CORFOOS: Cost Reduction Framework for Object Oriented System

Vedpal, Naresh Chauhan

Department of Computer Engineering, YMCA University of Science & Technology, Faridabad, India

Email: ved_ymca@yahoo.co.in, nareshchauhan19@gmail.com

Abstract

There are many constraints in developing software for an organization, such as time and budget. Due to these constraints and intricacies of advanced software development technology, it has become very challenging to complete such projects. To make these projects cost-effective, this paper presents a cost reduction framework (CORFOOS) which works at three levels. At the first level, Intermediate Requirement Dependency Value (IRDV) of each requirement is determined by creating the intermediate requirements dependency graph (IRDG). At the second level, the requirements are categorised and finally at the third level, the testing parameters are determined by analyzing the requirements. To analyze the requirements, the dependency model, interaction model, language specification model and fault model are used.

Keywords: *Testing of object oriented software, cost reduction, testing efforts, software testing, framework for cost reduction*

INTRODUCTION

Software development is very challenging now days due to advancement in technologies. Availability of different software with multiple features has raised the expectations of users. In order to satisfy the client expectations, developers need to incorporate complex software enhancements. Nowadays, many people have got used to the convenience of using online applications for their routine work like shopping, money transfer, ticket booking, etc.

So, due to complexity of software needed to satisfy different requirements of the user, testing of software has also become quite complex. Effective software testing consumes more resources including time and increases the overall cost of software development. Various researchers have presented many techniques for reduction of testing- cost. The studies show that if the faults are not fixed in their early phase, more cost is incurred to fix the faults in the later phases. Software maintenance phase is an expensive phase as it incurs an approximate 60% of the total cost of software development.

The researchers showed that regression testing takes almost 80% of the budget allocated for testing and up to 50% of the budget for software maintenance [1]. So, for reduction of testing cost the software must be developed in a way that it is open for extension but closed for modification, and there is less chance of changes in requirement so that the resultant alterations do not push the cost up. The various constraints in software development that need to be factored in for controlling costs are budget, time, quality, risk etc.

According to finding of sixth world quality report, average spending on QA as a percentage of the total IT budget has risen from 18% in 2012 and 23% in 2013 to 26% in 2014 [2]. The share of testing budget is expected to reach 29% by 2017. Due to increase of testing cost in software development, there is a need for a technique or a framework for reduction of testing cost. With that objective, a cost reduction framework for object oriented software is presented in this paper. This framework works at three levels. In the first level, an analysis of requirements is performed and the dependency values of all the requirements are determined. In the second level, proposed requirements are mapped with past implemented requirements. In the third level, the testing parameters of requirements are determined by analysing the requirements using models.

RELATED WORK

Samaila Musa *et al.* presented a framework for regression testing of object oriented software [3]. They used the System dependency graph model to detect changes in the method of a program which occurs due to data dependency, control dependency and dependency caused by object relations. For verification of any statement, slicing is performed on a constructed graph.

Yves Le Traon *et al.* presented a methodology for integration planning and regression testing of objects oriented software [4]. For the purpose of regression testing and integration, the classes are ordered on the basis of the proposed model.

Recardo Terra and Macro Tulio Valente presented domain specific language to restrict the spectrum of dependencies that are allowed in object oriented system [5]. They also explained a checking tool. The violations of proposed constraints are detected by this tool.

Sunil L. Bangare *et al.* proposed a metric for object oriented software for measuring the quality of modularization [6].

Michela Pedroni *et al.* analyzed the dependency structure of the object oriented concept [7]. By an analysis of the dependency structure, they found that basic object oriented concepts are tightly interrelated.

Ranjita kumara swain *et al.* proposed an approach for generating the test data [8]. They first created the transition graph from the state chart diagram. The test cases are generated by extracting the required information from the state chart.

Amaranth Singh *et al.* presented a metric which helps to identify the critical elements [9]. They used intermediate graph representation of a program. The influence of class is found out through a forward slice of the graph.

Xiaolan Wang *et al.* presented a method to make dependency graph of the error statement [10]. The proposed method is based on the symbolic execution and constraint solving. It can be used in different systems and is able to detect errors in different languages.

