Comparison of Modernization Approaches: With and Without the Knowledge Based Software Reuse Process

Meena Jha  
CQU University and The University of New South Wales, Sydney NSW 2052, Australia  
m.jha@syd.cqu.edu.au

Liam O’Brien  
Geoscience Australia, Canberra, ACT 2609, Australia  
liamob99@hotmail.com

Abstract—The ever increasing demand for improvements in software maintainability and modernization cannot be met through traditional techniques of software development and modernization. Most approaches to software development and modernization do not explicitly address software reuse however new approaches that address issues and concerns of software reuse must be developed. The benefits of software reuse are widely accepted by software engineers and developers. However based on our previous work in software reuse for the modernization of legacy systems, we have identified the need to build a knowledge based software reuse process and a reuse repository that manages reusable artefacts and components. Software engineers find difficulty in locating reusable software components (code artefacts and components. Software engineers find understanding and adapting the required reusable software artefacts to the difficulty of retrieving, software artefacts to the difficulty of retrieving, the scarce availability of reusable components and other problems that still limit software reuse. These range from the scarce availability of reusable components and other software artefacts to the difficulty of retrieving, understanding and adapting the required reusable software artefacts and components. Software engineers find difficulty in locating reusable software components (code related) and reusable software artefacts (non-code related). The results from our surveys support this finding.

It may not be possible to redevelop business critical legacy systems, rather than modernize them, due to the risks involved in doing so. Some of the major risks are:

- Current system may not be well documented and specifications may need to be redeveloped and this may introduce errors in the system;
- Current system’s documentation, design etc. may not conform to the running system and any redevelopment may create problems;
- Critical data and business logic may not be replicated;
- The size and complexity of the legacy system may have grown beyond a comprehensible level to understand and analyse.

With the changing paradigm of software development software reuse is required for software development and for modernization of legacy systems. To make software reuse a complete 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 paper compares our software modernization approaches based on software reuse. We have developed two modernization approaches. The first “Reusing code for modernization of legacy systems” which we term Modernization Approach I (Modernizing for reuse) in the
remainder of the paper [11]. The second is “Modernization with software reuse (Modernizing with reuse)” which we term Modernization Approach 2 in the remainder of the paper [1]. In the Modernization Approach 1 we had no software artefacts ready for reuse. In the process of modernizing the legacy system using the Modernization Approach 1 we identified/ restructured software artefacts for reuse to modernize the legacy system. The Modernization Approach 2 was built with software reuse as an integral part. In the process of modernizing the legacy system using Modernization Approach 2 we incorporated our reuse process and repository, which we have called the Knowledge Based Software Reuse (KBSR) Process and KBSR Repository.

The remainder of the paper is structured as follows. Section 2 describes the case study on which we applied our two modernization approaches. Section 3 describes our modernization approach “Reusing code for modernization of legacy system” (Modernizing for reuse). Section 4 describes our modernization approach with KBSR Process” (Modernizing with reuse). Section 5 compares the two modernization approaches on different attributes which have been identified from our previous work as major issues in software reuse from different development communities. Finally Section 6 concludes the paper.

II. THE CASE STUDY SYSTEM - ACRSS

The system used in the case studies is the Automatic Cane Railway Scheduling System (ACRSS) [12]. ACRSS is a computer-based system developed in 1987 to solve the cane railway scheduling problem. ACRSS was developed to schedule operations involved in the transport of cane from field to factory. ACRSS uses data describing the cane railway layout, harvesting patterns of the relevant growers and some operational parameters to produce a schedule. We could get some details of ACRSS from its documentation and user’s guide. We also had an access to the running program and its source code subroutines written in FORTRAN 77. ACRSS consists of 194 subroutines and about 50,000 lines of code.

We applied our modernization approaches on the ACRSS system in two separate case studies: One Reusing code for modernization of legacy systems and another on Modernization with KBSR Process. Below we discuss our approaches to legacy system modernization. Each modernization phases and activities are described.

III. Modernization Approach 1: Reusing code for modernization of legacy systems

Our first modernization approach consists of 4 Phases that were applied sequentially. These phases are:

- Phase 1: Analyse the legacy system
- Phase 2: Reconstruction of the legacy system
- Phase 3: Design structure : Restructuring
- Phase 4: Transformation (Procedural ->OOP)

Phase 1: This phase analyses legacy systems to capture their structure and to identify problems caused by the past development and evolution. This task includes gathering all application artefacts such as source code, test cases, design documents, DFD’s ERD’s, statistics about the size, complexity, amount of dead code or unused code [1], and amount of bad programming for each program such as dead code, messy chaotic code, bad variable names, poor documentation etc.. In the analysis the important part is to create a description of each module and each data item.

- Phase 2: This phase of the modernization discovers the design of the legacy system. The Architecture Reconstruction Mining (ARMIn) tool [13] is used to reconstruct the legacy system. Identifying all external dependencies that a module has is important when considering modernization. Of particular importance are the dependencies between subroutines that are candidates for restructuring.

