Legacy Source Code Migration to .NET Architecture

K.Gowthaman[1], K. Mustafa[2], R. A. Khan[3]

[1]Department of Computer Science Jamia Millia Islamia New Delhi, India gowthaman@computer.org

[2]Department of Information Technology, Al-Hussein Bin Talal University, Ma'an, Jordan kmfarooki@ahu.edu.jo

[3]Department of Computer Science, Jamia Millia Islamia, New Delhi, India rakhan_cs@jmi.ernet.in

Abstract

Dot NET is one of the key products that enable application development under the new vision. However, .NET is not quite backward compatible with prior versions like Visual Basic version 6. This makes migration a serious issue. Dot NET has its built-in migration tool, which performs the vital task of converting the source code syntax. But that's just half of the work done, rest the developer needs to enable it to smoothly fix lot of issues to fit into .NET architecture. In our efforts to find out solutions to these migration issues, a reengineering Migration Model for Legacy Source Code (MM^{LC}) has been proposed in this paper. Proposed model has been further validated using a in-house project at one of the leading software development organisation. It is envisaged from the experimental try-out that the model would help the developer community to easily convert their legacy source code to target .NET code.

Key Words: Coding Standard, Compatibility Check, Forward Engineering, Refactor, Reverse Engineering.

1. Introduction

The architecture of .NET offers several advantages, such as object-oriented features, ease of developing and deploying Windows and Web applications, ability to develop Web services and mobile applications, improved security features, ability to access data using disconnected record sets, backward compatibility etc. With all the above features, it becomes necessary to convert the legacy source code into .NET framework to sustain and improve the business [13].

It is evident form the literature review that there are two ways in which re-engineering approaches are followed by Developers. **Figure 1** depicts a broad view of the reengineering approach. Most of the legacy applications are re-designed and redeveloped without using the migration tools. This approach is more expensive and may take more time for development. In this paper, we focus on availing the migration tool (shown by the dotted line) with certain human efforts to achieve the desired target as follows:

1. Evaluate the complexity of the application under consideration. In addition, analyze the feasibility of migration in relation to the time and effort involved in migration.
2. Modify the code, if necessary, before the upgrading process. This helps to reduce reworking time after the upgrade process.
3. Upgrade the legacy application to .NET framework using the Upgrade wizard.

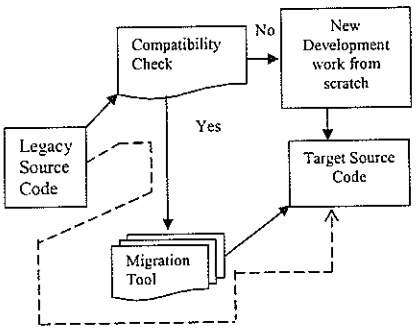

Figure 1 : Type of Re-Engineering approaches for Legency source code to target architecture

4. Modify the migrated code based on the results of the upgrade report that defines the migration issues. Section 2 provides the methodology to be followed before and after migration process for the long-term benefits of reengineering process. This model covered the various phase like Assessment, Reverse Engineering, Forward Engineering and Focus on Future Reengineering. Section 3 we compare our experimental result with one of in-house project.

2. A Migration Model For Legacy Source Code To Target .NET Architecture (MM^{LC})

Figure 2 depicts a simplistic view of a reengineering Migration Model for Legacy Source Code (MM^{LC}) to .Net Architecture. This model is based on the migration guidelines **[6][7][3]** and its related software engineering methods **[2] [14]**. However, there are a few issues that are to be taken care of while migrating from legacy code to .NET.

2.1 Phase I: Assessment

Before the migration process, the developer needs to get familiarised with the .Net Migration tools and their compatible issues **[13]**. Based on this knowledge the developer may be able to identify the complexity modules and their related debugging activities for the success of the migration process. This is called as Assessment phase of a migration project. It involves developing an understanding of the existing system **[17]**. This can be carried out through various activities such as interviews, application demonstration, meeting with system experts, and evaluation of proposed (new) system.

