Development of a Change Impact Model for Regression Testing using UML Diagrams

Jayaprakash Krishnan, Gunasekar Subramani, Sapna P.G.
Department of Computer Technology and Applications
Coimbatore Institute of Technology
Coimbatore, India

ABSTRACT

Regression testing involves testing not only the functionality containing a defect but also testing related functionality to check if a change has introduce side effects. In order to check for the above, a change impact model has been developed using the artifacts built for the software during the design phase. Using both static and dynamic diagrams of UML, it is possible to understand the effect of a change. Also, this serves as a mechanism to ensure customer requirements are satisfied. In this paper, a fine-grained assessment of system change is given at the activity, class and functionality (i.e. use case) level using design models. When a defect is notified, first the activity or method containing the defect is identified. The impact of the defect on other methods is calculated and risk level assigned. Further, the same is used to calculate risk level at the class level and then at the use case level. An indication of the level of risk the method, class and use case incurs due to the change is indicated aiding in selection of regression tests.

Keywords

Regression Testing, UML, Change Impact Model, Risk, Ripple Effect

1. INTRODUCTION

Regression testing involves re(verifying) software to ensure that a change has not introduced defects. Thus, regression testing involves not only retesting part of the code that had defect(s) but also parts of the code that might get affected due to changes. However, re-execution of all test cases during regression testing is not possible given the constraints of cost, time and resources. In such a situation, it would be beneficial to identify parts of code that could be affected and run test cases related to them, thereby maximizing chance of detecting defects. Also, reducing the number of regression test cases is an important factor that ensures reduced cost.

The main objective of building an impact model is to identify parts of code that have a high chance of defects and hence require rerun of test cases. This approach is called safe regression testing [6] as it involves exercising all test cases that may reveal a defect. To achieve the above, there is need to classify methods, classes and use cases in a manner that aids testing by priority. Classes and methods are classified according to their relationship with the affected class and method, respectively, as follows :
(1) High : Class (or) method has a high chance or influence of having a defect and has to be tested definitely.
(2) Medium : Chance of defect in a class B being introduced due to defect in class A is medium.
(3) Low : The chance of the class being affected due to a defect is low and hence testing of such classes can be accorded low priority.

Regression testing is usually based on control flow or data flow obtained from the source code. Based on the Control Flow Graph(CFG), changes at source code level are identified and their impact on other parts of code assessed. Thus, this approach is code based. An alternative would be to use the analysis and design artifacts to understand the influence of a change. In this approach, a change is analyzed from a higher level perspective. Analysis is done to understand the impact of a change on design by studying methods, classes and functionality that may be influenced or affected by the change and defining a magnitude level for the methods, classes and use cases. The assumption made in such an approach is that changes in design during coding are reflected in the analysis and design document thereby maintaining traceability between design and development stages. The advantage of an analysis and design based approach is that changes can be analysed better in terms of impact at the design level and an overall perspective obtained to decide on change requests.

A second advantage is that the approach aids in test case prioritization and selection as it identifies parts of code that may be affected. This aids in selecting test cases that need to be rerun and those test cases that can be removed from the test suite as they do not hold validity in the new context. By identifying a subset of test cases to be rerun, the cost of regression testing is kept low.

In this work, the focus is on introducing a change impact model using design artifacts (Unified Modeling Language) as basis, thereby aiding the regression testing process. The Unified Modeling Language (UML) has been chosen as it is becoming the de-facto standard used in the industry for analysis/design[1].

Thus, the motivation for the current work are as follows :
a. Estimation of regression test effort is a crucial factor. To aid this activity, UML diagrams are used to identify methods and classes that will need to be tested when a change is made.
b. Analysis of interactions both static and dynamic can be done using UML diagrams.
c. Enabling managers to plan and allocate resources for testing activities at an early stage.
d. Ripple effect of a change can be analyzed and hence testing can be planned accordingly.

The paper is organized as follows: Section 2 gives the background study in the area of regression testing. Section 3 discusses related work in the area. Section 4 gives details of our approach on determining causes of defect and identifying methods and classes that need to be tested. Results of the work are discussed in section 5. Conclusion is given in section 6 along with the issues we intend to handle in future.

