Requirements Sensemaking using Concept Maps

Shamal Faily[1], John Lyle[1], Andre Paul[2], Andrea Atzeni[3], Dieter Blomme[4], Heiko Desruelle[4], and Krishna Bangalore[5]

[1] University of Oxford, Oxford UK OX3 0NH,
`firstname.lastname@cs.ox.ac.uk`
[2] Fraunhofer FOKUS, 10589 Berlin, Germany,
`andre.paul@fokus.fraunhofer.de`
[3] Dip di Automatica e Informatica, Politecnico di Torino, 10129 Torino, Italy,
`andrea.atzeni@polito.it`
[4] Ghent University/IBBT, B-9050 Gent, Belgium,
`firstname.lastname@intec.ugent.be`
[5] Technische Universität München, 85748 Garching, Germany,
`krishna.bangalore@in.tum.de`

Abstract. Requirements play an important role in software engineering, but their perceived usefulness means that they often fail to be properly maintained. Traceability is often considered a means for motivating and maintaining requirements, but this is difficult without a better understanding of the requirements themselves. Sensemaking techniques help us get this understanding, but the representations necessary to support it are difficult to create, and scale poorly when dealing with medium to large scale problems. This paper describes how, with the aid of supporting software tools, concept mapping can be used to both make sense of and improve the quality of a requirements specification. We illustrate this approach by using it to update the requirements specification for the EU *webinos* project, and discuss several findings arising from our results.

1 Introduction

Techniques, tools, and artifacts may come and go, yet requirements are a consistent element in a software development project. The needs and expectations of stakeholders motivate the elicitation of requirements that software developers use as a contract for what a system should do. This makes a requirements specification the authority upon which architectural design and software implementation decisions are made.

The perceived usefulness of requirements, however, often fails to extend beyond the initial stages of system design, especially as these are often construed negatively from both a usability and developer perspective. To usability professionals, specifications of system behaviour are agnostic of usability concerns and say little about the needs of people using it. To software developers, they are seen as too abstract to effectively form the basis of an implementation; developers may feel it is easier to prototype a solution and make their own sense of the domain and what needs to be built. Because of this, maintaining system requirements is considered by many to be an unnecessary and cumbersome activity that adds comparatively little value. While this position may be reasonable in projects where all stakeholders are directly involved in the system design,

for many projects this is not the case. Customers and prospective end-users may want a precise description of what a system does when explaining the system to other people, or evaluating the system for concerns such as regulatory compliance or value for money. Similarly, security engineers need models of system behaviour upon which their own analysis can be based. In such cases, prototypes that embody what they think the system should do are a poor substitute for a description of what the system is expected to do.

Requirements are seen as static artifacts which, while useful early in a project's life for motivating initial architectural design, are too unwieldy to maintain as a going concern. This perception is not helped by the fact that, as natural language text, they are often hidden away in word processors, spreadsheets, or databases. As developers become more detached from the requirements that form the basis of their work, the rationale for design changes can either become lost or become encapsulated in other design elements. Because of this, when requirements do need to be updated, the revision process can be both tedious and time-consuming. Although research by the Requirements Engineering community has been concerned with how these requirements traceability problems can be prevented, there has been comparatively little work on how these traceability challenges can be addressed once they do occur in real projects. The possible implications arising from missing, superfluous, or ambiguous requirements means that we need creative approaches for addressing these challenges. If such approaches are to be successful, we need to engage the stakeholders responsible for specifying and maintaining these requirements by helping them make better sense of what they are.

In this paper, we present an approach for using concept mapping as a sensemaking technique for requirement specifications. This improves the quality of a requirements specification while also increasing its visibility among project team members. In Section 2, we describe the motivating literature, before presenting our approach in Section 3. In Section 4 we present the results of applying our approach to update the requirements specification for the EU FP7 *webinos* project, before reflecting on these results and proposing follow-on work in Sections 5 and 6 respectively.

2 Related work

2.1 Requirements maintainability and traceability

Rather than being concerned with the behaviour of the proposed system, Requirements Engineering is concerned with defining the problem being solved [1]. While it is tempting to think that, once defined, requirement specifications will be *good enough* to sustain software engineering activities, changes to the problem space, or modifications to environmental conditions affecting system design might be significant enough to challenge assumptions underpinning a requirements specification, therefore motivating the need to re-evaluate it.