R. Krishnamoorthi *et al.* proposed a model for prioritizing the system test cases by considering six factors, which are customer priority, implementation complexity, completeness, traceability, and fault impact [11]. They validated the proposed model with two different validated techniques and experimented with some projects.

Varun Gupta *et al.* presented dynamic cohesion metrics [12]. The introduced metrics provide the scope for measurement of cohesion up to class level. Their analysis was based on application of the proposed dynamic cohesion metrics on 20 Java programs and found that dynamic cohesion metrics are more accurate and useful.

PROPOSED FRAMEWORK

The proposed framework works at four levels. At the first level, requirements are analyzed and a requirement dependency graph is plotted. By using the requirement dependency graph a requirement dependency metric will be created that shows dependency value of requirements. There may be some requirements which are already implemented by the organization. In the second level, all the requirements are mapped with the past implemented requirements. After mapping, requirements will be divided into three categories: partial modified requirements, unmodified requirements and new requirements.

By using requirement dependency metric, dependency of unmodified requirement is determined. If the dependency of unmodified requirements is zero, there is no need to test them. But if the dependency value of requirement is non zero, then suitable testing strategy is required to test the requirements. The test cases are selected from the previously tested cases. In the case of partial modified requirements, an appropriate regression technique is applied to identify the affected part of requirements and for testing of requirements as a whole.

For the new requirements three models are used: dependency model, interaction model and Language specification model. After analysis of these models, complexity of new requirements and the faulty model of requirement are determined. By using the identified complexities and faulty model, the requirements are prioritized and suitable testing strategy is selected as shown in Figure 1.

