Facilitating tacit-knowledge acquisition within requirements engineering

Abdulmajid Hissen Mohamed
Department of Computer Science
Sebha University
PO Box 18758, Sebha, LIBYA
abdulmajid.h@gmail.com    http://www.sebhau.edu.ly/cs/faculty.html

Abstract: - Software maintenance represents one of the most challenging tasks for software engineers. This can be attributed to many problems related to how software applications are built. However, the lack of enough historical knowledge about legacy software projects is a major software maintenance issue. Though software documentation is heavily used to guide maintainers tasks, but it only cater for documented experience knowledge in the form of diagrams, code, test cases, etc. On the other hand valuable experience knowledge can not be recalled simply because it is implicitly embedded in the minds of expert software engineers. This includes views, assumptions, and observations made as part of managing legacy software projects. The lack of such valuable experience knowledge during software maintenance would certainly lead to misinterpretations and wrong assumptions about the software being maintained. Within the software lifecycle, software requirements phase accommodates extensive expert deliberations. This represents a major source of software tacit or undocumented knowledge. Capturing tacit knowledge in the form of requirements rationale is expected to provide greater help for software maintainers to understand the complexity of the software application being maintained. This paper presents an approach for capturing experts' tacit knowledge. It is aimed to provide the ability to capture requirements tacit knowledge resulted from the collaborative requirements verification and validation.

Key-Words: - Requirements engineering, Knowledge Management, tacit knowledge, knowledge reuse

1 Introduction
Software engineering is a team-based process, and any collaborative task involves great part of deliberation and discussion between members involved. Meanwhile, huge volume of professional knowledge is usually communicated as part of the software team deliberations. Usually part of this knowledge is explicitly documented in the form of meeting minutes, modelling diagrams, test cases, code, etc. This explicitly documented knowledge can be organized and shared easily. But, Substantial experience knowledge remains undocumented and implicitly kept in software engineers' minds. This experience knowledge is classified as tacit knowledge, which is usually communicated orally or through observation. Though its importance, capturing tacit knowledge has twofold challenges, firstly it is unseen and secondly it is usually unconsciously exploited by knowledge experts. In other words it is hardly explicated. This characteristic is best reflected by Polanyi’s theory of personal knowledge “we know more than we can tell” [1]. In fact, usually experts practicing their craft demonstrate know-how and do so without conscious reflection [2]. From the viewpoint of organizations, knowledge is central to the competitive advantage of organisations [3], and therefore the issue of tacit-knowledge mismanagement forms a major threat for organisations. Because though experts' know-how should be considered as part of the organizational memory, but organizations have no control on the experience knowledge kept in experts' minds. This is especially applicable to knowledge-intensive organizations such as software organizations. According to Hoffman et al[4], such organisations are subjected to lose their ability to conduct business as their workforce ages and critical knowledge walks out the door. The rest of the paper is organised as follows: an overview of knowledge management is highlighted in Section 2. An overview of software requirements engineering is presented in Section 3. Sections 4 discusses the characterization of tacit knowledge generated as part of the requirements engineering phase. The proposed approach to tacit-knowledge management is presented in Section 5 followed by overview of related research. The paper ends with a conclusion and suggestion for further work.

2. Knowledge management
Currently, knowledge management is a very active multidisciplinary research. It aims to formulate knowledge models and group-communication frameworks to manage knowledge creation and reuse. Nonetheless, the term knowledge still sometimes considered as a buzzword. According to Fenstermacher, despite debating the topic for millennia, philosophers have yet to agree on a definition of knowledge themselves [2]. Traditionally, knowledge is described hierarchically with the concept of data, information and knowledge [5]. And in regard to knowledge taxonomy, knowledge management researchers classify knowledge as explicit (i.e. formal) and tacit (i.e. informal) knowledge. Formal knowledge is the stuff of books, manuals, documents, memos, white papers, plans and training courses, whereas informal knowledge is the knowledge that is created and used in the process of creating the formal results. It includes ideas, facts, assumptions, questions, guesses, stories, points of view, etc.[6]. In other words, tacit knowledge constitutes what Koskinen describes as the practical know-how [7], which cannot be transferred simply by symbolic communication [8]. However, tacit and explicit knowledge tend to co-exist [9], because tacit knowledge is often crucial for the interpretation of the explicit knowledge” [10]. It forms what Gal et.al calls the guidance of human behaviour [11], because in any problem solving process, experts usually rely on the experience they had which deeply embedded in their minds.