As cases like the Ariane 5 failure have shown, failing to properly analyse existing requirements for their continued relevance can lead to catastrophic results [2]. Consequently, the importance of rationalising requirements has motivated the need for managing requirements traceability: the ability to describe and follow the life of a requirement, in both a forward and backwards direction [3]. When properly maintained, requirements

traceability provides the rationale for requirements, provides a means for checking their satisfaction, and provides a framework for software tools. Unfortunately, while there is no shortage of requirements traceability frameworks and models, software tools for supporting them are usually restricted to requirements management tools [4]. While requirements managements tools are reasonably effective at indicating how requirements are operationalised, Winkler and von Pilgrim found that pre-requirements traceability remains poorly tool-supported. This finding confirms an argument first made by Gotel and Finkelstein [3], who claimed that pre-requirements specification traceability problems are hard to address because the dynamic nature of the sources and environment from which requirements are drawn makes categorising traces difficult.

One way for dealing with pre-requirements traceability is to employ design rationale tools to externalise the arguments underpinning requirements. For example, Burge [5] has discussed how tools like Compendium [6] provide a visual language for capturing group discussions and facilitate requirements negotiation. Similarly, Eclipse IDE plugins like Software Engineering Using RATionale system (SEURAT) can be used to capture the source of requirements, in addition to evaluating alternatives and flagging errors when alternatives violate requirements [7]. More recently, work on the open source Computer Aided Integration of Requirements and Information Security (CAIRIS) requirements management tool [8,9] demonstrated how requirement models can be motivated by usability artifacts such as scenarios and personas [10], as well as how argumentation models can be used to structure qualitative data and assumptions underpinning the characteristics of personas [11,12].

2.2 Sensemaking and concept mapping

While pre-requirements traceability remains a challenge, there is evidence that sensemaking — the process of searching for a representation and encoding data in that representation to answer task specific questions [13] — might assist in understanding where requirements have come from, and where they might ultimately lead. In a long-term study of software development within a company, it was found that relying on rational decisions alone to simplify assumptions increased misunderstanding of products as systems grew [14]. Consequently, the company found that rather than making what appeared to be the most rational decision, it needed to rely on sensemaking techniques to choose the most appropriate course of action. This is because trying to understand the rationale behind a particular decision may lose sight of the contextual factors at play when a decision is made. In the broader context of software engineering, collectively re-organising and synthesising information can help groups of stakeholders make sense of what this information means [15]. Sensemaking techniques have been used to supplement various aspects of usability evaluation in software design, ranging from re-designing the information architecture for a software platform [16], through to evaluating the usability of APIs [17].

In a sensemaking task of any significant complexity, external representations are important for supporting the cognitive demands made on users, but creating such a representation is itself a significant problem [13]. A popular visual representation for facilitating sensemaking by helping designers absorb knowledge is *concept mapping*. Concept maps are sensemaking tools that connect many ideas, objects, and events within

a domain and, as a result, help organise and visualise knowledge [18]. Concept maps were first proposed by Novak [19] as a learning tool which allows students to build and reflect on the conceptions of a domain. For example, in a case study in organisational learning [20], individuals made sense of concepts before exposing it to the influence of others and sharing discourse. As such, the technique was able to bring people together and synthesise different views.

Although software tools such as CmapTools [21] can be used to collaboratively build concept maps, these rely on using keyboard and mice as input, potentially making concept mapping less fluid, hindering the co-creation process. Together with work by Klemmer et al. alluding to the benefits of combining the affordances of paper with electronic media [22], this has stimulated a growing interest in the use of tangible interfaces for concept mapping. For example, Tanenbaum and Antle [23] created a prototype tabletop system and software application to support the creation and revision of concept maps. This study served as a useful proof-of-concept, but it also highlighted problems such as slow object tracking, and restrictions adding new content; these suggest possible scalability problems when working with larger maps. This scalability problems were confirmed in a more recent study [24] by Oppel and Stary comparing the use of tabletop systems for concept mapping with screen based equivalents. Despite the scalability problems, the study found that the time spent on discussion was significantly higher for tabletop concept maps, and information was less likely to be filtered by 'operators' running the software tool to support the concept mapping process. It was, however, noted that participants of screen based tools tended to focus more on the process of representing the model itself. This issue of representation is important from a Requirements Engineering perspective; concept maps need to be both used and maintained as a system design evolves, and by a potentially larger range of people than those engaged in participatory concept mapping sessions.

3   Approach

Our approach uses concept mapping to both make sense of and improve the quality of a requirements specification. The approach assumes that an initial requirements specification has already been created, and that the specification is sub-divided into different functional categories. We carry out concept mapping to make sense of a system's requirements, together with the traceability associations that motivate and connect them. In carrying out this exercise, we glean a better understanding of the problem domain the system is situated in. This approach is supplemented by software tools which synthesise concept map data with other requirements artifacts to form a coherent requirements model.