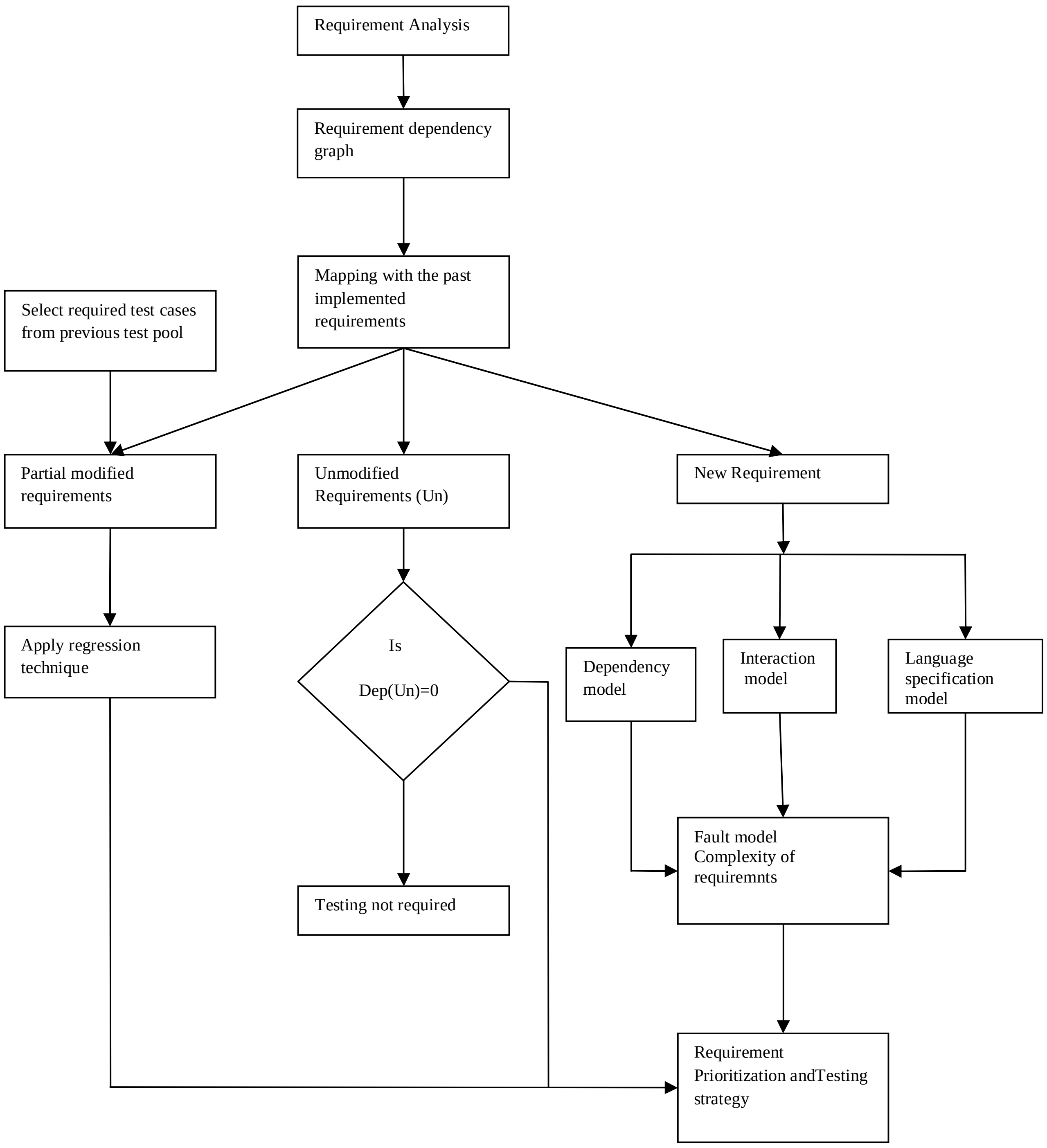

Requirement Analysis
Requirement dependency graph
Mapping with the past implemented requirements
Select required test cases from previous test pool
Partial modified requirements
Unmodified Requirements (Un)
New Requirement
Apply regression technique
Is Dep(Un)=0
Dependency model
Interaction model
Language specification model
Fault model Complexity of requiremnts
Testing not required
Requirement Prioritization andTesting strategy

***Fig.1:** Framework for Cost Reduction.*

REQUIREMENT ANALYSIS AND REQUIREMENT DEPENDENCY GRAPH

In this phase, an analysis of requirements is performed first of all. The analysis of requirements is performed for identifying the purpose of developing the software. After analysing the requirements, an intermediate graph for determining the dependencies between the requirements is constructed. In the intermediate requirement dependency graph (IRDG), the requirements are denoted by the node and dependencies between the requirements are shown by the directional edges. After constructing the IRDG, degree of each node is counted. The degree of each node is the sum of in degree and out degree of a node. This degree of requirements is termed as intermediate requirement dependency value (IRDV). In this way, the intermediate requirement dependency value (IRDV) metric forms using IRDG.

PARTITION OF REQUIREMENTS BY MAPPING THEM WITH PAST IMPLEMENTED REQUIREMENTS

In this phase, the requirements are mapped with past implemented requirements. Mapping is based on functionality and implementation platform of a requirement. After mapping, the requirements are categorised as new, partially modified or unmodified.

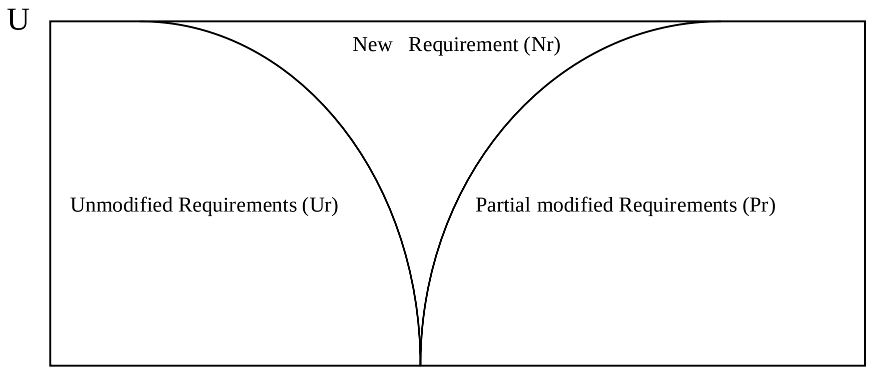
- **New Requirements** are the emerging requirements which are never implemented by the organization.
- **Partially Modified Requirements** are those requirements which were implemented earlier by the organization, but now there is a scope for quite a few changes.
- **Unmodified Requirements** are those requirements that are implemented without any changes.

