ISSN No. 0976-5697

Volume 4, No. 3, March 2013 (Special Issue)

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Analysis of Impacted Classes and Regression Test Suite Generation

Aprna Tripathi
Department of Computer Science and Engineering
MNNIT Allahabad Allahabad, India
rcs1051@mnnit.ac.in

Dharmendra Singh Kushwaha
Department of Computer Science and Engineering
MNNIT Allahabad Allahabad, India
dsk@mnnit.ac.in

Arun Kumar Misra
Department of Computer Science and Engineering
MNNIT Allahabad Allahabad, India
akm@mnnit.ac.in

Abstract: Software needs to be changed over time to deal with new requirements, existing faults and change requests. Change made to software will inevitably have some unforeseen and un desirable effects on other parts of the software. Software Change Impact Analysis (SCIA) is an approach used to identify the potential effects caused by change made o software. As any change is requested by the client or user the software project team have not only the objective to incorporate that change in the existing system while to maintain the software quality is also the other objective. The paper proposes an approach to find the impact set of the change requested by user or client. Author uses the impact set of the requested change to prepare the test suite for regression testing. The results of proposed approach are illustrated with a case study. The approach used in this paper finds the regression test suite required for regression testing based on the impact set that is the sub set of the existing test suite of the system.

Keywords: Software Change Impact Analysis, Regression Testing, Class, Regression Test Suite, Change Element and Impact set.

I. INTRODAUCTION

Software maintenance [1] has been recognized as the most costly and difficult part of software development. Software needs to be changed over time to deal with new requirements, existing faults and change requests, etc. Sometimes it may require addition of new requirements while in other cases amendment in the exiting is sufficient. For a successful and qualitative maintenance, it is essential that before moving towards change implementation, analysts should thoroughly analyze the impacts of requested changes on the existing software. Changes made to software have some unforeseen and undesirable effects on other parts of the software. A major problem for developers in an evolutionary environment is that seemingly small changes can ripple throughout the system to have major unintended impacts elsewhere. The complex relationship among classes makes it difficult to anticipate and identify the ripple effects of changes. Software Change Impact Analysis (SCIA) is a solution to identify unpredicted and potential effects caused by software changes. It starts with a set of changed elements in a software system, called the change set, and attempts to determine a possibly larger set of elements, called the impact set. In this paper we have used the class name as the change element for the change set. The impact set is the collection of the classes that would be impacted with reference to the class where the change occurs. After implementation the assurance is needed that the functionality of change is as needed and for this we perform regression testing. Thus it is essential to know which subset of existing test suite is needed to perform the regression testing

This paper proposes a method for preparing the regression test suite for a requested change before actual implementation of that change. The rest of the paper is organized as follows. Section II summarizes the related work of impact analysis and test suite reduction for regression testing. Section III details the proposed approach. Section IV and V presents the case study and results. Section VI, the last section of the paper, outlines conclusions and future work.

II. STATE- OF - THE- ART

SCIA is an approach used to identify the potential effects caused by changes made to software. Angelis and Wohlin [2] discuss the importance of the change impact analysis issues during software change implementation process. Ali [3] deals with impact analysis approach during project changes and identifies issues of impact analysis. Kenichi [4] defines the impact scale to quantify the change impact. After changes have been implemented in the original system, SCIA can be applied to guide regression, select test cases, perform change propagation, and ripple effect analysis. Xiao-Bo [5] proposes a method for SCIA based on program dependence graph (PDG). PDG is used for analyzing the data and control dependencies. Proposed approach is considering the fine grain level of the software that gives efficient knowledge about change impact but it gives equal credence to each type of change,