- Phase 3: This phase of modernization involves a restructuring process which consists of a series of semantic preserving decompositions and compositions of ‘processing elements’. If functions are in the same logical unit then through abstraction and grouping of the functions within the unit then ARMin can be used to generate a view that shows the logical connection. Four types of relationships are extracted using ARMin. They are:
  - Common relation: a subroutine sends information to another through a global component.
  - Call relation: a procedure imports another subroutine’s computation to execute its functions; a subroutine calls another subroutine.
  - Sequential relation: an output of a subroutine is passed to another subroutine as an input; an output of a subroutine is used as an input of another subroutine, and
  - No relation: two subroutines do not have any of above relations

- Phase 4: Once all the above phases are completed we get Structured Object Model. Not all code can be turned into OO because of some internal dependencies. Phase 2 has identified which modules are the suitable candidates for restructuring. Phase 3 has restructured the selected modules into Structured Object Model. The object can be viewed as an abstract data type, encapsulating a set of data (i.e. attributes) and a corresponding set of permissible actions on the data (i.e. methods). After data item is defined for each object, the next step is to define the methods for each object. The methods are determined using both the invocation statements and the bodies of the subroutines. The invocation statements are used to provide the proper mapping of formal parameters to actual parameters while the bodies of the subroutine are considered line-by-line to define the actual methods. The objects generated are reused in the modules to see the working/ running of the system. The three independent subroutines generated as objects from ACRSS system are SALE, PAY and PROFIT. Objects in Object-Oriented programming (OOP) are essentially data structures together with their associated processing routines. For instance in our case subroutines are the objects – a collection of data and the associated statements.
IV. Modernization Approach 2: Modernization with the KBSR Process

The growing concern in finding reusable software artefacts and the complexity of managing these software artefacts for reusability [2] and [3] has led us to devise the KBSR Repository which reduces the complexity of identifying and managing the software reusable artefacts. In this modernization approach we have specifically included a software reuse process, the KBSR Process with an associated KBSR Repository for storing and managing the reusable components and artefacts. The KBSR Process involves two necessary software reuse phases to help software engineers develop or modernize a software system with reuse. These phases are:

- **Phase 1:** Develop the KBSR Repository *(for reuse)*, and
- **Phase 2:** Use the KBSR Repository in the modernization of a system *(with reuse)*.

Our modernization approach 2 incorporates the use of the KBSR Repository in the KBSR Process. To develop the KBSR Repository which is Phase 1 of our modernization with KBSR Process, there are three activities involved. These activities are:

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

The products of each activity of developing the KBSR Repository serve as an input to next activity. These activities are developed to address the issues identified by our survey respondents [2] and [3] such as: software engineers cannot find what software artefact to reuse and providing a repository in which to describe and find reusable software artefacts.

V. Comparing Software Modernization Approaches based on Software Reuse

In this section we compare the modernization approaches we applied to our case study the ACRSS legacy system on different attributes. The first time we modernized the legacy system was without having any KBSR Process or KBSR Repository. And then we again took the same case study using the KBSR Process and KBSR Repository once we had developed them and incorporated them into our modernization approach.

We collected the set of attributes used for comparison purposes from the outcome of our surveys so that the issues and problems associated with reuse could be addressed. The software reuse surveys were carried out within two software development communities (Conventional Software Engineering community and Software Product Line community). Some of the major concerns shown are lack of tool support, the Not-Invented-Here (NIH) syndrome, case tools are not promoting reuse, no reuse education, and no reuse repository and no systematic reuse process [2] [3]. With the development of KBSR process we have addressed the no reuse repository and no systematic reuse process concern of our software development communities. The comparison attribute “integration of software reuse in modernization and SDLC process” addresses the issue and concern for “software reuse management and measurement”, the comparison attribute “ad-hoc reuse, no strategy for software reuse” address the issue and concern of “disadvantages of software reuse”, the comparison attribute “domain based” address the issue and concern for “is software reuse domain based?”, the comparison attribute “Planning required”, addresses the issue and concern for “reuse and software quality”, and the comparison attribute “language specific” addresses the issue and concern for “is software reuse language specific?”.

Modernization with the KBSR Process and KBSR Repository has software reuse as an integrated phase as software reuse components were already identified for reuse. This modernization approach is based on with software reuse. It saved us time and cost as software reusable artefacts were already there in the repository. While using modernization without KBSR Process and KBSR Repository, we required extra time and effort to find out what software artefacts are available to reuse. This process is very resource intensive.

<table>
<thead>
<tr>
<th>Comparison Attributes</th>
<th>Approach 2: Modernization with KBSR Process and KBSR Repository</th>
<th>Approach 1: Modernization without KBSR Process and KBSR Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration of software reuse in modernization and SDLC process</td>
<td>Reuse was integrated as we had components identified for reuse. The modernization approach used software with reuse.</td>
<td>We required extra time and effort to find out what software artefacts are available to reuse. In the process we developed software for reuse.</td>
</tr>
<tr>
<td>Ad-hoc reuse, no strategy for software reuse</td>
<td>No ad-hoc reuse was done. Strategy was followed to modernize the system with reuse.</td>
<td>Strategy was followed to identify reusable artefacts. Again it was time consuming and human efforts were used. Human efforts were totally dependent on the expertise the people have and the complexity of the legacy software.</td>
</tr>
<tr>
<td>Domain Based</td>
<td>Reusable components such as Employee class from KBSR Repository were used in ACRSS and another application, the Theatre System, to check the functionality. So Software reuse is not</td>
<td>Reusable components were extracted to be reused in the same system.</td>
</tr>
</tbody>
</table>
Our second approach 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 and being incorporated into software developed and software modernization approaches.

REFERENCES


