TAGGING KNOWLEDGE ACQUISITION SESSIONS TO FACILITATE KNOWLEDGE TRACEABILITY

HEATHER RICHTER, GREGORY ABOWD
College of Computing and GVU Center, Georgia Institute of Technology
Atlanta, GA 30332-0280 USA
{hrichter, abowd}@cc.gatech.edu

and

CHRIS MILLER, HARRY FUNK
Smart Information Flow Technologies
2119 Oliver Avenue South, Minneapolis, MN 55405-2440 USA
{cmiller, hfunk}@sift.info

Knowledge Acquisition (KA) is important throughout systems development for gathering expert domain knowledge that is incorporated into the requirements and design of a system. There are problems ensuring that accurate and useful knowledge is captured initially, refined as needed, and transferred to later development efforts in a usable format. We present a method, called tagging, for addressing these problems without undue burden on the KA practitioners, along with initial studies to examine the feasibility of real-time tagging and to inform the design of a tool called TAGGER. TAGGER operates by permitting KA discussions to be “tagged” as they happen with concepts and groupings relevant to software development.

Keywords: knowledge acquisition, knowledge management, knowledge capture

1. Introduction

Large systems development projects involve a great deal of knowledge acquisition (KA) to understand the domain, the problem, and user needs. This knowledge must then be refined into domain models and requirements specifications that feed design and development. KA sessions always involve at least one designer and one Subject Matter Expert (SME), but for complex domains, and large knowledge-based systems, sessions may involve many participants and be distributed in time and space. Different sessions may well contradict each other. Knowledge, once captured, can be used in non-obvious ways and is rarely traced through the whole lifecycle of design, usage, test and evaluation, and re-use or modification. This can result in misuse of knowledge and duplication of effort gathering knowledge over the lifecycle of the project. There are substantial tools and environments to facilitate tracing requirements through the software lifecycle, but comparatively little support for tracing the knowledge expressed by an SME into requirements and beyond.

In this paper, we introduce the concept of “tagging”, chunking and indexing knowledge acquisition dialogue using structures that are relevant to software development. We are developing a tool, called TAGGER, to capture knowledge acquisition sessions and allow individuals to tag these sessions as they occur, increasing the usefulness of captured knowledge to software developers and improving the traceability of knowledge back to its initial acquisition without additional time involvement of the KA team. In this paper, we introduce the tagging concept and describe two initial studies showing the feasibility of real-time tagging during knowledge acquisition sessions.

In Section 2, we describe the knowledge acquisition process, and in particular, the problems associated with this activity. In Section 3, we introduce the concept of tagging,
and detail the structures that we are proposing. In Section 4, we assess tagging against a number of KA problems to demonstrate how we believe tagging will benefit knowledge acquisition and software development. In Section 5, we discuss the TAGGER prototype, and two initial studies that demonstrate the feasibility of tagging and will help inform our design of the prototype. Finally, we conclude with a discussion of related work in Section 6 and plans for continuing this research in Section 7.

2. The Knowledge Acquisition Process

The lifecycle of knowledge for a knowledge-based system begins as knowledge is acquired and recorded through knowledge acquisition sessions. Knowledge is then passed along to software development efforts that must translate “raw” knowledge acquired on video or audio tape, in natural language transcripts, or in KA session notes, into whatever representation is ultimately used in design and implementation. Several rounds of testing for validation and then use for verification may follow, with subsequent knowledge tuning phases. From any of these steps, an evaluation of the existing knowledge can be performed and new knowledge requirements can be identified, which restarts the knowledge cycle.

We began our work by identifying a set of problems and challenges in knowledge acquisition and knowledge transfer from the literature [1, 2, 3] and from a specific program, the Rotorcraft Pilots Associate (RPA) [4, 5], an extremely large and knowledge intensive effort conducted by the U.S. Army’s Aviation Applied Technology Directorate. We performed a series of interviews with the leaders of KA and software development on the RPA project, and reviewed the KA documents produced by the RPA team to help identify shortcomings in the RPA knowledge management process. The set of KA problem types we identified include:

