LiSER: A Software Experience Management Tool to Support Organisational Learning in Software Development Organisations

Abdulmajid Mohamed, Sai Peck Lee, and Siti Salwah Salim
Faculty of Computer Science and Information Technology, University of Malaya, Malaysia

Abstract: *The efficient management of experience knowledge is vital in today's knowledge-based economy. This paper is concerned with developing a software experience management tool as an organisational memory subsystem. The tool aims to support Knowledge Management (KM) and Organisational Learning (OL) activities in a typical software organisation. It is specifically targeted to capture the pearls of tacit knowledge in the form of Knowledge Asset (K-Asset), which only surface as the outcome of collaborative analysis and refinement of the captured knowledge. The prototype tool is based on the framework for collaborative organisational learning we developed in previous research.*

Keywords: *Knowledge management, organisational memory systems, tacit knowledge, organisational learning, ontologies.*

Received July 27, 2003; accepted February 9, 2004

1. Introduction

This paper describes LiSER (Living Software Experience Repository) as an Organisational Memory (OM) subsystem. It aims at supporting knowledge management activities in a typical software organisation. LiSER can be described variously as an "experience rationale capture tool", a "knowledge management tool for software development knowledge" or a "software experience repository tool". It is intended to provide means to capture both tacit and explicit knowledge in a way they act as correlated information sources. However, LiSER is not meant to be a tool for software project management or a Document Management System (DMS) for software development projects. While these two systems form an essential part of the organisational memory approach, they mainly focus on explicit knowledge and they pay less attention to tacit knowledge. LiSER on the other hand put more emphasis on tacit knowledge management where the research contributions are less.

By explicit knowledge we mean "knowledge that has been captured and codified into manuals, procedures, and rules, and is easy to disseminate" [13]. While tacit knowledge represents the undocumented information that usually reside in workers' minds. Tacit knowledge is usually embedded as insights, views, know-how, etc. This type of knowledge is usually an organisation-based (i. e. cross projects) knowledge while explicit knowledge is largely project-based knowledge.

According to the KM literature, more attention has to be paid to managing tacit knowledge as it has more influence on upcoming practices. For the case of software development, non functional requirements are largely realised in a tacit form, and it forms the greater percentage of reusable software knowledge, while project-based knowledge in the form of functional requirements is project specific and is less reusable in subsequent projects. Thus LiSER does not put the same emphasis on both types of knowledge. LiSER only makes reference to explicit or documentary knowledge that supports any tacit knowledge captured as a wisdom or best practice or a lesson learned or any K-Asset type.

As the word 'living' in the name of the tool indicated, an Organisational Memory System (OMS) should not act as a passive repository of organisational historical knowledge. It has to be a 'living' technical organism, because captured knowledge losses its relevance as time passes. Therefore, unless organisational decisions are made based on continuously updated knowledge, organisations cannot escape repeating similar previous mistakes and/or caught in the act of "reinventing the wheel". This issue raises two questions: What is the level of "knowledge up-to-datedness" of a particular OMS? And what mechanisms it offers to maximise the group awareness in the host organisation? The integrity and consistency of LiSER's knowledge repository is maintained by series of learning cycles and feedback mechanisms defined by the framework of collaborative organisational learning presented in [10]. The issue of group awareness is tackled through the competence-based collaborative knowledge filtering groups set up in this framework. This paper is organised as follows. In section 3 we present a description of the software

knowledge management. In section 4 we describe our ontology-based knowledge model which defines the basic ontologies representing the characterised knowledge fragments. The underlying system architecture is described in section 5. Features of the prototype tool are presented in section 6. A short discussion of implementation issues is presented in section 7. Finally, in section 8 we give some concluding remarks in section 8 end up the paper.

2. Related Research

The work described in this paper has roots in a number of research topics, including Case-Based Reasoning (CBR), organisational learning and organisational memory, Software reuse, Computer Supported Collaborative Work (CSCW), Artificial Intelligence, decision theory and design rationale. There are many KM systems which are available either as commercial tools or as research prototypes. These approaches vary in the types of information, implementation technologies, and the application domains. From the viewpoint of information types, some systems only support textual data [3], while others are also capable of processing hypermedia data [1, 14]. From the perspective of the implementation technologies, several techniques were used either individually or as a combination of different technologies. Among these technologies are: Hypertext [3], Ontologies [9, 15], Email [3] and Case-Based Reasoning (CBR) [6, 7].