CONFERENCE PAPER
II International Conference on
"Advance Computing and Creating Entrepreneurs (ACCE2013)"
On 19-20 Feb 2013
Organized by
2nd SIG-WNs, Div IV & Udaipur Chapter , CSI , IEEE Computer Society Chapter India Council ,
IEEE Student Chapter Geetanjali Institute of Technical Studies, Udaipur, Rajasthan, India

which is not practical. Ceccarelli [6] proposes a novel change impact analysis method based on the idea that the mutual relationships between software objects can be inferred with a statistical learning approach. Acharya [7] propose a novel framework for change impact analysis based on static program slicing, while addressing its performance and accuracy issues. Sun [8] proposes a new SCIA technique based on lattice of class and method dependence (LoCMD). The proposed approach can effectively capture the dependences between classes and methods. Based on the LoCMD, SCIA technique calculates a ranked list of potential impacted methods according to a metric, impact factor, which corresponds to the priority of these methods to be inspected. He does not claim that technique could be generalized to an arbitrary software system. Li [9] proposes a solution for analyzing SCIA of object oriented software.

As a solution author gives a definitions for object-oriented data dependency graphs, a set of algorithms that allow software developers to evaluate proposed changes on object-oriented software, a set of object-oriented change impact metrics to quantitatively evaluate the change impacts, and a proof-of-concept tool (ChAT) that computes the impacts of changes but this approach is applicable only if codes are available, and the case where component based software development is used, this approach fails. Liu [10] proposes a method to analyze the change impacts caused by woven aspects. The changes introduced by aspects are assessed from the perspectives of control and data interactions between the base code and aspects.

The simplest way to gain the bug free software after regression testing is to retest the complete test suite. Several techniques based on code and model exists in the literature. In software testing, there are the relationship between modules, and modules associated with test cases [11]. Xuepin [12] uses the association graph to represent relationships between modules, when requirements change, before modify codes of a module, according criterion of positive association is used to assess its effect. Gorthi [13] proposes an approach for regression test suite selection that utilizes Unified Modeling Language (UML) [14] based Use Case Activity Diagrams (UCAD). Chen [15] proposes a new pairwise interaction of requirements based coverage criterion (PWIC) and a pairwise interaction based test suite reduction approach (PWIR). PWIC generates all the interactions of arbitrary two requirements (i.e., pairwise interaction) covered by test suite T. The new coverage criterion is designed with respect to the observation that covering all the interactions of requirements helps to detect more faults than only covering all the individual requirements.

III. PROPOSED APPROACH

It is not preferable to run complete test suite during regression testing. There are various techniques for selecting test cases from the existing test suite for regression testing. These techniques are broadly classified into the three classes: prioritization techniques, selection techniques and reduction techniques. To preparing the regression test suite, we use the test suite reduction approach. As the first step, we find the impact set with respect to the requested change. The aim of finding the impact set is to provide the earlier generation of regression test suite based on the volume of impact set. For this we followed the following steps:

a. Generation of XML representation of class diagram
b. Finding dependencies between classes
c. Finding classes impacted by requested change
d. Locating error prone segment in the code
e. Preparation of test suite for regression testing form impact set and
f. Analysis of number of classes impacted with respect to a changed class.

Figure 1 depicts the work flow of our proposed work. In this work we considered the XML of class diagram for finding the impact set, the test suite of the existing system for finding test suite for regression testing and the existing system code to analyze the potential areas of failure as inputs of our proposed approach. The whole work is divided into the following sub - activities: finding dependencies among classes, classes impacted by the requested change, percentage reduction in existing test suite and finally the error prone segments of code where the possibility of error lies.

A. Generation of XML Representation of Class Diagram:

Before moving towards applying our approach we generate class diagram using a plug-in in Eclipse called ObjectAid [16] that generates class diagrams. Internally, this class diagram is represented in the form of an XML file, which contains the information about classes, their methods, attributes and dependencies, associations and generalizations between them. Each class has an id, and each dependency has a source and a target, both of which is an id of the class which acts as the source or the target of the dependency. Association (a relationship between classes of objects that allows one object instance to cause another to perform an action on its behalf) and generalization (shared characteristics, especially methods and attributes, usually as an outcome of inheritance between classes) are also a form of dependency between classes, and has been taken into consideration. In the following section, we illustrate how the dependencies between the various classes in a given software have been extracted using its class diagram.

