International Journal of Soft Computing 11 (6): 391-396, 2016
ISSN: 1816-9503

GA Factor: A Generic Automated Refactoring Tool for the Legacy Software Systems

[1]M. Srinivas, [1]G. Rama Krishna and [2]K. Rajasekhara Rao
[1]Department of CSE, KLEF, K L University, Andhra Pradesh, India
[2]Sri Prakash College of Engineering, Andhra Pradesh, India

Abstract: Over the last two decades, many business organizations had noticed that a generous amount of non-trivial legacy software frame works fail due to unstructured architectural design. Moreover, refactoring is professional procedure for managing the software systems. Indeed, programmers practice regularly with refactoring tools in two different occasions-normal program development phase whenever and wherever design problems arise. Secondly these toolsare needed at the time of code duplication, specifically when adding a new feature, the programmer need to remove the duplication using the re-factor tool. Based on level of automation, refactoring can be classified into three categories-fully manual refactoring, semi-automatic refactoring and automatic refactoring. However, fully manual refactoring and semi-automatic refactoring tools are underused, because sometimes fails to recognize the legacy code and chasing the error messages that leads to more error-prone. This study proposed a novel refactoring tool called GA factor. The GA factor system detects a developer's legacy code, reminds to the programmer that the automatic refactoring is available and if the programmer accepts then GA factor complete the refactoring automatically. GA factor automatically performs static analysis for analyzing the flowof knowledgeof the code that saves the software engineer from doing erring

Key words: Legacy system, refactoring, GA factor, switch, engineer

INTRODUCTION

Refactoring is that the method of improving the inner structure of the code in such how that's doesn't alter the external behavior of the system (Srinivas *et al*., 2016a, b). Over the last two decades, many business organizations had noticed that a generous amount of non-trivial legacy software frameworks fail due to unstructured architectural design. Moreover, research suggests that refactoring is considered a best-method for managing the software system. Indeed, programmers practice regularly with refactoring tools in two different occasions-normal program development phase, whenever and wherever design problems arise. Another is at the time of code duplication, when adding a feature, then the programmer need to remove that duplication using re-factor tool. In addition the key advantages of refactoring are making software easier to understand to find defects, improves the design of software and helpsuser to program faster.

Based on level of automation, refactoring can be categorized into three categories-fully manual refactoring, semi-automatic refactoring and automatic refactoring. However, fully manual refactoring and semi-automatic refactoring tools are underused because these two refactoring technique sometimes fails to recognize the legacy code and chasing the error messages that leads to more error-prone. In this study, we proposed a novel refactoring tool called GA factor. This GA factor system detects a developer's legacy code, reminds to the programmer that the automatic refactoring is available and if the programmer accepts then GA factor complete the refactoring automatically. GA factor automatically performs static analysis for analyzing the flow of data of the code that saves the programmer from doing error-prone work. Second, the GA factor is applicable for both application programmers and developers. Third, the GA factor keeps the 90% of configuration defaults and will not be changed when programmers use the tools until the application programmers invokes the commit within the tool in Fig. 1.

Literature review: Many researchers studied refactoring and analyze the impact of refactoring on software quality. Few of them are discussed below. (Tourwe and Mens, 2003) described three phases of refactoring perceiving when the refactoring should be applied, identifying which methodology to apply and finally carrying out therefactoring process. The

Corresponding Author: M. Srinivas, Department of CSE, KLEF, K L University, Andhra Pradesh, India