From the point of view of the targeting domain, some systems are generic while others targeted specific domains. Some of the systems reviewed were not exclusively developed to facilitate KM practices in software development domain, but they still can be used to play the same role. Systems like Answer Garden [1] and Know-Net [9] exemplify the generic OM systems. Approaches such as BORE [6, 7], Designer Assistant [14], TeamInfo [3] are primarily meant to support knowledge management in software development domain. However, due to the complexity of the software development process, some systems only target at certain phases of the software life cycle. These systems addressed either the software design phase [2, 8, 14, 16] or the requirements specification phase [11], or they just focus on capturing expertise and competencies available at an organisation [15].

3. Software Knowledge Management: A Review

Traditionally, knowledge creation and exchange in software organisations is communicated through natural language either verbally or vocally. Verbal knowledge is usually presented in plain text augmented with diagrams and software engineering notations. Some of this knowledge is stored electronically and others may be kept as hard-coded documents (for example, personal notes). The knowledge documented electronically is stored in various formats processed by different tools (i. e. word processors, drawing software, project management tools, and CASE tools).

It is believed that good software documentation would help software developers make good decisions in upcoming projects. However, in spite of strict documentation policies imposed by some software organisations, there is one type of knowledge that is hardly captured. For instance, a huge part of meeting details are unrecorded and only resided in the developers' minds. This limitation deprives the organisation of very important information. This includes assumptions, alternatives and views behind software decisions taken. The Rationale Management is introduced as one of the software engineering topics that tackle this issue. It aims to 'improve' the quality of decisions by making decision elements, such as criteria, priorities, and arguments explicit [4]. There are many models proposed to represent rationale knowledge, but since the rationale management systems was not meant to represent a corporate memory, the captured rationale is not structured in a way to cater for this capability.

4. LiSER's Knowledge Model

LiSER's knowledge model represents the different data structures and relationships that govern the generation and sharing of organisational knowledge. It is used to guide knowledge generation and sharing in software organisations Figure 1 represents the Meta model describing the knowledge skeleton of LiSER's repository. This model represents the domain ontology that describes different constituent ontologies used to symbolise basic ingredients of the experience drawn from the software production line.

4.1. Characterisation of LiSER's Knowledge Assets

Unlike information management systems where all aspects of organisational data are considered, in knowledge management systems, the focus should be on knowledge fragments rather than information fragments. Knowledge fragments can be defined as the knowledge pieces that were proved useful through experience. These fragments are created as a result of intensive and critical communications between respective knowledge workers. In other words, organisational knowledge is the organisational information enriched with different criteria and assumptions that represent context within which that knowledge was created. Much of the effort in the investigation and design of OM systems has been lacking such a comprehensive view, which we hope to offer in this paper.

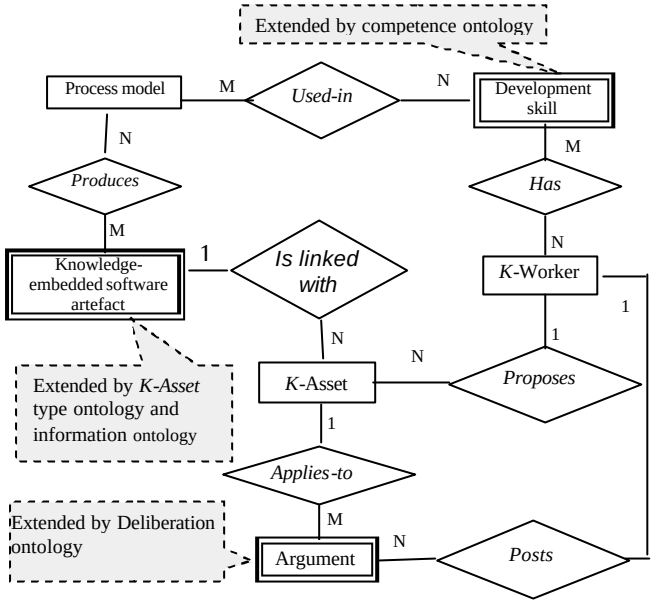

Figure 1. LiSER's knowledge model (higher level domain ontology).

