http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 4 (4): 420-427, 2005
ISSN 1812-5638

Source Code Migration to DOT NET Framework: A Re-engineering Application Perspective

[1]K. Gowthaman, [2]K. Mustafa and [3]R.A. Khan
[1]Department of Computer Science,
[2]Department of Information Technology, Al-Hussein Bin Talal University, Ma'an, Jordan
[1,3]Jamia Millia Islamia (A Central University), New Delhi, India

Abstract: DOT NET is one of the key products that enable application development under the new vision. However, DOT NET is not quite backward compatible with prior versions like visual basic version 6. This makes migration a serious issue. Converting existing source code to DOT NET architecture is not just a matter of loading it to the new version. 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. But before the converted code is actually compiled, the developer needs to enable it to smoothly fix lot of issues to fit into DOT NET architecture. In present efforts to find out solutions to these migration issues, a re-engineering Migration Model for Legacy Source Code (MM^{LC}) has been proposed in this study. 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 DOT NET framework.

Key words: Coding standard, compatibility check, forward engineering, refactor, reverse engineering

INTRODUCTION

The architecture of DOT 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 DOT NET framework to sustain and improve the business[1].

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 re-engineering 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. Here we focus on availing the migration tool (shown by the dotted line) with certain human efforts to achieve the desired target as follows:

- 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.

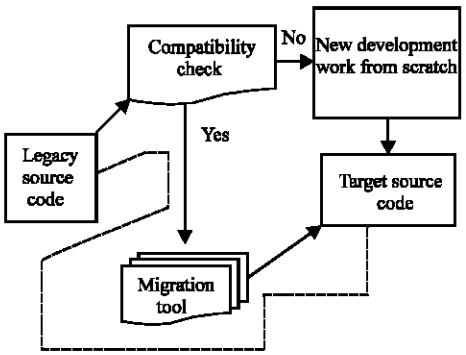

Fig. 1: Type of re-engineering approaches for legacy source code to target architecture

- Modify the code, if necessary, before the upgrading process. This helps to reduce reworking time after the upgrade process.
- Upgrade the legacy application to DOT NET framework using the upgrade wizard.

Modify the migrated code based on the results of the upgrade report that defines the migration issues.

Legacy system migration encompasses many research areas. A single migration project could, quite legitimately, address the areas of reverse engineering, business re-engineering, schema mapping and translation, data

Corresponding Author: K. Gowthaman, Department of Computer Science, Jamia Millia Islamia, (A Central University), New Delhi, India Tel: +91 11 29532992

transformation, application development, human computer interaction and testing. Due to space limitations, only a brief outline of research directly related to legacy system migration will be presented.

Tilley and Smith[2] discusses legacy system re-engineering from several perspectives: engineering, system, software, managerial, evolutionary and maintenance. A framework for legacy system re-engineering is proposed for each perspective. Using the system re-engineering framework, the implication is that the legacy system will operate normally while the target system is developed independently. When the target system is complete, the legacy system will be shut down and the target system switched on. However, the proposed frameworks are presented at too high a level to be applied in practice and no consideration is given to the migration of legacy data. This is unacceptable when considering migration of mission-critical legacy systems.

Csaba[3] presents in an approach for user interface re-engineering, in order to transform the DOS screens in Windows user interfaces. The approach uses specific Microfocus COBOL applications utilizing ADIS libraries for the management of the user interface input/output operations. It is based on the transformation of the user interface instructions into calls to functions of a C library for windows user interface.

Gold and Mohan[4] have proposed a frame study for understanding conceptual changes in evolving software systems. The framework attempts to characterize types of conceptual changes via transformations in the locality and interpretation of regions of source code. Their study is derived from a case study of commercial cobol systems. The author's approach is to create a set of concepts for a given software application and then create a set of source code indicators for the concepts. The source code is then analyzed to look for indicators of the various concepts; if one is found and matched, then the position in which it occurs is noted and a hypothesis for the concept is generated. This model can then be applied to different versions of the software system and can be used to trace the morphological evolution of the system's concepts. Often, this means that an initially well-designed software system can be seen to exhibit a visible loss in conceptual integrity as it ages; this is because concept indicators tend to become scattered as the code segments that contain them are added, deleted, moved, split and merged.

