I.J. Modern Education and Computer Science, **2015, 2, 24-31**
Published Online February 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2015.02.04

Security Improvement of Object Oriented Design using Refactoring Rules

Suhel Ahmad Khan[1], Raees Ahmad Khan[2]
Department of Information Technology, Babasaheb Bhimrao Ambedkar University (A Central University), Lucknow, India-226025
Email: ahmadsuhel28@gmail.com[1], khanraees@yahoo.com[2]

Abstract—The main component of study is to confirm that how developed security model are helpful for security improvement of object oriented designs. Software refactoring is an essential activity during development and maintenance. It promotes the reengineering measures for improving quality and security of software. The researcher made an effort in this regard to develop security improvement guideline using refactoring activities for object oriented deign. The developed guidelines are helpful to control design complexity for improved security. A case study is adopted from refactoring example by fowler to implement the Security Improvement Guidelines (SIG). The developed Security Quantification Model (SQM^{OODC}) is being used to calculate the quantified value of security at each step. The proposed model SQM^{OODC} calculates the effective security index by ensuring that revised version of object oriented design is being influenced through security improvement guidelines. There is some possibility that original code segment may have some security flaws, anomalies and exploitable entities or vulnerable information that may influence security at design stage. SIG is helpful to cease the security flaws, anomalies, exploitable entities into refactored code segment. Each refactored steps of case study match the prediction of the impact for refactoring rules on security and the impact study for security through SQM^{OODC} model legalize the effectiveness of developed model and security improvement guidelines. The validated results of statistical analysis with different case studies of object oriented designs reflect the usefulness and acceptability of developed models and guidelines.

Index Terms—Security, Object Oriented Design, Security Quantification, Security Improvement, Refactoring

I. Introduction

An inherent dimension of software in present scenario is its need to develop. As the software is improved, modified and personalized with innovative ideas, the code becomes more complex. This unsolicited complexity will lower the quality and security of the software because huge amount of development cost is born out in the maintenance phase[3-4]. To avoid this undesirable complexity of software applications, there is an urgent need to develop a technique that cuts complexity by incrementally improving the internal software quality. A recent technique called 'Restructuring' is providing a better solution for such questions. The research domain that addresses restructuring more specifically in case of object oriented software development is called refactoring.

Security is a multidimensional attribute. The indispensable purpose of security is to control digital access of valuable property. Software security is about understanding software-induced security risks and how to manage them. As functionality of application travels to its average intensity, security issues becomes more highlighted agenda for researchers and industry people and to those who are dealing with digital technology. Using the concept of software security estimation during development of software, security can be measured by analyzing the design activities, measurement of security attributes and its impact on software. A quantitative approach can be much better than conceptual method to develop and deliver a truthful technique which can assess the actual level of security measure in newly developed software as well as in existing. Without quantification nothing can be predicted. Therefore, quantification of security has become an urgent to help predict the immunity and resilience of the software.

Security enhancement strategies are extremely enviable for improving internal structure, design simplicity, flexibility or other features of application software's. Improving applications potentialities are the key issues in the context of reengineering object-oriented software[5]. In this endeavor, refactoring provides a novel vision of object-oriented software development process. Programs are designed to satisfy immediate need and future changes can be done later if they are really needed. That is, encompass adjusting of design to hold the changes of requirements and features by applying refactoring.

The assessment of program security at design time is more efficient in the relation of improvement under the aegis of restructuring and refactoring. Refactoring and restructuring are also used in the environment of reengineering, which is the assessment and amendment of existing system to restructure it in a new appearance and the successive realization of the new form[6-7]. This refactoring made changes to the internal structure of a program to make it easy to understand and economical

for change without changing its observable behavior. It is widely used to improve the reusability, flexibility, extendibility effectiveness of the software but the quantitative assessment is more helpful to know the effects of refactoring for security design improvements[13].

II. Design Of Experiments

Software refactoring is a vital activity in the course of development and maintenance. It engineers the reengineering measures for improving quality and security of software[8]. Refactoring can reduce the effects of design corrosion, but this process requires significant effort on the part of the maintenance programmer. Design-level refactoring is also possible, but this approach operates on design models and does little to help in the subsequent refactoring of the source code[9]. A novel refactoring approach is being used to improve security of object oriented class diagrams both on its desired design and on its source code. The researcher first generates a desired design for the software grounded on the current software design and their understanding of how it may be required to evolve. Then, the source code is refactored using the desired design as a target. This resulting source code has the same behavior as the original, but its design more closely correlates to the desired design.