Traditionally, documenting explicit knowledge is the common practice. But recently there is a growing recognition that tacit knowledge management is expected to provide great improvement to computer supported decision making. According to Zack [12], explicating tacit knowledge so it can be efficiently and meaningfully shared and reapplied, especially outside the originating community, is one of the least understood aspects of knowledge management.

As part of the conscious and unconscious use of experience knowledge, experts’ knowledge tends to develop from tacit to explicit and vice versa. This form of knowledge dynamics is depicted by Nonaka’s model of knowledge creation and transformation a.k.a. SECI [13]. As shown in Fig. 1, Nonaka defined four modes of knowledge conversion, firstly, in the socialisation mode (tacit to tacit), knowledge workers acquire new knowledge directly from each other. Secondly, the externalisation mode represents the articulation of tacit knowledge into tangible form. Thirdly, in the combination mode, different forms of explicit knowledge are combined to generate new factual knowledge. Finally, the internalisation mode (Explicit to Tacit) comes as a result of the three previous modes. Through experience, workers enrich their understanding and new tacit knowledge is embedded into their mind as a result. Notice that Nonaka’s model considers tacit knowledge as mainly generated and reused as part of the socialization cycle. Because humans naturally share knowledge by telling stories [14] and debating.

3. Software requirements engineering

Software requirements engineering (RE) is the initial phase of software development lifecycle. It is the phase where customers’ requirements are identified. This process involves lengthy customer-developer and developer-developer deliberations. The aim is to conclude complete, accurate and unambiguous list of software project requirements. An individual software requirement can be defined as a capability or a condition needed by a client to accomplish software facilitated tasks. Meanwhile, the requirements engineering process is concerned with the identification, modelling and verification of the functionalities of a software system. This includes the context within which the system will be developed or operated. RE has four main tasks includes requirements elicitation, negotiation, specification, and validation/verification [15]. There are many requirements elicitation techniques available such as Joint Application Development (JAD) [16; 17], Storyboarding [18], and Rapid Application Development (RAD) [16; 19]. The objective of these techniques is to provide requirement engineers or system analysts a platform to conclude final list of requirements collaboratively. However, in terms of capturing tacit knowledge, none of these techniques pay attention to documenting the rich collaborative discussions held during the RE process.

4. The Characterization of tacit knowledge embedded in RE phase

Because requirements engineering involves intensive discussions and deliberations, this makes it the richest software development phase in terms of tacit knowledge generation. Meanwhile, numerous studies asserted that higher percentages of software failures are attributed to poorly articulated
requirements. According to Grünbacher and Briggs, one common cause of poor requirements is that critical knowledge of stakeholders remains often hidden and unshared in the course of a negotiation [20]. This is happened because conflict is inherent in any team-based project such as software engineering projects. As part of the requirements verification and filtering, requirements engineers need to examine huge matrix of features, technical, and domain constraints. Within this collaborative process, arguments and conflicts arise naturally to form the requirements rationale. Traditionally, the rationale behind the concluded list of requirements is not documented.

The representation of tacit knowledge in the form of requirements rationale is a very complex process, because it may take many forms including gestures, signs, and other forms of personal expression. Accordingly, it is hardly possible to manage the mapping of the full richness of discourse elements into a formal representation. However, simple discourse ontologies can be employed to grab significant part of tacit knowledge, keeping in mind that users tend not to disclose all information they know.

5. Our approach
Our approach relies on the use of IBIS model [21] as an ontology to represent requirements' rationale. IBIS is initially proposed as generic deliberation ontology to capture design rationale. Fig. 2 shows our adaptation of the IBIS deliberation ontology.

---

**Tacit knowledge**

<table>
<thead>
<tr>
<th>Socialization</th>
<th>Externalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internalization</td>
<td>Combination</td>
</tr>
</tbody>
</table>

**Explicit knowledge**

<table>
<thead>
<tr>
<th>Req. Engineer</th>
<th>Posts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related to N</td>
<td>Replied to N</td>
</tr>
</tbody>
</table>