Ahmed and Richard[5] formalized a process through the introduction of the concept of water transformations. These transformations enable a light weight and formal process to convert a web application to a traditional single language system. Water transformations and it migration process can be used more generally to migrate programming languages that are embedded inside other programming languages (for example to migrate embedded SQL to newer version of SQL) by using the ideas of pre processing of the input ?le and the placement of place-holders to be later replaced in the Post processing stage after the transformations are done. Also the authors present an approach to migrate between various object oriented frameworks. The approach uses traditional programming language transformation techniques to migrate from VBScript to JavaScript and object model mapping based on a developed reference object model for web application frameworks. The approach automates error prone and low level migration details. Developers can concentrate on more interesting and complex problems in the migration process. This migration process ensures that the generated code is maintainable by preserving the structure of the migrated code and the relative position of comments in the generated code. This approach has been used successfully to build a prototype tool to migrate their application to Netscape Server Pages (NSP). We share with all the discussed approaches the idea for legacy source code migration to DOT NET architecture.

A MIGRATION MODEL FOR LEGACY SOURCE CODE TO TARGET DOT NET ARCHITECTURE (MM^{LC})

Figure 2 depicts a simplistic view of a re-engineering Migration Model for Legacy Source Code (MM^{LC}) to DOT NET Architecture. This model is based on the migration guidelines[6-8] and its related software engineering methods[9,10]. However, there are a few issues that are to be taken care of while migrating from legacy code to DOT NET.

Phase I: Assessment: Before the migration process, the developer needs to get familiarised with the DOT NET Migration tools and their compatible issues[1]. Based on this knowledge the developer may 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[2]. This can be carried out through various activities such as interviews, application demonstration, meeting with system experts and evaluation of proposed (new) system architecture. 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[8]:

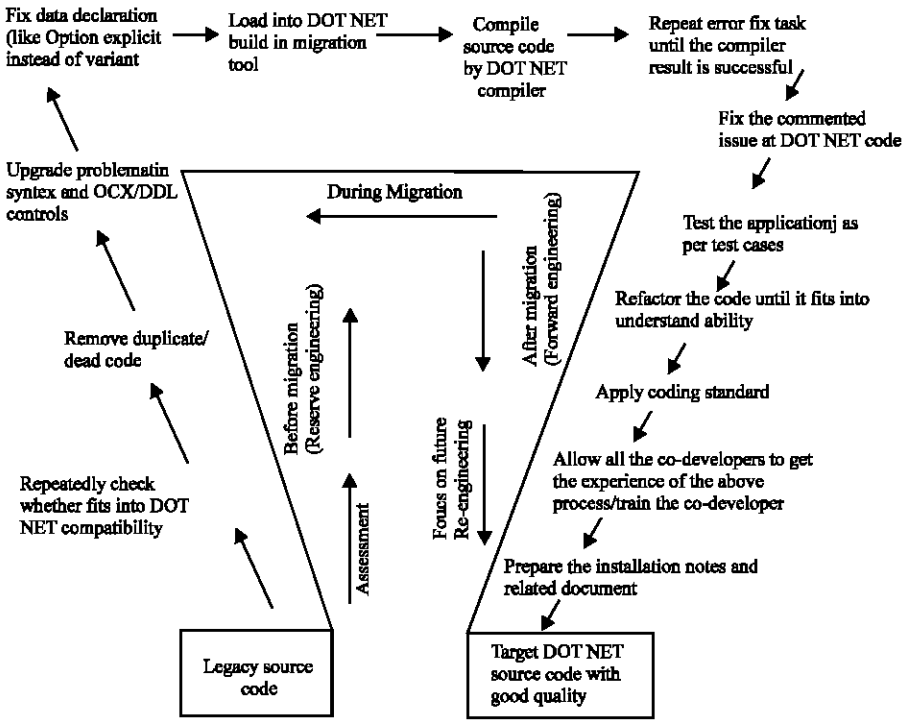

Fig. 2: Migration model for legacy source code to target DOT NET architecture (MM^{LC})

- 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).