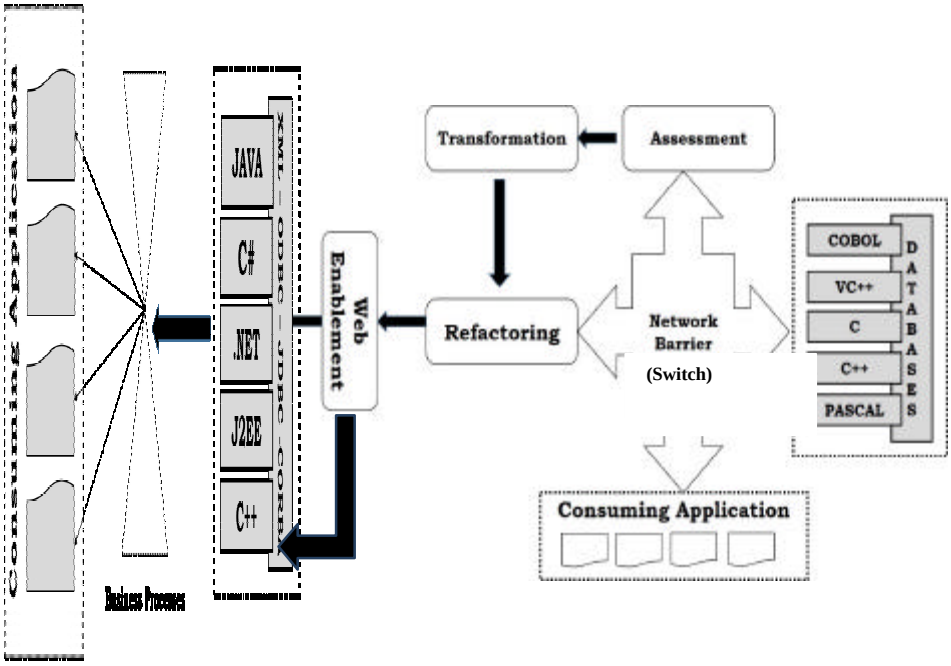

Fig. 1: Architecture of the GA refactoring system

steps in this simpler model correspond tothe identify, initiate and execute, respectively. Demeyer (2005) shows that refactoring can have avaluable influence on software performance (e.g., compilerscan optimize better on polymorphism than on simple ifelsestatements). Mens and Tourwe (2004) develop a frame workfor investigating the effects of refactoring on internal qualitymetrics but again, they have not provided an sexperimentalsubstantiation in an industrial environment.

Stroggylos and Spinellis (2007) evaluated source code versioncontrol system logs of popular open source software systems to detect changesmarked as refactoring's and examined how the software metrics are affected by this process. DuBois studied the impact of refactoring on cohesionand coupling metrics in and identified the benefits that can follow and defined the application of refactoring could improve selected quality characteristics (Bois, 2006). Fontana and Spinelli (2011) studied the effect of refactoring applied to reducecode smells on the quality assessment of the system Kataoka *et al.* (2002) and colleague's introduced a 3-step model identification of refactoring candidates, validation of refactoringeffect and application of refactoring. This corresponds to the identify, interpret results and execute steps, respectively. Vakilian proposed a compositional model for refactoring (automate individual steps and let programmers manually compose the steps into a complex change) and implemented a tool to support it. Henkel implemented a framework which captures and replays refactoring actions.

Simon *et al.* (2001) investigated how metrics were affected and found that size and coupling metrics of their system decreased after the refactoring process. Mens and Tourwe (2004) studied the effects of refactoring on internal quality metrics based on a formalism to describe the impact of a representative number of refactoring's on an AST representation of the source code. Bois proposed refactoring guidelines for enhancing cohesion and coupling metrics; they obtained promising results by applying these transformations to an open-source project. Moser studied the impact on quality and productivity as observed small teams working in similar, highly volatile domains and assessed the impact of refactoring in a close to industrial environment. Their results indicate that refactoring not onlyincreases software quality but also improves productivity in Fig. 2.

MATERIALS AND METHODS

Modeling a Generic Automatic refactoring (GA factor) tool: We designed a novel refactoring tool called GA factor. This GA factor system detects a developer's legacy code, reminds to the programmer that the automatic refactoring is available and if the programmer accepts then GA factor complete the

