Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 9 (22): 4086-4090, 2009
ISSN 1812-5654

A Novel Approach for Replacing Legacy Systems

Y. Moghaddas and H. Rashidi
Department of Computer, Islamic Azad University, Qazvin Branch, Qazvin, Iran

Abstract: Modernizing and reengineering on the legacy systems is an essential need in software industries. The replacement of legacy software systems is one of the most effective parts in process of modernizing software systems. In this study, a useful method for replacement of legacy large systems is proposed. In this method, the incremental approach is used to create interface and control layers on the corresponding sectors in two systems. With considering a new system in under-controlled condition and testing with real data and finding possible errors, replacement process faces with low risk. In comparison of proposed method with other traditional methods, two main differences are considered. In proposed method, data bases and resources of new system shall be different with legacy system. Second difference is finding errors to make final systems ready to decrease the risks and increase reliability.

Key words: Software reengineering, replacement, legacy systems

INTRODUCTION

Increasing the needs of organizations and their expectation levels from software system cause the increase of usage from new technology and advanced programming tools. Most important system which controlling important parts in modern society, is among systems are developed during last years, need support and maintenance to continue life. Always in these changes and supports, systems are gradually faced with some problems and limitations cause to increase the level of financial costs for their maintenance. These systems are called legacy systems. Generally, any information based systems that resist against changes and developments are called legacy system (Davis and Alken, 2000).

Set of maintenance activities of software systems are divided into three parts: corrective, adaptive and perfective maintenance. Corrective maintenance is referred into removing reported problems by users from system, in this type of maintenance there shall not be many structural changes in system. Adaptive maintenance is concerned with adapting the software to a new environment (such as new platform or OS). Also, it seems that considered changes in platform of software systems is not referred into implementing of system and there shall not be any changes in this part. But most of the times, this process are caused into some fundamental change in system and even its implementation will be impossible and it needs re-carrying out of system for new platform. Finally, in Perfective maintenance, including 65% of available maintenances shall be several changes and implementing of new needs in system. Thus, these changes does not cause fundamental changes in structure of system, but, as a result of increasing these kinds of changes is important in supporting unit (Bisbal *et al*., 1999; Pressman, 2001).

The aim of proposed method is risk reduction and increasing the reliability of legacy systems replacing process. For this propose, the effort was to reduce the rate of random faults of new system as low as possible and to remove the regular data invalidity while replacing the new systems.

MODERNIZATION OF LEGACY SOFTWARE SYSTEMS

Usually, most legacy software systems are faces with some problems such as dependency on hardware, lack of document, impossibility of tracing the program, lack of integrity and unity of system, increasing the time and supportive costs. These problems cause that organizations perform several activities for modernizing their legacy information based systems. To remove these problems, system developers, use one of these approaches: wrapping, migration, reengineering or re-developing. These four approach are followed each other and on the basis of level of changes and its dependency with legacy system, they shall be different with each other (Chikofski and Cross, 1990).

Wrapping and changing in available system, being under especial condition, are used for removing problems, sectionally. Infect, this approach is considered following the reforming and re-constructing legacy system and

Corresponding Author: Yaghoub Moghaddas, Department of Computer, Islamic Azad University, Qazvin Branch, Qazvin, P.O. Box 51745/534, Tabriz, Iran Tel: +98 912 677 4147

continuing the work with new condition. As a result of lower executive cost and it's simplify, it shall be preferred into other ways.

Migration approach transfers components of legacy system into environment having more facilities and flexibility, but data and other parts of main system are kept with out change. In this approach, components of legacy systems reforms gradually and inducts into new system. Also, it can be used in long-term for removing problems of legacy systems. In spite of that this way is more flexible, but it shall not be considered as a comprehensive method, because, although it has some standard frames for improving these processes, but, as a result of lack of re-structuring the final system structure and not main changing in code, so, it can not be used in any software projects, especially in modernizing the critical projects of legacy systems.