- 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

- 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.

Once decided to move on to the DOT NET framework, we suggests for development team to do the certain task at reverse engineering phase.

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[11].

The following steps will form a major part of this phase[8]:

- Carry a program-level code walk-through and detailed analysis.
- Based on information recovered from the program-level code walk-through, regenerate functional specification document and other higher level software artifacts of the current system.
- Create the functional specification for the new system by Adding new requirements to recovered functional specification and Remove obsolete functions.

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 re-engineering process, the developer uses the migration tool to convert the existing source code into a new target code without following the re-engineering 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 re-engineering task .

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[13]. 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[6,14]. 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 RaiseEvent to raise the event. All existing event handlers are not executed.

Dead return value: A function's return value is not used by any of its callers. Review the function and the callers to determine whether the return value should be used or whether the function should be rewritten as a Sub. This problem does not apply to VB3 where return values must be used at all times.

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 the exiting VB application to make the upgrading process much more efficient. The developer need to concentrate on the compatibility check as per DOT NET framework[1].

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. The existing code is still in VB3 or VB4 and it must port it to VB6 first. Obsolete syntax like GoSubs need to be removed and there is need to prepare the conditional compilation (#If statements). But fortunately, needs to be upgraded. Only those things that the VB.NET upgrade wizard doesn't handle.

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[16]. It will also enable the upgrade wizard to properly port this code.

Phase-III: Forward Engineering: The aim of the forward engineering phase was 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[9]. 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[1]. The developer can count on Macro issues arising in code that uses ActiveX documents, graphics, RDO/ADO data binding and GoSub. If the application uses any of the above, it will take significant time and labour to refactors the source code manually.

However, there are a few issues that are to be taken care of while migrating from legacy code to DOT NET. As mentioned, porting the application using the Migration tool wizard approach does automate the process to an extent, but some modifications must be made to complete the upgrade process. The wizard produces a report in HTML format with the information of the places where it failed to migrate the application to DOT NET. 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[15-17]:

- 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 text box 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.
- Use constants instead of underlying values. VB6 constants will convert to the correct value when upgraded to VB.NET, but if actual values are used then it may end up with wrong hard-coded values. For example, for Boolean values using-1 and 0 instead of True and False and will have an adverse effect in VB.NET because inVB.NET True is 1.
- 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.

Phase-IV: Focus on Future Re-engineering (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[18], 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 and compile the documentation: The converted source code must be refactored as per need of source code understand ability matrix[3,19].

The term refactoring specifically refers to a common activity in programming and software maintenance: changing the structure of a program without changing its semantics[14]. 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[20], such as:

- Developers can recognize changes and by-passes inside the code base that are introduced as part of a large refactoring. Therefore they use the refactoring plan and connections between the plan and the code[21].

- Developers can easily see which large refactorings are not yet finished. This prevents the team from forgetting unfinished large refactorings.
- The risk of getting lost within a large refactoring is reduced by the refactoring plan. Developers can watch the plan while diving down into the refactoring. They can check whether the current activity really provides a benefit for the overall refactoring or not.
- The team can track the progress of the refactoring. This can help to plan the refactoring effort to spend within current and future iterations[21].

Focus on Future Re-engineering (FFR): The re-engineering project may fail due to one of the following reasons[22] :

- Legacy system developer were no longer available (resigned or left the organization)[22]
- Legacy system developer is not full knowledge of entire modules[22] .
- Documentation is poorly written[23]. Finding useful content in documentation can be so challenge that people might not try to do so[24].
- Available documentation is out of date[24].
- Difficult to understand the source code[25].

To prevent the failure at future we recommend another phase called as Focus on Feature Re-engineering (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[25] 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[24].

As per this approach, once the above phases are performed methodically, the conversion from legacy source code to DOT 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.

RESULTS AND DISCUSSION

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 DOT NET environment with same additional requirements.

The project (RRF .NET) was developed in DOT 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 (Table 2).