Refactoring works on code segments that improves quality and security without changing the behavior of software's at code level at which the software design is associated [10]. Software refactoring is commonly used in agile software processes to improve software quality[9]; it is used for continuous improvement of the software design structure. The principle of these modifications is to renovate a program structure into improved security after fitting defects such as bad smells, anti-patterns, flaws, pitfalls, anomalies, and ill-nesses[10-11]. This sort of reconstruction decreases the cost and endeavor of software maintainability for the extended run by keeping software complexity within adequate levels[12]. Refactoring has been used in practice to improve software security for commercial and open-source software systems.

III. Formulation Of Rules

Different works are identified which are helpful to improve quality of object oriented design using a novel refactoring rules. In this regard Raed Shatnawi presented work to estimate the quality of software using refactoring activities for object oriented designs[13]. Raed's work is inspired by Jagdish Bansia hierarchical model for quality estimation[1]. Raed adopted the core quality factors and metrics of Jagdish Bansia and study the impact of refactoring activities by establishing refactoring heuristics. Raed uses only 43 refactoring activities out of 72 activities to fix 22 code bad smells of fowler catalogs[14]. As per security concern, restructuring or refactoring have received relatively little care at code level. The impact analysis of code level refactoring may influence the design structure of software.

The assessment of refactoring rules at code level through security metrics is capable to produce a quantitative analysis of information security. From this point of view a work carried out by Bandar Alshsmmari, that establishes a theoretical background of refactoring rules for security at first sight and develop security metrics accordingly. Bandar calculated the metric values for given java program using static tool analyzer. Bandar uses the code refactoring rules in context of security assessment and recalculated the metric values on the basis of security assessment guidelines inspired from refactoring rules to validate the results for security. Bandar picks only 16 refactoring activities and reframe his observation for security restructuring[15]. Another work developed by Maruyama that aims to improve the security of a given program's code by identifying vulnerabilities by using design set of secure refactoring rules[16]. Hafiz's work also uses secure transformation rules for secure refactoring[17]. The detection of highly secured classes in real time large applications due to sharing libraries and code between them is one of the challenging issues discussed by smith[18].

On the basis of above discussion, it is proved that very little work has been done to examine the impact of refactoring activities for security improvement. It is evident form literature that there is an urgent need to develop security improvement strategies on the basis of refactoring activities. The researcher made an effort in this regard to develop security improvement guidelines using refactoring activities for object oriented deign. Researcher adopted and extracted the set of refactoring activities and case study from Fowler catalogs[14]. The identified set of activities are analyzed to expose the impact of design security rules in context of security improvement[14, 19-24].The details of identified refactoring activities and its effect have been discussed in Table 1.

Table 1. Impact analysis of refactoring rules for design security
*Rules having positive or negative impact based on condition

S. No.	Refactoring activity	Impact analysis	Impact on Security
1	Extract Method	Increases accountability of classes by fragmenting the long methods into small methods.	↑
2	*Extract Class	Create a new class and move the relevant fields and methods from the old class into the new class. This activity used to break a large class.	↑↓
3	Inline Class	Inline class moves all features into another class and removes it.	↑
4	*Move Method	Create a new method with a similar body in the class it uses most. Either turn the old method into a simple delegation, or remove it altogether.	↑↓
5	Move Field	A class is being used by another class more than the class on which it is defined. Create a new field in the target class, and change all its users	↑
6	Replace Temp with Query	Extract the variable's initializer expression into a method, and replace all references to the variable with the calls to the extracted method.	↑
7	Encapsulate Fields	If there is a public field. Make it private and provide accessors methods.	↑
8	Replace type code with state/strategy	If you have a type code that affects the behavior of a class, but you cannot use sub classing. Replace the type code with a state object.	↑
9	Replace conditional with polymorphism	Move each leg of the conditional to an overriding method in a subclass. Make the original method abstract.	↑
10	Replace inheritance with delegation	Create a field for the superclass, adjust methods to delegate to the superclass, and remove the subclassing. It allows removing a class from inheritance hierarchy, while preserving the functionality of the parent.	↑
11	Replace data value with object	A data item that needs additional data or behavior. Turn the data item into an object.	↑