Reengineering approach includes a combination of reverse engineering process, re-documentation of system, re-structuring the system, forward engineering on new documents for the purpose of producing new system and replacing final system with previous. The aim of this approach is understanding available legacy system (including characteristics, designs and its implementations) and re-carrying out of system for improving its efficiency. Re-developing or buying a new system is considered as a step for avoiding form the problems of previous system and replacing all system with a new one. It is necessary that before any selection form mentioned approaches, developers consider available risk and dangers, costs and level of accuracy of each method (Chikofski and Cross, 1990; Davis and Alken, 2000).

REENGINEERING

Reengineering of legacy systems plays an important role in modern industry of software engineering. Chikofsky and Cross (1990) introduced reengineering as a process, considering original system, making it into a new form and implementing its characters in frame of this new form. In 1993 to complete the present definitions, Arrnold introduced reengineering as changing of a software production after recognition of problems, so that it causes to improve the qualities of other norms of system or it's adaptation with new platforms (Chikofski and Cross, 1990; Stevens and Pooley, 1998).

Life cycle of reengineering: As shown in Fig. 1, general process of reengineering includes three phases: abstraction, alteration and refinement. In abstraction

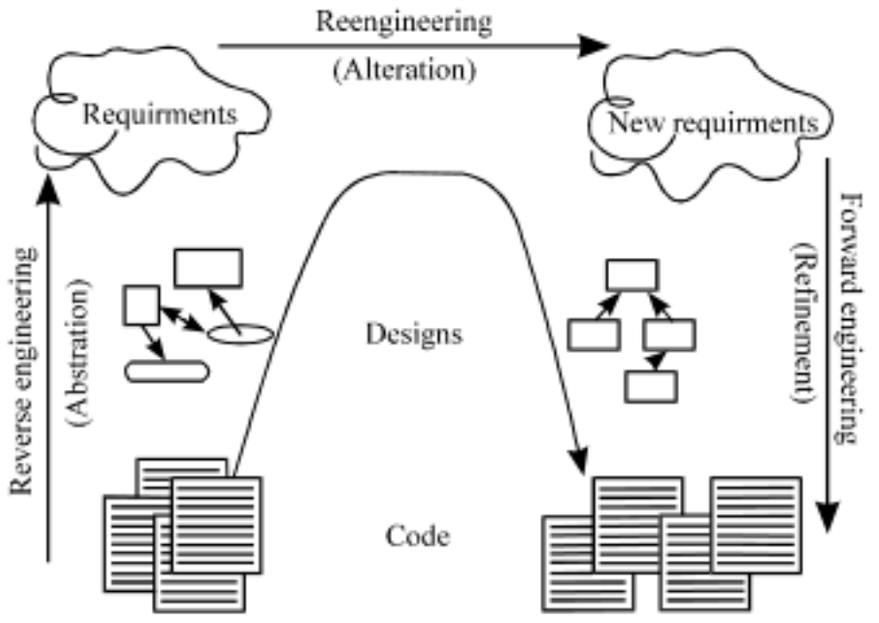

Fig. 1: Life cycle and general model of reengineering process

phase, complete documents are extracted from system by performing reverse engineering on available legacy system. In alteration phase, new documents are extracting from traditional documents and system requirements. In refinement phase, final system will be produced by forward engineering on new documents. Life cycle of reengineering is introduced in six steps that cause to achieve final system.

Requirements analysis: In this phase, reasons of performing reengineering are considered and analysis on legacy system and new requirements of system shall be recognized.

Recognition of system model: For making any changes on a system, it is necessary to determine the design structure, architecture and relation between different components of system. One of the most important problems of legacy software systems is lack of documents. Thus, initial structures and models witch are extracted from these documents may not be able to define the system completely.

Problem and fault detection: Existence of problems in legacy systems is the main reason of requesting new requirements. To perform reengineering process, it is necessary to gather complete information from these problems. For this purpose, use tools and methods carrying out some activities on structure of software system, such as leveling, testing and visualizing, is possible.