This will help in deciding the migration path from an existing system to a new system based on the methodology provided in this document. Assessment activities can be classified into three categories **[5]**:

1. Understanding the existing system and its development environment.
   - This is achieved through review of existing system documentation
   - Interviews, meetings, and application demonstration with system experts (such as business leaders, developers and maintainers, users, etc.)
   - Review of system history records (change log, error log, maintenance records)
2. Analysis and decision making. Based on the analysis of the existing system, business goals and objectives, customer needs, and migration recommendations, the following decisions are made:
   - Migration goals and objectives
   - Scope and extent of migration efforts
   - Migration strategies and technical approach
   - Development environment and architecture of new system
   - Critical success factors for the migration effort
3. Planning. Once the decision to migrate the existing system is affirmed, the following plans for the migration efforts are prepared:
   - Reverse-engineering plan
   - Forward-engineering (development) plan
   - Test plan
   - Configuration management plan
   - Data conversion plan
   - Installation and cutover plan Training plan

As part of reverse engineering phase at source code level we suggest the following steps for further process.

2.2 Phase –II: Reverse Engineering

The purpose of reverse engineering is to recover and reconstruct software design artifacts such as DFDs, business and validation rules, and key data elements of

the existing system. Types of the artifacts to be recovered and the effort involved depend upon the goals and objectives of a migration project and also upon the gap between existing system documentation (such as functional specification, design document, etc.) and the running system [4].

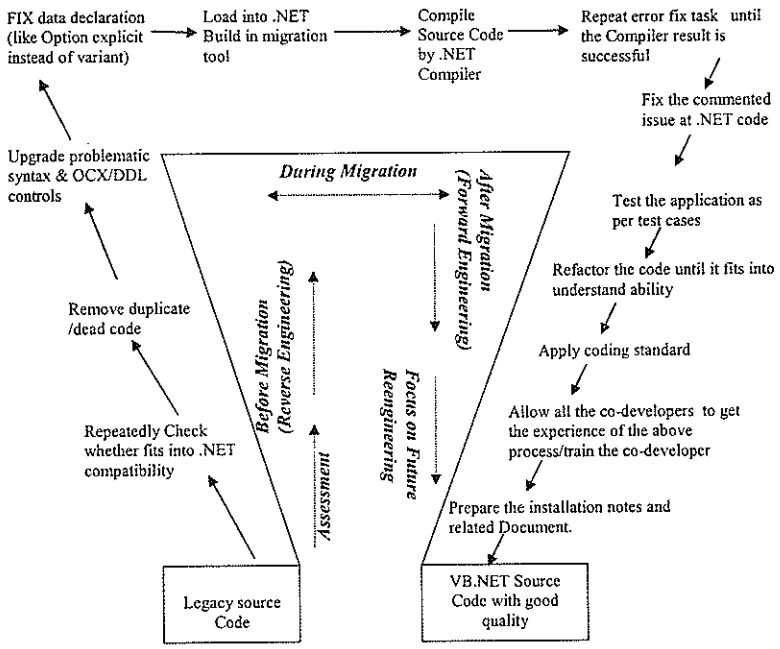

Figure 2 : A Migration Model For Legacy Source Code To Target .NET Architecture (MM^{LC})

As part of reverse engineering phase at source code level we suggest the following steps for further process.

Step-: 1 Remove dead and duplicated code

Analogously to existing applications, the redundant code is introduced by Developer's copy-and-paste practices. It is not unusual that developers indiscriminately duplicate controls without following systematic development and maintenance methods [12].