In regard to explicit (documentary) knowledge, and as we stated earlier, LiSER is not meant to act as a DMS; it does only make links to documentary information as references to what is being seeded as a knowledge asset. LiSER users can make a hyperlink reference to the URL of any document, but it does not make any conversion or importation to such documents as the DMS does. LiSER can make reference to documents/ files of any format:- office documents, scanned images, technical drawings (i. e. UML diagrams), video clips, and sound files.

Basically, LiSER's repository is built on the notion of Knowledge Asset (K-Asset) as the basic building block. K-Assets represent the smallest level of granularity in LiSER's knowledge base. As such, the K-Asset can be any useful proven fragment of software development knowledge. Any lesson learned or knowledge-embedded software artefact can be considered as candidate K-Asset.

4.2. Realisation of LiSER's Knowledge Assets

As it is cited by Conklin, the biggest barrier to knowledge sharing is the "lack of shared understanding, especially about key concepts and terms" [5]. Research about ontologies aims to overcome this limitation. By definition, ontology is a "formal and explicit specification of a shared conceptualisation" [15]. It symbolises the entities and relationships that define any particular domain. In LiSER, all stored K-Assets are linked to the defined ontologies, which will be used later to search through the mass of K-Assets held in the resultant OMS.

Based on LiSER's knowledge model, an individual K-Asset is described by four types of ontologies:

1. Competence ontology.
2. Information ontology.
3. Type ontology.
4. History ontology.

Firstly, the information ontology illustrates the attributes used to describe any K-Asset contents. Different attributes are used to characterise different K-Assets based on the K-Asset types represented by the type ontology. Attributes are filled in by the author of any K-Asset before being submitted to the repository. Figure 2 represents the information ontology of a K-Asset characterised as a lesson learned. Secondly, the competence ontology (see Figure 3) is mainly used as an indexing schema for all K-Asset types. Instances of this ontology are arranged as taxonomy of software competences in the form of *is-a* and part-*of* hierarchy similar to the object-oriented structuring of elements.

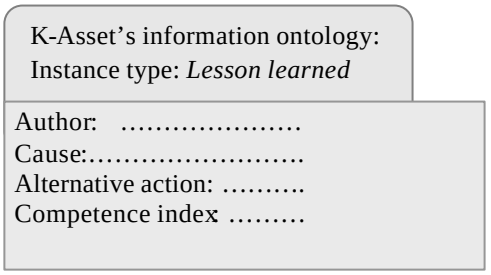

Figure 2. An information ontology for a lesson learned k-Asset type.

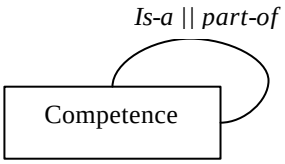

Figure 3. Competence ontology.

Thirdly, the type ontology represents the K-Asset types based on the defined types of the knowledge-embedded software artefacts. In this era of COT-based software development which is basically code-based reuse, reusing functional diagrams, data models and other know-how information has become a necessity. Figure 4 depict candidate K-Assets as represented by the K-Assets' type ontology. They include process models, software artefacts, and lessons learned from the software development process. Software artefacts represent any potential reusable software artefact. They include data models, test suites, screen shots, tables, tool recommendations, database, code and functional diagrams. Process models also represent a major source for learning the skills and know-how. They include any process description or installation procedures or bug workarounds. Lessons learned as K-Asset type represent descriptions of what could be considered by developers as lessons learned. Each lesson can be thought of as an avoidable negative practice. Each K-Asset characterised as a lesson learned includes descriptions like the causes of the

problem, its symptoms and alternative actions that could be taken to avoid the lesson reoccurrences.

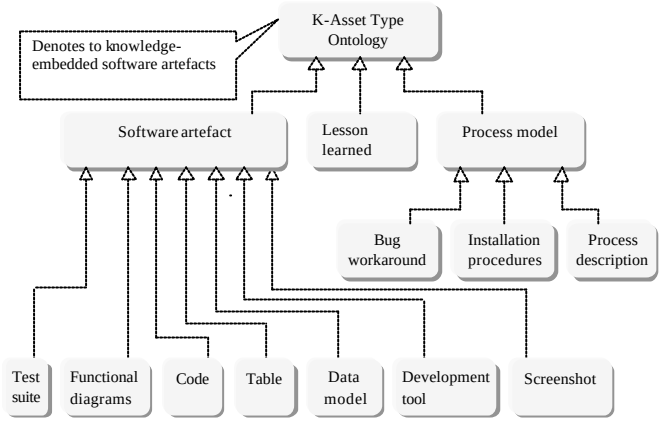

Figure 4. K-Asset's type ontology.