Refactoring activities are set according to their composition of methods and objects, movement between objects, conditional expression and coverage with generalization and organization of methods & class behavior. This limited refactoring activity is engaged to quantifying and improving class diagrams using object oriented design complexity attributes. Refactoring is an art to know the potential impact of risk for application and provides preventive measures for secure designing. This will assist developers when to refactor, how to refactor and where to refactor in small steps to avoid bugs into code for improved software design. The research identifies the limited set of refactoring activities and evaluates the technicalities of these refactoring activities that are applicable to measure the impact of security for object oriented design perspective. We documented the quantitative effect of each refactoring activity on design properties for security.

Extract Method, a common refactoring activity increases the accountability of classes by fragmenting the long methods into small methods. These small methods increase the reusability and minimize the design complexity for easier visibility and understandability. Preferences to use short methods increase the acceptability of chances that other methods can use a method when the method is finely grained. Overriding is also easier when the methods are carefully grained. If extracting improves clarity, do it, even if the name is longer than the code you have extracted. Make a new method, copy the extracted code from the source method into the new target method. Scan the extracted code for references to any variables that are local in scope to the source method. Pass into the target method as parameters local-scope variables that are read from the extracted code. Replace the extracted code in the source method with a call to the target method. This activity may increase coupling between objects. The overall discussion concludes that it extract method have positive impact on design security.

Extract Class is used to break a large class, i.e., the class is doing a work of two or more classes and should be divided into more classes. Create a new class and move the relevant fields and methods from the old class into the new class. The steps fallowed in Extract Class are: First, decide how to split the class. Second, make the new class. Third, make a link between the two classes. Fourth, move fields to the new class. And last, move methods to the new class. This activity increases the number of classes. Coupling between objects is increased due to linkage of two classes. Moving methods and fields to the new class increases the cohesion of both classes. This refactoring activity has no effect on the inheritance measures. It increases number of classes in the system as well as coupling and cohesion among methods. Due to unwanted anomalies or security flaws, the behavior of methods or attributes can be vulnerable. As per the possibility of security flaws increases, the unwanted design complexity leads to less secure design. This

refactoring activity may have negative impact on security. Conversely, inline class moves all features into another class and removes it, reduces the number of classes in design. Therefore from this point of view, it reduces the flaws by minimizing exploitable classes of design for improved security.

Move Method is used by more features of another class than the class on which it is defined. Create a new method with a similar body in the class it uses most. Either turn the old method into a simple delegation, or remove it altogether. Move method is applied when classes have too much behavior or when classes are collaborating too much and are too highly coupled. By moving methods around, the classes become simpler and they end up being a crisper implementation of a set of responsibilities. Such kind of effort reduces coupling, responsibilities of class and complexity of the refactored classes. This refactoring activity may increase the security of design.

Move Field, organized in such a way that a class is being used by another class more than the class on which it is defined. Create a new field in the target class, and change all its users. The fundamental nature of design exhibits state and behavior using field's movements in class structure. This will help to distribute the responsibilities of classes in design structure. Allocation of fields or attributes to other classes increases the coupling and cohesion among methods. The exploitable attributes may be avoided through refactoring the design with move field for security enrichment. This can be used to minimize design complexity of object oriented class diagram.

Replace Temp of Query activity substitute's temporary variable with a method (query). Temporary variable is being used to clutch the result of an expression. Extract the expression into a method. Replace all references to the temp with the expression. The new method can then be used in other methods. Adding new methods increases the responsibilities of classes. Replace Temp with Query often is a vital step before Extract Method. This refactoring activity simplifies methods but having possibility to increase the number of classified method that sometimes not good for secure design.

Encapsulate fields says that there is a public field. Make it private and provide accessors methods. One of the principal tenets of object orientation is encapsulation, or data hiding. This says that you should never make your data public. When the data is made public, other objects can change and access data values without the owning object's knowing about it. This separates data from behavior. This is seen as a bad thing because it reduces the modularity of the program. When the data and behavior that uses it are bundled together, it is easier to change the code, because the changed code is in one place rather than scattered all over the program. Encapsulate Field begins the process by hiding the data and adding accessors. But this is only the first step. A class with only accessors is a dumb class that doesn't really take advantage of the opportunities of objects, and an object is terrible thing to waste. Once programmer has done Encapsulate field, it looks for methods that use the new methods to see whether they fancy packing their bags and moving to the new object with a quick Move Method. It surges data encapsulation and responsibilities of a class. This may helps to increase the security of design.

