Knowledge Transfer Model for the Development of Software Requirements Analysis CASE Tools to Be Used in Cross Time-Zone Projects

Zenon Chaczko[*1], Jenny Quang[1], Bruce Moulton[*2]
[*1] ***Corresponding author*** *School of Electrical, Mechanical and Mechatronic Systems,*
[2,] *School of Computing and Communication,*
Faculty of Engineering and IT, University of Technology, Sydney, Australia
brucem@eng.uts.edu.au
doi: 10.4156/jdcta.vol4.issue1.1

Abstract

This article describes work undertaken to evaluate an approach for developing collaborative requirements-analysis CASE tools that are specifically designed to address the needs of cross-time-zone development teams, that is, teams spread across different geographical locations around the world. Few of the software requirements analysis computer assisted software environment (CASE) tools readily available are designed specifically for cross-time-zone development activities. We propose a specifically tailored data and knowledge-transfer model, and investigate its suitability for the development of a cross-time-zone oriented CASE tool. The approach was used to develop a working prototype. The approach and prototype will be further evaluated in a collaborative undertaking involving the Wroclaw University of Technology, the University of Technology Sydney and the University of Arizona (UA).

Keywords

Knowledge transfer, requirements analysis, CASE, cross-time-zone

1. Introduction

It is well known that doubling the size of a team does not halve the development time. To reduce development time, organizations have increasingly been adopting a practice which makes use of additional teams located in various spots around the world. This "24 hour" mode of working is commonly found in open source development projects, and increasingly used by large companies. While one team sleeps, another can continue the development during its daylight hours. The Open Source community (For example the Linux kernel and Apache web server projects) has worked in this fashion for years. Many large organizations including IBM, Sun Microsystems, Cisco Systems, Nokia and Google also use geographically and temporally spaced development teams [1]. Twenty four hour continuous development is ideal for tasks that have hard-deadlines or require work completed as soon as possible. If a functional or security bug is discovered in a mission critical application, there is a need to find a solution within the shortest period of time. For example, the approach might enable a "three day solution" to be completed in a 24-hour period. The two day difference might be extremely valuable in terms of down-time costs.

However, the process whereby teams work in different locations has significant effects on the way that the work is structured and organized. For example, the project leader is not available to the "night" team. Each team must work independently of the other, and each must hand over the work to the other at the end of the shift.

The emerging trend in cross-time-zone development is sufficiently prevalent that several curriculum developers are exploring ways of enabling students to gain experience in this mode of working (e.g. [2][3][4]).

It is proposed that the development process might benefit from efforts to acknowledge and convey both explicit knowledge and tacit knowledge. Tacit knowledge can be understood as un-codified knowledge that leaves when employees leave the project). This problem is ordinarily handled by attempting to "convert" tacit knowledge into codified knowledge, by way of documentation. However prior research suggests certain aspects of tacit knowledge can only be transferred through face-to-face contact [5]. Nonaka and Nishiguchi suggest that most knowledge is created not by individuals, but by interaction and dialogue among several people [6]. Distributed teams have limited opportunities for face to face contact, so such knowledge transfer issues are

particularly relevant for cross-time-zone development projects.

Prior work suggests that certain environments have specific characteristics that render formal methods of knowledge transfer inadequate. For example, one study suggests that some engineering sites experience difficulties during attempts to codify/document certain aspects of their more experienced employee's knowledge for simulation or formal training purposes [7]. It has been proposed that inability to transfer knowledge can be a hazard where there are safety critical operations, and that this must be taken into account during the design of workflow processes [8][9]. Brown and Duguid suggest that knowledge transfer is less facilitated by converting tacit to declarative knowledge than by aligning the goals and practices of employers and employees [10].

A vast range of computer assisted software environment (CASE) tools can be used to assist knowledge management during software development. Many of the tools focus on providing support for a range of complex abstract concepts and representations, for example, UML, SDL, Z-spec etc. A characteristic ordinarily exhibited by such tools is that the complex levels of functionality can cause the tool to be a hindrance to split teams where each hands over work to the other at the end of each shift. At the time of writing, very few of these tools appear to be specifically designed to support cross-time-zone software development, and this was the motivation for evaluating a data/knowledge management approach for the development of a software requirements analysis CASE (SRAC) tool.