Variance of time consumption of newly developed RRF DOT 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:

Table 1: Details of hours spent for application development (fresh development) without using migration tool

Tasks	Effort (h)
System study and requirements analysis	
System study of legacy application	8
Study of SRS from legacy application	8
Review and rework on SRS	8
Customer review	8
Customer review feedback implementation	8
External technical review and rework	8
Prepare system test cases	32
Review and rework on system test cases	8
Test plan	
Prepare test plan	8
Review and rework on test plan	8
Design	
Preparation of High Level Design (HLD)	40
Functionality review and rework (Inspection technique)	16
Technical review and rework (Inspection technique)	8
External technical review and rework	4
Low level design (Program specifications)	24
Peer review	8
Prepare component/module test cases	8
Review and rework on component/module test cases	8
Phase: Construction	
Coding	320
Code walkthrough and rework	40
Prepare unit test cases	16
Review and rework on unit test cases	8
Unit testing	32
Rework	16
Testing	
Component testing with stubs	24
Rework	8
Completion of component testing	8
Integration testing-1 cycle	4
Rework	4
Completion of integration testing	8
System testing	8
Rework	8
Completion of system testing	8
Review of acceptance test cases	8
Package and release	24
User acceptance testing	
Support testing at customers site	16
Total (97.5 man days)	780

Table 2: Details of hours spent for application development using MM^{LC}

Tasks	Effort (h)
Migration tool study (Assessment)	
Study of migration tools study and training (Forward engineering)	40
System study of legacy application	8
Study of SRS from legacy application	8
Review and rework on SRS	8
Customer review	8
Customer review feedback implementation	8
External technical review and rework	8
Prepare system test cases	32
Review and rework on system test cases	8
Migration analysis and Prepare the detail migration plan	16
Test plan	
Prepare test plan	8
Review and rework on test plan	8
Design	
Preparation of High Level Design (HLD)	40
Functionality review and rework (Inspection technique)	16
Technical review and rework (Inspection technique)	8
External technical review and rework	4
Low level design (Program specifications)	24
Peer review	8

Table 2: (Continue)

Tasks	Effort (h)
Prepare component/module test cases	8
Review and 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 and forward engineering)	16
Post migration action as per compiler report (Forward engineering)	16
DOT 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)	16
Review and rework on unit test cases (Forward engineering)	8
Unit testing and rework(Forward engineering)	48
Knowledge Transfer (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	16
Rework	4
Completion of component testing	8
Integration testing - 1 cycle	4
Rework	4
Completion of integration testing	8
System testing	8
Rework	8
Completion of system testing	8
Review of acceptance test cases	8
Package and release	24
Training	
Allow all the co-developers to get the KT	36
User acceptance testing	
Support testing at customers site	16
Total (95.5 Man days)	764

- Developer gains knowledge about migration model.
- The trained developer can be reused for similar re-engineering projects.
- Time consumption and manpower requirement is also reduced.
- Also cover the Focus on Future Re-engineering (FFR) which yields additional resources to knowledgeable persons.

CONCLUSIONS

Exhaustive review of literatures on migration shows that there are several reasons and advantages in migrating legacy applications to DOT NET. This study has taken a critical look at the issues associated with such a migration on re-engineering perspective. Proposed model has been validated theoretically and experimentally. The results from experimental try-out shows that DOT 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 DOT NET framework.