**Figure 1: Nonaka’s model of knowledge creation**

**Figure 2: An adapted version of IBIS argumentation model**
The adapted model provides requirement engineers simple vocabularies to express their argument details. It encourages team members to debate the validity of elicited requirements. Basically, the model provides debaters a formalism to represent issues that need to be debated, and members’ positions in response to raised issues. Positions represent arguments to support or disprove these other members’ positions [21]. Our adaptation attached deliberation details to individual software requirement items proposed by requirement engineers. Fig. 3 shows part of our implementation of the proposed approach in capturing software requirement’s tacit knowledge. The screenshot represented by Fig. 3 shows a sample of an individual functional requirement. The lower pane of the screenshot shows deliberations conducted as part of the verification and the approval of the sample software requirement. The title of the sample software requirement is ticket payment. It is part of the requirements list of the budget public bus system (BPBS). Each software requirement can be debated while its associated arguments are captured in a structured format showing its approval’s rationale. RE team deliberation is started by one of the team members raised the issue of what forms of ticket payment should be considered (i.e. cash or prepaid tickets or credit cards). In response, a team member replied by supporting the previous argument. He/she suggested the use of machine-readable prepaid tickets. As shown in Fig. 3, the two followed arguments are posted in favour of the use of machine-readable prepaid tickets. Both justified their arguments by avoiding cashed change or paying extra service and telecommunication cost if credit cards are used. Each argument is visually recognised by a one of the symbolic icons shown in Table 1.
expected to help maintenance engineers to have exposure to all historical issues related to the software requirements being maintained. In addition, this would also help apprentice requirement engineers to learn from experience of skilled requirement engineers.

<table>
<thead>
<tr>
<th>Argument type</th>
<th>Symbolic icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td>![Issue icon]</td>
</tr>
<tr>
<td>Supporting argument</td>
<td>![Supporting icon]</td>
</tr>
<tr>
<td>Objection argument</td>
<td>![Objection icon]</td>
</tr>
</tbody>
</table>

6. Related work

There are various attempts made to capture tacit knowledge. A generic approach is exemplified by OMEX [14], which is a web-based knowledge acquisition tool aimed to build a large-scale commonsense knowledge base. OMEX's knowledge base is populated by descriptions and explanations of everyday, commonsense experiences from volunteer contributors distributed across the Internet. Readers[10], is also a tacit-knowledge management approach aims to replicate and transfer of experimental know-how issues in the form of software Lab Packages. Each laboratory package describes an experiment in specific terms and provides materials for replication, highlights opportunities for variation, and builds a context for combining results of different types of experimental treatments. Our previous work, LiSER [22], also represents a similar approach to tacit-knowledge management, however, the scope of LiSER includes knowledge artefact in all software development phases. Asgari et. al. [23] proposed the “tribal lore” or “folklore” which constitutes experts’ knowledge collected through surveys and group discussions.

Another domain-specific approach to capture tacit knowledge is proposed by Abidi et al [24]. It is based on defining a health-care “scenario” which is a goal-oriented description of the problem situation. Each scenario includes the “environmental context; the problem description in terms of actors, role of actors, temporal events and inputs; and the problem’s solution in terms of the expert’s interventions and outcomes”. Each scenario goes through a crystallization process during which it is assessed and validated by experts and practitioners, and finally made available for downstream knowledge sharing and utilization. The approach introduced by Friedrich and Poll [25] is the nearest to our approach. They too focus on the requirements engineering phase, but they mainly focus on capturing customer's tacit knowledge rather than the tacit knowledge owned by software engineers. Their assumption is that requirements engineers need to tap into customers' tacit knowledge in order to maintain full understanding of the application domain. In many situations customers presume that requirement engineers are familiar with certain domain-specific business details, so they do not elaborate on that. But what might be 'obvious' to customers is necessarily the case for software engineers.

7. Conclusion

Organisations competitiveness is under threat as a result of workforce aging and other management practices such as downsizing and layoff. Critical experience in the form of workers' tacit knowledge could be lost consequently. Software engineering is a very knowledge-intensive task and a great portion of software engineering experience is usually held in professionals’ heads as practical know-how. This paper introduces an approach to capture tacit knowledge resulted as part of the requirements engineering process. We adapted the IBIS argumentation model for the characterisation of requirements tacit knowledge in the form of asynchronous arguments posted by software requirement engineers. Eventually, the captured deliberations represent the rationale associated with each individual requirement of software projects. Managing the corpus of the captured tacit knowledge is then expected to provide software maintainers with relevant historical knowledge which is very critical to accomplish software maintenance easily.

References:

Hawaii International Conference on System Sciences, IEEE.