- **Distributed Expertise, Distributed Knowledge Capture and Knowledge Integration**: Knowledge must come from experts distributed in time or place, leading to distributed knowledge acquisition activities. This potentially leads to a lack of coordination and/or duplication of effort. Accurately integrating all the distributed knowledge is challenging and problematic.
- **Contradictory Expertise, Individual Differences, and Conflict Resolution**: Experts disagree and have different preferences. These disagreements can be difficult to discover and then need to be resolved once detected.
- **Over-generalization and Under-generalization**: Incorrect belief that previously captured knowledge covers the current case, or the failure to recognize when knowledge that has been captured is similar and does cover the current case.
- **Knowledge Reuse and Knowledge Transition**: Projects can fail to reuse previously captured knowledge for many reasons. Reusing knowledge previously captured for a new platform or system is particularly difficult.
- **Knowledge Requirements Detection**: Ability to determine that additional knowledge is needed.
- **Knowledge Testing**: Difficulties in validating knowledge for a large, complex and context-sensitive system.

The RPA program, using advanced recording and document management tools, captured huge amounts of data—video and audio recordings, transcriptions and some processed documents. Various attempts to structure this data after the fact were seen as generally unuseful both because they were time consuming and because they did not
provide the organization that software engineers needed or wanted. Indeed, the single biggest problem in knowledge acquisition for RPA was agreed to be the useful transfer of knowledge from KA to software development. As in any large development project, the rich body of knowledge acquired in KA must inevitably be reduced to a compressed set of requirements for further software development. But this is inherently a lossy process; much of the knowledge about why requirements are as they are, and how they might be changed, is removed. Yet some form of compression is inevitably required since the raw KA information is too unstructured for the software development team to navigate efficiently. This problem manifested itself in a variety of ways on the program, including:

- KA was seen as little or no use to development.
- Large amounts of data were captured but never looked at again.
- Software development had problems getting answers to their questions.
- Duplication of effort by KA teams.
- An inability to trace and understand the rationale of requirements.

Also, now that the RPA program is attempting to transition its technology to new applications, subsequent developers are finding it difficult to determine or recall why specific requirements and design decisions are in place. These problems are certainly not unique to the RPA program, although they were perhaps magnified by the size of this effort. Despite these problems, KA was profitable, and formed the basis for RPA's functionality. A better process, however, would doubtless have alleviated many of the problems.

3. Tagging

What is needed is a means of facilitating the transition of all information from the initial KA to the software development effort, but in a form that developers will find useful and usable. This means taking knowledge initially captured and facilitating its transition to representation and implementation in a way which maintains and supports navigation back to the richer context of initial knowledge acquisition. This may, in turn, require greater communication and sharing of representations between KA and software development.

Within development, a similar need for communication support exists between requirements developers, designers, implementers, and evaluators. This recognized need has seen the emergence of Computer Assisted Software Engineering (CASE) tool suites designed to facilitate this communication. While these tools have great value within software development, they fail to span the gap between software development and KA.

The missing link is a capability to tie the knowledge captured in KA, the mounds of video, transcripts, and documents, into the structures used in software development. This would create traceability not only from implemented software to an initial requirement, but also from the requirement to the set of (potentially distributed and contradictory) statements that were used to produce that requirement. Thus, this approach differs from requirements traceability approaches, in that it enables the requirements engineer, designer and developer to resort to the earliest source material—that is, to the initial knowledge acquisition sessions—when needed. It would also enable software developers to communicate more precise knowledge needs to the KA team, and help the KA team know when contradictory and/or missing knowledge must be resolved.

The first and most critical step in providing this traceability is to ensure that raw KA discussions are not merely recorded and never viewed again, but are “chunked” –
separated into meaningful units, and “tagged” – linked to descriptors or identifiers to provide traceability back to the initial KA sessions and forward into the requirements and development process. “Tagging” is thus a process akin to making an index for a book—specific terms which are expected to be useful for later readers are organized in an easily searchable fashion along with references to where they are discussed in the text, or in this case, the recording or the transcripts of the KA session. It is not expected that every term in a book would be present in the index—only ones the author deemed particularly salient or important are chosen. Similarly, while some relationships between indexed terms might be reflected in an index, there is no attempt to replicate or summarize the complex relationships from the book itself. These attributes should hold true for conversation tagging as well.

While this sort of indexing can be done after the fact via text or video markup tools, indexing in real time, during the KA, offers several advantages. First, the ability to tag a conversation in real time would greatly speed the process of getting the results of a KA session into the development process, saving both time and money. Second, those present in KA discussions are arguably best suited to process all of the nuances of the KA session—including both technical details and human interaction. Post-processing of video or text transcripts is, all too frequently, left to junior members of a project or secretarial staff—those who arguably have the least awareness of the personalities involved, the background domain, or even of how the knowledge is to be used.