2. Method

A primary goal was to investigate the suitability of a data/knowledge-transfer model for the development of a SRAC tool. The work was to focus not on just the analytical parts of the documentation, but rather the entire shape of it. It was envisaged that the SRAC tool should be suitable for diverse skill sets, and suitable for the support of requirements analysis involving a minimum of overhead while facilitating effective document standardization and sharing of information (requirements artifacts). The process was to draw from IEEE Recommended Practice for Software Requirements Specifications (IEEE 830-1998) [11]. The tool was intended to provide a framework for standardization of system/software requirements documentation at both local and global levels, and at the same time remain a shared data repository that enables exchange of information across different locations, time zones, system development environments and documentation formats.

The approach included considerations relating to iterative and incremental development processes that would be suitable for a cross-time-zone collaborative development environment. It was proposed that such teams would benefit from a structure for documenting small iterations in development (8 hour shifts) and methods for allowing for periodic resynchronization. It was envisaged that it would be valuable to permit a way for inter site issues arising from shift to shift to be identified, documented and perhaps isolated and planned to be rectified by the same or successive shifts. An agile software development methodology known as Scrum was chosen, in part because it focuses on managing complex development processes iteratively. Issues relating to handover-synchronization can be handled using the Scrum process skeleton. From a project management perspective, it was proposed that the Scrum methodology may assist in synchronizing intensive development tasks.

It was proposed that it would be useful from an organizational knowledge point of view if data (for example files) that was produced during a shift, and if subsequent alterations to this data, could be captured and correlated against work done in previous shifts. The approach viewed the modifications to data as being similarly noteworthy as the data itself. This approach is different from existing tools, which over a long period of time record only the persistent data of the project.

It was noted that decision making in distributed teams can also be fragmented, and individual teams may make decisions that affect the entire project and must be recorded and distributed to all teams.

To handle the above considerations we introduced the concept of *Eventflows*. The Eventflows concept is an adaptation of the Lifestreams concept coined by Freeman and Gelernter [12]. Where Lifestreams record the digital events of a single person, Eventflows record the digital events of a project and the project artifacts. Eventflows capture events and periods within the project's global system, and capture and distribute project knowledge (Figure 1). Events were classified as any significant occurrence on the project that can be captured or recorded by a development environment, for example the login or logout of a system, the commit of changes to a version control system or the modification of a project artifact. A period is the linking of two key events where on their own has little or no value. Eventflows can be captured through automated systems or through manual creation from users. Eventflows can also be linked against individual or groups of tasks defined in project management tools such as Microsoft Project, thereby showing the actual work that was required and accomplished to complete the task.