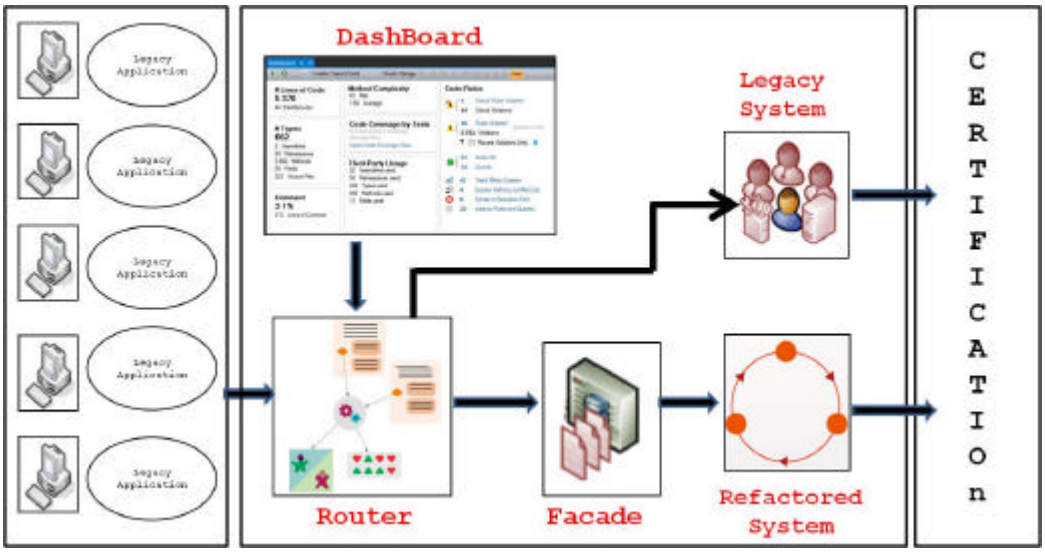

Fig. 2: Working scenario of a network switch barrier components

refactoring automatically. GA factor will overcome the burden of underuse problem that occurs in both manual and semi-automatic refactoring. To use a GA factor refactoring tool, a developer must recognize that GA factor tool is available and should select the Network Barrier (switching key) to perform refactor the legacy code. The advantage of using GA factor refactoring tool over manual refactoring first, the GA factor automatically performs static analysis for analyzing the flow of data of the code that saves the programmer from doing error-prone work. Second, The GA factor is applicable for both application programmers and developers. Third, the GA factor keeps the 90% of configuration defaults and will not be changed when programmers use the tools until the applicationprogrammers presses the commit within the tool.

This proposed GA factor tool uses a component called Switch which helps in toggling between legacy and refactored systems in a convenient and effective manner providing service certification and allowing the client to migrate from legacy to refactored system. The main functionality of this switch component is to migrate from legacy systems to services and provides backward compatibility. Moreover, if the developer want to invoke or to undo the entire legacy code that has already made then he can use same switching as depicted in Fig. 3.

The network barrier (Switch) components has several sub components such as Router, Dashboard, Facade, Messenger, Certification, Metrics which helps in achieving different functionalities of

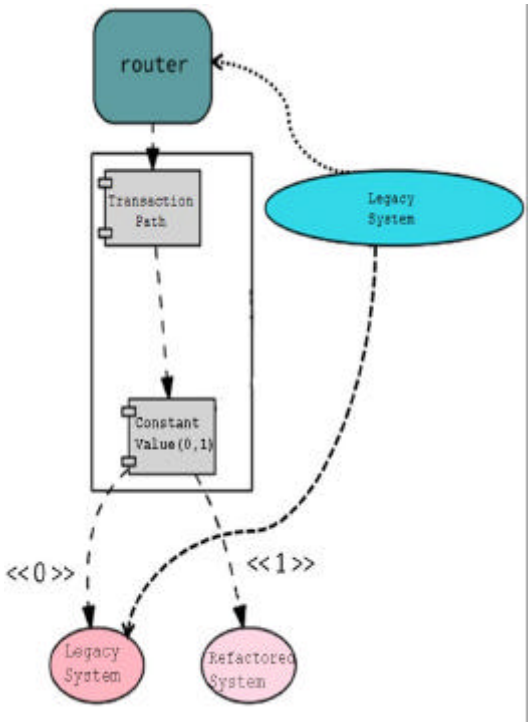

Fig. 3: GA factor refactoring process

GA factor as depicted in Fig. 3. Let us give brief each sub-component-router is the heart of the switch which is responsible mainly for routing the requests to either legacy or refactored systems. Moreover, Router will identify the transaction path based on the defined mappings at the dash board level. We will