By using software tools to support rather than mediate the concept mapping process, we gain the benefits of tabletop concept mapping without the suffering the scalability problems associated with tangible interfaces, or filtering problems associated with screen based concept mapping tools. The approach also has three other benefits from a broader Requirements Engineering perspective. First, the process of building the concept map also identifies and addresses duplicate or conflicting requirements. When the model is complete, the relationships between the requirements becomes apparent, as

well as the most appropriate means for addressing them. Second, because this is largely a group-based exercise, an understanding of what the requirements are and how they contribute towards system design is shared by everyone involved in the process; this also helps to establish what the current scope of the system is, and to address any areas people feel should be addressed but currently are not. Third, the resulting concept map provides a basis for examining how other design artifacts fit into this conceptual model from a traceability perspective. For example, can scenarios or use cases be built into this concept map without changing the structure itself? Does the concept map align with the conceptual integrity of the planned architecture, and do the implementation level items used to drive the implementation align with this model?

This approach is applied in three stages. In the first stage, concept names are elicited from the existing requirements, and concept maps are created for each category of requirements. In the second stage, the concept maps are consolidated onto a single diagram and, as associations are made between concepts from these different sub-maps, the consolidated concept map itself evolves. In the final stage, we validate the concept map by examining how well they align and account for other requirements artifacts, and generate the resulting requirements model.

This is not the first time that concept mapping has been proposed as a technique for supporting Requirements Engineering; Kof et al. [25] have proposed concept mapping as a generating traces between requirements artifacts. However, our approach differs in two respects. First, Kof et al. are concerned with using concept mapping primarily for supporting the traceability between artifacts at different levels of abstraction, whereas our approach is primarily designed to help designers make sense of the requirements. Second, while Kof et al. uses a frequency profiling algorithm to identify concepts, our approach involves manually inspecting each requirement to infer a suitable concept characterising each requirement. This makes the process more time-consuming, but it better sensitises designers to the requirements themselves.

3.1 Category concept mapping

Before concept maps can be created, it is necessary to create a concept name or phrase characterising each requirement. For example, the requirement *The train control software shall control the acceleration of all the system's trains* can be characterised using the phrase *Acceleration control*. This short phrase becomes the canonical identifier for each requirement.

For each functional area associated with the specification, concept maps are then created. This involves preparing paper cut-outs of the concept names associated with each requirement, placing them on a large piece of paper and, using a pencil, drawing and annotating the relationship between each concept. This exercise is carried out in groups of between two - four for each part of the problem domain represented by the requirements category. During the exercise, the requirements specification associated with the category is available for consultation. Requirements that are redundant or out of scope are removed from the concept map, and a reason for their omission is noted. Requirements that are unclear or ambiguous are discussed within the group and, where necessary, the requirement's concept name or description is updated as appropriate.

Once each map is completed, it is transcribed to a machine readable format. For this, we use Graphviz [26]; this is a suite of graph visualisation tools, which can layout graphs written in a simple textual language called DOT. Figure 1 illustrates an excerpt of a Graphviz concept map.
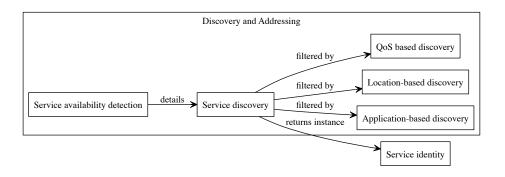

Fig. 1. Example Graphviz concept map

3.2 Concept map consolidation

Once all the concept maps have been created, they are then consolidated onto a single concept map. Like the category concept mapping exercise, this involves drawing relationships between paper cut-outs of requirements. However, this exercise begins by re-creating each of the individual concept maps based on their respective Graphviz models. Using a similar sized group to the first stage, which includes participants involved in the development of the category concept maps, the concept maps are reviewed to clarify relationships both within and between each category. Like the category concept mapping exercise, unclear requirement descriptions are updated and, where appropriate, those deemed duplicate, redundant, or out-of-scope are removed. Similarly, like the first stage, once this exercise is complete the Graphviz versions of each category concept map are updated to incorporate any revisions and relationships between categories.

