Journal of Computer Science 8 (6): 822-827, 2012
ISSN 1549-3636

Automatic Specification Evaluator for Effective Migration

Sudhakar, P. and P. Sakthivel
Depatrment of ECE, Faculty of Information and Communication Engineering,
Anna University, Chennai, India

Abstract: Problem statement: Software Reengineering is an effective technique for reuse the older application in the new environment. Nowadays, Reengineering techniques are increasing in spite of many difficulties and issues arise when the older application is converted to newer one. So there is a need to enhance the new system to satisfy the user requirements and quality aspects. **Approach:** For this enhancement of new system, we propose a method namely Automatic Specification Evaluator (ASE) where the interference and their effects on the new system were identified by their attributes and modify the interference if necessary. The accuracy of the migration was further increased by reimplementation of the same method. **Results:** After the proposed ASE method, the system interference was reduced and the efficiency of the new system was improved. In many migration situations, ASE produces the target system with zero interference. **Conclusion:** Our proposed method gives a good performance in the new system and hence the new system can adopt the properties of the legacy system and also satisfies the user requirements.

Key words: Software reengineering, interference, LOC, ASE, recursive ASE

INTRODUCTION

The main aim of the Software Reengineering (SR) is reusability. Legacy system is the one where the system is not supported and compatible with the new, modern environment. So, there is a need to migrate older system to a new system which is otherwise known as Forward Engineering. When the components migrated to new platform, there are several challenges such as the assembling of new application with their requirements is extremely tedious. Consequently, adaptation and integration does not perform effectively when the SR technique is directly implemented. Due to these constraints, it is not possible to implement the direct reengineering technique in the older system. All the converted systems need some structured arrangement to implement the new system to overcome the various issues. Nowadays, reengineering techniques are widely needed for services like web technologies, business and enterprise technologies. In organizations, legacy systems are valuable where each and every module plays a vital role. Direct implementation of SR in this legacy system leads to several problems. The entire software products constantly changes because of update and regularities measures. Because of these change factors in a system, the new system should not adopt the legacy system requirements and peculiarities. To implement a reengineering technique for an organization it needs enormous data and schedules. Apart from these, the method and size also constantly changes in real time situations. There are some critical information and structures which is very difficult to migrate from legacy system. There are some similarities and distinct between the new one and old ones. So maintaining the balance between the old and new system is difficult. Despite of many methods, there is no perfect and accurate method of bug free migration. So we propose a method namely Automatic Specification Evaluator (ASE) for enhancing the migrated new system considering the efficiency and accuracy as migration factors. As the name implies, this ASE method directly retrieves the specification of a legacy system from the new system and evaluates the specification for various modules in the proposed approach. This method also didn't give accurate results but it is easy to access and give efficient results in target system compared to direct reengineering. This method improves the quality of migrated system by reducing the various defects in migrated system.

When we convert the legacy system to new one, the primary problem evolved is interferences. There are several reasons for interferences as one of them is changes of updating between legacy and new system. While reengineering changes takes place in the new system and these modifications affect the new system behavior semantically and also in other system measures. These changes are indicated as interferences. Our proposed approach will reduce these interferences and enhance the new system performance behavior.

Corresponding Author: Sudhakar, P., Department of ECE, Faculty of Information and Communication Engineering, Anna University, Chennai, India