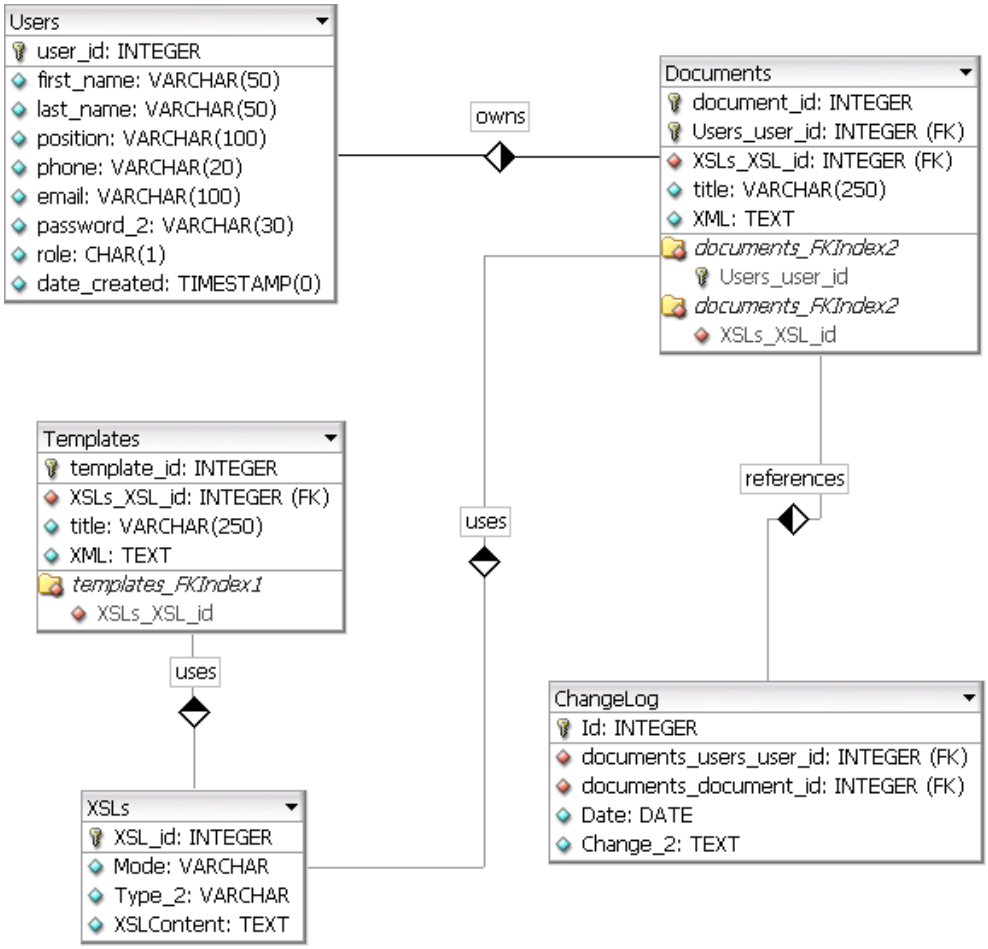

Figure 1. Entity Relationships arising considerations relating to the data/knowledge-transfer model

3. Evaluation of the approach

Considerations relating to the Eventflow methodology led to the following attributes/constraints: (1) an Eventflow must consist of both human readable and application readable data; (2) it must include date and time of creation; (3) it must include a human readable description of event; (4) it must include a machine readable description of event (implemented as serialized object); (5) it must include a project identifier; (6) it must include an artifact identifier.

The evaluation suggests that the database design plays a crucial role in the design of a SRAC, because it decides what data will be stored in the system, what type of user queries will be easily provided and how the rest of the system will interaction with the database. Key data persistent components identified during the evaluation of the approach include: (1) account information and details pertaining to a user are stored in a table, and each user is identified by a unique key; (2) templates are stored table, each has a unique id, entire templates are kept in text/XML format, each has a reference to the XML stylesheets in an 'XSLs' table; (3) documents created from templates are stored in a 'Documents' table, and documents are kept in text/XML format and updated each time the document is revised. Like the templates, each document has a reference to the XML style-sheets in the 'XSLs' table; (4) to capture the historical changes made to an existing document, when a change occurs, it is logged in a "Change Log" table.