Replace type code with state/strategy speaks that if a type code which affects the behavior of a class, but difficult to use sub classing. Get the type code replaced with a state object. The mechanism is first Self-encapsulate the type code, then Create a new class, and name it after the purpose of the type code. Add subclasses of the state object. Create an abstract query in the state object to return the type code. Create a field in the old class for the new state object. Adjust the type code query on the original class to delegate to the state object. Adjust the type code setting methods on the original class to assign an instance of the appropriate state object subclass. This will increase the number of classes, data encapsulation, coupling, cohesion, polymorphism, responsibilities of a class, while it reduces the complexity of the class. This could be helpful to increase the security of design.

Replacing Inheritance with Delegation puts that a subclass uses only part of a subclasses interface or does not want to inherit data. Create a field for the super class, adjust methods to delegate to the super class and remove the sub classing. This activity increases coupling and accountabilities of a class, while decreases number of hierarchies, utilization of inheritance and polymorphism. Replace Data Value with Object rule states that a data item needs additional dada or behavior. Encapsulate the data item in its own object. The numbers of classes gets fueled due to this activity and it also increases data encapsulation, coupling, among objects, cohesion among methods, use of composition and responsibility of class.

Replace conditional with polymorphism states that a conditional that chooses different behavior depending on the type of an object. Move each conditional to an overriding method in a subclass. Make the original method abstract. This action increases the number of classes, utilization of inheritance, polymorphism while it reduces the complexity of the class. In the context of above discussion a security improvement guideline has been proposed on the basis of using refactoring rule for object oriented design complexity. The proposed security improvement guideline using refactoring rules (SIG) are mentioned below:

Security Improvement Guidelines Using Refactoring Rules (SIG):

Extract Method Rule: Fragmenting long methods into small methods increases reusability and minimize the design complexity for improved security.

Extract Class Rule: Extraction may lead number of classes in system as well as coupling and cohesion among methods while decreases accountability of classes. This may lead unwanted complexity so keep it low as much as possible.

Inline Class Rule: This action decreases number of classes, coupling between classes and cohesion among methods while increases responsibilities of classes for improved security.

Move Method Rule: Simplicity can be achieved by moving methods around the classes. This action shortens coupling, responsibility and complexity of refcatored version. This will help to increase security of design.

Move Field Rule: Allocation of fields or attributes to other classes increases the coupling and cohesion among methods. The exploitable attributes may be avoided through refactoring the design with move field for security enrichment. This can be used to minimize design complexity of object oriented class diagram.

Replace Temp with Query Rule: Adding new methods increases responsibilities of a class. Try to minimize the possibility to increase the number of classified method for secure design.

Encapsulate Fields Rule: This activity increases data encapsulation and responsibilities of a class. Higher values put refactored design much secure.

Replace type code with state/strategy Rule: It increases number of classes, data encapsulation, coupling, cohesion, polymorphism, responsibilities of a class, while it reduces the design complexity of the class for improved security.

Replace conditional with polymorphism Rule: This movement increases number of classes, utilization of inheritance, and polymorphism, while reduces the design complexity of the class.

Replace inheritance with delegation Rule: It increases number of hierarchies, utilization of inheritance and polymorphism, while decreases coupling and responsibilities of a class for improved security.

Replace data value with object Rule: This is responsible for security improvement by increasing data encapsulation, cohesion, and use of composition and responsibility of classes.

This may helpful to increase the security of design. Thus researcher has find that the selected refactoring rules can potentially affect the security of programs, if applied to a security critical code segment. This will helps to improve software design after providing changes in code segment without changing its functionality and behavior.

IV. Implementation Of Rules

A case study movie rental system in fig 1 is taken to implement the refactoring rules for security improvement. This case study is extracted and adopted from refactoring example by Fowler to implement the security improvement guidelines (SIG) using refactoring rules and Security quantification model (SQM^{OODC})[25] for object oriented design. The developed security quantification model (SQM^{OODC}) is being used to calculate the quantified value of security at each step. The proposed model SQM^{OODC} calculates the effective security index by ensuring that revised version of object oriented design is being influenced through security improvement guidelines. There is some possibility that original code segment may have some security flaws, anomalies and exploitable entities or vulnerable information that may influence security at design stage. SIG is helpful to cease the security flaws, anomalies, exploitable entities into refactored code segment.