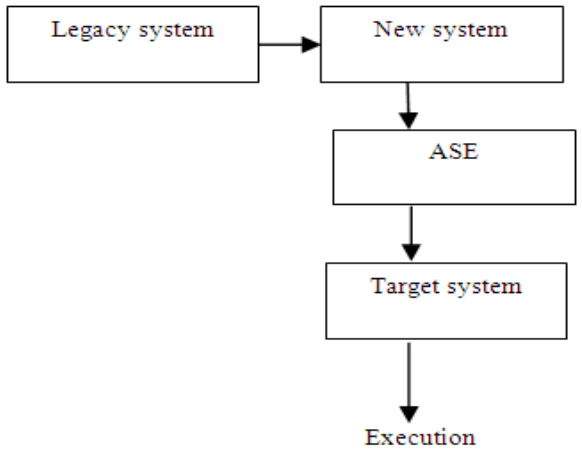

Fig. 1: Mechanism of proposed ASE

The data and some functions didn't compatible with each other and so they get overlapped with each other. This overlapping introduces some serious defects in reengineering such as loss of data, security measures. Our proposed method identifies the various interferences also reduces them smartly and the resultant system is referred as new target system. In this study, we introduce some metrics to ensure the quality of new system performance. ASE method evaluates the specification in the low level by introducing some granules in the new system and gives some breakpoints in the new system too. In general, a system has many attributes and peculiarities. To evaluate the migration efficiency, we check these attributes whether they are correct as of the legacy system. Our ASE method easily obtains these data without complications. Further the categorization as positive and negative enhancement of this method as well as the target system. To get a fine process we also iterate the process. The below Fig. 1 gives the overview of the proposed process.

When the legacy system is converted to new system using any reengineering technique then it undergoes our proposed ASE process and finally we will get the desired accurate new system as the target system. ASE takes place prior to the execution of the new system which is an added advantage of the proposed system. This ASE process is operated in all environments and also suited to integrate in any SR technique. It is quite easy to simulate this ASE as a tool by using packages that includes these functions as libraries.

Related work: Many existing research works elaborate how new system or process is extracted from older ones. For reusing the existing application, there are several approaches have been proposed. Most of the applications take a complete legacy system and transform into the new one. Moreover these approaches act as a functionality to convert a legacy system into new one. Some work gives about the model checking process of the migration. There are few works also available to enhance the reengineering technique. Some of the literatures are discussed and reviewed here.

Stilkerich *et al*. (2011) discussed about how to combine isolated legacy components with the mixed mode operation.

Jain *et al*. (2011) propose a method to extract information from legacy C++ source code and making a new system without making a new system.

Hwang *et al*. (2009) focused on improving reusability and extensibility to legacy system and proposed an automated approach to migrating legacy systems.

Chen *et al*. (2010) developed a method to class diagram and sequence diagram from the Java binary byte code**.**

Gowthaman *et al*. (2005) discussed various demerits and limitations of the current reengineering techniques. They also identified a method how to convert the legacy source code to model driven architecture.

Nagy *et al*. (2011) gave a method to technology development and functionality for effective Reengineering of Legacy systems.

Meng *et al*. (2011) introduced a method for efficient migrating of legacy system to web applications.

Zhang *et al*. (2006) made an attempt for the analysis of extracting reusable object oriented legacy code segments from their legacy programs through wrappers.

Zahi *et al*. (2009) discussed about the business process retrieval from the legacy information system through functional analysis of stable components of the legacy system.

MATERIALS AND METHODS

The proposed ASE process and their steps are summarized in the Fig. 2. During the operation of this ASE process, it ensures the legacy system and target system memory locations as the execution of the proposed ASE takes place directly to the memory location. It is easy to setup the desired location of execution as of the user criteria. The tools for garbage collection is optional to delete the no longer required memory areas also they help for common errors. It is possible to break the proposed ASE in any modules as it is a linear and conquer approach.

When the new system obtained from any reengineering technique, the first step is to retrieve the attributes of those systems. When the new system is simple, then it is easy to obtain the attributes by means of manual process like debugging.