3.3 Concept map validation and model generation

Building the consolidated concept map not only shows how requirements are connected, it also helps align them with other design artifacts. This is due to the better understanding that the design team gains developing the visual model. For example, zooming out of the concept map can help retrospectively determine which scenarios the requirements were designed to support. Zooming in explores how the elicited use cases operationalise the requirements. This alignment activity also acts as an additional form of validation; by casting the concept map in a different light, existing concepts or relationships are challenged, and new requirements and traceability associations may be

identified. Therefore, the final stage involves identifying design artifacts associated with each requirement concept, and updating each category DOT file to include the aligning artifact references. Our approach supports three types of aligning concepts: scenarios, use cases, and backlog items. Backlog items are items of work which are carried out when developing software as part of the Scrum method [27]. This approach does not pre-suppose Scrum is employed, and this concept can be used synonymously for any downstream design and development artifact.

If we reflect on Figure 1 then we discover that while Graphviz is useful for modelling concept maps, something more is necessary for representing requirements traceability. For example, the association between *Service discovery* and *Service identity* indicates that concepts are related to different categories, not just those associated with a given DOT file. Pasting the contents of all DOT files into a single file is feasible, but with concepts cross-referenced across different files then some duplication of associations is inevitable. It also desirable to filter these maps in some way, or visualise the consequences of the traceability alignment. Moreover, because other requirements artifacts might be maintained using different representations, there needs to be a coherent requirements models that designers can reference internally and present to others externally.

Because it supports the aligning concepts used by this approach, artifacts were converted to XML and imported into the open-source CAIRIS tool [9]. To better fit this approach, a number of modifications were made to CAIRIS. To complement its pre-existing requirements visualisations, an additional graphical view was added to CAIRIS to support the visualisation and manipulation of concept maps; this included the filtering of maps by name or functional category. In this view it is also possible to quickly assess the quality of the requirements themselves using an automated quality gateway check; the metrics are described in more detail by [28], and are based on requirements completeness, the presence of an imperative phrase, and ambiguity. These resulting metrics were visualised using cartoon Chernoff Faces [29], where each part of the face represented a different requirement quality variable. Eye-brow shape indicates the completeness of a given requirement. If no text is found in certain fields, or phrases like "TBC", "none", or "not defined" are present, the completeness score is marked down accordingly, and the eye-brows convey a negative mood. The eye shape indicates whether or not an imperative phrase exists in the requirement description. If such a phrase exists then the eyes become vertically elongated. The mouth indicates the presence of weak or fuzzy phrases, such as mostly, appropriate, normal, or adequate; the presence of these phrases turn the smile into a frown. Examples of how Chernoff Faces visualise these quality attributes are given in Figure 2. These Chernoff Faces are also coloured based on their level of pre- and post-requirements traceability.

Requirements with no evidence pre- and post- requirements traceability were coloured red. Requirements which are motivated by other requirements, based on scenarios and use cases, or refined to product backlog items were coloured blue. Requirements with both pre- *and* post- requirements traceability are coloured green. While it would be unreasonable to expect pre- and post- requirements traceability at early stages of a project, this level of traceability is important for high priority requirements. With this in mind, the labels of high-priority requirements without both pre- and post-requirements trace-
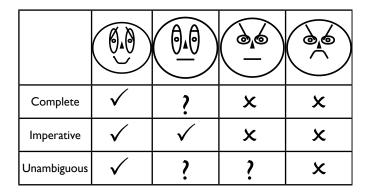
	😊	😐	😠	😠
Complete	✓	?	✗	✗
Imperative	✓	✓	✗	✗
Unambiguous	✓	?	?	✗

Fig. 2. Example Chernoff Faces mapping to requirement quality attributes

ability are highlighted in red. This is illustrated in Figure 3 which illustrates a Chernoff Face augmented concept map based on the initial concept map in Figure 1.

4 Results

We evaluated our approach by using it to update the requirements specification for the EU FP 7 *webinos* project. *webinos* is a federated software platform for running web applications consistently and securely across mobile, PC, home media, and in-car systems. More information about the project is described in [30].

This requirements specification was initially derived from a set of use cases and scenarios [31] and contained 330 requirements; these requirements were sub-divided under eight functional categories: Device and Service Functional Capability (CAP), Discovery and Addressing (DA), Identity (ID), Lifecycle (LC), Negotiation and Compatibility (NC), Policy and Security (PS), Remote Notifications and Messaging (NM), and Transfer and Management of State (TMS). These requirements were used to devise a software architecture for the *webinos* platform.

As architectural design progressed, the project team obtained a better understanding of the problem domain; this challenged many of the assumptions underpinning existing requirements artifacts. However, time constraints during the architectural design meant that the requirements could not be revisited until software development of the platform had already commenced. The platform was implemented using an iterative model based on the Scrum method [27]. Scrum teams were created for representative *webinos* platforms and the core architectural components and features developed were derived from the architectural design documentation.