SQM^{OODC} model is being used to calculate the effective security index at each step to check whether refactored version of design is being improved or not. The original and refactored code segment of case study is adopted from Fowler[14]. A. Each refactored steps of case study match the prediction of the impact for refactoring rules on security and the impact study for security through SQM^{OODC} model legalize the effectiveness of developed model and security improvement guidelines. The above mentioned refactoring rules are implemented for design security improvements using different case studies. The impact of refactoring rules on security has been studied in previous section and implementation of those rules for secure design improvements is applied on the case study of Movie Rent System.

Decomposing the statement () method: The statement () method in the class Customer is too long and we use Extract Method to create a new method called amount For ().

It comprises the whole switch statement. The variables this Amount and each in the new method amount For () are not very meaningful. We change their names to result respectively rental to reflect their uses.

Moving the amount calculation: The method amount For () in the class Customer uses information from the rental, but does not use information from the customer. Because of this, moves this method to the class Rental, where it will surely feel better, and rename it to get Charge (). To keep the functionality of the customer, delegate calls to amount For () in Customer to the new method get Charge () in Rental. Unnecessary delegations like this should be avoided, and search all calls to the method in Customer and replace the calls with the delegation code itself. After that, delete the now useless delegation method in Customer. In the method statement replace every use of the variable this Amount with the query each. get Charge() (Replace Temp with Query).

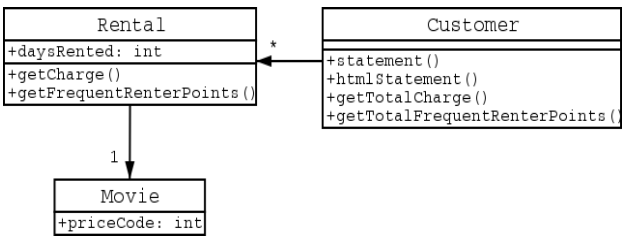

Fig. 1. Class diagram Movie rental System before refcatoring

The while loop in the Customer's statement () method mixes presentation and business logic. Copy the loop along with some needed accompanying lines and paste it to a new method called total Amount (). This method does the business calculation. Remove all this stuff from the original loop, so that it only covers presentation. To get the total amount owed, Call the new method. It is

almost identical to the statement () method, only that it has HTML tags in it to do the layout on the screen.

Replacing the conditional logic on price code with polymorphism: To avoid switch statement, replace the explicit logic with implicit logic by using polymorphism. Because the decision bases on data from a movie, move the method get Charge () onto movie. A delegation in the old method ensures compatibility with our tests and other callers. This effort produces refactoring successfully for given application. While these refactoring cleaned the design considerably, it was not enough to support easy change of classification of movies. So some more refactoring, coupled with a design pattern, led us to a really elegant program, in contrast with the starting code. The refactored version of class diagrams and security improvement analysis is depicted in Fig. 2, 3 and 4 respectively.

