Re-engineering Legacy Systems for Modernization:The Role of Software Reuse

Meena Jha and Liam O'Brien

Abstract*—In this paper we outline our legacy modernization approach that incorporates our reuse process and repository which we have called the Knowledge Base Software Reuse (KBSR) Process and the KBSR Repository. The KBSR Process and Repository aim to give software engineers easy access to reusable software artefacts and reusable components within a defined process which we have incorporated into our modernization process. We outline how software re-engineering of legacy systems is used to populate the repository through the use of architecture reconstruction techniques to identify and categorize legacy components and other artefacts and save the components and associated information in the KBSR Repository to support modernization. The KBSR Repository can contain all categories of reconstructed software artefacts which have the potential to be reusable assets. In the context of modernization software re-engineering through architecture reconstruction has a major role to play in software reuse. We illustrate the use of the software re-engineering through software architecture reconstruction and the KBSR Process and Repository with a case study.

***Keywords*— Software Re-engineering, Software Reuse, Knowledge Based Software Reuse Repository, Knowledge Based Software Reuse Process.**

I. Introduction

The lifetime of a software system is very variable. Some organizations, banks and governments still rely on software systems that are more than 20 years old. Many of these legacy systems are still business critical [1, 2]. Yet, the value of the legacy system investment tends to decline over time. According to *Lehman's first law* [17] software must be continually adapted or it will become progressively less satisfactory in "real-world" environments.

The benefits of software reuse have been widely accepted. Re-engineering reclaims the software artefacts. O'Brien and Smith [30] have emphasized the need of software architecture reconstruction as a decision making tool to identify and document components dependencies to get better understanding of the potential for software reuse within a legacy system.

Meena Jha
UNSW and CQUniversity
Australia, m.jha@cqu.edu.au

Liam O'Brien
Geosciences
Australia,
William.OBrien@ga.gov.au

We have developed the Knowledge Based Software Reuse (KBSR) Process and KBSR Repository for legacy system modernization based on software re-engineering through software architecture reconstruction. We validated our KBSR Process on a case study using the Automatic Cane Railway Scheduling System (ACRSS).

The remainder of the paper is organized as follows. Section 2 outlines related work on modernization and software reuse and the motivation for developing our approach. Section 3 outlines an overview of our KBSR Process. Section 4 describes an overview of the KBSR Repository. Section 5 describes the case study the modernization of the Automatic Cane Railway Scheduling System (ACRSS) using the KBSR Process. Section 6 describes the phases to develop the KBSR Repository to be used in the KBSR Process for modernization of legacy system. Section 7 presents the KBSR Process for modernization of a legacy system using ACRSS as a case study. Section 8 summarizes and discusses our experience and concludes the paper.

II. Background and Motivation

Re-engineering legacy systems for modernization aims to retain and extend the value of the legacy systems investment as it reuses the software artefacts already developed [8]. Modernization of legacy systems by reusing existing software functionality and making it available as web services or using it in other web environments has been done. Sneed and Sneed [29] outline an approach for creating web services from reusing host programs. Kontogiannis and Zhou [3] outline an approach to migrate legacy applications through identification of the major legacy components and reusing these procedural components in an object-oriented design, specifying the interfaces, automatically generating the wrappers and seamlessly interoperating them via HTTP based on SOAP messaging. Litoiu [4] outlines issues such as performance and scalability related to reuse and migration of legacy applications to web services.

Given the attractive payoff of reusing software artefacts, there have been several efforts undertaken to discuss the topic of reusability [39, 40and 41], including software reusability in practice [42, 43]. Developers are adopting many of these reuse approaches, including reuse in product lines [44] and design patterns [45]. Software modernization uses all the phases of software development life cycle. software development has been strongly criticized [15, 46]. One of the reasons of criticism is the long development time. An obvious goal is to speed up development processes by reusing already developed software artefacts. Our survey has also reported many benefits

of software reuse such as quicker time to market, better use of resources, increased quality, reduced software risk, and reduced development cost [4, 5, and 16].