2. DETERMINING IMPACT OF CHANGE

2.1 Regression Testing

Regression testing is done to test a new version of software in order to verify that existing functionalities have not been affected by addition or change to system feature[5]. Regression testing cannot be done exhaustively due to constraints of cost, time and resource availability. The question to be asked is, ’Should all test cases be re-executed each time a change is affected to a program?’. Running all test cases is not an effective strategy as it might include obsolete test cases as well as test cases related to functionality not affected by the change. Thus, a retest all strategy assumes that a change might introduce faults anywhere in the program and hence needs to be tested completely. Thus, the cost of testing here is equal to or exceeds the cost of system testing which is not feasible. An alternative approach is to reduce testing to those parts of the system that might have been influenced//compromised due to the change. Such an approach is called a selective retest strategy. In this approach, an analysis is done to identify parts of the code that might have been compromised due to a change. Here, a subset of test cases are executed with the effect that the defects unearthed
will be the same as using the entire test set.

Several approaches to regression testing can be found in literature, wherein various software artifacts are used for selecting test cases. Some techniques use requirements, while others use design artifacts. Most work in literature use code as the basis for regression testing. In the case of requirements and design based approach, the changes are identified at a higher level of abstraction whereas the code based approach aids in identifying defects at a lower level of abstraction i.e. statement/method level. The methodology followed is to obtain a control flow graph of the code before and after the change and compare them in order to find those parts of the code that have changed. However, such an approach is confined to the coding phase and provides change impact information at a statement level.

In this work, an approach to identifying change impact and regression testing is evolved at the design level. The advantage of such an approach is that aspects of design along with the code can be considered providing a better approach to regression testing. Kung et al in their work list five phases for the regression testing process. They are :

(1) identification of changed classes
(2) identification of affected classes
(3) generation of an order for testing classes
(4) selection of test cases
(5) test case modification and generation

The current work looks at a design based approach in order to minimize testing effort at the same time maximizing the chance of detecting faults. The work proposed in this paper concentrates effort on steps 2 and 3 considering as input a set of changed classes. Given a set of defects found, the approach identifies affected methods, lists methods, classes and use cases by an order of priority by which they have to be tested and also lists functionalities (scenarios) to be tested.

3. RELATED WORK

Regression testing strategies work at two levels : white-box and black-box. A major part of regression testing use white-box testing strategies [2] and perform code analysis. Most regression testing techniques are code-based, i.e., these techniques select test cases using the source code of the original and modified programs [11][12][13][14][15]. Specification-based regression testing techniques[4] use specification as basis for regression testing. Most of these techniques select regression tests using only the modified system specification. However, it has been shown that code based testing and specification-based testing complement each other[4]. Most code based regression testing techniques thus use a call graph, control flow graph and the dependence graphs as the basis. The advantage of using white-box testing strategies is that the approach is fine-grained viz. changes are monitored at the code level. However, a disadvantage is the loss of functional perspective.

A second approach is the black-box approach where changes are monitored at the functional level. Here, functionality that can be affected are studied and the same used for regression testing[4]. Change impact model has been used for predicting the effect of change in [6][7][8][9][10]. The change impact model has primarily been used around code as well as design[10]. Briand et al[4] has used the class diagram and sequence diagram to build an impact model and select test cases for regression testing.

4. APPROACH TO REGRESSION TESTING

4.1 Change Impact Model

A change impact analysis is defined as determining the set of locations in the software that may be affected due to change. To build the change impact model, three types of UML diagrams have been considered: use case diagrams, class diagrams and activity diagrams. A use case represents a functionality and is elaborated using activity diagrams. A class represents the structure for similar objects and is defined as a set of attributes and methods. Classes are related to one another based on the following types of relationships: association, dependency, aggregation and inheritance. Composition relationship is considered along with aggregation as it is a strong form of aggregation.

An example of the relationships between classes is discussed through a class diagram for a library management system as shown in Figure 1. The relationship between the classes ’Librarian’ and ’Book’ is an association type of relationship. In case of association, a class uses the methods of another class to provide a service.