Of course, tagging in real time imposes huge constraints on the tools and the humans who must do that tagging. Not the least of these is that, to be efficient, sessions must be able to be tagged quickly, possibly limiting the complexity of the tags. Tags, like the index terms in a book, must be carefully selected so as not to impose undue workload constraints on human users during real-time conversations, and yet provide significant value for later phases of project development. The set of tags we propose below has been carefully architected to provide value and yet be usable by those engaged in KA sessions. The experiments reported in this paper are designed to establish whether or not we have been successful in creating a usable set of tags; subsequent experiments will establish whether or not those tags provide value.

Another problem with this vision of tagging during initial KA is that the project-specific structures used in software development typically do not exist when the KA team does its job—they have yet to be developed. However, several types of structures, useful for software development, do exist before any design or implementation is done. These simplified structures turn into more formalized representations throughout development. One example is the domain-specific structures that knowledgeable experts have imposed on the domain itself—their jargon terms and labels for the things, actions and conditions in their environment. A second example are the raw, domain-independent structures used in object-oriented software engineering—objects, actors, methods, and relationships. These structures will be imposed on the knowledge of the domain that the experts have and that the KA team will acquire, likely instantiating much of the domain-specific knowledge that is gathered. Finally, there are the conversational structures of the discourse itself, the issues and resolutions that the KA team and experts discuss. Not only do many of the resolutions lead to specific requirements, but such structures will be critical for imposing a high level organization on unfolding conversation, and will be important for tracing relationships in the conversation after the fact.

Table 1 summarizes the set of tags we propose to structure a KA dialogue. Next, we define each tag and present an example tagged discussion. Figures 1, 2, and 3 represent the application of many of these tags to dialogue from a realistic KA session.

### Table 1. Proposed tag categories and types.

<table>
<thead>
<tr>
<th>1) Domain Specific Tag Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Thing</td>
</tr>
<tr>
<td>b) Actions (Tasks)</td>
</tr>
<tr>
<td>c) Conditions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2) Domain Independent Tag Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Classes</td>
</tr>
<tr>
<td>b) Attributes</td>
</tr>
<tr>
<td>c) Operations</td>
</tr>
<tr>
<td>d) Generalization Associations (an ‘is-a’ link.)</td>
</tr>
<tr>
<td>e) Multiplicity (associations between classes)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3) Conversational Tag Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Issues (Open, Closed, or Tabled)</td>
</tr>
<tr>
<td>b) Resolutions (Consensus, Partial or Rejected)</td>
</tr>
</tbody>
</table>

#### 3.1. Domain-specific tags

Domain specific tags denote the specialized terminology of the domain and the unique aspects of the environment. These tags indicate something the developer likely needs to understand, and may become an important reference term for finding relevant passages in the future. Domain specific tags can be of three types: things, actions, and conditions. Things are used to denote concrete nouns in the domain, terminology that is new or used in new ways. Actions are unique tasks or activities performed in the domain. Conditions are areas of dependency, where different decisions or facts would hold under different circumstances.

In the example in Figure 1, military terminology like “friendlies” and “enemy units” are tagged as a thing. Other terms, such as “mobility projection areas” or “movement areas” are also important objects in this discussion. An activity of the domain, “fratricide” is tagged as an action. Additionally, SME3 says that under certain conditions, namely wearing “Night Vision Goggles”, the use of red may be problematic. Thus, “Night Vision Goggles” is marked as a condition.

#### 3.2. Domain-independent tags

Domain independent tags are based on the structures of object oriented programming and design [6]. Their aim is to help form the model of the project domain. Thus, the tags are class, attribute, and operation. Classes are the objects of the domain, objects that one would expect to instantiate within a system. Attributes are characteristics of a class, including specified ranges or default values of such descriptions. Operations are actions that a class may take, things that can be done with or to the entities represented by a class. Classes can be related via generalization, showing inheritance or the “is-a” relationship.

---

KA1: What color would you like mobility projection areas to be on the TSD?

SME2: They're related to enemy units, right? Enemy units are red, so movement areas should be some shade of red, maybe mauve.