The risks of redevelopment suggest that there is a need to reuse existing software artefacts, components, software assets, application requirements, source code, etc for modernization of the legacy systems. To make software reuse an integral phase in software development or in legacy system modernization all reusable software artefacts, components, assets etc. should be made easily available to software engineers. This can be made possible only if a reuse repository is developed to store all of the knowledge base of reusable software artefacts, reusable components, previous software development experiences, etc. We propose such a repository and call it the Knowledge Base Software Reuse (KBSR) Repository. The KBSR Repository can contain all categories of reusable software artefacts.The development of a KBSR Process which uses software re-engineering along with an associated KBSR Repository will help to systematize the software reuse process and provide the repository to store the reusable components and other reusable software artefacts and capture current and past knowledge of software reuse.

There are several areas in which knowledge bases can be used to support the software development process. These areas include supporting the expert nature of software design and coding, facilitating the reuse of software components and artefacts, and providing domain knowledge to support software reuse and in the implementation effort [17]. Our literature survey [18, 19, and 20] shows that there are several software reuse processes. Bauhaus [18] is a knowledge-based software parts composition system shell. LaSSIE [19] is a knowledge based software information system. It has knowledge representation for software objects and its relations, and it provides functions to query and browse software objects. Software Components Catalogue [20] is another knowledge-based system for software reuse. It utilizes a conceptual dependency database describing software components and their relations, than matches users requests for software components with description of components which satisfy these requests.

Our KBSR Repository re-engineers and stores knowledge developed during software development for reuse and for modernization of legacy system KBSR Process is used where KBSR Repository is an integral part. None of the modernization approaches make the use of re-engineering and knowledge-based software reuse.

I. An Overview of the KBSR Process

During software development or modernization, software engineers and developers first select a software process e.g. agile vs waterfall that is tailored for the project goals and other Resource constraints and then enact the process as a guide for software engineers and developers. Software developers generally follow the process for the roles they play as to what development activities to perform and when to perform them [22]. When the Knowledge Based Software Reuse (KBSR) Process is utilized, management and reuse of software artefacts becomes a necessity.

A KBSR Process involves two necessary software reuse phases to help the software engineers and developers develop a software system with reuse. These phases are:

i. Re-engineer the legacy system components to develop the KBSR Repository (*for reuse using Software Architecture Reconstruction*), and
ii. Use the KBSR Repository in the modernization of the system (*with reuse*).

Figure 1 shows the overview of the KBSR Process. It includes the KBSR Repository and a complete framework where the KBSR Repository is being used for modernization of a legacy system or development of a new software system based on the reusable artefacts found in the KBSR Repository. Once the KBSR Repository is populated it is going to store the reusable software artefacts and components. This KBSR Repository can then be used in the KBSR Process *for reuse/with reuse* development or modernization of the software.

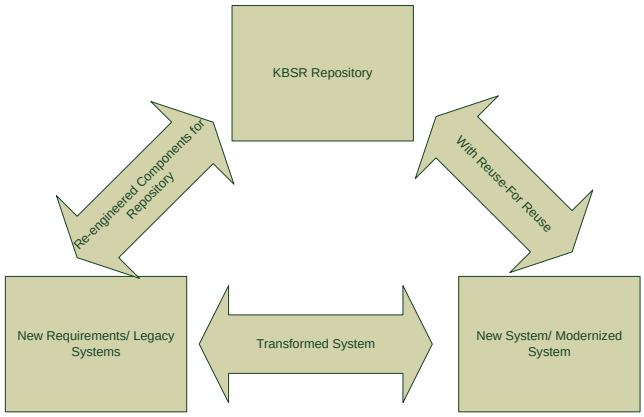

Figure 1: Overview of the KBSR Process

Our studies have [4, 5] shown the problems related to vertical and horizontal reuse. Our KBSR Process can be used for both types of reuse. It allows the reusable components from a system/domain and from outside the domain to be placed in the repository. It supports the reuse of software components and the development of software components for reuse. So software with reuse and for reuse are also categorized and incorporated in the development of a software system

Many legacy systems can be modernized using our KBSR Process and with time our KBSR Repository is going to grow large to store most of the required reusable software artefacts. It can serve as one single point to look for reusable software resources for all software engineers and developers within an organisation. Our approach of using a KBSR Repository may significantly improve the software reuse in the software industry. It is well understood that present software development or modernization approaches are not adequate for meeting the software reuse demand [17].