B. Finding Dependencies between Classes:

We can generate the class diagram for any given software. We now have the xml representation of the class diagram for the software as well. We use the XML DOM (Document Object Model) parser API in JAVA, which is included in org.w3c.dom package for JAVA.

Firstly, all nodes that have the tag-name as Class are extracted, along with the id given to that class. These classes are stored in an array, with index corresponding to their respective ids.

After all the classes have been extracted, all nodes that have tag-names as dependency, association or generalization have been extracted. These nodes contain, within their source and target attributes the ids of classes. Using these values, a two dimensional dependency matrix DM has been created, which is a matrix of 0's and 1's, where row represents the source class, the column represents the target class, and the value 1 of a particular cell shows that there exists a dependency from the ith row to the jth column, and 0 shows the independency between two classes. So, by parsing the XML file, we now have the list of all classes in the software as well as the dependency matrix for those classes. In the following section, we figure out, that if a change is made to a particular class, which classes would be impacted due to that change.

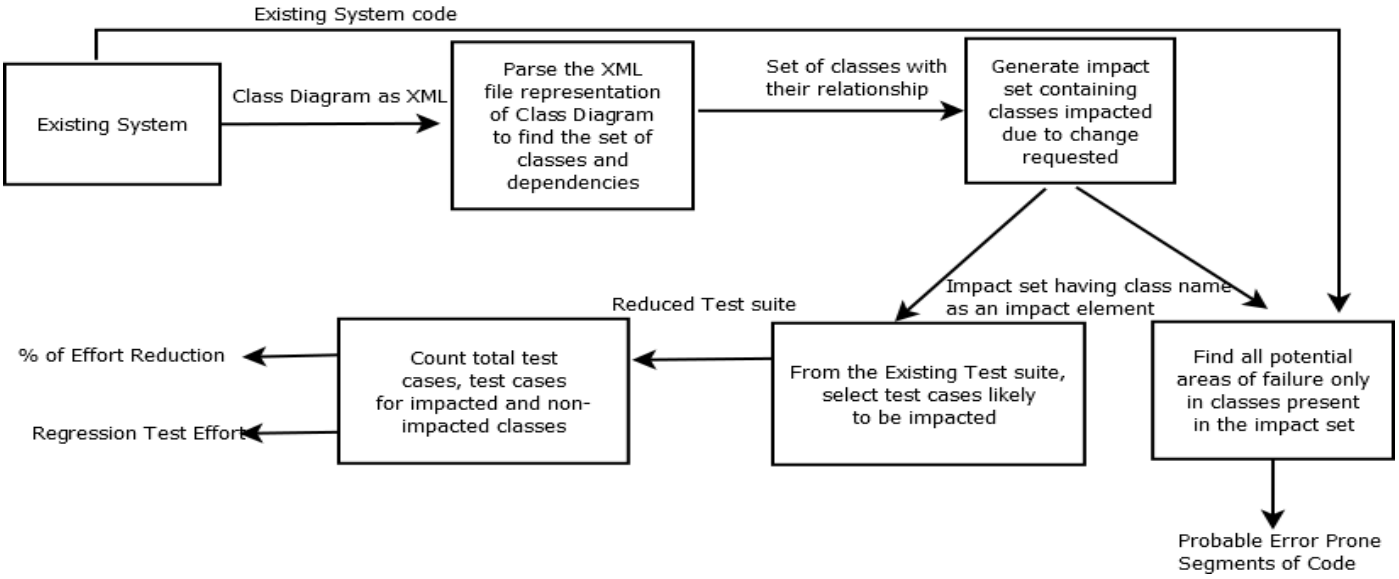

Figure: 1 Proposed Approach Framework

C. Finding Classes Impacted by Requested Change:

These would include the classes that are directly dependent on the modified class, as well as the transitively or indirectly dependent classes. The algorithm shown in figure 2 illustrates the process to find the classes impacted by the requested change. E.g. in Figure 3, when a change is made to the Equality class, then the impacted classes would be the Equality class itself, Bitwise class, which is directly dependent on it, and the Multiplicative class, which in turn is dependent on Bitwise class, and hence indirectly on Equality class and so on.