Fourthly, the history ontology is used to capture the history of argumentations about the validity of any particular K-Asset. Since K-Assets are usually created or modified in a collaborative manner, any knowledge generated as the outcome of such collaborative knowledge filtering has to be captured as well. Capturing this type of knowledge shall be the responsibility of the *history ontology*. This ontology includes information related to rationale behind individual K-Assets. This part is the most important part as it plays the main role of weighing the relevance of particular K-Assets. Details of this ontology are represented by an IBIS-based deliberation model that we proposed in [12]. Components of this model are shown in Figure 5.

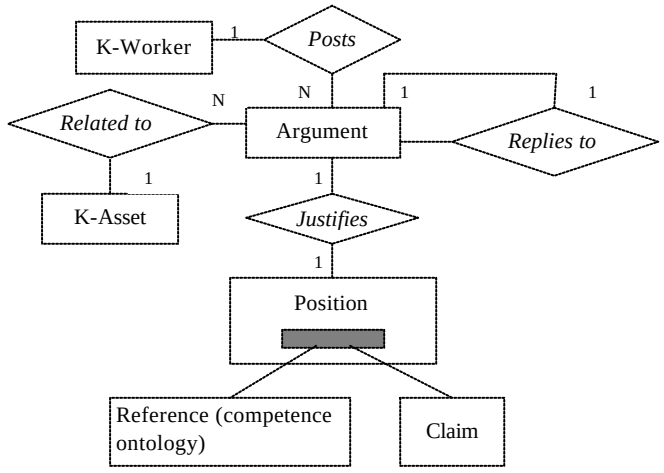

Figure 5. The proposed IBIS-based argumentation model (deliberation ontology).

5. LiSER's Architecture

A three-layered architecture is followed in the implementation of LiSER software. All these layers function in an integrated fashion to enable the learning cycles and feedback mechanisms set by the KM framework [10]. The layers are: the presentation layer, the logic layer and the information layer as shown in Figure 6.

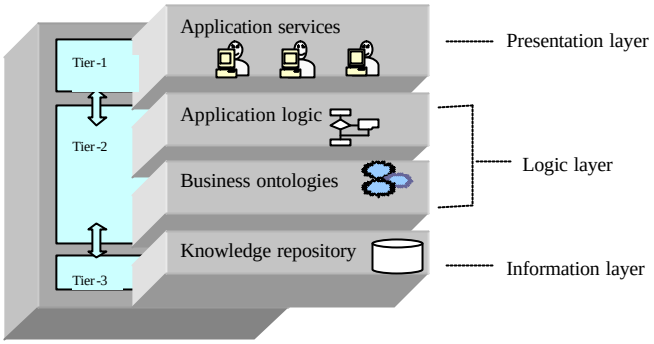

Figure 6. A layered architecture of LiSER.

The presentation layer comprises the user interface components for users to interact with LiSER. This layer provides knowledge workers and the knowledge manager with means of viewing, manipulating, and interacting with the information provided by the other two layers. The logic layer represents the business logic of the tool. It describes both the flow of the business logic (i. e. access rights, interaction rules, reasoning) and the conceptual knowledge taxonomy of business domain. The purpose of this layer is to provide a definition of the ontologies and the related semantics for the identification and classification of different types of K-Assets. These ontologies are integrated to form a semantic net through which conceptual search can be guided. KM dedicated agents are also defined at this layer; they are used to qualify candidate *K*-Assets based on qualitative assessment of the argumentation elements related to respective *K*-Assets. The qualitative assessment can be done based on many proof standards of decision theory. We adopted Scintilla of Evidence (SoE) as a standard to qualify active *K*-Assets.

Basically, LiSER regards K-Assets submitted by Knowledge Worker (K-Workers) as knowledge seeds as an analogy to growing plants. Seeds can only grow up and rise as trees based on the existence of certain success factors such as the climate, soil fertility, and of course, the smart farmer. We also consider newly added K-Assets as a K-Asset seed. It is initially considered inactive and it has no value until it is collaboratively scrutinised by members of respective community competence. Community members can either support or oppose or raise further issues before K-Asset candidates (seeds) are activated and grown up as knowledge pearls.