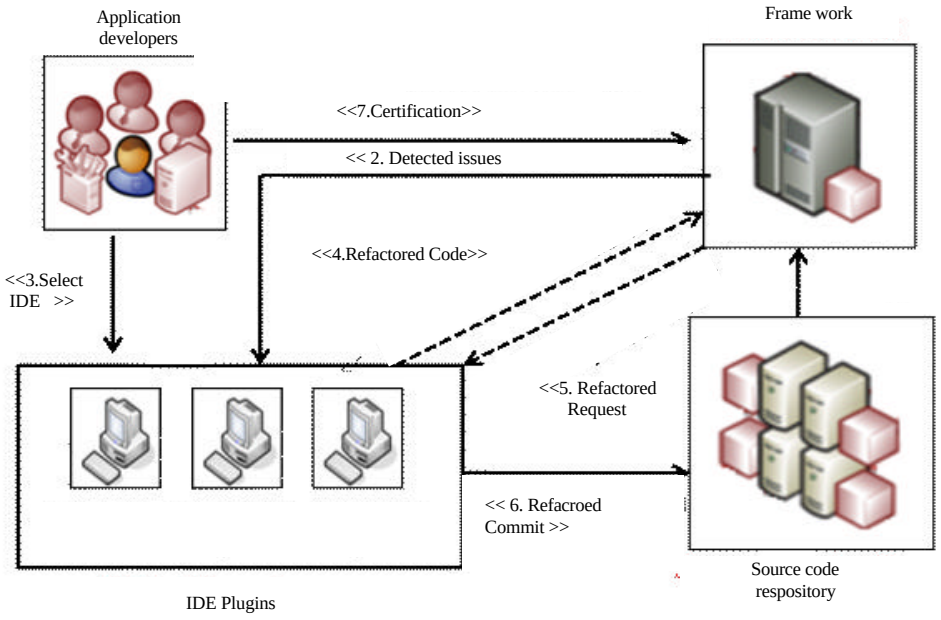

Fig. 4: Working process of a router

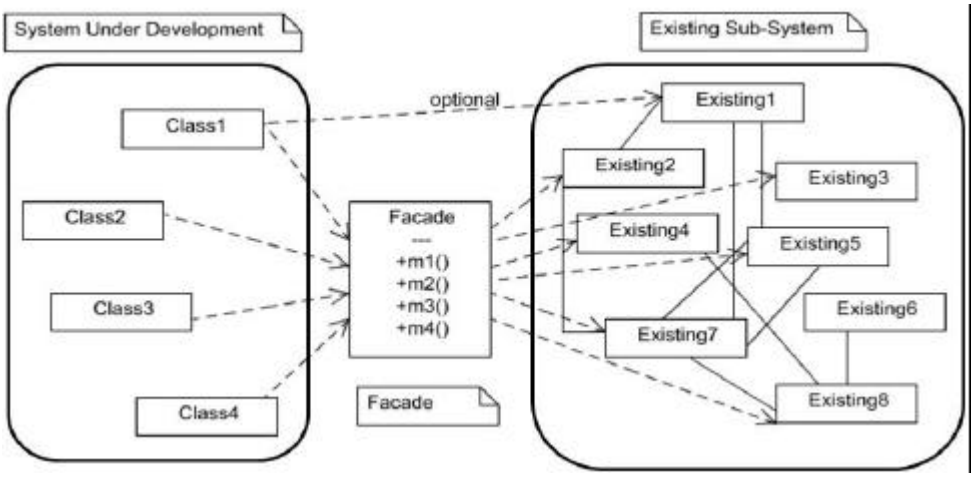

Fig. 5: Working process of a facade

use to constant values 0 and 1 to identify the route for legacy and refactored systems respectively as shown in Fig. 4. Second sub-component is facade which provides backward compatibility for legacy applications whenever there is a need to refactor the legacy systems to services, there also definite need to provide backward compatibility for existing legacy applications. This operation can be performed by façade which is shown in Fig. 5. Moreover, this façade will interact with messenger to drop the request to either legacy or refactored system and get the response back from these system. Once the response from the refactored system is received, it is validated against the configuration file to check whether there is a need to provide backward compatibility, if yes apply the configurations on the response received from the refactored system so that it will be apple to apple match to the legacy system. Next, subcomponent is messenger which is used to communication between switch and the enterprise communication systems and final one is dash board which is mainly used by the switch for mapping that identifies the transaction path of all the request that are sent by the consumer/application programmer shown in Algorithm 1.

Algorithm 1:

```
Public Class Facade {
Public void m1 ()
{ / / make  all Calls into the existing System ,
/ / hiding the Complexity
}
Puublic String m2 ()  {
 / / make all Calls into the existing System ,
/ / Converting the  return
 }
}
```

Transformation module: The main goal of the Transformation phase in GA factor is to develop a new system that has been improved in some way. What that improvement goal is depends on the individual project but it could include improved quality attributes like higher modularity, lower complexity, less replicated code), etc. As part of this phase, the quality metrics must be determined both at the beginning of the transformation and then through, out the iterative transformation steps. In this way, it can be verified that the transformations are making the desired improvements.

RESULTS AND DISCUSSION

Designing of GA factor tool using network switch barrier: Network switch barrier has an interface that is very similar to that of legacy systems so that the client applications don't need to change any connectivity logic. Each application programmer will migrate from the legacy system to switch. Once programmer connects with the switch, one has to establish a mapping in the dashboard for each application within switch. Upon the receiving the request from the application programmer, switch validates the application name against the dashboard mappings and identify a route (legacy or refactored system). If the route is to legacy application, switch will then post a request directly to the legacy system. If the route is to refactored system switch will post the request to the façade. This facade will do the transformations needed and drops the request to the messenger. The messenger will identify the transport channel and the connection details from the runtime variables and then post the request to the refactored system using these transport details. Once the request is served by the refactored system, the messenger posts the response to the facade which in turn apply backward compatibility configurations to the response and sends the reply back to the switch and switch will pass on the response to the application programmer as shown in Fig. 2. Finally, a switch is a temporary component which will be alive only till the legacy systems are in place. Once the legacy system is retired switch can also be retired.