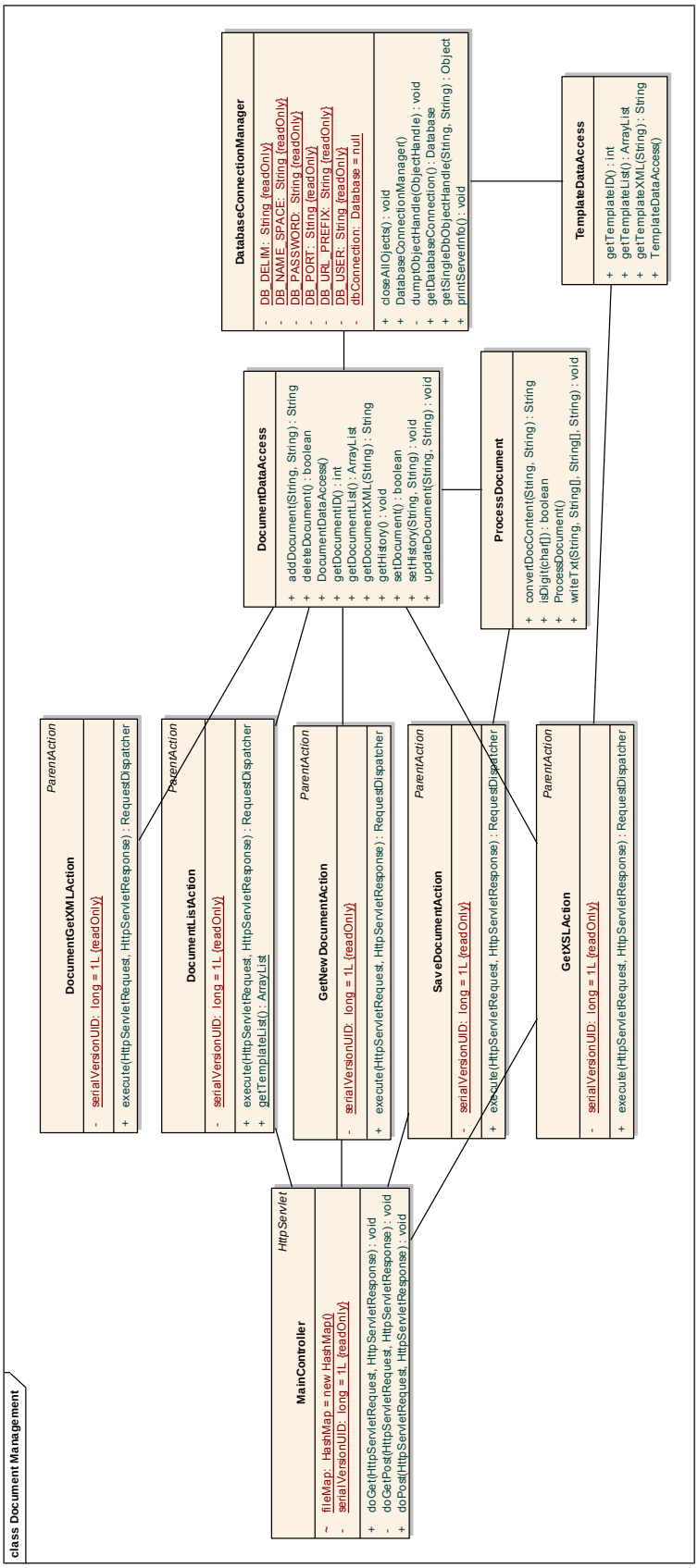

Figure 2. Diagram regarding document management, arising from considerations relating to the data/representation model

Each entry is identified by a unique identification number and also holds the identification number of the document it corresponds to; (5) templates and documents are kept as XML in the data tables. XML is human readable, however it is not aesthetically formatted for viewing. To provide styles and formatting for XMLs, the 'XSLs' table contains XSL style sheets.

The evaluation considered diagrams providing views of document management and associated entities, including those given in Figures 1 and 2. The relationships between significant components of the system architecture can be best described using collaboration diagrams, and the basic dynamics can be demonstrated using high level sequence diagrams. For example, the Participants component is shown using collaboration diagrams, and its dynamic is visualized using in high level sequence diagrams. The approach was evaluated for its suitability for implementing views of various layers. The View layer contains all the components associated with presenting the user interface that allows the end-user to view and interact with the system. Of the two distinct interfaces that make up the interaction screens to the tool, first enables the user to manage the application users, and the second is for the creation, and viewing of the templates and documents. The Controller layer brings the model and view together and integrates the application. The Model layer contains the business logic and components that access the data in the database and manipulates the data. The Data View depicts the key persistent elements of the system.

For the purposes of this project, we estimated that the average number of users supported by the tool on the project would be 100, while the rate of document creation is 3 per day. (The expected volumes of traffic would vary depending on the type and size of the development project.) Even though the functions available within the tool are not strictly time critical, the performance of the system is still expected to be responsive to a user's actions. It was estimated that the system would easily respond to a user's action in less than 0.5 seconds.