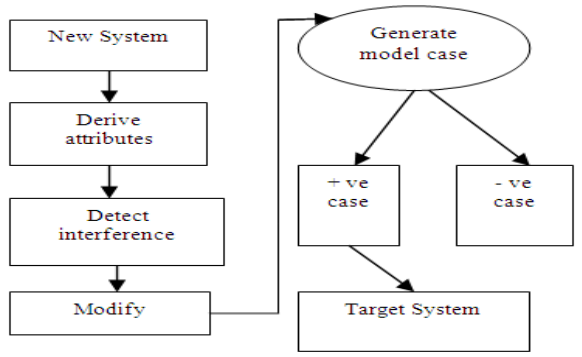

Fig. 2: Process of AS

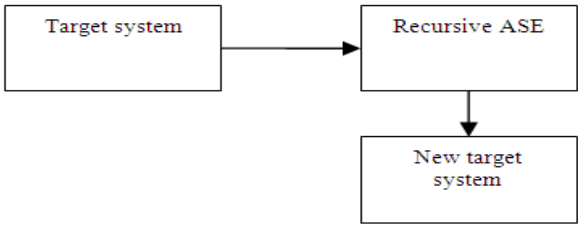

Fig. 3: Mechanism of Recursive ASE

If it is of large scale having enormous data, then the several tools and metrics are available to retrieve the attributes of the new system. Each and every action of the old system procedures and components are referred as attributes. (Example of attributes is statements, operations) Moreover these attributes are the main components of the legacy system. We have to ensure that all the available components are reengineered to the new system. We apply any slicing algorithm or any checking tools for a reengineering process. By obtaining these attributes, we are able to detect the interferences in the new system. There are few methods available to identify the interferences in the old system but this proposed method ASE with SR is a promising approach. For detecting the interferences in a system first we analyze the attributes thoroughly. We have to spend sufficient time to analyze the attributes. By doing so it is easy to find the changes and redundancy in the new system. When identifying the redundancy, take the necessary action to modify such as deletion and conversion of new process. After the modification, we should derive and generate a model system so as to perform the migration. Once the model is generated, allow the model system to undergo and check design cases with positive and negative cases inbuilt with ASE. The methods like dependency graph or any efficient measurement metrics is required to design the positive and negative cases. When the new system falls under the positive case, then execute this system as target system if it satisfied the user requirements.

ASE process removes the interference from the new system but still there are some redundancy remains in the target system. This situation arises because of overlapping and inter-dependencies of system components. When we apply our proposed ASE method again to the system, we further reduce this interference as much as possible. In many cases, the ASE gives the redundant free new system. The mechanism of recursive ASE is given in Fig. 3.

Comparing to the direct reengineering methods, implementation of this ASE is quite time consuming. But when we consider some large scale process and systems, the manual debugging and other corrective measures should take enormous time to complete the debugging process successfully. In this connection, the proposed ASE is an automated task and it will complete the process successfully with minimum time when comparing to manual debugging.

RESULTS

Our proposed ASE is experimented where CPP considered as Legacy and JAVA as a new system. So, we retrieve a JAVA program from a CPP program. The average errors predicted from various Legacy CPP programs before ASE is 0.58 which is reduced by ASE. For our experiment, we took many programs in CPP and ASE generates 18% of program as negative cases which also efficiently removed after implementing ASE. There are various issues arise when the source code is converted to JAVA. Before implementation of this method, the importance of older CPP program was considered and new JAVA functionalities so as to recover the program without loss and corrective measures.

The summarized results are discussed in Table 1 and 2, Fig. 4 and 5. Consider the following CPP program where it undergoes reengineering program with ASE approach:

```
#include<iostream.h>
#include<conio.h>
int mul(int p,int q,int r);
int mul1(int p,int q);
int main()
{
int s,w;
cout<<"s="<<mul(2,3,5)<<"\n";
cout<<"w="<<mul1(3,2);
}
int mul(int p,int q,int r);
{
int s;
s=p*q*r;
```

```
}
int mul1(int p,int q);
{
int w;
w=p*q;
}
```

Program P1: Function Overloading

Table 1: Retrieving of attributes

Legacy CPP	Derived attributes	Recovered in JAVA
Data	A1	
Methods	A2	
Statements	A3	
If….else	-	Not available
Switch	-	Not available
Operators	A4	
Data structures	-	Not available

