural elements using graph-theoretic terms as "tree", "leaf", and "node". Note that Luhmann worked on paper, without a computer. For his realization by a numbering scheme see [1] and [2].

The slip box consists of a tree *T*, where the nodes and leaves are slips full of textual notes. The tree *T* is established by the following basic operations:

- extending a leave or a node: additional text can be attached to a slip any time.
- branching from a node or a leave (attaching slips deeper in the hierarchy). This allows to extend the structure of the hierarchy any time.
- referencing from any slip to any slip (the opportunity to place hyperlinks).

T continuously establishes a **hierarchy without names and without formal is-a or part-of relation**. Luhmann tried to avoid any pre-supposed ontology, world-model or domain-model to be capable of truly restructuring sociological theories [1]. However, he headed at a still formalized hierarchical order on top of a full text index he maintained. The hierarchy expressed his personal associations of the slips. The structure *T* resembles a wiki owned and maintained by a single user – but **without**

- dedicated names for sections and wiki pages. Access would be by tags or by full text search.
- the ability to delete text. Luhmann considered the surprising return of forgotten slips as a source of creativity.

The main difference of Luhmann's slip box model to requirements repositories is the number of accessing users. Requirements elicitation has a multi-reader access: many people give their requirements to the moderator of a requirements process, who establishes readability by a requirements tool, e.g. a wiki. There are requirements processes with less degree of freedom, like a moderator maintaining a legacy of reusable requirements. However, the authoring and the moderating role, which are melted in Luhmann's system, are separated in requirements engineering. Transferring to the organization of requirements, there are consequences regarding the understandability of each requirement. Where Luhmann appreciates remembering the contents of a slip in a potentially new context by himself, requirements methodology introduces approaches of requirements testing to maximize a common understanding in a group. We take this clarity requirement on requirements processes as fulfilled.

For our purposes, the main benefit of Luhmann's approach is the notion of an unnamed hierarchy in *T*, which organizes the slips. We consider this to be a useful mechanism for the organization of requirements and as a complement to full-text search or ontology-based access as discussed for example in [3]. Other structured browsing approaches as Amazon-like recommender systems are not directly applicable to requirements. And hyperlinks only may cause a "lost in hyperspace" effect.

3. Related Work

Requirements processes as described by Robertson/Robertson [4] mention the benefits of spontaneously created hierarchies in mind mapping [5] as a universal tool for the requirements engineer, but do not see this as a continuous collaborative effort of people bringing in requirements. Nguyen and Swatman point out the permanent potential for re-ordering requirements, which create "catastrophic", i.e. disruptive reorganization activities in the requirements process, which are not supported by CASE tools [6]. They see this fact as an open research question. We try to construct ideas for this "formality-creativity" gap.

Laddering and card sorting as described by Maiden and Ncube [7] have been a source of inspiration for the mechanism presented in section four. However, we step back from [7] abstracting from the moderating role of a requirements engineer facilitating and documenting the project-specific requirements hierarchies.

4. Collaborative asynchronous sorting

We turn the activity of maintenance into an asset. The structures we are using come from Formal Concept Analysis (FCA) [8]. FCA provides means to transform tables of objects with binary attributes into concept lattices. A formal concept has an extension (a set of objects as instances of the concept) and an intension (a set of attributes with value "true" for the concept). The concept lattice shows the formal super-concept and sub-concept relations. When browsing via an edge in the formal concept lattice, one is able to see, which objects and attributes enter or go lost for the more special or more general concepts. We differ from other collaborative approaches to FCA [9] by the way we establish the attributes without any tagging or keywords on the objects.

The core idea is, to produce binary attributes for the single requirements, by a card sorting technique. Whenever a new requirement *R*x is entered into the repository, a user A gets a presentation of a set *E*i of existing requirements. Each *E*i has the size two or more and belongs to an overall set *E* of randomly chosen pairs or triples of requirements. We assume this to be a pair and denote the requirements in *E*i by (*R*i1 and *R*i2). The attributes forming the concept lattice are of shape "being more similar to one of the requirements *R*i1 or *R*i2 in *E*i". I.e. the card sorting action lets the user determine, which of the elements of *E*i s/he considers to be the most similar one to *R*x. This sorting is continuously extended for already existing requirements: the user is prompted to sort a second requirement already in the repository, for which the value of *E*i is not determined as an attribute yet.

The approach also foresees a way to drop the partitions of the cards, which are too ambiguous among the users. Along with bringing in a new requirement, the user gets a presentation of a third requirement already in the repository and already sorted by *E*i by another user B. I.e., the value of *E*i is determined as an attribute yet - but the sorting result of user B is hidden to user A. If the sorting result equals among users A and B, all attributes in *E* stay the same. If not, *E*i will not be used for future sorting actions and be replaced by a new *E*j containing the new requirement *R*x. Thus, the user has three sorting actions and is confronted with three requirements to sort and two to "sort against" (*E*i) in total without any further necessary explanation from her/his side.