II. An Overview of the KBSR Repository

The KBSR repository consists of following reusable software artefacts:

- Components from the market
- Components from other sources
- Software patterns from other sources
- Software patterns internal to organization
- Internal Component libraries
- Other reusable artefacts from legacy systems

All the artefacts and components from any software organization can contribute to our KBSR Repository. Once the artefacts and components are re-engineered they should be made available for/with reuse so that other organizations involved in the development of a new system or modernization of legacy system should able to reap the benefits of already developed components.

III. A Case Study-ACRSS System

The case study used is the Automatic Cane Railway Scheduling System (ACRSS) [21]. ACRSS is a computer-based system developed in 1987 to solve the cane railway scheduling problem. ACRSS consists of 194 subroutines and the 50,000 lines of code. All the reusable artefacts extracted from ACRSS legacy systems are stored in the KBSR Repository *for/with reuse*. Below we discuss each activity in detail and how we applied the KBSR Process to our case study of ACRSS in the following sections. We have worked at the subroutine level to re-structure the code for the ACRSS system [23].

IV. Re-engineer the Legacy System to develop the KBSR Repository

Re-engineer the legacy system to develop the KBSR Repository using SAR requires three activities. These activities are as follows:

- Activity 1: Identify Reusable Artefacts,
- Activity 2: Classify Reusable Artefacts,
- Activity 3: Store Reusable Artefacts in the KBSR Repository.

The growing concern in finding reusable software artefacts and the complexity of managing these software artefacts for reusability [4, 5] has led us to devise the KBSR Repository which reduces the complexity of identifying and managing the software reusable artefacts. The products of each phase of developing the KBSR Repository serve as an input to next phase. These phases are developed to address the issues identified by our survey respondents [4, 5] such as: software engineers cannot find what software artefact to reuse and where to find reusable software artefacts.

V. Use the KBSR Repository in the Modernization of Legacy System

The activities involved in using the KBSR Repository in the modernization are:

- Activity 1: Analyse Problem
- Activity 2: Retrieve Reusable Artefacts
- Activity 3: Understand Reusable Artefacts
- Activity 4: Select Best Reusable Artefacts
- Activity 5: Adapt and Reuse Reusable Artefacts

Below we illustrate the use of of the KBSR Repository in the KBSR Process to modernize the ACRSS legacy system.

A. *Activity 1: Analyse Problem*

In this activity we analyse the problems associated with the legacy system. The first question is why modernization is required for the existing system. What problems are being faced while keeping the existing system running? There could be number of quality issues such as reliability, maintainability, security, dependability, etc. There could also be a requirement to modernize the legacy system to be compatible with the new technology. Whatever may be the reason for modernization of legacy systems the problem needs to be analysed as the first phase to use the KBSR Repository for the modernization of legacy systems.

Software architectures can be developed for enhanced understanding and looking in depth analysis of the problem. In order to modernize the legacy system we need to identify the dependencies of legacy components. Dependencies include:

- Data dependencies where global data is shared between a component and other parts of the system.
- Functional dependencies where a component uses other parts of the system in order to carry out its functionality or other parts of the system use the component.

The visualization of architectural views can help in finding out the answers to the questions, which can form the basis of the restructuring. Some of the questions include:

- What are the subsystems or components of the software system?
- How should the interfaces between components be structured?
- What are the characteristics of the component communication?

Pinkney [21] worked on the modifications of the source code using FORTRAN 77. Values from previous versions of the ACRSS system were analysed to assess maintainability of the system.

We analysed the ACRSS system for modernization for better maintainability. The existing problem descriptors were old monolithic legacy source code which stopped evolving and were difficult to maintain. We chose a few subroutines as described in Section 5 to modernize using the KBSR Process.

B. Activity 2: Retrieve Reusable Artefacts

The finding process involves more than just locating an exact match. It includes locating highly similar components because even if a target component must be partially redeveloped—rather than be reused in total—an example similar to the ideal component can reduce the effort and eliminate many defects [29]. Due to the difficulty of finding an exact match, artefacts become less and less reusable the more specific they become. Atkinson [35] has suggested a unifying model for retrieval from reusable software libraries. His retrieval method is based on identifying the nature of the indices used as representation of components such as external indices, internal static indices, and internal dynamic indices. External indices are keywords faceted [30] and feature based [31] classification techniques all seek to find relevant components based upon controlled vocabularies, properties and ontologies external to the class. Internal Static Indices is based on structural matching techniques, most probably signature [32] and specification matching [33, 34], techniques, seek to find relevant components based upon elements of the structure of the software components. Internal dynamic Indices are based on behavioural retrieval technique to take advantage of distinguishing property of software-executability. Behaviour based technique [35, 36, 37] seek to find relevant components by comparing input and output spaces of components.