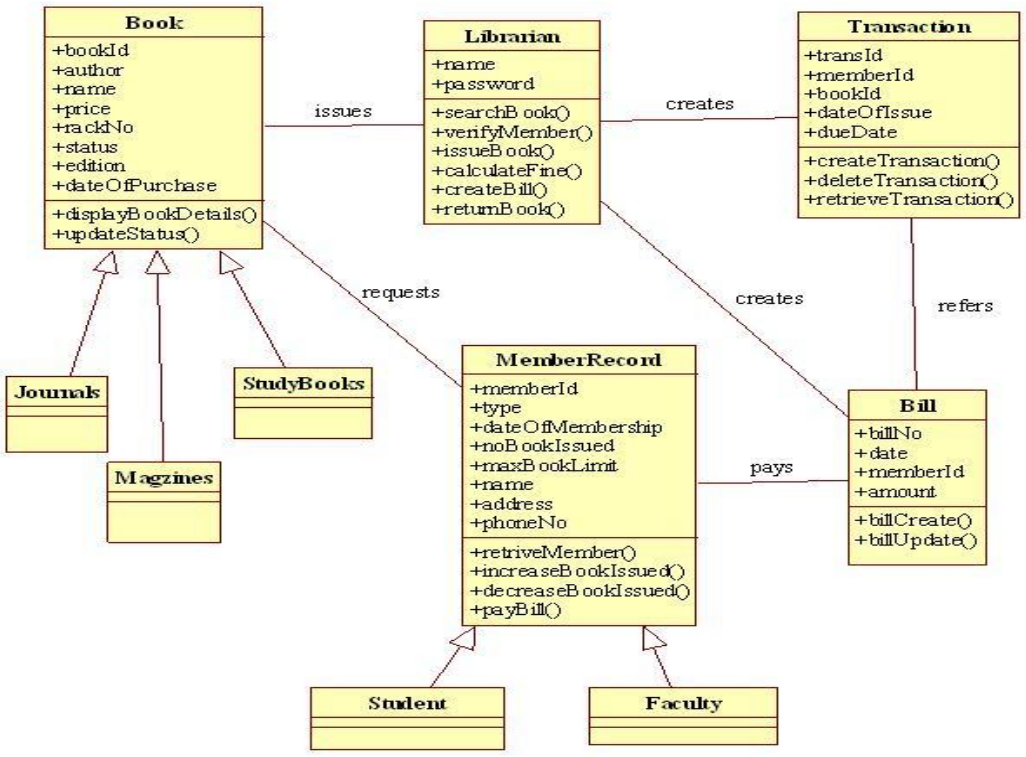


Fig. 1. Class diagram for library management system

The relationship between the classes ’Journals’ and ’Book’ is an inheritance relationship. Here, ’Journals’ is considered as a type of ’Book’. Therefore, the attributes and methods of the class ’Book’ get inherited by the class ’Journals’. An aggregation is a relationship where an object of one class becomes a member(i.e. attribute) of another class. Thus, in all of the mentioned relationships, it can be inferred that a change in one class will affect the interacting classes. A change to a class can be a change to one or more attributes or to a method. Examples of change includes: adding an attribute, deleting an attribute, changing a method’s scope and changing the relationship between classes.