SME3: Red’s going to cause problems with the Night Vision Goggles. Besides, movement lines and arrows are always green—they should be some shade of green.

SME1: Yeah, but that’s only because we only do movement rings for friendlies now. If we do them all in green, they’ll be confusing and make fratricide more likely.

SME2: I think I agree. Can we do some shade of red without obscuring all the other symbology?

KA1: We’ve used this semi-transparent, strawberry milkshake color in the past and gotten good visibility.

SME2: So, red it is then—or that pink color, right?

SME3: I guess so.

---

Figure 1. Domain specific tags.

between two classes. Additionally, multiplicity is tied to associations between classes and indicates how many objects may participate in the association.

There is likely to be a correlation between these tags and the domain specific tags as domain dependent entities are instantiated in a domain model. For example, objects and attributes may correspond to things while operations are likely to be actions. Additionally items tagged as conditions may correspond to attributes of relationships between objects. This redundancy is intentional and important because it allows different types of tags to be assigned by different users—some of whom may be familiar with the categories of object oriented programming and others not.

In Figure 2, “mobility projection areas” and “TSD” have been tagged as classes while “color” is an attribute of mobility projection areas. The colors “red” and “green” are also attributes, specific colors that are being discussed. The relationships between classes, and between the attributes and classes, are shown as a line between tags.

It is worth noting that these tags represent a highly simplified version of the concepts and relationships available in object oriented software engineering tools and techniques. Their expressive power is correspondingly weak compared to other diagramming systems, but this is intentional and desirable. We don’t expect the KA team to build class diagrams on the fly during a KA session, but the simple tagging of concepts that occur during discussion with some or all of the syntactic structures that software development provides will both identify and categorize those concepts for later software development and improve traceability of these concepts back to the original experts.

Finally, there is no example of an operation in this conversational fragment, though in the broader conversation tasks such as “reconnaissance” or specific actions taken with the objects discussed above (e.g., “predict enemy location” via mobility projection areas) would be examples.

3.3. Conversational tags

The goal of the conversational tags is to help capture the structure or flow of the conversation and record the arguments and outcomes of the discussion. These tags are derived from the rationale capture annotations described by Conklin and Begeman [7].

An issue is, basically, a topic of investigation or discussion. In knowledge acquisition sessions, issues frequently correspond to questions. Each issue tag has a status associated with it. The status can be:

- open—currently being discussed;
- closed—a resolution has been decided and the issue is not expected to produce further discussion; or
- tabled—discussion of the issue has been abandoned without resolution.

Additionally, issues can be related to each other, as one may follow on another, or be a sub-issue of another issue.

Resolution tags are used to mark the end or closing of discussion on an issue. Thus, they should be linked to a prior issue. Resolution tags also have a status. The status is used to reflect the tagger’s opinion of how universal the agreement was on the resolution. They are:

- consensus—the group generally agrees with this outcome;
- partial—there is majority agreement, but some hesitancy, reluctance or partial disagreement; or
- rejected—there is no general agreement to this outcome, a significant number of participants hold different opinions.

In the example in Figure 3, the conversation revolves around the issue of what color to make the mobility projection areas, with a sub-issue of whether some shade of red would work. A shade of red is suggested, but is only partially agreed upon.
Issue: What color would you like mobility projection areas to be on the TSD?

SME2: They’re related to enemy units, right? Enemy units are red, so movement areas should be some shade of red, maybe mauve.

SME3: Red’s going to cause problems with the Night Vision Goggles. Besides, movement lines and arrows are always green—they should be some shade of green.

SME1: Yeah, but that’s only because we only do movement rings for friendlies now. If we do them all in green, they’ll be confusing, and make fratricide difficult.

SME2: I think I agree. Can we do some shade of red without obscuring all the other symbology?

KA1: We’ve used this semi-transparent, strawberry milkshake color in the past and gotten good visibility.

SME2: So, red it is then—or that pink color, right?

SME3: I guess so.

4. Tagging Assessment

The example shown in Figures 1, 2, and 3 illustrates several potential problems in knowledge acquisition -- dissenting opinion, individual preferences and differences in assumptions that are sometimes lost in a KA session, and are frequently lost when the knowledge is ‘boiled down’ to a set of requirements.