C. Activity 3: Understand Reusable Artefacts

We generated the complete description of the reusable artefacts as stated in Table 1 so that the software engineers can understand what objects have been retrieved. We used the ARMin reconstruction tool [38] to re-construct the architecture of the ACRSS system. The architectural analysis of the ACRSS system made it more understandable. We documented our findings and worked on what subroutines are loosely cohesive, and highly coupled. We also documented the dependency graph as the type of software artefact which suggested the characteristic of each subroutine under examination.

D. Activity 4: Select Best Reusable Artefacts

All system development is driven by requirements, whether they are documented or not. Even verbal directions count as system requirements, albeit ones having a high tendency to be misunderstood. Architectural analysis uses these requirements to derive a system structure that provides an understanding of the system as the software engineers created code. This system form may be a layered architecture, where all of the system components reside in well-defined system layers, or other structures (such filter/pipe architectures for flow-through signal processing). The key to this analysis is to identify the requirements that have the highest impact on overall system structure. These "architectural drivers" are typically the ones that use the highest amount of system functionality, and are the more complex areas of the application.

At the end of this phase we could select the best reusable artefacts. We had very old documentation of the ACRSS system which was out of sync with the running system. We updated it and made it suitable for reuse.

E. Activity 5: Adapt and Reuse Reusable Artefacts

Adaptation, the lifeblood of software reusability, is the customization of the reusable artefacts to fit the new problem. This changes perception of a reusability system from a static library of rock-like building blocks to a living system of components that spawn, change, and evolve new components with the changing requirements of their environments [42]. The major steps involved in adaptation are figuring out what to adapt and adapting it.

Logical design of the legacy system was very complex to understand. There were lot of dependency between functions/methods. We created clusters of re-structured methods to combine them in a class and worked on generalization and specialization rules so that inheritance, encapsulation and polymorphism of OO design can be applied.

VI. Discussion of the Result and Conclusion

The development of a Knowledge Base Software Reuse Process with an associated KBSR Repository systematizes the software reuse process and provides the repository to store the reusable components, reusable software artefacts and capture current and past knowledge of software reuse with the help of software re-engineering and architecture reconstruction.
This also provides a mechanisms to locate reusable software artefacts and components from reuse repository, adapt them (if necessary) and even create new ones making use of the information provided by other similar software reusable components and software reusable artefacts. Software engineers now know exactly where to look for reusable software artefacts. This addresses the major issues and concerns of software reuse which was hindering software reuse from being a systematic process.

References

[1] R.W. Selby, "Enabling Reuse-Based Software Development of Large Scale System", *IEEE Transaction on Software Engineering*, Vol 31, No 6, June 2005.

[2] W.B. Frakes, and K. Kang, "Software Reuse Research: Status and Future", IEEE *Transaction on Software Engineering*, Vol 31, No 7, July 2005.

[3] N.Y. Lee, and C.R. Litecky, "An Empirical Study of Software Reuse with Special Attention to Ada", *IEEE Transactions on Software Engineering*, Vol 23, No 9, Page(s):537-549, September 1997.

[4] M. Jha, L. O'Brien, and P. Maheshwari, "Identify Issues and Concerns in Software Reuse", *Proceedings of the Second International Conference on Information Processing (ICIP'08)*, Bangalore, India, 2008.

[5] M. Jha, and L. O'Brien, "Identifying Issues and Concerns in Software Reuse in Software Product Lines", *11th International Conference on Software Reuse (ICSR)* , Virginia, USA 26-30 September 2009.

[6] M. Griss, and M. Wosser, "Making Reuse Work at Hewlett- Packard", *IEEE Software*, Vol 12, No 1, Page(s):105-107, 1995.

[7] M. Griss, "Reuse Comes in Several Flavours," *presented at the Flashline white paper*, 2003.