Let R be a set of requirements, such that

R = {R1, R2, R3, R4, R5, R6, R7, R8, R9}

Set Pr is a set of partial modified requirements, Ur is set of unmodified requirements and Nr is the set of new requirements, such that

Pr= {R1, R4, R5}, Ur = {R5, R8, R9}, Nr = {R2, R3, R6}


U
New Requirement (Nr)
Unmodified Requirements (Ur)
Partial modified Requirements (Pr)

IDENTIFICATION OF CRITICAL REQUIREMENTS

By using the IRDV, any dependency of unmodified requirement is identified. If no dependency of unmodified requirement is found, then there is no need to test them. But if dependency is found, then these requirements are put in a pool of requirements to be tested and mapped with the fault model of past implemented requirements.

COMPLEXITY OF REQUIREMENT

To find out the complexity of requirements, three models namely Dependency model, Interaction model and Language Specification model are used to calculate the testing parameters of requirements. Higher the scale of testing parameters, more are the chances of errors to occur. By using these models, the developer can identify the types of errors that might occur. Testers are able to design test cases and developer can code the requirements on the basis of these test cases as well as expected faults. This type of coding helps the developer to avoid these faults from occurring.

Dependency Model: It helps to detect the structural dependency. Software architects always specify a set of structural constraints for the target system. Source code and related information like classes, sequence diagram and high level modules such as package and component diagram must be analysed by the architects. Analysis of dependency includes the control dependency of the program, data dependency and dependency between the classes, method to class, method to method, polymorphism interdependency, implementation dependency, contractual dependency, dependency of program on external system call, functional dependency, etc. The control dependency covers exception handling, multithreading and synchronization. The data dependency model helps to identify the cohesion of each class and coupling between the classes which helps to determine the complexity of the programme.

Interaction Model: Interaction model is used to identify the different types of interactions presented in the program. As object oriented language provides various features such as inheritance, polymorphism, message passing, and encapsulation, it is complicated and prone to errors. By using the interaction model, different types of interactions between the programs are identified. The interaction model describes the communication between the classes. The classes communicate to each other by passing messages. These messages represent the interaction between the objects. There are various types of messages in object oriented language as shown in Table 1.

Table 1. *Types of Messages and Interaction in Object Oriented Language.*

Message	**Interaction**
Simple Message	Interaction between the classes
Synchronous Message	Interaction between the classes and interface
Asynchronous Message	Interaction between the different objects of the program
Reflexive message	Interaction between the program and native method
Return message	Interaction between the classes and distributed class

Language Specification Model:
Language specification model explains the model of the language used for implementing the requirements. It helps to identify the specified feature of language that is going to be used. Every language has a set of rules to use the various features of the language. If the specified rules for use of feature are not followed, then it will become a source of error. Using this model helps the designer to find out which features should be used to implement the requirements for getting a quality product. The language specification model also shows which feature is prone to error and the steps to follow for using the feature in an efficient and error free manner.

Fault Model: The fault model is used to determine types of faults which are usually found during testing. The fault model shows the types of fault and reason of the faults in the software. By using the fault model, the developer or tester can analyze the software and take the required steps for reducing the faults.

OBJECT ORIENTED DESIGN PRINCIPLE [13]

- ***Single Responsibility Principle*****:** A class should be designed only for a single responsibility because each responsibility is a cause of changes in a class. The classes become large and complex if many responsibilities are handled by a single class. For avoiding this situation, it is mandatory to ensure that the code is simple.
- ***Open Closed Principle*****:** Software entities like classes and modules should be designed in such a way that they are open

for extension and closed for modification. All new functionality should be added in the code by adding a subclass to the existing class without making any change in existing classes.

- ***Liskov Substitution Principle***: The instance of super class is replaced by the instance of the derived class. If this is not followed, the class hierarchies become messy.
- ***Interface Segregation Principle***: The class should depend on the smallest possible interface.
- ***Dependency Inversion Principle***: Modules that implement the high level policy should be dependent on a well-defined interface rather than on modules that implement low level polices.
- ***Principle of Package Cohesion***: If the classes are changed or reused at the same time, only then they should be grouped together, otherwise they should not be grouped together.