The example dialogue concerns a disagreement among several subject matter experts over what color the mobility projection areas for threats should be on a certain display. While agreement was reached on a specific color, there were issues raised that may not have been fully addressed during the KA session, and it was unclear that all participants fully agreed to the outcome. Thus, a stated requirement from this session would likely call for a pink color, yet problems may arise if this color were to be used with Night Vision Goggles, for example.

In such a situation, the goals of any recorded KA information should be to help development team members find the agreed upon resolution and be aware that there were alternatives and possible issues left open concerning this decision. We will detail how tags may help solve these problems.

*View the agreed upon resolution:* The conversation tags mark the issue of color, and the agreed upon resolution. Additionally, the domain-specific tags mark a new term – “strawberry milkshake pink” – that also marks this decision. The domain independent tags enable us to see that this conversation was about the color attributes for the mobility projection ring symbology, and to discover which color attributes were discussed.

*Be notified that alternatives of red and green were considered:* Again, the domain-independent tags show that several alternative colors were discussed prior to the decision of pink.

Be notified of an open issue over compatibility with Night Vision Goggles: While the domain-independent tags show several alternative colors were discussed, the domain specific tags mark a particular condition, the Night Vision Goggles. Additionally, the sub-issue of obscuring other symbology emphasizes that there is concern, and the partial resolution in the conversational tags shows that not everyone agreed that the chosen color met this issue.

Table 2. Tagging addressing knowledge acquisition problem types.

<table>
<thead>
<tr>
<th>KA Problem Type:</th>
<th>Relationship to Tagging to problem type and software development transition</th>
<th>Tagging Solves?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed</td>
<td>Better integration between KA and development will help organize, prioritize and identify knowledge that needs to be captured and where it ‘goes’ in development once captured</td>
<td>Very High</td>
</tr>
<tr>
<td>Expertise and Distributed Knowledge Capture</td>
<td>With tagging, and with a shared repository of structured knowledge once captured, multiple views of that knowledge will be enabled, and knowledge integration will be improved.</td>
<td>Very High</td>
</tr>
<tr>
<td>Knowledge integration</td>
<td>Better integration will help identify the role of knowledge once captured—identifying disagreements and creating better indexes so contradictory knowledge gets captured and accessed better</td>
<td>Very High</td>
</tr>
<tr>
<td>Contradictory Expertise or Individual Differences</td>
<td>Tagging will offer no particular benefit to solving the problem beyond improved detection and tracing of it as mentioned above.</td>
<td>Very Low</td>
</tr>
<tr>
<td>Conflict Resolution</td>
<td>Tagging will help with improved organization and linking new knowledge against previous knowledge. However, incorrect beliefs are a largely a human difficulty, and will be hard to overcome.</td>
<td>Low</td>
</tr>
<tr>
<td>Overgeneralization and Undergeneralization</td>
<td>Improved capture and structuring of knowledge may reduce duplication and improve the spotting of re-use opportunities.</td>
<td>Moderate</td>
</tr>
<tr>
<td>Knowledge Reuse and Knowledge Transition</td>
<td>By linking captured knowledge into emerging structures, tagging can potentially facilitate the identification of missing knowledge.</td>
<td>Moderate</td>
</tr>
<tr>
<td>Knowledge Testing</td>
<td>Tagging is not targeted at this problem.</td>
<td>Very Low</td>
</tr>
<tr>
<td>KA Seen as Not Useful</td>
<td>Tagging organizes the knowledge in structures already used by software development, improving the relevance to development.</td>
<td>Very High</td>
</tr>
<tr>
<td>KA Structure Ignored</td>
<td>Structure used by KA is already in terms used by software development.</td>
<td>High</td>
</tr>
</tbody>
</table>

In Section 2 we listed multiple problems types of knowledge acquisition, including several we found in examining the RPA program. In Table 2, we present these problem types as well as the relationship that we believe tagging has in addressing them. Tagging obviously does not help solve every KA problem. It is designed to help structure

captured knowledge to improve the integration with software development efforts. Thus, we do believe tagging addresses many important problems by improving the accessibility, traceability, and relevance to software development.