[8] C. McClure, "*Software Reuse*", Wiley-IEEE Computer Society Press, New York, 2001.

[9] Y. Kim and E. A. Stohr, "Software Reuse: Survey and Research Directions " *Journal of Management Information Systems*, Volume 14, Page(s): 113-145, Spring, 1998.

[10] W. Frakes and K. Kang, "Software Reuse Research: Status and Future," *IEEE Transactions on Software Engineering*, Volume 31, Number 7, Page(s): 529-536, 2005.

[11] M. Morisio, M. Ezran, and C. Tully, "Success and Failures in Software Reuse," *IEEE Transaction on Software Engineering*,Volume 28, Number 4, Page(s): 340-357, April 2002.

[12] R. van Ommering, "Software Reuse in Product Populations," *IEEE Transactions on Software Engineering*, Volume 31, Number 7, Page(s): 537-550, 2005.

[13] M.F. Dunn, and J.C. Knight, "Software Reuse in an Industrial Setting: A Case Study," *Proceedings of 13th International Conference on Software Engineering*, IEEE CS Press, Page(s): 329-338, 1991.

[14] E. Henry, and B. Faller, "Large-Scale Industrial Reuse to Reduce Cost and Cycle Time," *IEEE Software*, Volume 12, Number 5, Page(s): 47-53, 1995.

[15] D.D. McCracken, M.A. Jackson, " Life Cycle Concept Considered Harmful", *ACM SIGSOFT*, Volume 7, Number 3 Page(s):29-32, April 1982.

[16] I. Sommerville, *Software Engineering*, 7th edition, Addison-Wesley, 2009.

[17] M.T. Harandi, "Building a Knowledge Based Software Development Environment," *IEEE Journal on Selected Areas of Communications*, Volume 6, Number 5, Page(s): 862-868, 1988.

[18] B.P. Allen and S.D. Lee, A Knowledge-based Environment for the Development of Software Parts Composition Systems. *In Proceedings of the 11th International Conference on Software Engineering*, Page(s): 104-112, Pittsburgh, PA, May 1989.

[19] P. Devanbu, R.J.Brachman, and etc. LaSSIE: A Knowledge-based Software Information System. *In Proceedings of the 12th International Conference on Software Engingeering*, pages 249-261, Nice, France, March 1990.

[20] M.Wood and I. Sommerville, A Knowledge-based Software Components Catalogue. In P. Brereton, editor, *Software Engineering Environments*, Page(s): 116-133. ELLIS Horwood Limited, 1988.

[21] A. J. Pinkney, An Automatic Cane Railway Scheduling System, *MSc Thesis*, Department of Mathematics, James Cook University of North Queensland, Australia. December 1987.

[22] P. Mi and W. Scacchi, Modeling Articulation Work in Software Engineering Processes. *Proceedings of the 1st International Conference on the Software Process*, Page(s): 188-201, October 1991.

[23] M. Jha, and P. Maheshwari, "Reusing Code for Modernization of Legacy Systems", Proceedings *of IEEE Conference on Software Technology and Practice 2005* Budapest, Hungary, 2005.

[24] G. Booch, "Architectural patterns" http://www.rational.com/products/whitepapers/390.jsp 2001-0730, 1998.

[25] J.M. Moore and S. C. Bailin, "Domain Analysis Framework for Reuse*", Domain Analysis and Software Systems Modeling, IEEE CS Press*,1993.

[26] H. Bruyninckx. Software patternshttp://www.orocos.org/patterns.html, 2002.

[27] W.W. Agresti and F.E. McGarry, "Minnowbrook Workshop on Software Reuse: A Summary Report," *Software Reuse: Emerging Technology*, Will Tracz, ed., Page(s): 33-40, 1988.

[28] Arun Sen, "The Role of Opportunism in the Software Design Reuse Process*", IEEE Transaction on Software Engineering*, Volume 23, Number 7, July 1997.

[29] T. Biggerstaff and C. Richter, "Reusability Framework, Assessment, and Directions," *IEEE Software*, Volume 4, Number 1, Page(s): 41–49, March 1987.

[30] R. Prieto-Diaz and P Freeman, "Classifying Software for Reusability" *IEEE Software*, Volume 4, Number 1, Page(s): 6-16, March 1987.