As an example for the essentials of the process sketched above assume the requirements repository to collect requirements on a bottle. When a new requirement (*R*x = "the bottle does not spill liquor, when it falls and hits the ground") enters the repository, the user is for example asked, if s/he considers *R*x to be more similar to *R*i1 = "the bottle is closable" or *R*i2= "the bottle has readable instructions, how to conserve its contents over a longer period". Let the user chose *R*i1 as more similar. Quite probably, the pair (*R*i1 and *R*i2) proves distinguishable for other requirements, too, which is checked by letting the user sort a second requirement from the repository, e.g. "the bottle is transparent". The system checks, if the user agrees with the old sorting by another user, if this is more similar or related to readable instructions, which is probable. If

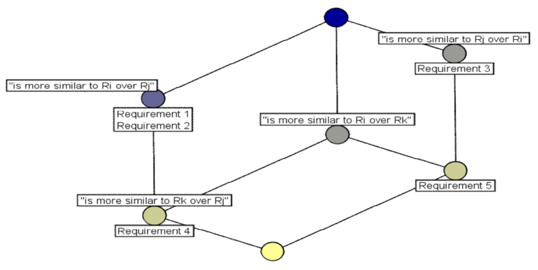
"is more similar to Rj over Ri"
Requirement 3
"is more similar to Ri over Rj"
Requirement 1
Requirement 2
"is more similar to Ri over Rk"
Requirement 5
"is more similar to Rk over Rj"
Requirement 4

Figure 1. Browsing requirements.

*R*i1 would have been "the bottle is re-closable" and *R*i2 would have been "the bottle is unbreakable", an existing requirement like "the bottle is safe for children" might cause different sorting by different users. Such a combination would be dropped as attribute and replaced by a combination-for-comparison of "the bottle does not spill liquor, when it falls and hits the ground" and a random requirement. Besides that, the user is prompted to sort something completely independent from the pair (*R*i1 and *R*i2).

Considering the results for browsing the resulting hierarchy, users get a presentation of the resulting concept lattice, which is unusual as the attributes are pairs (or n-tuples) of objects. This means, that the resulting browsing actions offer the following steps: 1) from a requirement to a set of requirements that are similar and 2A) to a further abstraction of this set of requirements or 2B) via a specialization to 3) other similar requirements. The trigger for step 1) is *E*i, i.e. the first browsing action will guide users to other requirements stably similar to *R*i1 or *R*i2. These requirements bare other similarities/attributes and users have the choice to keep (2A)) or drop (2B)) *E*i. This is visualized in figure 1 and table 1.

Table 1: SIMILARITY OF REQUIREMENTS

Requirement	is more similar to…
1	*R*i over *R*j
2	*R*i over *R*j
3	*R*i over *R*j
4	*R*i over *R*j, *R*k over *R*j, *R*i over *R*k
5	*R*j over *R*i, *R*i over *R*k

5. Conclusion and future work

The main conclusion we draw from the sketch of the system described here is, that the unnamed hierarchy, which is characteristic for the slip box, can be established by formal means and by multiple users. It can be kept as a data structure and mechanism in parallel to other knowledge representations to order repositories of requirements.

We feel the urgent need to gather empirical data on sorting, hierarchies and the stability of the distinctions as well as on a quantitative measure like "number of new requirements resulting from the browsing actions". We intend to apply the sorting method on a small set of requirements first and measure the time needed to define a fixed number of new ("new" in the sense of new fit criteria of the Volere shell) requirements by browsing the unnamed hierarchy in n steps versus access by full text search without creativity trigger only - and finally both variants against a random set of requirements of size n as creativity trigger.

Acknowledgements

APOSDLE is partially funded under the FP6 of the European Commission within the IST work program 2004 (FP6-IST-2004-027023).

6. References

[1] N. Luhmann: Kommunikation mit Zettelkästen, in Universität als Millieu, reprint, Haux, Bielefeld 1998

[2] M. Schiltz, F. Truyen, H. Coppens: Cutting the Trees of Knowledge: Social Software, Information Architecture, and Their Epistemic Consequences, Thesis 11, Vol. 89, (1), 2007

[3] S.W. Lee, R.A. Gandhi: Ontology-based active requirements engineering framework, 12th Asia-Pacific Software Engineering Conference, Teipei, Taiwan, 2005

[4] J. Robertson, S. Robertson: Mastering the Requirements Process, Addison Weasley, 2006

[5] T. Burzan: The mind map book, Pearson Education, 2006

[6] L. Nguyen, P.A. Swatman: Managing the requirements engineering process, Requirements Engineering, Vol. 8 (1), Springer, 2003

[7] N.A.M. Maiden, C. Ncube: Acquiring COTS software selection requirements, IEEE software, vol. 15(2), 1998

[8] U. Priss: Formal Concept Analysis in Information Science, Annual Review of Information Science and Technology, Vol. 40, No. 1. 2006

[9] A. Hotho, R. Jäschke, C. Schmitz, G. Stumme: BibSonomy: A social bookmark and publication sharing system, In: CS-TIW '06, Aalborg, Aalborg, 2006