Analysis of problems: Developers must determine the reasons of existing mentioned problems and their effects on system process by complete analysis on problems of

legacy system. Also, it must be determined that solving the specified problems by supplying the new requirements is possible or not.

System restriction: To achieve final system, it's necessary to make some changes on structure of legacy systems, based on new documents.

Publication and acceptance: This phase includes all acceptance processes and testing the validity of new system and its adaptation with presented exceptions and finally replacement with legacy system (Demeyer *et al*., 2008; Müller *et al*., 2000).

REPLACING METHODS

One of the important problems which developers of legacy systems are faced with them is state of replacing of legacy and new system with together. For this purpose, developers usually use some general approaches including Big-Bang, incremental and evolutionary.

Big-Bang approach: As shown in Fig. 2, in Big-Bang approach, which is called lump sum, all parts of legacy system are replacing with a new system. This way is used in projects which problems must quickly remove in system or new facilities are added into system, immediately.

One of the advantages of this approach is quick making the system ready after completing the other steps of reengineering. But, this method is not desired for critical systems and because of high risks in replacing process, losing system data is possible.

Incremental approach: In incremental approach which is called phase-out in reengineering, considered system are divided into some logical parts by reverse engineering on legacy system. As, shown in Fig. 3, after performing software reengineering on each part, any of them will be added into legacy system as a part of new system.

By applying incremental approach, components of system are produced quickly and testing the system to find problems before the system become complex is so easy. According to this fact that temporary versions of program are produced after completing each part, customers can observe the steps of system development and determine the faults of system, quickly. But, it is not possible to perform structural changes on whole of the system and developers have to make structural changes on each part separately.

Evolutionary approach: Finally in evolutionary approach, which is the completed version of incremental approach, reengineering process and required structural changes are performed on whole of the system. In this approach

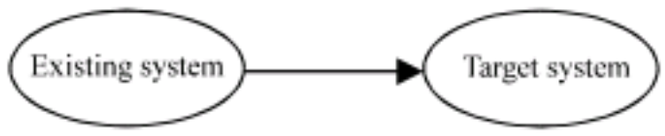

Fig. 2: Big-bang replacement approach

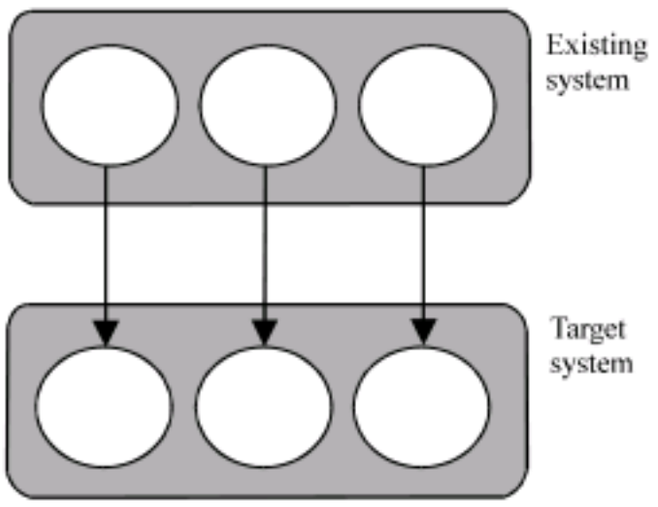

Fig. 3: Incremental replacement approach

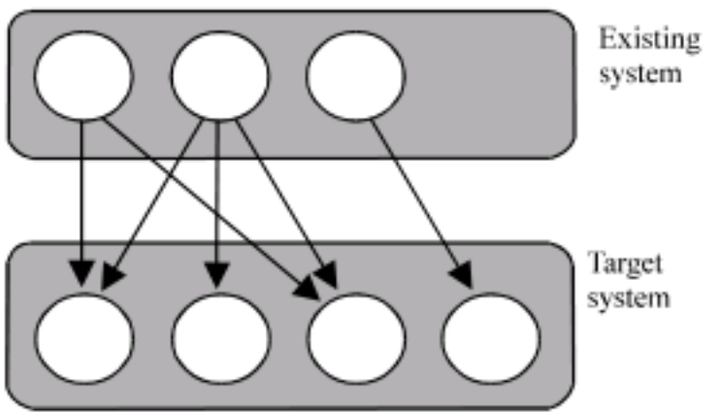

Fig. 4: Evolutionary replacement approach

(Fig. 4), it is possible to keep a part of legacy system in final system with same functionality, compose some parts of legacy system into a single part in final system or divide a part in legacy system into several parts in final system.