5. Designing TAGGER

“Tagging” is the concept of assigning one or more of the tag types to a dialogue of a knowledge acquisition session. To tag effectively and efficiently, we argue that the initial identification and linking of important information should be done in real time – during the KA session. This ensures that those already participating in a KA session are helping to structure the captured knowledge, and requires no additional time for the KA team. Some amount of structuring and organizing of recorded KA audio or video is certainly performed as a part of KA activities today—but almost always after the fact when some usually junior member of the KA team is required to wade through hours of captured information to derive those portions deemed relevant. This process is inefficient and lossy, and often doesn’t occur because of the time and effort involved. Instead, we believe we can ensure tagging richness and comprehensiveness by enabling multiple knowledgeable participants to do that tagging from multiple viewpoints, simultaneously, during the KA session.

We are developing a tool, TAGGER (Team-Aware Acquisition Guide for Goals, Entities, and Relationships), to capture and record knowledge acquisition sessions, and enable tagging of that recording in real time. TAGGER will consist of hand-held devices, either PDAs or tablets, for individuals to perform tagging. In one context of use, all meeting participants might have access to all tag types and thus be able to assert any of them at will. This would allow great flexibility but would probably not result in very “complete” tagging. If the goal were to have a thoroughly tagged record of a KA session, then it would be likely that specific individuals attending the meeting be assigned the responsibility of tagging. This is not inconsistent with the practice of assigning a specific note takers in current KA practice in large projects. We further expect that different users will be responsible for tagging different tag categories, thus reducing the amount of tagging required of any individual, yet allowing multiple viewpoints to be tagged. The ultimately envisioned system will automatically record audio and video, transcribe the audio via speech-to-text transcription software, and synchronize all of the streams of tagging and input. Additionally, we are examining technology that could provide some automated tagging using dictionaries of terminology for example. Thus, TAGGER will not only enable full capture of knowledge acquisition sessions, but real time structuring and indexing of those recordings to provide traceability and highlight important aspects of KA sessions.

In designing and building TAGGER, we will be examining how users are able to recognize and annotate the various concepts, including their understanding of the different tags, the interactions that facilitate tagging, and the session variants that affect the quality of tagging. In the following sections we describe our first two studies investigating tagging to show the feasibility of this concept, to inform the design of TAGGER and to ascertain the viability of the set of tag types we have designed.

5.1. Tagging transcripts

In designing TAGGER, our first goal is to understand users’ behavior in tagging real KA sessions, their understanding of the various tags, and their ability to tag in real time.
These results will provide data to serve as a baseline for understanding the performance of future TAGGER prototypes and may signal the need to modify the proposed tag set as well.

We performed a small study using twelve computer science graduate students at Georgia Tech. These students have backgrounds similar to that of a junior member of a KA team – a graduate in computer science with some experience in development. Each subject was given a training document explaining knowledge acquisition sessions, the tagging concept, and details and examples of each tag. They were then given a paper transcript of a knowledge acquisition session to tag by marking up the transcript. Two conditions were examined: real-time and non-real-time. Six subjects took as much time as they liked to tag the discussion; the other six tagged the paper transcript in real time, as a video of the discussion played, and were required to stop when the video ended. We hypothesized that those with time constraints would tag less than those without the time constraints.

The recorded KA session was an hour-long interview between one of the authors and two airline pilots, focusing on take off procedures and checklists with the purpose of ascertaining how they might use an electronic checklist system for the cockpit. While the session was not part of a real project, the interview itself is realistic, involving real pilots giving real answers to questions about their domain and a potential system to be used in it.

From that KA session, we selected several multi-minute dialogues on different topics to use for this study. Non-real-time subjects were asked to read the transcript of approximately 11 minutes of dialogue and tag it for one category of tags at a time. They then re-read the same transcript, and tagged it with another category. The order in which they were asked to assign tag categories was varied from subject to subject. In the real time condition, subjects used one tag category at a time, and tagged three separate segments of dialogue, each approximately 5 minutes long. All subjects tagged by circling or underlining tags, and using an abbreviation to mark specific tag types within the general category. To relate, or link, tags they drew lines between them.

Subjects reported having no difficulty understanding the tagging concept, or any of the individual tags. There was much variation in what was tagged, yet there were no clear mistakes or errors made in tagging. In other words, all tags asserted can be interpreted as reasonable. The multiplicity tag was rarely used. This may be due to a lack of applicable instances in the transcript, but also suggests that removing this tag from the tag set may be warranted. In the non-real-time condition, subjects reported that conversation tags were the most difficult to assign because their use spans greater distances in the transcript in order to track and link issues and resolutions. However, this was not the case for the real time subjects, who instead found conversation tags much easier to keep up with, and domain independent tags the most challenging. Relating classes, attributes, and objects required users to look back in the transcript, find the tag, link it, then catch back up with the conversation.