The process was applied over a two month period by a team of nine part-time and one full-time team member; the part-time team members were involved in platform development activities, while the full-time team member was responsible for co-ordinating the overall approach.
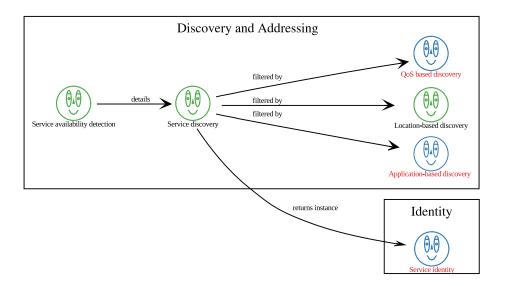
Fig. 3. Chernoff Face concepts

4.1 Preparatory activities

Before applying our approach, a number of preparatory activities needed to be carried to better facilitate collaboration between the team. The first step involved converting the scenarios, use cases, and requirements needed to a more accessible format given the distributed nature of the project team. These were originally documented in Microsoft Word which proved to be too unwieldy. It was, therefore, decided to use the project wiki to maintain these. Unfortunately, because there was no easy programmatic way to access content on the wiki, a git repository [32] was used to maintain the underlying page source. Moreover, while the largely textual structure of the wiki was a fit for storing scenarios and use cases, a more structured format was felt more useful for editing requirements. Consequently, requirements for each category were stored in spreadsheet files which, like the wiki page source files, were also kept under configuration control. Scripts were written to convert both the page sources and the spreadsheets into an XML format compatible with CAIRIS.

4.2 Category concept mapping

Before concept names could be created to characterise each requirement, it was first necessary to improve the quality of many of the original requirements. In most cases, this involved little more than simplifying the requirements text, and breaking down large requirements descriptions into multiple requirement statements. At this stage, ten new requirements were added, but existing requirements were not removed, nor were their priorities changed. Also, in addition to the inclusion of a spreadsheet column for

the concept name, an additional comments attributes was added to each requirement to indicate what original requirement each updated requirement was based on.

The team met over a two day period to begin the category concept mapping process. To acquaint the team members with the process of concept mapping, three concept maps (LC, NC, and PS) were created . As Figure 4 (left) shows, print-outs of the requirements for each category were available for participants to refer to during this exercise; these were used to refer to the requirements description and, when the group felt this text was ambiguous or inaccurate, these were revised. At the end of each session, the concept map was transcribed to DOT. A laptop was also available for directly updating the requirements spreadsheets. Each concept mapping session took approximately half a day with the exception of the Policy and Security concept mapping session which, due to the number of requirements, took one day. Following this workshop, responsibility for completing the remaining five concept maps, together with updating the DOT files and requirements, was distributed among the five partners. The partners then completed these remaining concept maps over a three week period.

Fig. 4. concept mapping session (left) and category concept map (right)

4.3   Concept map consolidation

Once the remaining concept maps were developed, the team met over a further two-day period to consolidate the concept maps. The session was attended by eight of the ten team members, where each person worked on at least one of the contributing concept maps.

Like the category concept mapping exercise, this involved drawing relationships between paper cut-outs of requirements. This exercise, however, began by re-creating each of the individual concept maps. In most of the cases, this involved re-drawing the concept map based on the optimised Graphviz layout although, in two cases, partners brought their physical maps with them. Each individual concept map was placed on a table covered with butcher paper, and marginally trimmed to fit the space available. Because it contained the most requirements, the PS concept map was the most central, while other concept maps were situated around this. Also, by covering the table with

paper, it became easier to draw associations between concepts. Where drawing associations were difficult to follow, then a duplicate concept label was added to the table to simplify reading of the concept map.

Rather than drawing concept mapping to a close, consolidating the maps instead caused the original concept maps to be re-examined in a new light. While participants expected to be asked to comment on the contribution their work areas made to the collective map, many did not realise that they needed to become knowledgeable in the concepts associated with the other maps. Consequently, much of the first day was spent walking through and discussing each of the five concept maps developed remotely, as well as reviewing the three concept maps created when the team last met. The first half of the second day was spent exploring how different concept maps were connected and, where necessary, updating the requirements as a better understanding was gleaned of them. The second half of the second day was spent walking through a selection of scenarios, use cases, and backlog items to check the concept map accounted for the concepts described in these artifacts. Finally, the remaining requirements were reviewed to ensure they were still within scope, their description was accurate, and they were appropriately prioritised.

Besides updating most of the requirements to improve their comprehensibility and clarity, a significant number of requirements (120) were defined as out-of-scope and removed from the specification. The most common reasons for deletion was that a requirement was either superseded or duplicated by other requirements, and what were originally understood to be platform requirements were requirements for end-user applications instead. To ensure that the reason for removing requirements was not forgotten, an *out-of-scope* spreadsheet, and moving the requirement into this sheet, together with an explanation for why the requirement was being removed.