In this method, final system is usually designed modular and is suitable for final systems which are based on object-oriented technologies (Nierstrasz and Ducasse, 2004; Rahgozar and Oroumchian, 2003).

In most online software systems, occurring any problem will be faced the system with a critical crash. In this case, developers use incremental or evolutionary approaches to decrease the risk of Big-Bang replacement. Applying incremental or evolutionary approach make more operational overhead. But, it is justifiable on the basis of the advantages of it is usage.

Simplest method of using these methods is carrying out all these parts and suddenly replacement of new part(s) with old part(s) in legacy system. This method has many limitations and it is usable only when input and output of two parts (from new and legacy system) are same. Also, the method has high risk in replacement phase.

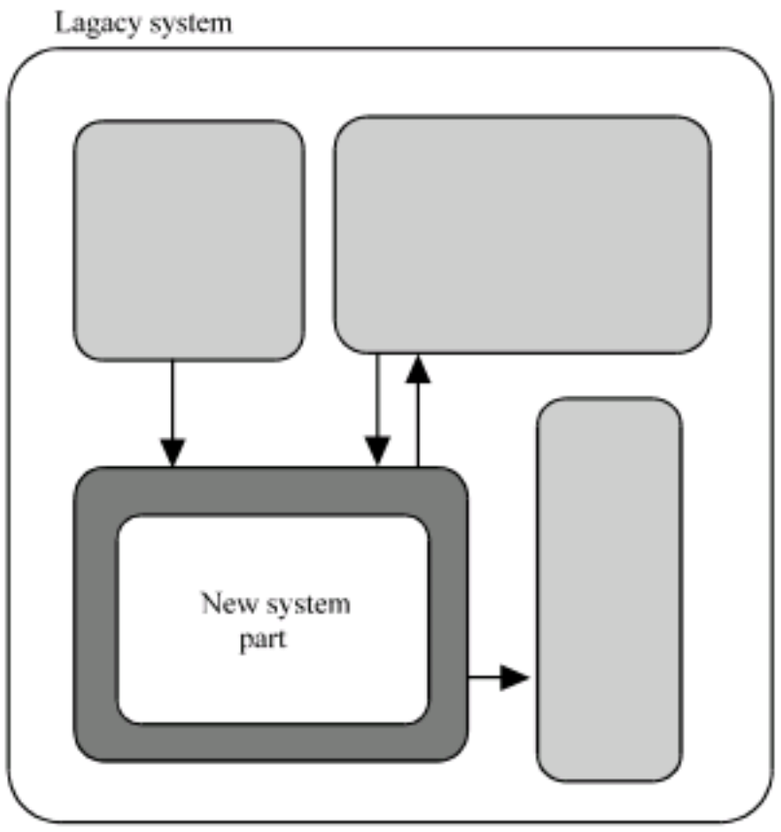

Fig. 5: Using interface in incremental replacement

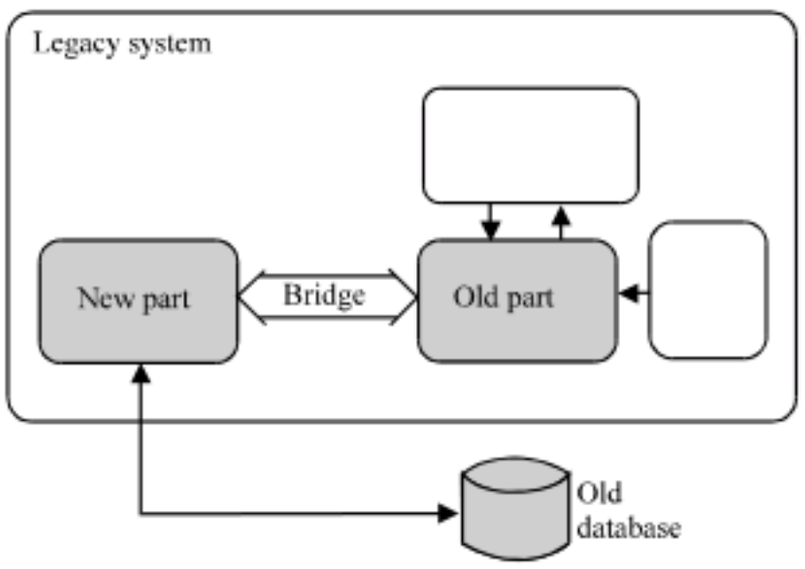

Fig. 6: Making a bridge between two systems method

As shown in Fig. 5, in another method, an interface layer is used for making relation between new part and other old parts. Thus, when inputs and outputs of a part in legacy and final system are not same, it is possible to create proper relations via interface layer.

Although, this method could remove one of the most important problems in replacement phase by applying an interface layer, but it can not be used as a suitable method for large scale projects.

In a comprehensive method, which is called making a bridge between two systems, as shown in Fig. 6, it tries to solve problems by creating a communication bridge between corresponding parts in two systems. In this method, old parts are applied as an interface for new parts and transfer requests via the bridge.

Also, there are some other methods to replace legacy and new databases with together such as: Chicken Little, Cold Turkey and Butterfly (Keller, 2000; Valenti, 2002; Demeyer *et al*., 2008).

PROPOSED METHOD

Traditional methods for replacing of new parts of software system and removing most available problems, there was no solution for removing the high risk of suddenly replacement. Even for decreasing these risks, any way for testing these parts in real environment without making any problem in legacy system is not proposed. On the other hand, traditional methods pay attention to software or data base systems separately.