During the reengineering process, the developer uses the migration tool to convert the existing source code into a new target code without following the reengineering guidelines. As a result, the converted new code also gives poor quality of codes i.e. hard stuff to do the test, maintain, reuse and further reengineering task [6]. The dead code means unnecessary, inoperative code that can be removed which, after removal, doesn't affect the functioning of the software application, i.e. doesn't harm the intended application [16]. The removal of the dead code becomes necessary in order to save excessive memory use and speed up the execution process [12]. Leaving dead code in a completed project often means higher maintenance costs and carrying untested code around. Over and above dead code may inadvertently become operative later, leading to a possible source for errors [6] [11]. A few examples of dead code removal are given below:

Dead DLL/Procedure/Function: Assume a DLL is referred into the Reference section, but it is not used in any part of this project and is also not required at any point. So it is called Dead DLL. It consumes more memory.

Dead variable/constant: A variable or constant is declared but not used in any part of the project. So this could be

removed which not affect the functionality of existing programs.

Event not raised: An event does not fire. The class does not call Raise Event to raise the event. All existing event handlers are not executed.

Dead class: A class is not used, but it is not used in any part of this project and also it is not required at any feature.

Class not inherited: A class is defined "MustInherit" but it is not inherited by any child class. As it is an abstract class, this cannot instantiate and is also of no use at run-time. Verify how it is being used to decide whether a child class should be added, remove the "MustInherit" keyword or remove the class together with the users

Step-2: Assuming that the existing application that uses UserControls, WebClasses, DHTML pages, VB's add-in model and other special techniques cannot easily be upgraded. There are a number of manual code modification tasks to be performed in the exiting VB application to make the upgrading process much more efficient. The developer need to concentrate on the compatibility check as per .net framework **[13]**.

Step-3: Upgrade problematic syntax and controls. Use of certain syntax and control versions may cause a lot of trouble. Take for example, the Data control or VB5 Common Controls controls (ComCtl32.ocx or ComCt232.ocx). This must be upgraded to ADO data control or MsComCtl.ocx and MsComCt2.ocx, respectively.

Step 4: Fix data declarations

If existing code has a lot of Variants and undeclared variables, the project is almost a failure. Now those Option Explicit statements and proper type declarations should be added to every Dim statement which will this make this code more robust and optimized **[15]**. It will also enable the upgrade wizard to properly port this code.

2.3 Phase –III: Forward Engineering

The aim of the forward engineering phase is to design, develop, and test the new system and migrate the existing system. The design artifacts of the existing system recovered in the reverse engineering phase and the functional specification of the new system form the major inputs to this phase. The developer needs to study and experience the micro and macro issues during a migration process. Micro issues are the type which generally require a simple replacement of one data type for another and which can be easily fixed in the tool. It scans the code and displays a list of the errors found during the test. The errors are integrated with contextual help, so the developer can get more information by clicking on any error in the list and pressing help button. The developer can then use the tool to select the appropriate fix for each micro issue and it will be automatically replaced in a single batch. Then it is just a matter of testing and retesting the code. Macro issues are the ones to fear, as they are not the sorts that can be solved with a global replacement in the wizard. These require refactoring of the application to change its reliance on unsupported technologies before the migration wizard can be of any assistance **[15]**. There are a number of changes that can be done to the existing Visual Basic application to make the upgrade process much more efficient. Some important aspects are listed below **[5] [15]**:

- Avoid using late binding. This is because properties and methods cannot be verified during the upgrade process.
- Specify default properties. VB6 specifies a default property for every component. For example, the default of Textbox is Text property and Caption is

default for Label component. But in VB.NET there is nothing called as default property.

- Use Zero-Bound Array. In VB we can declare an array with any positive integer as its lower bound. But in VB.NET all the arrays are zero bound.
- Examine API calls with fixed length strings in VB application because VB.NET doesn't support fixed length strings.
- Lines and Shapes are not supported in VB.NET and hence cannot be upgraded. Instead, a graphic object is provided for shapes.
- May be taken online, i.e. web application. VB.NET provides web forms that can be used both for web applications and VB applications. This gives it dual manageability.