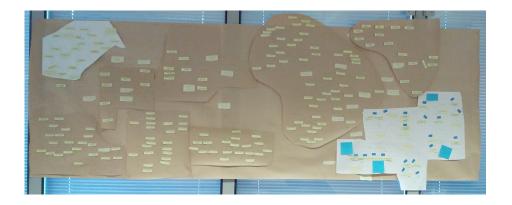The final consolidated map is illustrated in Figure 5.

Fig. 5. Consolidated concept map

4.4 Concept map validation and model generation

Before departing the consolidation workshop, the group carried out a worked example of how the semantic zooming exercise should be carried out, and how the alignment associations between scenarios, use cases, and backlog items should be specified using DOT. Following the workshop, each partner took responsibility for carrying out this alignment activity for one or more functional categories. Because of their familiarity with the respective functional categories, the same breakdown was used as for the category concept mapping.

The participants found that aligning scenarios and use cases to requirements was straightforward because, having gleaned a better understanding of the requirements for their particular area, it was now easier to put the scenarios and use cases in context. However, the lack of post-requirements traceability meant that aligning requirements to the backlog items and implementation code was much harder. To remedy this, the lead developers were invited to review both the concept maps and the Scrum backlog list to both identify requirement concepts they believed were missing, or requirements that motivated backlog items they were working on. Working with their feedback, the participants updated the DOT concept maps for the aligning concepts based on the comments received.

Because of the wide range of artifacts and technologies, and the desire for consistency, additional work was needed to synthesise the data before importing into CAIRIS. A *build* script was created to import all of the requirements artifacts (scenarios, use cases, and requirements) that were stored in the git repository into a single requirements model in CAIRIS; the script also exported the scenarios, use cases, and requirements into both a single specification document, and a collection of wiki pages that project team members could review. When responding to comments from either the document or the wiki page, respondents were asked to update the artifacts in git rather than the wiki pages itself. To validate that people were updating the git-based artifacts, the build script was run daily to re-generate a CAIRIS model based on the contents of the git repository. When the script failed, it was often because of typographical errors in use case, scenario, or requirement identifiers; these caused referential integrity errors when importing data into CAIRIS' database schema.

5 Discussion

In this section, we discuss some of our findings and lessons we learned applying our approach.

5.1 Modelling fatigue

Although concept mapping is easy to explain and, to a passive bystander, concept mapping *looks* easy, it is a cognitively demanding process. Moving and linking concepts forced participants to make and justify value judgements *in-situ* within a group setting, rather than at the individual's leisure while working offsite. This led to two consequences. First, there was a tendency to under-estimate the amount of time needed for

concept mapping. The category concept mapping sessions took twice as long as initially planned because of the discussion generated during the sessions. Second, in a number of cases, time pressures and mental demands meant there was sometimes a tendency to designate borderline requirements as out-of-scope. These cases were most apparent in categories with large numbers of requirements. For example, the original Policy and Security category contained 115 requirements; following the update, only 58 requirements remained. In some cases, subsequent reviews meant that requirements placed out-of-scope needed to be re-introduced into the specification.

Despite the challenges, participants were still positive about their experiences applying the approach. One participant observed that although the time was under-estimated, delays were only in the region of hours, rather than the weeks and months it took to negotiate the original requirements. Participants also found that, as the concept maps became more elaborate and more requirements were incorporated into it, the complexity of the task did not increase. Moreover, participants that worked on multiple concept maps during the Catania workshop found that working on more maps helped them understand the other concept maps better, and aligning scenarios, use cases, and backlog items was considered trivial once the concept maps had been completed.

5.2   Marketing the approach

As a vehicle for updating the *webinos* requirements specification, most participants were initially sceptical about how useful the approach would be. While the theory behind the approach were clear, many participants failed to see how moving around pieces of paper, and drawing labelled arcs between them would improve the quality of the requirements specification. Participants were, however, also sceptical about how useful the approach taken to elicit and specify the original requirements specification would be for updating them. Given the options available, the concept mapping approach was the most palatable because it directly addressed traceability problems that were facing the project at that time.