Table 2: Raw collection of conducted experiments

Programs	LOC	Non ASE	ASE (Mins)	Time	Removal rate
P1	1800	158	56	7	3.248
P2	925	89	22	4	2.563
P3	1306	103	28	6.3	3.005
P4	982	76	11	2.1	2.632
P5	489	37	2	0.56	1.115

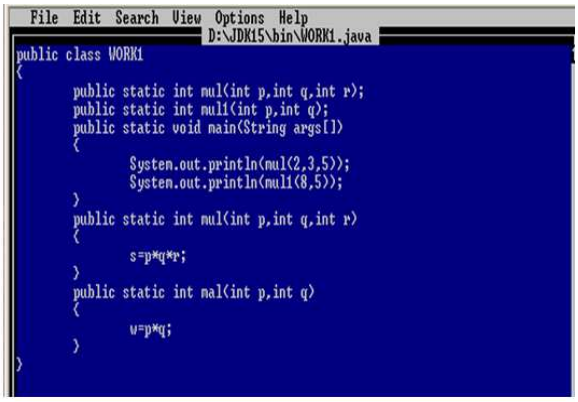

Fig. 4: New JAVA Program

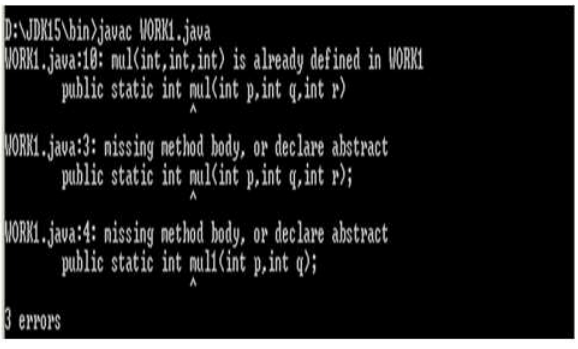

Fig. 5: Execution of JAVA program

For the above program, the derived and the recovery properties from the respected legacy program is shown in the Table 1. The program considered here is based on the function overloading conceept.

Table 1 gives the attribute satisfaction between CPP and JAVA. For a CPP program, ASE automatically retrieves all the Attributes named as A1, A2, A3 and A4. The different attributes identified in the program P1 are Data, Methods, Statements, Conditional statements such as if…else, switch, Operators and Data structures used in the CPP program. Table 1 concludes that the retrieved attributes of CPP are migrated completely. In attribute retrieval module, the attributes are identified successfully and the interference removal module is the next module of our proposed ASE approach. Consider the following CPP legacy code from the above P1:

```
int mul ( int p, int q, int r);
int mul1 ( int p, int q);
```

The above code undergoes some slicing algorithms and reengineering technique and produces a equivalent JAVA program which is given below.

The outcome of the above JAVA program is evaluated as follows:

```
p= 2;
p=3;
q=4;
q=2;
r=5;
s=p*q*r;
w=p*q;
```

When we execute the above JAVA program it will display the error result as follows.

To overcome the above errors, the second module of our work is implemented. Although the grammar is checked by the slicing algorithm there are some complicated tasks. When we analyze the migrating snippets, there are various issues and requires attention for further process. In the above transformed code, the variables p and q affect the value of s and which leads to interference. In the legacy CPP, the function overloading takes place to evaluate the result of s. But the target system i.e., JAVA doesn't support the overloading concepts and will affect the computation. It takes both the values resulting overlapping of each other by overlapping of these two values (for p 2, 3 and for q 2, 5) for a single variable. To overcome this error, we apply our ASE method. When our proposed ASE is applied to the above snippet, then the above code is modified as:

```
p=2;
q=3;
 r=5;
s=p*q*r;
```