According to the SoE, any K-Asset^{ka_i} is active, if at least one position argues in favour of it. I. e. the activation or qualification of candidate K-Assets is formally represented as:

$$active(ka_i) \Leftrightarrow \exists(p_j \wedge inFavor(p_j, ka_i))$$

Where p = knowledge workers' positions

Ka = candidate Knowledge Asset
While inactive (i. e. dead) knowledge assets are reasoned about as follows:

$$In\ active(ka_i) \Leftrightarrow \neg\exists(p_j \wedge inFavor(p_j, ka_i))$$

All active knowledge assets considered fit yet the degree of fitness vary between different knowledge assets. The degree of fitness (i. e. relevance weight) is calculated based on the percentage of the believers of any particular K-Asset. Basically we regard the captured knowledge as a set of beliefs. In other words the captured knowledge represents what knowledge workers believe to be true or otherwise. The values of beliefs held in the OMS changes as a result of the continuous collaborative knowledge filtering. The fitness of the captured knowledge assets changes as well. Arguments in favour of a particular knowledge asset represent the believers, while objection arguments put more weight on the disbelievers' side. According to this standard, any knowledge asset is regarded as relevant when believers outweigh the disbelievers of that particular knowledge asset. However, the degree of relevance varies based on the overall percentage of the relevance score. The relevance score is calculated as follows:

$$relevance\_score(ka_i) = \frac{\Sigma(active(p_j) \wedge in\_favour(p_j, ka_i)) \times 100}{\Sigma(active(p_j) \wedge in\_favour(p_j, ka_i)) + \Sigma(active(ka_i) \wedge aganist(p_j))}$$

Based on the calculated value of relevance score of each knowledge asset, any particular knowledge asset is assigned one of the degrees of fitness shown in Table 1. Notice that we also used visual symbols to provide visual assessment of the degree of relevance of retrieved knowledge assets. Mathematical or textual descriptions alone sometimes hinder the users' ability to value the relevance of retrieved K-Assets. This knowledge filtering strategy shall help the knowledge manager or any competence community leader, to review and discard less qualified or inactive K-Assets (i. e. dead seeds).

Finally, the information layer represents the knowledge repository of LiSER. This layer collects different types of organisational K-Assets including the activation history of each particular K-Asset.

6. Tool Features

LiSER provides four customised knowledge navigators. Based on the types of stakeholders defined in our framework for collaborative organisational learning [10], users can interact with the tool through the following navigators:

- Knowledge manager's navigator.
- Navigator of competence group leader.
- Knowledge worker's navigator.
- Customer navigator.

The Knowledge Manager's view helps the knowledge manager to authorise the tool access and to establish competence communities. He/ She will be able to navigate brows and maintain the competence ontology. Based on the established competence groups, each Group leaders are responsible for authorising the access to knowledge related respective communities. Only members of respective competence communities can participate in the collaborative knowledge filtering of particular communities

The knowledge Worker's navigator helps K-workers to populate or retrieve previously captured K-Assets. This navigator also provides knowledge workers a discussion area for sharing or maintaining arguments related to candidate K-Assets. This view is specifically meant to support asynchronous collaborative argumentations among members of respective competence communities. Based on the competence ontology and the K-workers profiles, LiSER can be used as a competence management system. Users can easily figure out who knows what among the available software experts. Currently, the customer navigator only enables customers to brows the captured K-Assets. However they are not allowed to participate in knowledge filtering sessions. In fact this view is only developed to provide customer with insights that might influence the non functional requirements of upcoming software projects.

In regard to KM strategies, LiSER adopts a combination of 'push' and 'pull' strategies. K-Workers can pull the knowledge fragments based on any criteria chosen, while any knowledge fragment which is related to K-Assets seeded or arguments posted earlier, shall be pushed to them through the internal E-mail system. To provide for the optimum search results, LiSER employs two different search mechanisms namely keyword based and ontology based. The keyword-based search allows implementing CBR searching strategy. On the other hand, domain ontology is used for implementing the conceptual search. This searching strategy enables a more precise searching, because information can be induced through the mapping between different types of ontologies. As a result, the retrieved knowledge shall include knowledge fragments that it would be difficult to retrieve in keyword-based search, unless precise keywords are provided.