[31] J. Borstler, "Feature Oriented Classification for Software Reuse", Proceedings of Seventh International Conference of Software Engineering and Knowledge Engineering, 1995.

[32] A. M. Zaremski, and J. M. Wing, " Signature Matching: A Key of Reuse" *Technical Report CMU-CS 93-151*, Carnegie Mellon University, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, May 1993.

[33] J. Jeng, and B. H. C. Cheng, "Using Analogy to Determine Program Modifications Based on Specification Changes", *Proceedings of IEEE 5th International Conference on Tools with Artificial Intelligence*, Page(s): 113-116, Boston, MA, November 1993.

[34] E.J. Rollins, and J. M. Wing, "Specifications as Search Keys for Software Libraries: A Case Study Using Lambda Prolog", *Technical Report CMU-CS 90-159*, Carnegie Mellon University, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, September 1990.

[35] S. Atkinson, and R. Duke, "Behavioural Retrieval from Class Libraries", *Australian Computer Science Communications*, Volume 17, Number 1, Page(s):13-20, January 1995.

[36] A. Podgurski, and L. Pierce, "Behavioural Sampling: A Technique for Automated Retrieval of Reusable Components", *Proceedings of the 14th International Conference on Software Engineering*, Page(s): 349-360, 1992.

[37] R. J. Hall, "Generalized Behaviour Based Retrieval", *Proceedings of the 15th International Conference on Software Engineering*, Page(s):371-380, May 1993.

[38] L. O'Brien, C. Stoermer, "Architecture Reconstruction Case Study", CMU/SEI-2003-TN-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, April 2003.

[39] T. Biggerstaff and A. Perlis, "Special Issue on Software Reusability", *IEEE Transaction on Software Engineering*, Volume 10, Number 5, September, 1984.

[40] "Special Issue on Software Reusability," *IEEE Software*, W. Tracz, ed., vol. 4, no. 4, July 1987.

[41] J. Bosch and N. Juristo, "Designing Software Architectures for Usability," *Proceedings of 25th International Conference on Software Engineering*, Page(s): 757-758,May 2003.

[42] N.Y. Lee and C.R. Litecky, "An Empirical Study of Software Reuse with Special Attention to Ada," *IEEE Transaction on Software Engineering*, Volume 23, Number 9, Page(s): 537-549, September 1997.

[43] J. Bosch, "Design and Use of Industrial Software Architectures," *Proceedings of Conference on. Technology of Object-Oriented Languages and Systems*, Page(s): 404-404, June 1999.

[44] J. Klein, B. Price, and D. Weiss, "Industrial-Strength Software Product-Line Engineering," *Proceedings of 25th International Conference on Software Engineering*, Page(s): 751-752, May 2003.

[45] J. Gustafsson, J. Paakki, L. Nenonen, and A.I. Verkamo, "Architecture-Centric Software Evolution by Software metrics and Design Patterns," *Proceedings of Sixth European Conference on Software Maintenance and Re-Engineering*, Page(s): 108-115, Mar. 2002.

[46] G.R. Gladden, "Stop the life-cycle, I want to get off", *ACM SIGSOFT*, VOL. 7, NO. 2, PP.35-39, April 1982.

About Author(s):

Meena Jha holds a Bach of Engineering (Electronics and Communication), Masters of Engineering (Instrumentation & Control), and on completion of her PhD in Modernization of Legacy Systems from UNSW Sydney. She has also been associated with industry on Legacy modernization process. She has 20 years experience in teaching and research. She is a member of IEEE and IEEE Computer Society.

Dr Liam O'Brien has over 23 years experience in research and development in software engineering. He is a Software and Applications Architect with Geoscience Australia and was previously Chief Software Architect with CSIRO and a Principal Researcher at NICTA's e-Government Initiative. He is also a Member-at-Large of the Service Science Society Australia which he co-founded in 2010. He has previously worked as s researcher with Lero (Ireland), Carnegie Mellon University's Software Engineering Institute (USA), CSIRO (Australia) and the University of Limerick (Ireland). His main areas of research include enterprise architecture, software architecture, SOA, service science, software reuse, software modernisation, and cloud computing. He holds a BSc and PhD from the University of Limerick, Ireland. He is a member of the IEEE and IEEE Computer Society.