With continuation of forward engineering phase in this section focus the importance of bug fixing issues, importance of Coding standard and refactoring the source code.

Fix the commented issues. The upgrade wizard provides comments in VB.net code. The Developer need to follow up the action as per given comments. Now it's also beneficial to review a list of expected behaviour changes, like events that won't fire.

Test. Apply all the test cases. Run and perform all unit test programs to make sure it works as expected. Testing policy encourages developers to explain their code using test cases. Also, the requirement that all tests must run 100% at all times ensures that the documentation via unit tests is kept up-to-date. With regular technical documentation and comments, nothing is more difficult than keeping them consistent with the source code.

2.4 Phase –IV: Focus on Future Reengineering (FFR).

In this phase we describes the required action when after completion of forward engineering phase.

Deploy the Coding Standard: A code guidelines document should describe recognized coding practices **[9]**, style guidelines, commenting guidelines and acceptable metrics for size and complexity. This should also provide a reference for reviews. Every organization that develops code should have code guidelines document that provides a set of coding conventions and standards that organization has adapted to standardize the programming style, to improve maintainability, and to enhance the reusability of the software.

Refactor the source code:

The converted source code must be refactored as per need of source code understand ability matrix **[17],[2].**

The term refactoring specifically refers to a common activity in programming and software maintenance: changing the structure of a program without changing its semantics **[11]**. Often, refactoring precedes a program modification or extension, bringing the program into a form better suited for the modification step. The impact refactoring plans can have on the development team dealing with large refactorings, such as Developers can recognize changes and by-passes inside the code base that are introduced as part of a large refactoring.

Focus on Future Reengineering

The re-engineering project may fail due to one of the following reasons [8]

- Legacy system developer were no longer available (resigned or left the organization) **[8]**
- Legacy system developer is not full knowledge of entire modules. **[8]**
- Documentation is poorly written. Finding useful content in documentation can be so challenge that people might not try to do so **[18]**.
- Available documentation is out of date **[18]**.
- Difficult to understand the source code **[17]**.

To prevent the failure at future we recommend another phase called as Focus on Feature Reengineering (FFR). In this stage the developer who currently dealing the project has to share knowledge and train the co-developers. So the entire team have the knowledge about the project. This will support the project even if the team member resigns or left the organisation.

Several research shows the effects of Knowledge Transfer(KT) among the team have been conducted [1] which reports that results in better code, fewer code defects, better design, better team building. The team also ensure the document process as per traceability matrix [18]. As per this approach, once the above phases are performed methodically, the conversion from legacy source code to .Net code can be said to be of better quality and these are the codes that should be followed, throughout the migration process. In other words, the delivered source codes are Clean, Understandable, Functionally identical, Maintainable, Compact and Low risk based to face further re-engineering.

3.0 Experimental Try-out And Discussions

We experimented this model with an in-house project at one of the leading Software Development organization. About the project: The project is an application namely Resource Request Form (RRF) which is basically developed for associates who are working for the organisation. The project has following features.

- Each request form will have a unique ID, which will be used for tracking and status viewing.
- Automates all functions of the workflow pertaining to hardware or software installation processing.
- Automatic mail generation between users to draw their attention.
- Approving Authorities name, time and date will be endorsed from the terminal.
- When the form is accepted / submitted by higher authorities, the associate will receive reply e-mail with date and time of the approval of the request.
- E-mail notification will be sent at every transition of the flow of the process.

The project was developed in VB, ASP with SQL Server environment and is being used successfully by the associates. Over the period, the higher authorities decided to redesign and redevelop this in .Net environment with same additional requirements.