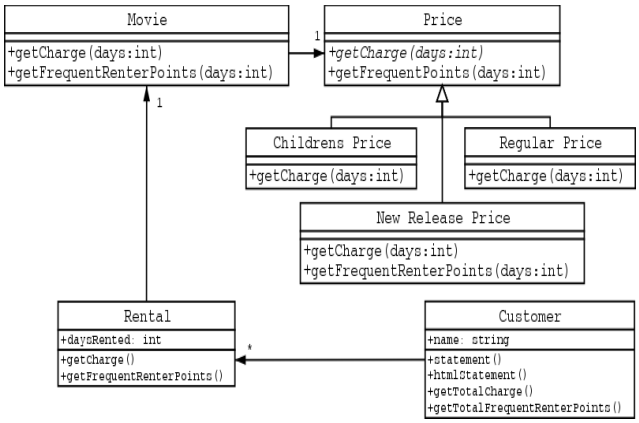

Fig. 2. Class diagram of Movie rental system after refactoring

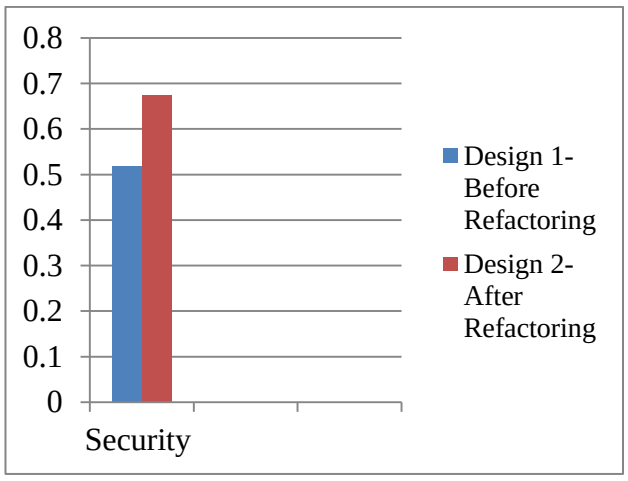

Fig. 3. Graphical representation of case study for security improvement

V. Improvements

To assess whether the proposed rules of refactoring is able to improve the security of design, different case studies have been conducted. The results of collected data for security are tabulated in Table 2.

Table 2. Security Improvement Analysis

Security Quantification / Design	Before Refactoring (Old Design)	After Refactoring (New Design)	Security Improvement (In Percentage)
Design 1	0.465	0.658	41.5%
Design 2	0.464	0.630	35.7%
Design 3	0.342	0.461	34.7%
Design 4	0.519	0.674	29.8%
Design 5	0.372	0.470	26.3%
Design 6	0.518	0.602	16.2%
Design 7	0.595	0.672	12.9%
Design 8	0.595	0.620	04.2%

VI. Statistical Anlysis

Statistics is a mathematical tool used for gathering, organizing, analyzing and interpreting numerical data. For the purpose of showing statistical significance or validation of the proposed refactoring rules is applied for security design improvements. As the sample size is small, the student t test is applied for finding out the level of significance and rejection of the null hypothesis[2]. The old values and new values gone under statistical analysis to draw the conclusion that whether there is a significant differences between the pre treatment data and the post treatment data. The obtained t value will determine whether to reject the null hypothesis and accept the alternative hypothesis.

Hypothesis Testing: A null hypothesis reflects that there is no significant relationship between two or more parameters whereas alternate hypothesis affirms the relationship. Rejection of a null hypothesis provides a stronger base to accept the relationship or to accept the alternate hypothesis.

Null Hypothesis (H_{01}): Security based refactoring guideline using Model SQM^{OODC} cannot helpful to quantify & improving security of object oriented designs.

Alternative Hypothesis (H_{11}): Security based refactoring guideline using Model SQM^{OODC} can helpful to quantify & improving security of object oriented designs.

Statistical Interpretation: If the security values in table are observed, it can be inferred very easily that the SIG (Security Improvement guideline) treatment for all the design has worked well. The new security values are relatively less than those old values. By reflection, it seems that the treatment worked. The security values in all the eight designs were increased and hence the security is improved. Fig. 2 represents the graphical representation of security data before and after refactoring. The initial claim that SIG is able to improve security proved true. All in all, the level of significance of the proposed approach must be computed. While studying inferential data analysis, it was found that the t-test for the situation given below is appropriate for the purpose: 'When the same group of individuals takes a pretest then the group is exposed to a treatment. The

group is again tested after treatment to determine whether the influence of the treatment has been statistically significant as determined by mean gain scores.' The t-test was carried out for drawing level of significance of the approach.

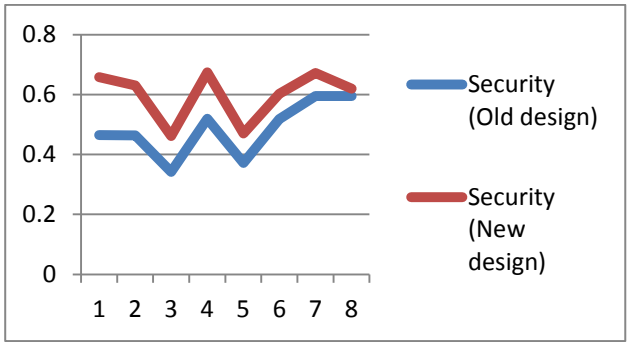

Fig. 4. Graphical Representation of Security Data Before and After Refactoring

Level of Significance of Security: To find out the worth of the difference between the means of old Security values and new security values, the means of both old and new security is calculated as shown in Table 3. Pearson coefficient of correlation comes out to be 0.814. The coefficient shows that the old Security values before treatment and new values of Security after SIG treatment are highly correlated. The degree of freedom for both security values is 7. This test provides the ground for applicability of t-test. The t value comes out to be 2.56. As the value exceeds the t critical value of 2.36 for a two tailed test at the 0.05 level for 7 degree of freedom, and the calculated p value is 0.02 which is < 0.05, thus the null hypothesis H_{01} is strongly rejected and the alternate hypothesis H_{11} is accepted.