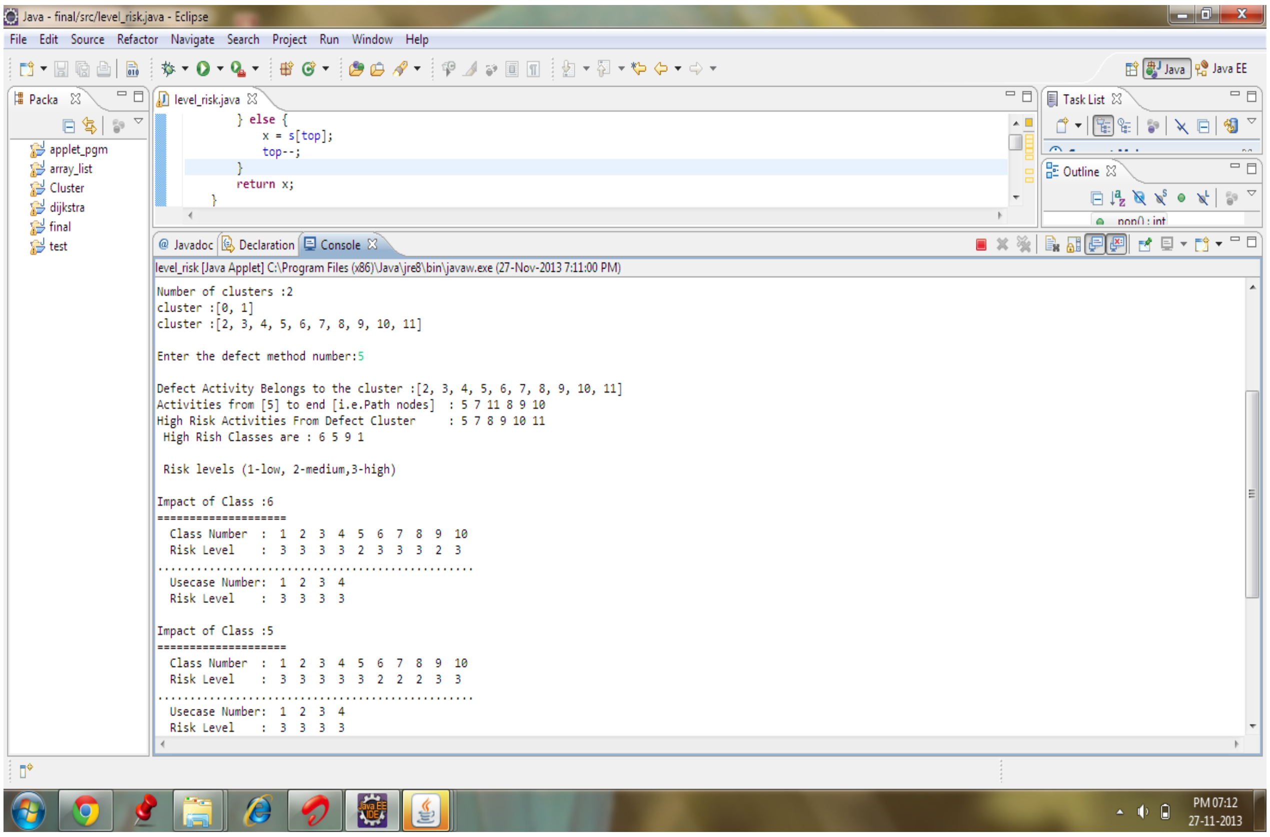
Activity diagrams elaborate a scenario in a use case diagram. Activity diagrams are dynamic in nature and show the interactions between objects of the system through method/message interactions. Method changes can be tracked from the activity diagrams. In an activity diagram, the type of construct a method that is being changed belongs to influences the methods that are part of the construct. For example, a change in an attribute type influences all methods that use the attribute. Similarly, a change in the predicate node of an activity diagram influences the branch nodes. Hence, the impact of a change at the method level can be assessed.

Use case diagrams represent functionality of a system with reference to users. Thus, activity diagrams elaborate use cases. Through the activity diagram and the class diagram, it is possible to define the methods and classes that can be affected due to a change. A set of classes can be considered as contributing to meeting a functionality of the system. Hence, a change in a class requires that functionality met by the class be re-tested. The above considerations are used to develop the impact model.

4.2 Ripple Effect

A ripple effect is a situation where, when a change is effected to a part of the system, the change is propagated to other parts of the system. ‘Ripple effect’\ was introduced by Haney and is a major problem due to dependencies that exist between classes, modules and subsystems. Consider the following example. A test case has thrown a defect and it has been identified in Class B. However, the cause of the defect maybe either in Class B or maybe due to the dependencies that exist between Class B and other classes. Further, a change made to fix the defect in Class B may influence other parts of the code i.e. interacting classes like the expanding ripples across water when an object is dropped into it. The influence of the defect reduces as the ripples expand outwards. The same can be followed when considering defects in a system. Based on the above, a set of rules can be evolved to understand the impact of a given change.