7. Tool Implementation

Several goals influenced our choice of the technical architecture to be used when implementing LiSER. First, LiSER must be a web based system to allow the tool access anytime and everywhere. This also contributes to fulfilling the portability feature, as the tool become platform independent. Secondly, the consistent growth of knowledge has to be ensured.

Thirdly, users must be able to seek and access K-Assets in an intuitive way. They also must be able to retrieve partial or similar information in addition to exact match search.

Table 1. Relevance degree of knowledge assets.

Relevance Score	Description	Visual Symbol
rs = 0	Inactive	
rs < 25	Sprouting	
rs = 25	Sprouting	
(rs > 25) AND (rs < 50)	Sprouting	
rs = 50	Borderline	
(rs > 50) AND (rs < 75)	Influential	
rs = 75	Influential	
(rs > 75) AND (rs < 100)	Influential	
rs = 100	Gem	

To accomplish the first goal, a combination of web programming tools was used in implementing LiSER. To accomplish the second goal, we restricted administration rights to the Knowledge manager only. To achieve the third goal, in addition to keyword-based search, an ontology-based search is provided to generate fuzzy and non-zero hit queries. Since the instances of the competence ontology is structured in OO-like hierarchy, inheritance rules can be used to include generic nodes to retrieve K-Assets similar to the target ones. For example, based on the instance of the competence ontology shown in Figure 7, instead of limiting the search through K-Assets annotated as *PHP scripts*, the node *web programming* is selected, and then all K-Assets annotated as *web programming* tools shall be considered among which is *PHP scripts*. Finally, the information layer is implemented through MySQL server which is an open source Relational Database Management System. HTML is used for Information rendering at the presentation layer, while we used PHP as a middleware server to connect the presentation layer with the information layer though the TCP/IP protocol.

8. Conclusion and Future Work

The prototype tool presented in this paper is an organisational memory subsystem aimed at facilitating knowledge management activities in software development organisations. The knowledge generation and sharing at LiSER are governed by a KM framework we proposed in a previous research. In regard to the structure of LiSER's knowledge repository, we proposed an ontology-based knowledge model. This model defines basic ontologies that represent salient software development knowledge.

As for further research, LiSER's functionalities can be augmented by synchronous argumentation through threaded discussions in the course of collaborative knowledge filtering. Secondly, our approach only relies on making references to explicit knowledge without any concern to the internal representation of these knowledge artefacts. The domain ontology can also be extended to include the modelling of explicit knowledge using XML annotations. Lastly, the integration between our approach and any DMS that has an Application Program Interface (API) shall extend the benefits further.

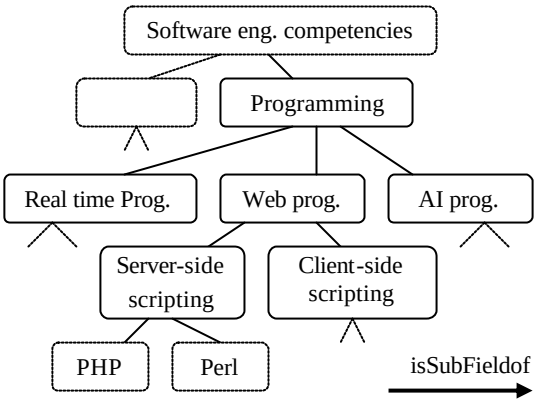

Figure 7. An instance of the competence ontology .

References

[1] Ackerman M. S. and Malone T. W., "Answer Garden: A Tool for Growing Organisational Memory," *in Proceedings of the Conference on Office Information Systems*, ACM, New York, pp. 31-39, 1990.

[2] Arango G., Shoen E., Pettengill R., and Hoskins J., "The Graft-Host Method for Design Evolution," *in Proceedings of the 15th International Conference on Software Engineering*, IEEE Computer Society Press, 1993.

[3] Berlin L. M., Jeffries R., O'Day V. L., Paepcke A., and Wharton C., "Where did you put it? Issues in the Design and Use of a Group Memory," *in Proceedings of the INTERCHI'93 Conference on Human Factors in Computer Systems*, ACM, New York, pp. 23-30, 1993.