In proposed method, after implementing a new part from system, to make sure about correct functionality, errors and inconsistency detection (related with other parts of system, database or other section), new parts be executed in parallel with legacy system parts. For this purpose (Fig. 7), a multi-layer control unit is used witch covers on corresponding parts in two systems. The control unit executes new parts as a virtual part and controls its functionality. Control unit includes two layers: interface and result controller.

Interface layer (activity emulator) is most important layer of control unit. Its main duty is executing the new and legacy system parts synchronously. This layer is similar to interface layer in traditional methods but executes legacy and new parts in parallel. By issuing new request from legacy parts, activity emulator delegates that request for new parts to debug new parts in real world conditions.

Verification and validation of new parts outputs is the main duty of result controller layer. This layer compares legacy and new parts' results with together to distinguish the differences. Most errors found in this unit are logical errors which based on problems in new parts. This layer increases the reliability and decreases risks of final system.

By tracking above steps, all parts of new system are combined with each other then, with solving problems and omitting applied layers, new system will be produced. With comparing the proposed method with others, two main differences are considered. In proposed method, data bases and resources of new system shall be different with legacy system. Traditional methods had proposed by the assuming legacy and new systems' data bases to be similar and in especial conditions, changes of databases were simulating with making interfaces. While most of the times, reengineering on legacy systems and databases are performed simultaneously.

Finding error to make final systems ready to decrease the risks and increase reliability is the second difference between proposed and traditional methods. It's clear that error detection in real world condition is not paid in traditional replacing methods.