Method Level :
a. If Method a() has a defect and is part of a sequence, then all outgoing message interactions from Method a() have a high risk factor.


Java - final/src/level_risk.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Packa
applet_pgm
array_list
Cluster
dijkstra
final
test
level_risk.java
Task List
Outline
Javadoc Declaration Console
Java
Java EE
PM 07:12
27-11-2013

```
            } else {
                x = s[top];
                top--;
            }
            return x;
        }
```

```
level_risk [Java Applet] C:\Program Files (x86)\Java\jre8\bin\javaw.exe (27-Nov-2013 7:11:00 PM)
Number of clusters :2
cluster :[0, 1]
cluster :[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Enter the defect method number:5

Defect Activity Belongs to the cluster :[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
Activities from [5] to end [i.e.Path nodes]  : 5 7 11 8 9 10
High Risk Activities From Defect Cluster     : 5 7 8 9 10 11
 High Rish Classes are : 6 5 9 1

 Risk levels (1-low, 2-medium,3-high)

Impact of Class :6
====================
  Class Number  :  1  2  3  4  5  6  7  8  9  10
  Risk Level    :  3  3  3  3  2  3  3  3  2  3
.................................................
  Usecase Number:  1  2  3  4
  Risk Level    :  3  3  3  3

Impact of Class :5
====================
  Class Number  :  1  2  3  4  5  6  7  8  9  10
  Risk Level    :  3  3  3  3  3  2  2  2  3  3
.................................................
  Usecase Number:  1  2  3  4
  Risk Level    :  3  3  3  3
```

Fig. 2. Snapshot showing identification of clusters and determining class impact

b. If Method a() is a predicate, then the entire condition has a high risk of having a defect.
c. If Method a() is at the head of a fork-join construct, then all methods part of the fork-join construct have a high risk of having a defect.

Class Level :
a. If Class A has a defect, then all classes directly associated with Class A have a high risk of defect. From Figure 1, Class B has a high chance of having a defect, given that Class A has a defect. However, classes associated with Class B do not have a high risk of defects.
b. If Class A has a defect, and class B is connected to class A through a dependency relationship i.e. class B is dependent on class A, then class A has a high risk factor and class B also has a high risk factor as class B is dependent on class A to provide a service.
c. If Class B has a defect, and Class B inherits from Class A, then Class A has a medium risk factor. However, Class C that inherits from Class B has a high risk factor.
d. If Class A aggregates Class B, and Class A has a defect, then the chance of Class B having a defect is medium. However, if Class B has a defect, then Class A has a high risk factor. Aggregation includes composition relationship as composition is a strong form of aggregation.

Functionality Level :
a. All scenarios, both valid and invalid related to a class having a defect has a high risk.
b. All scenarios, both valid and invalid related to a method having a defect has a high risk.

Thus, the ripple effect can be calculated at different levels of granularity, namely, method, class and functionality level. Steps involved in determining risk with respect to each activity are given below.
1. First, edges in an activity diagram are given weights [16].
$Wt(e) = (n_i)_{in} X(n_j)_{out}$
where $(n_i)_{in}$ is the number of incoming dependencies of node n_i and $(n_j)_{out}$ is the number of outgoing dependencies of node n_j and e is the edge connecting n_i and n_j .
2. The distance between any two edges are stored in an adjacency matrix.
3. The Nearest Neighbour algorithm is used to form clusters by determining the nearest neighbour using the adjacency matrix.
4. Using the node that has a defect, identify corresponding cluster. Based on rules defined at the 'Method Level' assign risk values to each activity node.
Figure 3 shows the ripple effect calculated at the activity i.e. method level. Using this, further, impact at the class and functionality level will be calculated. The steps involved in developing the change impact model is shown in algorithm given in Section 4.3.

4.3 Algorithm

Algorithm 1 below details the steps involved in identifying methods, classes and use cases with risk. For the system under study, UML diagrams, namely, use case, class and activity diagram are given. Also, the lists of defects identified during testing are listed. For each defect the corresponding location, here, activity is identified. To rectify a defect, changes are made in code. A change can introduce new defects through change in related activities or classes. To identify the impact of a change, a change impact model should consider the above. In the proposed approach, first risk level is calculated at the activity level by using a clustering technique (Nearest Neighbour Algorithm) and classified as high/medium/low. Further, the impact on classes and use cases is calculated.