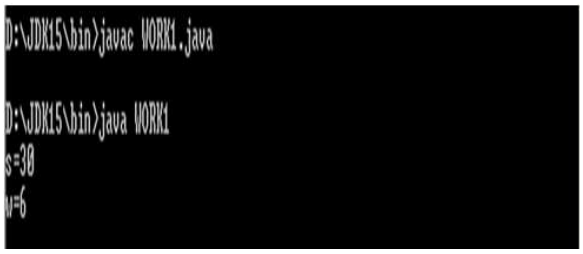

Fig. 6: Successful Execution after ASE

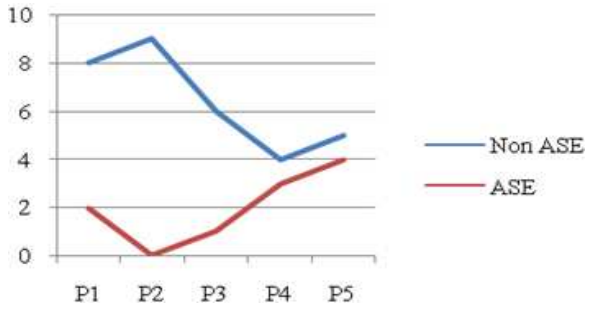

Fig. 7: Non ASE Vs ASE

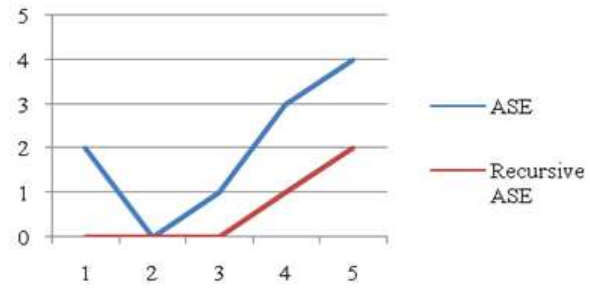

Fig. 8: ASE Vs Recursive ASE

After evaluates this snippets it evaluates other data as follows:

P=3;
q=2;
w-p*q;

For this process the execution process is given in Fig. 6.

By using our ASE the computation of s is done by two separate modules. In migration, the modification takes place automatically by deleting the snippet q: =4 in the first module and vice versa. After the modification, ASE generates the model case for user requirements satisfaction to avoid the abrupt migration. When the forthcoming code comes under positive case then it is ready for execution as the target system. We repeat the experiments for various programs.

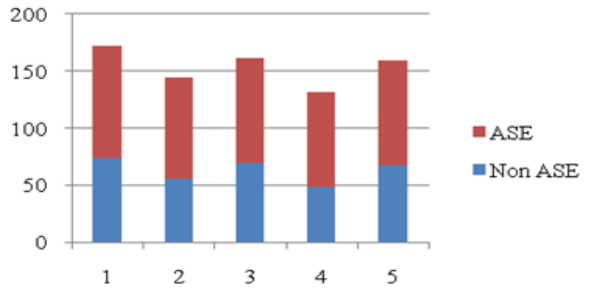

Fig. 9: Efficiency of ASE Method

Table 2 gives the raw collection of the conducted experiments and the results are discussed below.

We took various CPP programs with many attributes and constraints where the corresponding Lines of Code (LOC) of programs is estimated. The interferences are computed without ASE is denoted in the third column. Programs numbered as P1….P5. After applying the ASE method, the interference is reduced in the migration system. The time required for evaluation of this process is given in the next column. Finally the interference removal rate also computed for several programs. The below Fig. 7 shows the results of the experiments of ASE.

For the various programs P1....P5, the interference is computed with and without ASE and the above graph concludes that the ASE eradicates the interference compared to direct reengineering obtained from legacy system. The interference is very much reduced in the obtained output and hence it yields an improved new system. The added advantage of our work is after getting the resultant of the method, again gives this process as input to ASE. By doing this recursive process will give the bug free target system. While doing this recurrent process, we must ensure that the peculiarities of the original system doesn't modify in the target system. The Fig. 8 gives the results of recursive ASE process.

DISCUSSION

When the interference and other abrupt errors are overcome by modifying the new system and so the efficiency and quality are also increased in the target system with our proposed ASE method which is discussed in below graph.