[4] Brügge B. and Dutoit A. H., *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*, Prentice Hall, USA, 2000.

[5] Conklin E. J., Designing Organisational Memory: Preserving Intellectual Assets in a Knowledge Economy, *Group Decision Support Systems*, URL: http://cognexus.org/dom.pdf, 1996.

[6] Henninger S., "Accelerating the Successful reuse of Problem Solving Knowledge Through the Domain Lifecycle," *in Proceedings of the 4th International Conference on Software reuse*, IEEE Computer Society Press FL, Orlando, pp. 124-133, 1996.

[7] Henninger S., "An Environment for Reusing Software Process," *in Proceedings of the 5th*

IEEE International Conference on Software Reuse (ICSR'5), Victoria, BC, Canada, 1998.

[8] Henninger S., Haynes K., and Reith M. W., "A Framework for Developing Experience-Based Usability Guidelines," *Symposium on Designing Interactive Systems (DIS'95)*, ACM Press, pp. 43-53, 1995.

[9] Mentaz G., Apostolou D., Young R., and Abecker A., "Knowledge Networking: A Holistic Solution for Leveraging Corporate Knowledge," *Journal of Knowledge Management*, vol. 5, no. 1, 2001.

[10] Mohamed A. H., Peck L. S., and Salim S. S., "A Framework for Collaborative Organisational Learning: A Catalyst for Continuous Software Process Improvement," *in Proceedings of the 1st International Conference on Information and Management Sciences*, Xi'an, China, pp. 1-10, 2002.

[11] Ramesh B. and Dhar V., "Supporting Systems Development by Capturing Deliberations during Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 498-510, June 1992.

[12] Sai P. L., Mohamed A. H., and Salim S. S., "Towards an Intelligent Organisational Memory System," *in Proceedings of the Knowledge Management International Conference and Exhibition (KMICE'2001)*, Lankawi, Malaysia, 2001.

[13] Stenmark D., "Turning Tacit Knowledge Tangible," *in Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS'33)*, Maui, Hawaii, January 4-7, 2000.

[14] Terveen L. G., Selfridge P. G., and Long M. D., "Living Design Memory: Framework, System, Memory: Framework, System, and Lessons Learned," *Human-Computer Interaction*, vol. 10, no.1, pp. 1-37, 1999.

[15] Vasconcelos J., Kimble C., and Gouveia F. R., "A Design for a Group Memory System using Ontologies," *in Proceedings of the 5th UKAIS Conference*, University of Wales Institute, Cardiff, McGraw Hill, April 2000,

[16] Vescoukis V., "A Data Model for Software Design Decisions Representation and Management," *in proceedings of the 5th Hellenic Conference on Informatics*, Athens, 1995.

Abdulmajid Mohamed is currently a PhD student at the Faculty of Computer Science and Information Technology, University of Malaya. He obtained his MSc degree in office automation and information systems from Leeds University in 1993. He worked as a lecturer in the Department of Computer Science, Sebha University, Libya in the period from 1994 to 1999. His current research interests include knowledge management, organisational memory systems, and ontology-based modelling.

Sai Peck Lee is currently an associate professor at the Faculty of Computer Science and Information Technology, University of Malaya. She obtained her MSc of computer science from University of Malaya in August 1990, her Diplôme d'Études Approfondies (DEA) in computer science from Université Pierre et Marie Curie (Paris VI) in July 1991 and her PhD degree in computer science from Université Panthéon-Sorbonne (Paris I) in July 1994. Her current research interests include software engineering, object-oriented methodology, software reuse and application framework, knowledge management, information systems and database engineering, object-oriented analysis and design for e-commerce applications and auction protocols. She has published an academic book and more than 70 papers in various local and international conferences and journals. She is a member of IEEE Computer Society, a founding member of Informing Science Institute, a member of the editorial board of a research bulletin, and she had served as the executive editor of a journal for 2 years, as well as an active member in the review committees and programme committees of several local and international conferences.

Siti Salwa Salim is currently an associate professor at the Faculty of Computer Science and Information Technology, University of Malaya. She obtained her PhD in computer science from the University of Manchester in 1998. Her current research interests include computer supported collaborative work/ learning, human computer interaction, web-agents, software requirements engineering, and usability engineering.