REFERENCES

1. MSDN Library, 2000. Preparing your visual basic 6.0 applications for the upgrade to visual basic. NET, Microsoft Corp., http://msdn.microsoft.com/library /default.asp?url=/library/en-us/dnvb600/html /vb6tovbdotnet.asp
2. Tilley, S.R. and D.B. Smith, 1995. Perspectives on legacy system reengineering. http://www.sei. cmu.edu/ -reengineering/lsyree, pp: 6-40
3. Csaba, L., 1997. Experience with User Interface Reengineering Transferring DOS Panels to Windows. Proc. 1st Euromicro Working Conf. Software Maintenance and Reeng. Berlino, Germania, IEEE CS Press, pp: 150-156.
4. Gold, N. and A. Mohan, 2003. A framework for understanding conceptual changes in evolving source code. Proc. Intl. Conf. Software Maintenance, pp: 431-439.
5. Ahmed, E.H. and C.H. Richard, 2003. Migrating web frameworks using water transformations. Proc. 27th Annu. Intl. Computer Software Applications Conf., pp: 296-305
6. Bergey, J., D. Smith and N. Weiderman, 199 9. DoD legacy system migration guidelines. Software Engineering Institute, Technical Report: CMU/SEI-99-TN-013, pp: 2-25.
7. Bergey, J., L. O'Brien and D. Smith, 2001. DoD software migration planning. Software Engineering Institute, Technical Report: CMU/SEI-2001-TN-012, pp: 10-18.
8. Katre, D., P. Halari, N.R. Surapaneni, M. Gupta and M. Deshpande, 2002. Migrating to DOT NET: A Pragmatic Path to Visual Basic DOT NET, Visual C++ DOT NET and ASP. NET, Prentice Hall PTR, ISBN: 0131009621, pp: 35-125.
9. Chikofsky, E.J. and J.H. Cross, 1990. Reverse engineering and design recovery: A taxonomy. IEEE Software, pp: 13-17.
10. Kazman, R., S.G. Woods and S.J. Carriere, 1998. Requirements for integrating software architecture and reengineering models: CORUM-II. Working Conf. Reverse Eng., pp: 154-163.
11. Tahvildari, L., K. Kontogiannis and J. Mylopoulos, 2001. Requirements-driven software re-engineering framework. Proc. 8th Working Conf. Reverse Eng., pp: 71.
12. Rieger, M., S. Ducasse and M. Lanza, 2004. Insights into system-wide code duplication. Proc. 11th Working Conf. Reverse Eng., pp: 100-109.
13. McConnell, S., 1993. Code Complete. A: Practical Handbook of Software Constructions. Microsoft press, pp: 695-725.
14. Fowler, M., 20021. Seperating user interface code. IEEE Software, pp: 96-97.
15. Hoag, S., 2002. Deploying hybrid visual basic 6.0/visual basic DOT NET applications, visual studio DOT NET technical articles. Available at MSDN Library, http://msdn.microsoft.com/library/ default.asp?url=/library/en-us/dv_vstechart/html/ vbcondeployinghybridvisualbasic60visualbasicnet applications.asp
16. Brown, G., 2002. Performance Optimization in Visual Basic DOT NET, Visual Studio DOT NET Technical Articles, MSDN library for visual studio 2003. http://msdn.microsoft.com/library/default.asp?url=/ library/en-us/dv_vstechart/html/vbtchperfopt.asp
17. O'Toole, P., 2003. Week 37, learn some of the changes from visual basic 6 to VB.NET by examining windows forms, VB.Net Advisor. http://zones. advisor.com/doc/13041
18. Mens, K., B. Poll, S. Gonzalez, 2003, Using intentional source-code views to aid software maintenance. Intl. Conf. Software Maintenance, pp: 169-174.
19. Tilley, S.R, S.B. Dennis, 1996. Towards a framework for program understanding. Proc. 4th Workshop Program Comprehension, pp: 19-28.
20. Arsenovski, D., 2004, Refactoring-elixir of youth for legacy VB code: The code project article. http://www.codeproject.com/vb/net/Refactoring_el ixir.asp
21. Wahler, V., D. Seipel, J.W.V. Gudenberg and G. Fischer, 2004. Clone detection in source code by frequent itemset techniques. Proc. 4th IEEE Intl. Workshop Source Code Analysis and Manipulation, pp: 128-135.
22. Bergey, J., L. O'Brien and D. Smith, 1999. Why reengineering project get failed, software engineering institute. Technical Report: CMU/SEI-99-TR-010, pp: 4-22.
23. Khan, R.A, K. Mustafa and H. Shalabi, 2004. Establishing software quality assurance program, developers IQ-a software magazine. Technmedia, pp: 72-75.
24. Lethbridge, T.C., J. Singer and A. Forward, 2003, How software engineers use documentation: The state of the practice. IEEE Software, pp: 35-39.
25. Arie, V.D., 2002, Program comprehension risks and opportunities in extreme programming. Proc. 8th Working Conf. Reverse Eng., pp: 176-182.