Algorithm 1 ImpactCalc()

```
Input : UML diagrams for system under consideration
        List of defects identified.
Output : List of methods, classes and functionalities that need to be retested along with risk factor

REPEAT
a. Identify method subject to change
b. For each method under change, identify impact of methods incident upon
Assess impact as high, medium, low

c. Identify class w.r.t. method under change
Assess relationship of class under change with associated classes
Assess impact as high, medium, low

d. Identify use cases w.r.t class undergoing change
Assess impact of scenario as high, medium, low

For each of the steps b,c,d
Identify related test cases and add to Test Suite
UNTIL all methods subject to change considered
```

5. RESULTS

Result of the work is shown in Figure 3 and Figure 4. In Figure 3, each activity along with corresponding risk level. Risk is categorized as high (level 3), medium (level 2) and low (level 1). Similarly, at the class level, for each class, risk level is given. Methods and classes with risk level termed ’high’ have a high chance of defects and hence need to be given priority during testing. Methods and classes with risk level ’low’ are not affected by the defect and hence have a low chance of regression defects. Hence, resources in terms of cost, time and manpower allocated to them can be reduced and effort can be concentrated on parts of code that have ‘high’ and ‘medium’ risk level. The impact model thus provides input to the managers to plan regression testing in an efficiency manner optimizing use of cost, time and resources.

6. CONCLUSION AND FUTURE WORK

In this work, a change impact model has been defined. The technique considers the impact a change has on methods, classes and functionalities of the system. A ripple effect model has been used to identify classes that are affected according to their distance from the affected class. The approach can be used in regression testing as it identifies a subset of test cases that needs to be re-run instead of the complete set.

Work done to date looks at design based regression testing. However, such an approach has its limitations as it lacks precision due to the high level of abstraction. To overcome this drawback, a hybrid approach that involves combining design and code based approach will be investigated. Secondly, the present work used a simple case study to understand the working of the technique. Ongoing work involves using a larger case study to determine the efficiency of the technique. Third, the results of the evaluation are shown in textual form. Future work will involve developing a visualization tool to make it easy for users to understand and comprehend the impact of change.

7. REFERENCES

[1] OMG. Unified Modeling Language (UML) Superstructure Specification, version 2.1. Technical report.

[2] Robert V. Binder. Testing Object-Oriented Systems Models, Patterns, and Tools. Addison Wesley, 1999.

[3] L. Briand and Y. Labiche. A UML-based approach to system testing. Software Systems Modeling, 1(1), 2002.

[4] L.C. Briand, Y. Labiche, G. Soccar. Automating Impact Analysis and Regression Test Selection based on UML Designs. Proceedings of the International Conference on Software Maintenance (ICSM ’02), IEEE Computer Soceity, pp.1-10, 2002.

[5] M. J. Harrold, J. A. Jones, T. Li, D. Liang, A. Orso, M.Pennings, S. Sinha and S. A. Spoon, Regression Test Selection for Java Software, Proc. ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA’01), 2001.

[6] Abdi, M. K.; Lounis, H.; Sahraoui, H., ”Predicting Change Impact in Object-Oriented Applications with Bayesian Networks,” Computer Software and Applications Conference, 2009. COMPSAC ’09. 33rd Annual IEEE International , vol.1, no., pp.234,239, 20-24 July 2009.

[7] M.K. Abdi, H. Lounis, H. Sahraoui: ”Analyzing Change Impact in Object-Oriented Systems ” In proceedings of the 32nd EUROMICRO Software Engineering and Advanced Applications Conference, Cavtat/Dubrovnik (Croatia), August 29-Sept 1, 2006.

[8] Lee, M.; Offutt, A.J.; Alexander, R.T., ”Algorithmic analysis of the impacts of changes to object-oriented software,” Proceedings. 34th International Conference on Technology of Object-Oriented Languages and Systems, 2000. TOOLS Vol. 34, pp.61-70, 2000.