As the participants became involved in the hands-on concept mapping process itself, they found the approach both rewarding and worthwhile. In particular, working with physical objects like papers, pencils, and scissors reinforced the tangibility of requirements, something not felt when working with screen based interfaces for manipulating requirements. Nonetheless, although the biggest benefit of this approach is obtaining a better understanding of a system's requirements, to gain initial adoption it was important to focus on the immediate tangible benefits which, for this approach, is tool-supported requirements traceability. Once participants had bought in to concept mapping, they encouraged others to get involved by joining sessions, and reviewing concept maps remotely, thereby improving the visibility of the process across the project. This was a notable result because, up to that point, the requirement specification was seen as something the project contractually needed to deliver, rather than something that added value in its own right.

5.3 Human-centered tool support

Although tools played only a supporting role in the process, it was invaluable for assisting what might otherwise have been tedious activities. These included printing out concept names, using spreadsheets to review requirements tables, de-duplicating traces that occurred in multiple DOT files, and generating a specification document. Consequently, rather than using software to automate the process of concept mapping, it was instead used to remove some of the drudgery associated with supporting the hands-on concept mapping sessions.

After reflecting on the sessions themselves, participants felt that, with so much information being discussed around the concept maps, software tools would have been useful for recording transcripts and annotating concepts with additional information. While someone fulfilling the role of scribe would have been useful at the formative stages of the concept mapping, participants appeared to self organise as the sessions progressed. For example, when the group decided a requirement was out-of-scope, there was usually a pause of around a minute as notes were taken to explain the rationale for de-scoping this requirements. When de-scoping took place when the concept maps were more advanced, concept maps needed to be updated to ensure links were not left hanging. By then, however, the group was confident enough to tidy up the concept map and reflect on the consequences of de-scoping the requirement while one person updated the requirements spreadsheets.

5.4 Requirements confidence

As both the category and the consolidated concept maps took shape, participants felt confident enough to address requirements which, up to that point, had been included in the specification but had not been properly addressed in the implementation. For example, the specification made several references to *context* and the need for *webinos* to be *context-aware*. When the map was consolidated, the participants found that the better understanding of the requirements had not helped clarify how to address requirements referencing this term. As a result, the team discussed what the term should mean and, once a proper definition had been agreed, revised both the concept map and the associated requirements based on this improved understanding.

Rounding off the consolidation workshop with the requirements prioritisation exercise re-affirmed the confidence that the team had developed about the requirements. Having now seen a big picture of the requirements, the team felt better equipped to re-prioritise them based on estimated delivery schedule of *webinos*, and their importance, rather than the urgency stated by certain stakeholders.

6 Conclusion

In this paper, we presented an approach where concept mapping was used to both improve the visibility of requirements and, with the aid of tool-support, improve their quality. Moreover, our discussion suggests that, as well as improving the quality of the requirements, the approach also served as a useful pedagogical tool about the value of

good requirements practice. By transposing requirements from text on a screen to a concept name on a piece of paper, and forcing debate around what particular requirements meant for the project, team members saw software requirements in a more positive light. As a foundation for future work, this paper makes two additional contributions.

First, we have demonstrated that sensemaking and organising learning techniques can provide practical support to participative Requirements Engineering activities. In particular, we found that concept mapping gave team members the confidence to make value judgements. We accept, however, when tired and under pressure, such judgements might be made too easily. For this reason, future work considering the group dynamics of concept mapping for supporting software engineering activities would be valuable.

Second, we have shown that lightweight software tools, where applied appropriately, can augment the concept mapping process without getting in its way. Our discussion suggests that desired tool-support shares many of the characteristics of Computer Aided Qualitative Data Analysis (CAQDAS) such as Atlas.ti [33]. While the ability to quickly search for concepts and other data was available once the data had been imported into CAIRIS, it wasn't during the concept mapping process itself. While recent work has looked at how CAQDAS tools might interface with requirements management tools when developing personas [12], our results suggest that understanding how tool interaction might be useful for supporting other design techniques warrants further investigation.

7   Acknowledgements

References

1. Cheng, B.H.C., Atlee, J.M.: Research directions in requirements engineering. In: Future of Software Engineering 2007, IEEE Computer Society (2007) 285–303
2. Nuseibeh, B.: Ariane 5: Who dunnit? IEEE Softw. **14**(3) (May 1997) 15–16
3. Gotel, O., Finkelstein, C.: An analysis of the requirements traceability problem. In: Requirements Engineering, 1994., Proceedings of the First International Conference on. (Apr 1994) 94–101
4. Winkler, S., von Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. Software and Systems Modeling (2009) 1–37
5. Burge, J.E., Carroll, J.M., McCall, R., Mistrik, I.: Rationale-Based Software Engineering. Springer (2008)
6. The Open University: Compendium web site. compendium.open.ac.uk (June 2012)
7. Burge, J.E., Brown, D.C.: Seurat: integrated rationale management. In: Proceedings of the 30th international conference on Software engineering. ICSE '08, ACM (2008) 835–838
8. Faily, S., Fléchais, I.: Towards tool-support for Usable Secure Requirements Engineering with CAIRIS. International Journal of Secure Software Engineering **1**(3) (July-September 2010) 56–70
9. Faily, S.: CAIRIS web site. http://github.com/failys/CAIRIS (June 2012)