Table 3. t-Test for Security Improvement Data Analysis

t- Test for Security							
	Mean	**Std. Deviation**	**Std. Error**	**No. of Samples**	**Pearson Coef.**	**Degree of Freedom**	**t- Values**
Security _(Old Values)	0.483	0.093	0.0328	8	0.814	7	2.56
Security _ (New Values)	0.598	0.085	0.0303				

VII. Conclusion

Validation of any new approach is directly linked to its acceptance by society or industry. It is the validation which demonstrates the usefulness of the approach in society or in industry. For testing the usefulness of the framework security quantification of an object oriented design and SIG, a systematic validation was carried out. As a primary step, empirical validation was carried out. Empirical validation involves pre tryout and tryout. Pre tryout encompasses a small set of data whereas tryout involves a larger set. The pre tryout was carried out on an object oriented design; the tryout was carried out on eight different designs. The designs were initially analyzed and the models being used to compute the values for security attributes. Again the designs treated by SIG and again the model used to compute security. The security values for pre treatment and post treatment were undergone statistical analysis to establish the fact that SIG treatment has successfully improved security. The t-test was carried out and it was found that the t-values obtained by computation performed on old and new security values were exceeding the t-critical values. Hence, the null hypothesis formulated at the beginning of statistical analysis rejected one by one and alternative hypothesis were accepted. Our claim that SIG are able to improve security of object oriented design proved true.

References

[1] J. Bansia, G.C. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transactions on Software Engineering, Vol. 28, No. 1, pp. 4-17, 2002.

[2] C R Kothari, Research Methodology: Methods and Techniques, Published by New Age International (P) Ltd, ISBN (13) : 978-81-224-2488-1, 1990.

[3] L. Tokuda, D. Batory, "Evolving Object-Oriented Designs with Refactoring", Department of Computer Sciences, University of Texas at Austin, Automated Software Engineering, Kluwer Academic Publishers, pp:89-120, 2001

[4] D. M. Coleman, D. Ash, B. Lowther, P. W. Oman, "Using Metrics to Evaluate Software System Maintainability", IEEE Computer, Vol. 27, No. 8, pp. 44–49, August 1994.

[5] S. Demeyer, S. Ducasse, O. Nierstrasz, "Object-Oriented Reengineering Patterns", Morgan Kaufmann and DPunkt, 2002.

[6] W. G. Griswold, D. Notkin, "Automated Assistance for Program Restructuring", Trans. Software Engineering and Methodology, ACM., Vol. 2, No. 3, pp. 228–269, July 1993.

[7] E. J. Chikofsky, J. H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy", IEEE Software, Vol. 7, No. 1, pp. 13–17, 1990.

[8] B. Alshammari, C. Fidge, D. Corney, "Security Assessment of Code Refactoring Rules", In Proceedings of WIAR-2012, Saudi Arabia, web address: http://eprints.qut.au/56382/, 2012.

[9] R. Moser, P. Abrahamsson, W. Pedrycz, A. Sillitti, G. Succi, "A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team", In Balancing Agility and Formalism in Software Engineering, Lecture Notes In Computer Science, (5082), Springer-Verlag, Berlin, Heidelberg, pp. 252-266, 2008.

[10] T. Mens, T. Tourwe, "A Survey of Software Refactoring", IEEE Transactions on Software Engineering, 30(2), pp. 126–139, 2004.