The evaluation also considered the extent to which the approach was suitable for the development of a system that is scalable, secure, reliable and portable. The prototype was a web based application, so security was imperative to ensure that any confidential information and intellectual property is more accessed by any unauthorized parties. The prototype was secured via authentication and authorization mechanisms. Following the MVC architecture will provide another level of security, where users cannot directly access specific sections of the CASE Tool. Reliability was also a key consideration because the prototype was intended to be available 24 hours a day. Portability was also a consideration so that developers in different locations could access the tool from different environments. The IEEE Recommended Practice for Software Design Descriptions (IEEE 1016-1998) [11] served as the basis for which the design description was written, and the design is customizable according to the particular attributes of the project. The design description describes the various classes to be built, how the database will be set up, what the system graphical user interfaces will look like, and what the interactions within the system are. Components relating to the View Layer of the system are made up customizable JSP/CSS files. The Template Management section of the system handles all the functionality surrounding the use of templates.

The approach was found to be suitable for producing documentation in accordance with IEEE standards for software design descriptions.

The resulting tools appear to be suitably scalable, and a range of features can be added including uploading of ready documents, incorporating an integrated help, notation toolkit, and interoperability with other products/systems.

Overall, the evaluation suggested that the data and knowledge-management approach had successfully led to the development of a simple and effective prototype.

4. Conclusion

This article considers a new approach for developing software requirements analysis CASE tools specifically intended to suit the particular needs of cross-time-zone development projects. The approach was evaluated in light of a data/knowledge-transfer model. The model is intended to enable developers to focus on issues relating to the codification and transfer of critical events and knowledge within and between development teams. The preliminary evaluation suggests that the model is suitable for informing relevant development-related considerations. This finding is consistent with the prior research. However further work is required to explore the limits of the model and determine the extent to which the model is appropriate for the development of tools that are scalable for large numbers of users.

5. References

[1] Gupta A. (2007) "Expanding the 24-Hour Workplace", The Wall Street Journal, September 15, 2007.

[2] Chaczko, Z., Davis J. D., and Scott C. (2004) New Perspectives on Teaching and Learning Software Systems Development in Large Groups -

Telecollaboration", IADIS International Conference WWW/Internet 2004 Madrid, Spain 6-9 October 2004.

[3] Chaczko, Z., Klempous, R., Nikodem, J. and Rozenblit, J. (2006) "24/7 Software Development in Virtual Student Exchange Groups: Redefining the Work and Study Week", ITHET 7th Annual Conference Proceedings, pp. 698-705.

[4] Gupta, A. and Seshasai, S. (2004) Toward the 24-Hour Knowledge Factory, MIT Sloan School of Management. USA. Available at http://papers.ssrn.com/sol3/papers.cfm?abstract_id=486127

[5] Roberts, J. (2000), 'From know-how to show-how? Questioning the role of information and communication technologies in knowledge transfer' Technology Analysis & Strategic Management, 12, 4, 429-444.

[6] Nonaka, I. and Nishiguchi, T. (2001), Knowledge Emergence: Social, Technical, and Evolutionary Dimensions of Knowledge Creation (New York: Oxford University Press).

[7] Moulton, B. and Y. Forrest (2005) Accidents will happen: safety-critical knowledge and automated control systems. New Technology, Work and Employment, Vol. 20, No. 2, 102-114.

[8] Moulton, B. (2009) Enabling safer design via an improved understanding of knowledge-related hazards; a role for cross disciplinarity. Australasian Journal of Engineering Education [in press, accepted 22.12.2008].

[9] Moulton, B. (2009) Conventions to achieve safer design and reduce catastrophic and routine harm to the environment 2009 International Conference on Environmental and Computer Science (ICECS 2009) Singapore 22-24 Jan 2009.

[10] Brown, J.S. and P. Duguid (2001) 'Structure and Spontaneity: Knowledge and Organization' in I. Nonaka and D.J. Teece (eds), Managing Industrial Knowledge: Creation, Transfer and Utilization (London: Sage), 44-67.

[11] Institute of Electrical and Electronics Engineers, (1998) IEEE Recommended Practice for Software Design Descriptions (IEEE 1016-1998), IEEE, New York.

[12] Freeman, E. and Gelernter, D. (1996) "Lifestreams: A Storage Model for Personal Data", SIGMOD Record, vol. 25, no. 1, pp. 80-86.