10. Faily, S., Fléchais, I.: A Meta-Model for Usable Secure Requirements Engineering. In: Proceedings of the 6th International Workshop on Software Engineering for Secure Systems, IEEE Computer Society (2010) 126–135

11. Faily, S., Fléchais, I.: The secret lives of assumptions: Developing and refining assumption personas for secure system design. In: Proceedings of the 3rd Conference on Human-Centered Software Engineering. Volume LNCS 6409., Springer (2010) 111–118

12. Faily, S., Fléchais, I.: Persona cases: a technique for grounding personas. In: Proceedings of the 29th international conference on Human factors in computing systems, ACM (2011) 2267–2270

13. Russell, D.M., Stefik, M.J., Pirolli, P., Card, S.K.: The cost structure of sensemaking. In: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems. CHI '93, New York, NY, USA, ACM (1993) 269–276

14. Jantunen, S., Gause, D., Wessman, R.: Making sense of product requirements. In: Requirements Engineering Conference (RE), 2010 18th IEEE International. (2010) 89 –92

15. Russell, D.M., Jeffries, R., Irani, L.: Sensemaking for the rest of us. In: CHI 2008 Sensemaking workshop. (2008)

16. Dubberly, H.: Using concept maps in product development. In Kolko, J., ed.: Exposing the Magic of Design: A Practitioner's Guide to the Methods & Theory of Synthesis. Oxford University Press (2011) 109–124

17. Clarke, S.: How usable are your apis? In Oram, A., Wilson, G., eds.: Making Software: What Really Works, and Why We Believe It. O'Reilly (2011) 545–565

18. Martin, B., Hanington, B.: Universal Methods of Design: 100 Ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions. Rockport (2012)

19. Novak, J.D., Gowin, D.B.: Learning How To Learn. Cambridge University Press (1984)

20. Sutherland, S., Katz, S.: Concept mapping methodology: A catalyst for organizational learning. Evaluation and Program Planning **28**(5) (2005) 257–269

21. IHMC: CmapTools web site. `http://cmap.ihmc.us` (June 2012)

22. Klemmer, S.R., Newman, M.W., Farrell, R., Bilezikjian, M., Landay, J.A.: The designers' outpost: a tangible interface for collaborative web site. In: Proceedings of the 14th annual ACM symposium on User interface software and technology. UIST '01, New York, NY, USA, ACM (2001) 1–10

23. Tanenbaum, K., Antle, A.N.: A tangible approach to concept mapping. AIP Conference Proceedings **1127** (2009) 121–132

24. Oppl, S., Stary, C.: Effects of a tabletop interface on the co-construction of concept maps. In: Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part III. INTERACT'11, Springer-Verlag (2011) 443–460

25. Kof, L., Gacitua, R., Rouncefield, M., Sawyer, P.: Concept mapping as a means of requirements tracing. In: Managing Requirements Knowledge (MARK), 2010 Third International Workshop on. (2010) 22 –31

26. AT&T: Graphviz web site. `http://www.graphviz.org` (June 2012)

27. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press, Redmond, WA, USA (2004)

28. Wilson, W.M., Rosenberg, L.H., Hyatt, L.E.: Automated quality analysis of natural language requirement specifications. In: Proceedings of Fourteenth Annual Pacific Northwest Software Quality Conference. Available at http://www.pnsqc.org/proceedings/pnsqc1996.pdf. (1996) 140–151

29. Chernoff, H.: The Use of Faces to Represent Points in K-Dimensional Space Graphically. Journal of the American Statistical Association (1973) 361–368

30. Fuhrhop, C., Lyle, J., Faily, S.: The webinos project. In: Proceedings of the 21st international conference companion on World Wide Web. WWW '12 Companion, New York, NY, USA, ACM (2012) 259–262

31. webinos Consortium: webinos project deliverable D2.1: Use Cases and Scenarios. `http://webinos.org/content/webinos-Scenarios_and_Use_Cases_v1.pdf` (2011 January)
32. Software Freedom Conservancy: git web site. `git-scm.com` (May 2012)
33. Muhr, T.: User's Manual for ATLAS.ti 5.0. ATLAS.ti Scientific Software Development GmbH, Berlin. (2004)