[11] B.D. Bois, T. Mens, "Describing the Impact of Refactoring on Internal Program Qualit", Proceedings of the International Workshop on Evolution of Large-scale

Industrial Software Applications (ELISA), Amsterdam, The Netherlands, pp. 37–48, 2003.
[12] Y. Kataoka, T. Imai, H. Andou, T. Fukaya, "A Quantitative Evaluation of Maintainability Enhancement by Refactoring", Proceedings of the International Conference on Software Maintenance (ICSM.02), pp. 576–585, 2002.
[13] R. Shatnawi, W. Li, "An Empirical Assessment of Refactoring Impact on Software Quality Using a Hierarchical Quality Model", International Journal of Software Engineering and its Applications, Vol. 5, No. 4, October, 2011, pp:127-149.
[14] M. Fowler, Refactoring: Improving the Design of Existing Programs, Addison-Wesley, 1999.
[15] B. Alshammari, C. J. Fidge, D. Corney, "Assessing the Impact of Refactoring on Security-Critical Object-Oriented Design", Proceedings of the Seventeenth Asia Pacific Software Engineering Conference, Sydney, 30 November-3 December (J. Han and T. D. Thu, eds.), (Los Alamitos, CA, USA), IEEE Computer Society, pp. 186–195, 2010.
[16] K. Maruyama, "Secure Refactoring Improving the Security Level of Existing Code", Proceedings of the Second International Conference on Software and Data Technologies (ICSOFT 2007), (Barcelona, Spain), pp. 222–229, 2007.
[17] M. Hafiz, "Security on Demand", PhD thesis, Graduate College of the University of Illinois at Urbana-Champaign, 2010.
[18] S. F. Smith, M. Thober, "Refactoring Programs to Secure Information Flows", Proceedings of the 2006 Workshop on Programming Languages and Analysis for Security, (Ontario, Canada), ACM, pp:75-84, 2006.
[19] B.D. Bois, S. Demeyer, J. Verelst, "Refactoring–Improving Coupling and Cohesion of Existing Code", Belgian Symposium on Software Restructuring, Gent, Belgium, pp. 144–151, 2005.
[20] J. Ratzinger, M. Fischer, H. Gall, "Improving Evolvability through Refactoring", Proceedings of the 2nd International Workshop on Mining Software Repositories (MSR'05), pp: 1–5, 2005.
[21] R. Moser, A. Sillitti, P. Abrahamsson, G. Succi, "Does Refactoring Improve Reusability?", Lecture Notes in Computer Science, 9th International Conference on Software Reuse, pp. 287–297, 2006.
[22] M. Alshayeb, "Empirical Investigation of Refactoring Effect on Software Quality", Information and Software Technology, 51 (9), pp. 1319–1326, 2009.
[23] F. Dandashi, D.C. Rine, "A Method for Assessing the Reusability of Object-Oriented Code Using A Validated Set of Automated Measurements", Proceedings of 17th ACM Symposium on Applied Computing, pp. 997–1003, 2002.
[24] K. Maruyama, K. Tokoda, "Security-aware refactoring alerting its impact on code vulnerabilities" , Proceedings of the 15th Asia-Pacific Software Engineering Conference (APSEC 2008), IEEE Computer Society-1488052 445-452, 2008.
[25] S A Khan, R A Khan, "Security Quantification Model", International Journal of Software Engineering, ISSN: 2090-1801, Volume 6, No. 2, pp: 75-89, 2013.

Authors' Profiles

Suhel Ahmad Khan is pursing PhD in Information Technology from Babasaheb Bhimrao Ambedkar University (A Central University), Vidya Vihar, Raebareli Road, Lucknow. He has been completed his MCA degree from Uttar Pradesh Technical University, Lucknow. Mr. Khan is young, energetic research fellow and has completed a Full Time Major Research Project funded by University Grants Commission, New Delhi. He has more than 5 year of teaching & research experience. He is currently working in the area of Software Security and Security Testing. He has also published & presented papers in refereed journals and conferences. He is a member of IACSIT, UACEE, and Internet Society.

Dr. Raees A. Khan has earned his doctoral degrees from JMI, New Delhi, India and he is currently working as an Associate Professor and Head in the Department of Information Technology, Babasaheb Bhimrao Ambedkar University (A Central University), Lucknow, India. His area of interest is Software Security, Software Quality and Software Testing. He has published a number of National and International books, research papers, reviews and chapters on software quality and software testing.

How to cite this paper: Suhel Ahmad Khan, Raees Ahmad Khan,"Security Improvement of Object Oriented Design using Refactoring Rules", IJMECS, vol.7, no.2, pp.24-31 2015.DOI: 10.5815/ijmecs.2015.02.04