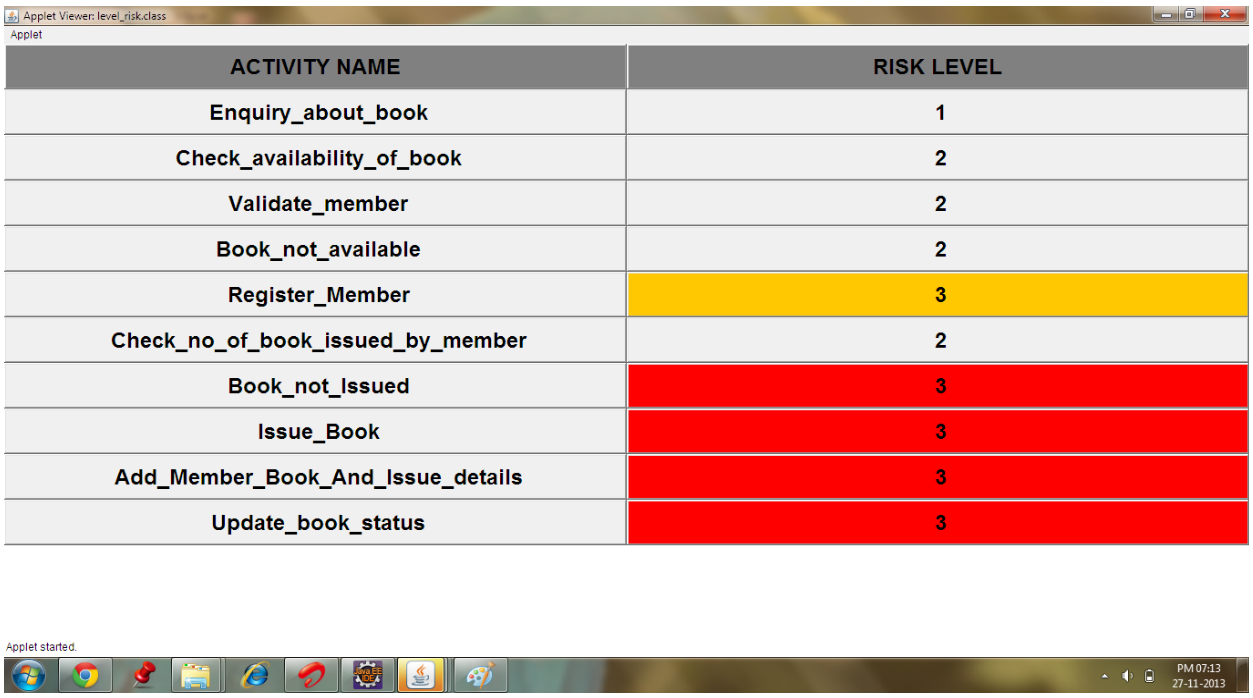
[9] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, JRipples: A tool for program comprehension during incremental change, in Proceedings of the 13th International Workshop on Program Comprehension (IWPC 05), pp. 149151, May 2005

[10] M. Hammad, M. L. Collard, and J. I. Maletic, Automatically identifying changes that impact code-to-design traceability, in Proceedings of the IEEE 17th International Conference on Program Comprehension (ICPC 09), Vancouver, BC, pp. 2029, May 2009.

[11] Baradhi, G., Mansour, N., ”A Comparative Study of Five Regression Testing Algorithms,” Proceedings of the IEEE Australian Software Engineering Conference, pp. 174-182, 1997.