Table 3 summarizes the tagging frequency results from both conditions, with column 1 showing the results of the non-real-time subjects, and column 2 showing the real time results. The numbers are in tags asserted per minute of conversation, and are shown for each category of tags. Despite our expectations that tagging frequency would be noticeably less, real time subjects performed about the same as non-real time subjects. In other words, the time constraints did not adversely affect tagging frequency.

Overall, this study showed that tagging is an understandable process, and that it can be done in real time. Yet the results do vary substantially from user to user -- what is important and how to tag it is open to interpretation—again, like creating the index for a book, some authors are better at it than others and some indices are better for a subsequent reader’s needs than others. This is not negative, as long as the knowledge acquisition team can have confidence that the session is being tagged reasonably. However, in the context of a real project, with a better understanding of the goals of the knowledge acquisition session, this variation may be reduced. Additionally, experience will also likely reduce variation as users learn how they utilize different tags. Subjects in this experiment felt that tagging was challenging, requiring a lot of thought. Thus, knowledge acquisition team members may not be able to keep up with such a high rate of tagging over a long period of time. Again, experience will likely make this task easier.

<table>
<thead>
<tr>
<th></th>
<th>Non-Real-Time Rate</th>
<th>Real Time Rate</th>
<th>Error + Delay Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-specific tags</td>
<td>7.5</td>
<td>9.4</td>
<td>5.1</td>
</tr>
<tr>
<td>Domain-independent tags</td>
<td>12</td>
<td>9.8</td>
<td>5.8</td>
</tr>
<tr>
<td>Conversation tags</td>
<td>1.8</td>
<td>3</td>
<td>2.5</td>
</tr>
</tbody>
</table>

5.2. Tagging with error and delay

In the initial study, subjects tagged human-transcribed, error-free records of the KA session which had been prepared in advance of the trial. Any foreseeable speech-to-text transcription software will never be error-free and will, inevitably, introduce some time lag between an utterance and its appearance in the transcript. Thus, in our next study we looked at the effects of the errors and delays that automated transcription might introduce.

Again, we used 6 computer science graduate students as subjects, and they tagged the same dialogue in the same manner as the real-time condition previously. However, in this case, the transcripts were created using existing speech-to-text software, and were delivered on a tablet computer, in chunks of 2 to 5 seconds, with a delay of 1 second. Thus, subjects would see text anywhere from 1 to 7 seconds after it had been said. Due to high background noise in our original video recording, we produced the transcripts by reading back the exact transcript—including all of the original “ums,” repeated words, etc.—into a version of Dragon’s Naturally Speaking 7.0 that was not trained for the speaker. Some of the dialogue was read at a slower and clearer pace than the original video. Thus, our study represents something approximating a best-case scenario of what might be produced in a multi-speaker (individually miked) real time conversation in the not-too-distant future. The transcripts were approximately 75% correct, but with no punctuation. However, many of the errors were not in the distinct, domain-specific
words, but in transitions or the beginnings of sentences, and thus, in words that were less likely to be tagged.

The tagging frequency results of this study are shown in Figure 3, the third column. In this case, performance was impacted for both domain independent and domain specific tags, but not conversational tags. As reported by the subjects, the delay in seeing the transcript did not cause many problems, as subjects had difficulty keeping up as it was. However, the errors in the transcript did have a major impact. As one subject reported, the errors were “distracting,” increasing the amount of time to find and tag phrases, and did cause subjects to tag useless phrases by mistake in several instances. However, given that many of the words likely to be tagged were recognized correctly, we don’t necessarily need a better recognition rate. Instead, punctuation, or some other form of phrase segmentation, may aid in matching what was heard with the text more easily, and thus, help users find the desired phrases to tag more quickly. Thus, even though performance was impacted by the transcripts, tagging was still able to occur and may be improved without requiring improved recognition. We also need to investigate opportunities for some types of automated tagging interactions to minimize the workload of asserting tags.