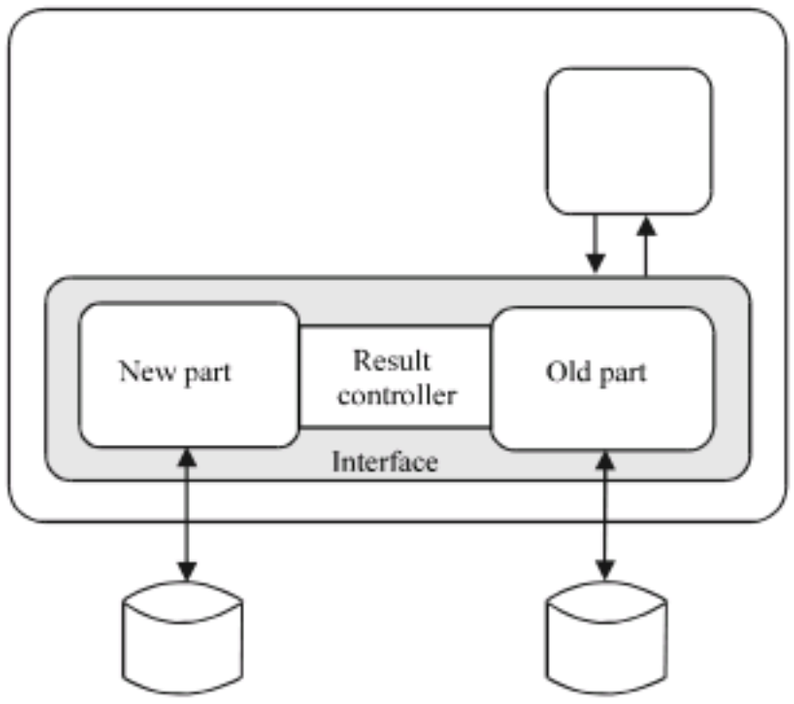

Fig. 7: Making a bridge between two systems method

CONCLUSIONS

In this study a novel method for replacing legacy software systems is proposed. In proposed method, a control unit (witch consists of two layers: Activity Emulator and Result Controller) covers on corresponding parts in legacy and new systems. The control unit executes new and legacy parts in parallel and controls the system functionality.

Proposed method has lower risk and high rate reliability in comparing with traditional methods. Using test layers in lower levels, decreasing the operational overhead of executing processes in parallel and customizing proposed method for multi-layer systems may improve proposed method.

REFERENCES

Bisbal, J., D. Lawless and J. Grimson, 1999. Legacy information system: Issues and directions. IEEE Software, 16: 103-111.

Chikofski, E. and J. Cross, 1990. Reverse engineering and design recovery: A taxonomy. IEEE Software, 7: 13-17.

Davis, K.H. and P.H. Alken, 2000. Data reverse engineering: A historical survey. Proceeding of the 7th Working Conference on Reverse Engineering, Nov. 23-25, Brisbane, Qld., Australia pp: 70-78.

Demeyer, S., S. Ducasse and O. Nierstrasz, 2008. Object-Oriented Reengineering Patterns. 1st Edn., Square Bracket Associates, Switzerland, ISBN 978-3-9523341-2-6.

Keller, W., 2000. The bridge to the new town: A legacy system migration pattern. Proceedings of the 5th European Conference on Pattern Languages of Programs, (PLP'00), Irsee, Germany, pp: 23-29.

Müller, A., H. Jahnke and B. Smith, 2000. Reverse engineering: A roadmap. Proceedings of the Conference on the Future of Software Engineering, (FSE'00), Limerick, Ireland, pp: 47-60.

Nierstrasz, O. and S. Ducasse, 2004. Object oriented reengineering patterns. Proceedings of the 26th International Conference on Software Engineering, (ICSE'04), USA., pp: 734-735.

Pressman, R.S., 2001. Software Engineering: A Practitioners Approach. 5th Edn., McGraw-Hill Series, USA., pp: 799-823.

Rahgozar, M. and F. Oroumchian, 2003. An effective strategy for legacy systems evolution. J. Software Maintenance, 15: 325-344.

Stevens, P. and R. Pooley, 1998. Systems reengineering patterns. Proc. 6th ACM SIGSOFT Int. Sympo. Foundat. Software Eng., 23: 17-23.

Valenti, S., 2002. Successful Software Reengineering. IRM Press, Italy, ISBN: 1-931777-12-8.