The above graph concludes that the efficiency is very much increased when the ASE is implemented in the new system. It is easily verified that the target system will get with requirements and functionalities of legacy system. Interference Removal Efficiency (IRE) is calculated as a percentage of the interferences identified and corrected inside ASE process with respect to the total interference in the complete target

system. The Fig. 9 concludes that ASE improves the efficiency of the migrated system.

CONCLUSION

This method gives a methodology how to reuse the components of the Legacy system effectively with reengineering technique. In our experience, this method works very well in all environments. This ASE method identifies the interferences from the new system. It also identifies the syntactic errors from a program whether a new system is a programming language. This method is very sophisticated where the older functionality doesn't change and make easy for the complex transformations also. This approach is further enhanced by embedding the method directly to the reengineering process. There are some issues have to be discussed such as time. Although it is an automated process, it occupies significant time which is managed in further work. Future work may need some semantic and syntactic analysis on the target system. There is no doubt that this process gives a great help to deliver the new process efficiency. The static method libraries are recommended to reduce a time of running a new system. The experimental results conclude that the ASE is correct, effective and suited for high level, complicated large scale systems.

REFERENCES

Chen, L., J. Wang, M. Xu and Z. Zeng, 2010. Reengineering of java legacy system based on aspect-oriented programming. Proceedings of the 2nd International Workshop on Education Technology and Computer Science, Mar. 6-7, IEEE Xplore Press, Wuhan, pp: 220-223. DOI: 10.1109/ETCS.2010.298

Gowthaman, K., K. Mustafa and R.A. Khan, 2005. Reengineering legacy source code to model driven architecture. Proceedings of the 4th Annual ACIS International Conference on Computer and Information Science, (CIS' 05), IEEE Xplore Press, pp: 262-267. DOI: 10.1109/ICIS.2005.108

Hwang, K.S., J.F. Cui and H.S. Chae, 2009. An automated approach to componentization of java source code. Proceedings of the Ninth IEEE International Conference on Computer and Information Technology, Oct. 11-14, IEEE Xplore Press, Xiamen, pp: 205-210. DOI: 10.1109/CIT.2009.19

Jain, A., S. Soner, A.S. Rathore and A. Tripathi, 2011. An approach for extracting business rules from legacy C++ code. Proceedings of the 3rd International Conference on Electronics Computer Technology (ICECT), Apr. 8-10, IEEE Xplore Press, Kanyakumari, pp: 90-93. DOI: 10.1109/ICECTECH.2011.5941963

Meng, X., J. Shi, X. Liu, H. Liu and L. Wang, 2011. Legacy application migration to cloud. Proceedings of the IEEE International Conference on Cloud Computing (CLOUD), Jul. 4-9, IEEE Xplore Press, Washington, DC., pp: 750-751. 2011. IEEE. DOI: 10.1109/CLOUD.2011.56

Nagy, C., L. Vidacs, R. Ferenc, T. Gyimothy and F. Kocsis *et al*., 2011. Solutions for reverse engineering 4GL applications, recovering the design of a logistical wholesale system. Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR), Mar. 1-4, IEEE Xplore Press, Oldenburg, pp: 343-346. DOI: 10.1109/CSMR.2011.66

Stilkerich, M., J. Schedel, P. Ulbrich, W. Schroder-Preikschat and D. Lohmann, 2011. Escaping the bonds of the legacy: Step-wise migration to a type-safe language in safety-critical embedded systems. Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), Mar. 28-31, IEEE Xplore Press, Newport Beach, CA., pp: 163-170. DOI: 10.1109/ISORC.2011.29

Zahi, A., A. Sarhan, Formalized model of stable reengineering information system functional elements (business processes). J. Comput. Sci., 5: 915-921. DOI: 10.3844/jcssp.2009.915.921

Zhang, Z., H. Yanf and W.C. Chu, 2006. Extracting reusable object-oriented legacy code segments with combined formal concept analysis and slicing techniques for service integration. Proceedings of the 6th International Conference on Quality Software, Oct. 27-28, IEEE Xplore Press, Beijing, pp: 385-392. DOI: 10.1109/QSIC.2006.29