Table 1: Details of hours spent for Application Development (Fresh Development) without using Migration tool

Tasks	Effort in hours
System Study & Requirements Analysis	
Study of SRS from Legacy application	16
Review & Rework on SRS	8
Customer Review Feedback Implementation	16
External Technical Review & Rework	8
Prepare System Test cases & Rework	40
Test Plan	
Prepare Test Plan, Review & Rework	16
Design	
Preparation of High Level Design (HLD)	40
Functionality Review & Rework (Inspection Technique)	16
Technical Review (Inspection Technique)	8
External Technical Review & Rework	4
Low Level Design	24
Peer Review, Component Test Cases	16
Review & Rework on Component/Module Test Cases	8
Phase: Construction	
Coding	320
Code Walkthrough and Rework	40
Prepare Unit Test cases	16
Review & Rework on Unit Test cases	8
Unit Testing & Rework	28
Testing	
Component Testing with Stubs & Rework	32
Completion of Component Testing	8
Integration Testing - 1 Cycle & Rework	8

Completion of Integration Testing	8
System Testing & Rework	24
Review of Acceptance Test Cases	8
Package & Release	24
User Acceptance Testing	
Support Testing at Customers Site	16
Total (97.5 man days)	780

The project (RRF .Net) was developed in .Net environment as per Software engineering process with various phases like System study, Requirement analysis, Documentation, Design, Construction, Testing, Deployment, Documentation for Installation notes, User Acceptance test and sign off. Table 1 gives the details of total hours spent for application development without using migration tool. The apparent reason for this new development was that the developer may not aware of migration tool and may not have the patient for migration.

Development Using MM^{LC}: The earlier version (RRF in ASP,VB) of legacy source code was taken and exercised as per migration model. The developers were trained to use the migration tool. They took 5 man days to study merits and demerits of the migration tool. Then they were requested to develop the same project (RRF .Net) using migration model. The source code was converted successfully as per the model. The details of hours spent for converting source code using migration model is shown in **table 2.**

Table 2: Details of hours spent for Application Development Using MM^{LC}

Tasks	Effort in hours
Migration Tool study (Assessment)	
Study of migration tools study & Training (Forward Engineering)	40
Study of Legacy SRS & Rework	24
Customer Review Feedback Implementation	16
External Technical Review & Rework	8
Prepare System Test cases & Rework	40
Migration Analysis & Prepare the detail migration Plan	16
Test Plan	
Prepare Test Plan, Review & Rework	16
Design	
Preparation of High Level Design (HLD)	40
Functionality Review & Rework (Inspection Technique)	16
Technical Review & Rework (Inspection Technique)	8
External Technical Review & Rework	4
Low Level Design (Program Specifications)	24
Peer Review, Component/Module Test Cases	16
Review & Rework on Component/Module Test Cases	8
Construction	
Remove duplicate /dead code at legacy application (Reverse Engineering)	16
Upgrade problematic syntax at legacy application(Pre-Migration process) (Reverse Engineering)	16
Load into Migration tool and compiling the code (Reverse & Forward Engineering)	16
Post-Migration action as per compiler report (Forward Engineering)	16
.Net coding task as per Customer additional request (Forward Engineering)	80
Code Walkthrough (Coding Standard) and Rework (Forward Engineering)	40
Prepare Unit Test cases (Forward Engineering), Review & Rework	24
Unit Testing and Rework(Forward Engineering)	48
KT session to Co-Developer (Focus on future Re-Engineering)	
Allow all the co-developers to get the KT	80
Testing	
Component Testing with Stubs & Rework	20
Completion of Component Testing	8
Integration Testing - 1 Cycle & Rework	8
Completion of Integration Testing	8
System Testing & Rework	24

Discussion: Variance of time consumption of newly developed RRF .Net project and developed using migration model were very less. But we found that the usage of this migration model has the following advantages over the new development. Developer gains knowledge about migration model.

- The trained developer can be reused for similar reengineering projects.
- Time consumption and manpower requirement is also reduced.
- Also cover the Focus on Future Reengineering (FFR) which yields additional resources to knowledgeable persons.