Using the abovementioned models helps to identify the testing effort of each requirement by as complexity of the requirements is calculated based on them. More the complexity of the requirement, more the effort required for testing; which increases the cost of testing too. Testing effort of a requirement can be calculated by incorporating the following factors:

1. No. of classes
2. Level of inheritance
3. No. of attributes used in each class
4. No. of methods used
5. No. of native methods used
6. External system call
7. Import of the package and API
8. No. of wrapper classes used
9. Multiple inheritance used
10. Method overloading and method overriding
11. Nested Classes
12. Expected Fault

Testing effort can be calculated by the following formula -

$$\text{Testing effort (TE)} = \sum_{j=1}^{n}(\text{fvalue}_{ij} * \text{fweight}_j) \text{ ----------------------------- (1)}$$

where fvalue is the value assigned to the considered factors and fweight is the weight assigned to the factors, and the weight is assigned based on the criticality of the factor. Factor criticality indicates the probability of error that different factors contribute. More the factor weight more the chances of errors to be introduced by the factors.

The requirements are prioritized and tested based on the calculated testing efforts. Value of testing efforts shows the complexity of the requirement.

RESULT AND ANALYSIS

Due to constraints of resources, the proposed approach is validated by applying it on the following given requirements of a project. The requirements dependency graph is shown in Figure 2.

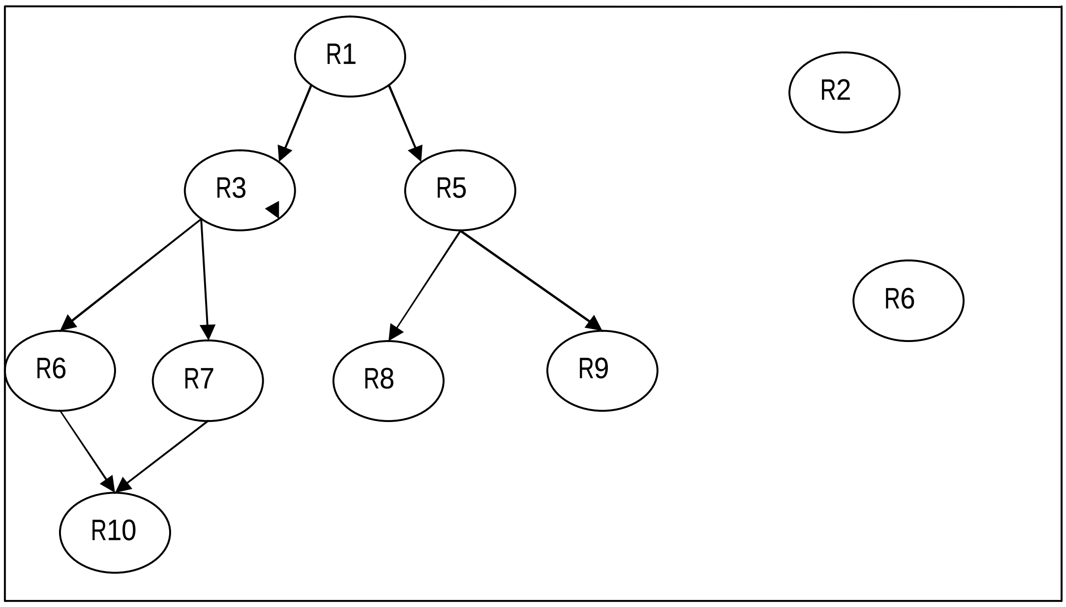


***Fig.2:** Intermediate Requirement Dependency Graph.*

As shown in Figure 2, there are 10 requirements. The requirements R2 and R3 are the independent requirements. The requirements and their dependency value are shown in Table 2.