Configuration information option: GA factor automatically collects this information option from the code deviations that the developer has already made. To perform an automatic refactoring, thisoption collects the information required. However, it is always possible to gather complete information essential to complete a refactoring.

Validation of GA factor tool: To validate this GA factor, The following sample configuration for account balance operation for bank application. In this we need to define a config entry for each service operation by using the identifiers service name and operation name. Every config entry can perform three actions namely-override, add, delete. Override used to override the refactored service response. add is used to add an object of simple type to the response. delete can delete a field from the refactored response in shown in the following code shown in Algorithm 2.

Algorithm 2:

```
<BackwardCompatability-Config>
< C o n f i g - e n t r y  S e r v i c e n a m e = " A c c o u n t "
operation_name="withdrawmoney" >
<override parent="Error" name="message" Value="Insufficient
Funds" >
<delet parent="Error" name="shortmessage" >
<add paren="Error" name="shorttex"value="ERROR"
</Config-entry>
</BackwardCompatability>
```

Here "Configuration Option file" is a configuration parameter that the application programmer can change. The data suggest that refactoring tools are configured infrequently. The actual message from refactored system for above example was retrieved by applying the GA factor as depicted in the following code shown in Algorithm 3.

Algorithm 3:

```
<error>
<message-id>6001</message_id>
<message>you hve insufficient funds to with draw</message>
<shortmessage>INSF FND</shortmessage>
</error>
```

Transformed message after applying backward compatibility configuration recovers the code to its original state shown in Algorithm 4.

Algorithm 4:

```
<error>
<message_id>6001</message_id>
<message>Insufficient Fund </message>
</error>
```

CONCLUSION

This study proposes a novel refactoring tool called GA factor. This GA factor system detects a developer's legacy code, reminds to the programmer that the automatic refactoring is available and if the programmer accepts then GA factor complete the refactoring automatically. GA factor automatically performs static analysis for analyzing the flow of data of the code that saves the programmer from doing error-prone work. As a future work, there is a need a certification mechanism to certify the refactored system that meets the requirements of enterprise and retaining the business logic of existing legacy systems.

REFERENCES

Bois, B.D., 2006. A study of quality improvements by refactoring. Ph.D Thesis, University of Antwerp, Antwerp, Belgium.

Demeyer, S., 2005. Refactor conditionals into polymorphism: Whats the performance cost of introducing virtual calls?. Proceeding of the 21st IEEE International Conference on Software Maintenance (ICSM'05), September 26-29, 2005, IEEE, Belgium, Europe, ISBN:0-7695-2368-4, pp: 627-630.

Fontana, F.A. and S. Spinelli, 2011. Impact of refactoring on quality code evaluation. Proceedings of the 4th Workshop on Refactoring Tools, May 22, 2011, ACM, New York, USA., ISBN:978-1-4503-0579-2, pp: 37-40.

Kataoka, Y., T. Imai, H. Andou and T. Fukaya, 2002. A quantitative evaluation of maintainability enhancement by refactoring. Proceeding of the International Conference on Software Maintenance, 2002, October 3-6, 2002, IEEE, Kanagawa, Japan, ISBN:0-7695-1819-2, pp: 576-585.

Mens, T. and T. Tourwe, 2004. A survey of software refactoring. IEEE Trans. Softw. Eng., 30: 126-139.

Simon, F., F. Steinbruckner and C. Lewerentz, 2001. Metrics based refactoring. Proceedings of 15th European Conference on Software Maintenance and Reengineering, March 14-16, 2001, Lisbon, Portugal, pp: 30-38.

Srinivas, M., G. Ramakrishna, K.R. Rao and E.S. Babu, 2016a. Analysis of legacy system in software application development: A comparative survey. Int. J. Electr. Comput. Eng. (IJECE.), 6: 292-297.

Srinivas, M., G.R. Krishna, K.R. Rao and E.S. Babu, 2016b. Gatalss: A generic automated tool for analysing the legacy software systems. Res. J. Appl. Sci. Eng. Technol., 12: 361-365.

Stroggylos, K. and D. Spinellis, 2007. Refactoring-does it improve software quality? Proceedings of the 5th International Workshop on Software Quality, May 20-26, 2007, Washington, DC, USA., pp: 10-16.

Tourwe, T. and T. Mens, 2003. Identifying refactoring opportunities using logic meta programming. Proceedings of the 7th European Conference on Software Maintenance and Reengineering, March 26-28, 2003, Benevento, Italy, pp: 91-101.