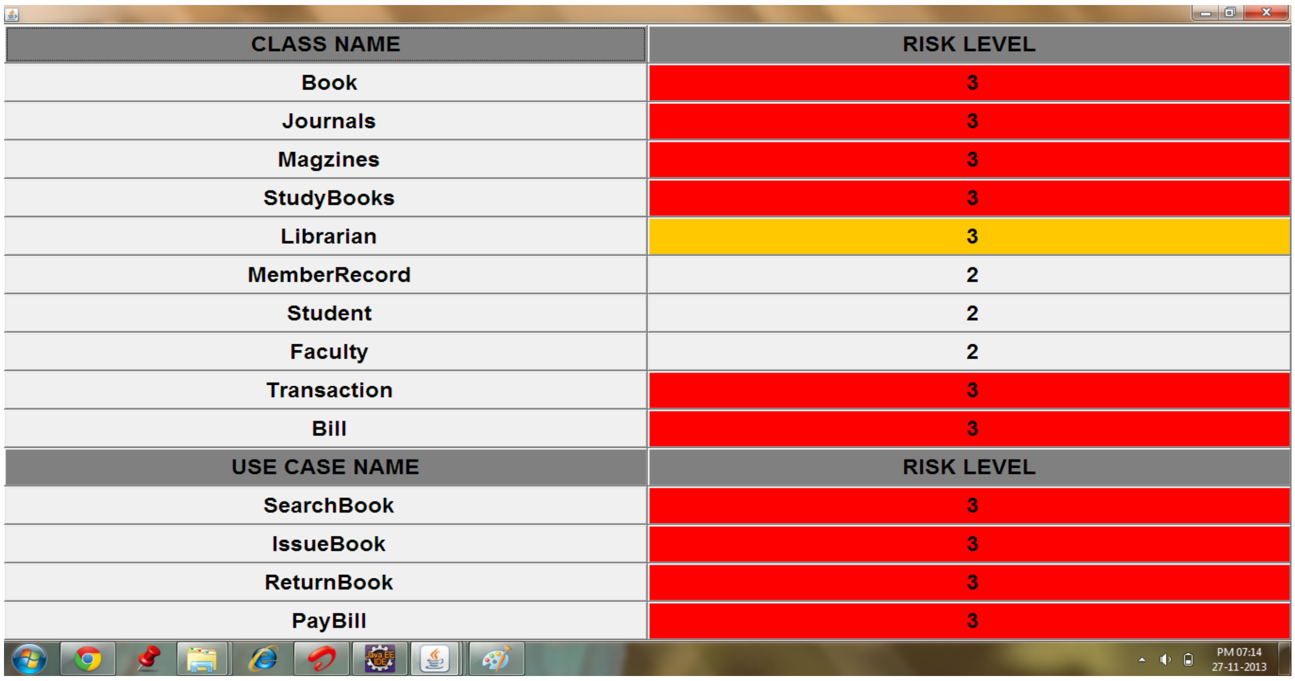
[12] Beydeda, S., Gruhn, V., ”An Integrated Testing Technique for Component-Based Software,” Proceedings of the ACS/IEEE Computer Systems and Applications International Conference, pp. 328 -334, 2001.

[13] Gupta, R., Harrold, M., Soffa, M., ”An Approach to Regression Testing Using Slices,” Proceedings of the IEEE International Conference on Software Maintenance, pp. 299-308, 1992.



ACTIVITY NAME	RISK LEVEL
Enquiry_about_book	1
Check_availability_of_book	2
Validate_member	2
Book_not_available	2
Register_Member	3
Check_no_of_book_issued_by_member	2
Book_not_Issued	3
Issue_Book	3
Add_Member_Book_And_Issue_details	3
Update_book_status	3

Fig. 3. Snapshot showing risk level calculated for each activity



CLASS NAME	RISK LEVEL
Book	3
Journals	3
Magzines	3
StudyBooks	3
Librarian	3
MemberRecord	2
Student	2
Faculty	2
Transaction	3
Bill	3
USE CASE NAME	**RISK LEVEL**
SearchBook	3
IssueBook	3
ReturnBook	3
PayBill	3

Fig. 4. Snapshot showing risk level calculated for each class

[14] Harrold M., Soffa M., 'An Incremental Approach to Unit Testing during Maintenance', Proceedings of the IEEE International Conference on Software Maintenance, pp. 362-367, 1998.

[15] Korel, B., AI-Yami, A.,"Automated Regression Test Generation,"Proceedings of IEEE International Symposium on Software Testing and Analysis, pp. 143-152, 1998.

[16] Sapna P.G., Hrushikesha Mohanty, "Prioritization of Scenarios based on UML Activity Diagrams", First International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), India, pp. 271-276, IEEE Computer Society, 2009.