4. Conclusion

Exhaustive review of literatures on migration shows that there are several reasons and advantages in migrating legacy applications to .NET. This paper has taken a critical look at the issues associated with such a migration on reengineering perspective. Proposed model has been validated theoretically and experimentally. The results from experimental try-out shows that .NET converted applications not only would enable to easily run on various platforms, but would be cost effective and would require less memory and maintenance with improved performance and faster speed in substantially lesser time period and lower risk rate. It is believed this migration model would be of initial support to the developer community to convert their legacy source code to target .NET code.

5. References

[1] Arie Van Deursen, Program comprehension Risks and Opportunities in Extreme Programming, Proceedings of the 8th Working Conference on Reverse Engineering 2002.

[2] Chikofsky, Elliot J. and James H. Cross II, Reverse Engineering & Design Recovery: a Taxonomy, IEEE Software, pp. 13-17, Jan1990.

[3] Dhananjay Katre, Prashant Halari, Narayana Rao Surapaneni, Manu Gupta and Meghana Deshpande, Migrating to .NET: A Pragmatic Path to Visual Basic .NET, Visual C++ .NET, and ASP.NET, Prentice Hall PTR,ISBN: 0131009621, Nov 2002.

[4] Dirk Draheim, Christof Lutteroth and Gerald Weber, A Source Code Independent Reverse Engineering Tool for Dynamic Web Sites, Proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR 2005)

[5] Gordon Brown, Performance Optimization in Visual Basic .NET, Visual Studio .NET Technical Articles September 2002, available at MSDN Library for Visual Studio 2003.

[6] John Bergey, Dennis Smith and Nelson Weiderman, DoD Legacy System Migration Guidelines, Software Engineering Institute, Technical Report: CMU/SEI-99-TN-013. Sept,1999.

[7] John Bergey, Liam O'Brien and Dennis Smith, DoD Software Migration Planning, Software Engineering Institute, Technical Report: CMU/SEI-2001-TN-012. Aug,2001.

[8] John Bergey, Liam O'Brien and Dennis Smith, Why ReEngineering Project Get failed, Software Engineering Institute, Technical Report:

[9] Kim Mens and Bernard Poll, Supporting Software Maintenance and Reengineering with Intentional Source-Code Views, 4th International Workshop on OO Reengineering, Germany July 2003.

[10] Ladan Tahvildari, Kostas Kontogiannis and John Mylopoulos, Requirements-Driven Software Re-engineering Framework, Proceedings of the Eighth Working Conference On Reverse Engineering (WCRE 2001)

[11] Martin Fowler, Seperating User Interface Code, IEEE Software Apr 2001.

[12] Matthias Rieger, Stephane Ducasse, and Michele Lanza, Insights into System–Wide Code Duplication, Proceedings of the 11th Working Conference on Reverse Engineering (WCRE 2004)

[13] Matt Pietrek, Avoiding DLL Hell: Introducing Application Metadata in the Microsoft .NET Framework, MSDN Magazine, Oct 2000, Volume 15

[14] Rick Kazman, Steven G. Woods and S. Jeromy Carriere, Requirements for Integrating Software Architecture and Reengineering Models: CORUM-II, Working Conference on Reverse Engineering 1998.

[15] Steve Hoag and Visual Studio Team, Deploying Hybrid Visual Basic 6.0 / Visual Basic .NET Applications, Visual Studio .NET Technical Articles July 2002, available at MSDN Library for Visual Studio 2003.

[16] Steve McConnell, Code Complete – A practical handbook of software constructions, Microsoft press, May 1993

[17] Tilley, Scott R.; and Smith, Dennis B, Towards a Framework for Program Understanding, Proceedings of the 4th Workshop on Program Comprehension (WPC), March 29-31, 1996, Berlin, Germany, pp. 19-28.

[18] Timothy C. Lethbridge, Janice Singer and Andrew Forward, How Software Engineers Use Documentation: The State of the Practice, IEEE Software, Dec 2003.