6. Related Work

Many of the common problems which TAGGER is designed to solve are understood and documented in the process of knowledge acquisition (e.g. [1, 2]). Much of the work in the KA community has focused on techniques for eliciting such knowledge, while much of the work in the software development community has focused on managing such information once it enters into the implementation and design process. Indeed, much of the recent development of integrated software development processes and the modeling languages and tools which support them, e.g. [8], can be seen as knowledge development processes, though they are frequently weak on how to obtain that knowledge in the first place. Our work, and the TAGGER concept, largely falls between these two camps and attempts to bridge them. As such, it is closer in spirit to early KA work such as Bullerme et al [9], which documents the advantages of using an Expert System engineering tool in a KA session, permitting the experts to discuss and validate each other’s knowledge and opinions.

Requirements analysis and requirements traceability are well-studied areas in software engineering. Various tools, such as DOORS [10] and RTM [11], among numerous others, are designed to help requirements engineers create, modify, and trace requirements back to original documents or scenarios. However, these tools were not meant to help structure and create requirements from highly unstructured and voluminous information, such as the transcripts generated from KA sessions. Thus, while these tools do aid in the traceability of the requirements back to their creation, they still do not bridge the gap back to the origin of the knowledge that went in to that requirement, the original discussion with the domain expert.

The TAGGER approach has, perhaps, more in common with Design Rationale Capture approaches. Here the goal is not so much to capture initial knowledge from a domain expert for use in design, but rather to capture the rationale for a design, wherever it might have come from, for use by later designers who need to know the motivations, assumptions, and decisions behind a system for further development. Many early approaches in this domain focused on formal representations of the issues, alternatives, and decisions of a design, such as IBIS (Issue Based Information Systems) [7]. Indeed,

our conversational tags are based on IBIS. However, a common problem with such approaches has been that they require much time and effort on the part of initial designers, while providing benefit only to other ‘downstream’ users. Tagging may well suffer from similar problems, but we are taking several related approaches to mitigating these problems. First, we are simplifying the set of tags which initial users are required to assert. Second, we have focused tagging at a point in the design process where knowledge must be transferred within initial product design, thereby making the benefits of good rationale capture more immediate to all project participants and worth the efforts of the KA team members. Finally, rather than placing all of the rationale into one representation, we are instead exploiting concepts already important to software developers to help structure the rich KA recording, thus making the knowledge and rationale contained within that recording more accessible.

The concept behind the tool TAGGER is similar to video annotation systems, or systems to aid in qualitative research. Systems such as Eva [12] and Marquee [13] allow researchers to annotate video of an evaluation session with keywords and notes. These systems may provide suggestions towards the design of TAGGER. However, little research has been done on the interface capabilities that facilitate this type of annotating, and on users’ performance when these activities are done in real time.

7. Conclusions and Future Work

Large systems development projects involve a great deal of knowledge acquisition in order to understand the domain, the problem, the needs of the users, and refine all of this knowledge into detailed domain models and requirements specifications that feed design and development. Yet in translating knowledge into requirements and design, much of the rich context of the original knowledge acquisition session is lost. Big programs like RPA illustrate the problems that happen when these knowledge acquisition discussions are not adequately disseminated among the lifecycle of the development effort and beyond. Yet, simply recording the conversations is not enough. We aim to improve the transition of knowledge into software development by organizing the original captured acquisition sessions with structures relevant to software development, subsequent project revisions and even later redesigns. Later knowledge inspection is facilitated, with links back to the original discussion that are available when needed.

We have proposed a set of tags – domain specific, domain independent, and conversation tags – that members of the knowledge acquisition team can assert during knowledge elicitation. We are designing TAGGER, a tool to capture KA sessions and allow individuals to assert tags against the ongoing conversation. As part of our design of TAGGER, we have completed initial studies that confirm that tagging can be performed by a KA team member, even in real time and with automatically generated transcripts.

We are continuing to design and build TAGGER. Our goal is to find the “sweet spot” where the benefits to later knowledge users are worth the work required to tag KA sessions in real time. As part of that effort we will be examining the interactions that the tool must provide to facilitate tagging, and minimize the workload of an individual tagger. Beyond building a tool to capture tagged knowledge acquisition sessions, we will also examine the use of tags in software development activities, such as requirements specification or domain modeling, to better understand how tags can be utilized and the value that tags provide.

Acknowledgements

This material is based upon work supported by the U.S. Army Aviation Applied Technology Directorate (AMCOM) under Contract No. DAAH10-02-C-0005.

References