***Table 2:** Intermediate Requirements Dependency Value.*

S. No.	**Requirements**	**IRDV**
1	R1	7
2	R2	0
3	R3	4
4	R4	0
5	R5	3
6	R6	2
7	R7	2
8	R8	1
9	R9	1

10	R10	2

For validation of the proposed requirements, partition of the requirements are shown below –

Un = R2, R4 Nr = R1, R3, R4, R5, R8 Pr = R6, R9, R10

Since the requirement set Nr is the set of new requirements that are implemented for the first time by the organisation, these should be analysed by applying three models: interaction model, dependency model and language specification model and be prioritized accordingly.

Suppose X be the total cost to test each requirement and Y be the cost incurred in regression testing of the software. Before applying the CORFOOS, total cost to test all the requirements will be 10 X.

After applying the proposed framework, the findings are:

1. The requirements R2 and R4 are the independent requirements so the testing cost of these requirements will be zero. So there is no need to test them.
2. The requirements R6, R9, R10 are partial modified, so cost incurred to test the partial modified requirements will be 3Y.
3. The Requirements R1,R3,R4,R5,R8 are new requirements and so, their testing cost will be 5X

So, after applying CORFOOS, total cost to test all requirements of the projects are 5X + 3Y where Y < X

If we do not apply the framework proposed above, then total cost to test all the requirements will be 10 X, which is greater than the cost estimated by applying the proposed approach, which are 5X – 3Y.

CONCLUSION

The proposed framework (CORFOOS) will work in three levels. At the first level, the Intermediate requirements dependency metric has been created. At the second level, the requirements are divided in three categories. In the third level, the testing efforts of the requirements are calculated and testing involves choosing various testing techniques. The proposed framework is validated by making some assumptions. It will help in reducing the testing cost of the software. When applied on software development projects, this framework can deliver cost-effective and quality work.

REFERENCES

1. G. Rothermel, M,j. Harrold, A safe efficient regression test selection technique," ACM Transactions on

software engineering Methodology 1997; 6(2): 173–210p.

2. Capgemini world quality report 2014 Hester Decouz www.worldqualityreport.com.
3. Samaila Musa, Abu Bakar M. D. Sultan, Azim Abd Ghani, Dr. Salmi Bahrom. "Regressiong testing framework based on extended system Dependence graph for object oriented programs" Proc. Of the intl. Conf. On advances in computer science & electronics engineering – CSEE2014.
4. Yves Le Traon, Thierry Jeron, Jean- Marc Jezequel, and Pierre Morel "Efficient Object oriented integration and regression testing" IEEE Transactions on reliability, 2000; 49(1).
5. Recardo Terra and Macro Tulio "A dependency constraints language to manage Object oriented software architectures" Software Practice and Experience 2009; 39: 1073–1094p. Willy interscience DOI: 10.1002/spe.931.
6. Sunil L. Bangare, Akhil R Khare and Pallavi S. Bangare " Measuring the Quality of Object oriented software modularization" International Journal of computer science and engineering (IJCSE).
7. Michela Pedroni and Bertrand Meyer Object-oriented modelling of Object-Oriented Concepts A Case Study in Structuring an Educational Domain http://link.springer.com/chapter/10.1007/978-3-642-11376-5_15#page-1.
8. Ranjita Kumar Swain, Prafulla Kumar Behera and Durga Prasad Mohapatra Generation and optimization of test cases for object orinetd software using state chart diagram http://arxiv.org/ftp/arxiv/papers/1206/1206.0373.pdf.
9. Amarnath singh, Biswajit Bishoyee, santosh kumar rath and Dharmananda Parida " International journal of computer science and information technologies, 2011; 2(5): 2055–2059p.
10. Xiaolan wang, Yanshuai Zhang and Hong he " Method of the object oriented Programs Exact testing Proceedings of the Second Symposium International Computer Science and Computational Technology (ISCSCT '09) Huangshan, P. R. China, 2009; 26–28p.
11. R Krishnamoorthi, S. A. Sahaaya Arul Mary Factor oriented requirement coverage based system test case prioritization of new and regression test cases. Information and Software Technology 51. 2009; 799–808p.
12. Varun Gupta and Jitender kumar Chhabra Dynamic cohesion measures for object oriented software Journal of system Architecture 57. 2011; 452–462p.

13. Object Oriented Design Principle http://www.oodesign.com/design-principles.html.