So, to find the set of impacted classes, we apply BFS (Breadth First Search) algorithm approach on the dependency matrix, with the source index as the id of the modified class to find all connected components in the matrix with root as given source id. In section D, we analyze the source code to find all areas which have the maximum potential of failure on a set of test cases due modifications made in a particular class.

Algorithm: IMPACT_SET_GENERATION

Input: Class Name (C) to which change is made, Dependency matrix (DM) of classes, Class Name versus ID mapping (M) and Class array (A)

Output: Set of classes impacted by that change

a. Let S be the initially empty set of impacted classes.
b. Let Q be an initially empty queue for temporary storage of nodes.
c. Let V be an array to mark visited nodes, initially having all 0 elements.
d. J=0
e. Q. Push(M[C])
f. While Q is not empty
    a. F = Q. Pop()
    b. S[J++] = A[F]
    c. Mark V[F] = 1
    d. For i = 1 to DM_MAXROWS
        i. If DM[F][i]=1and V[i]=0
            1 Q. Push (i)
            2 V[i] = 1
g. Return the required Impact Set S

Algorithm to find impact set for a modification made in a particular class

D. Locating Error Prone Segments of Code:

At this point, we are making an assumption that the maximum error prone segments of code of failure are those which have branching, looping, or a call to a function of some other class. We have already extracted the set of classes that are impacted when a class is modified for implementing the requested change. For all classes present in the impact set, we parse the source code of the class to find those areas, and the output of this section is the class name accompanied with the line number where the failure potential exists in the code.

E. Preparation of test suite for Regression Testing from the Impact Set:

Once we have the impact set for requested change, we will prepare the test suite for regression testing from the existing test suite. The testing can be reduced if we can identify those test cases from the test suite that may be impacted due to the modifications made. Other test cases need not be run again and again. From the existing test suite, we select only those test cases that test one or more of the classes available in the impact set. Thus the test cases that require to be run again, to test the correctness of the software are separated from those test cases which are independent of the change made. So, now we can say that the total reduction in testing is the ratio of test cases that are not covered by impacted classes to the the whole existing test suite.

Required in Regression Testing =

(Total number of test cases for impacted classes / Total number of test cases in Test suite)

In the next section we will analyze the area of the class diagram impacted because of the change class.

F. Analysis of number of classes impacted with respect to a changed class:

It is also an important dimension to analyze the content of impact set. There are two main observations. It might be possible that the classes having the same size of impact set but not have the same size of test suit for regression testing and the second is that the class that have the largest impact set must have the largest test suite for regression testing.

IV. CASE STUDY

We demonstrate the proposed methodology work using a small self-made mini-application Computer Operations Tutorial (COT) on JAVA that takes a mathematical expression as input, parses it and displays the solution as the output. For example: 122+23 to compute the sum of 122 and 23. The exit condition for the application is to give the input "exit".

This example was chosen as the inputs and outputs, and thus the test suite created for this application is easy to understand. The main focus of the project is to efficiently represent the impact of change requested to the application.

The class diagram for the same is shown in Figure 2. The Main class gets a mathematical expression from the user as input, which is then sent to be computed and is parsed and handed internally by the various other classes having their respective functions. The other classes like Additive, Arithmetic, Bitwise, Compute, Main, Multiplicative, Prefix, Shift, Unary and Equality in the application provides the different functionalities needed for computing the user given mathematical expression.

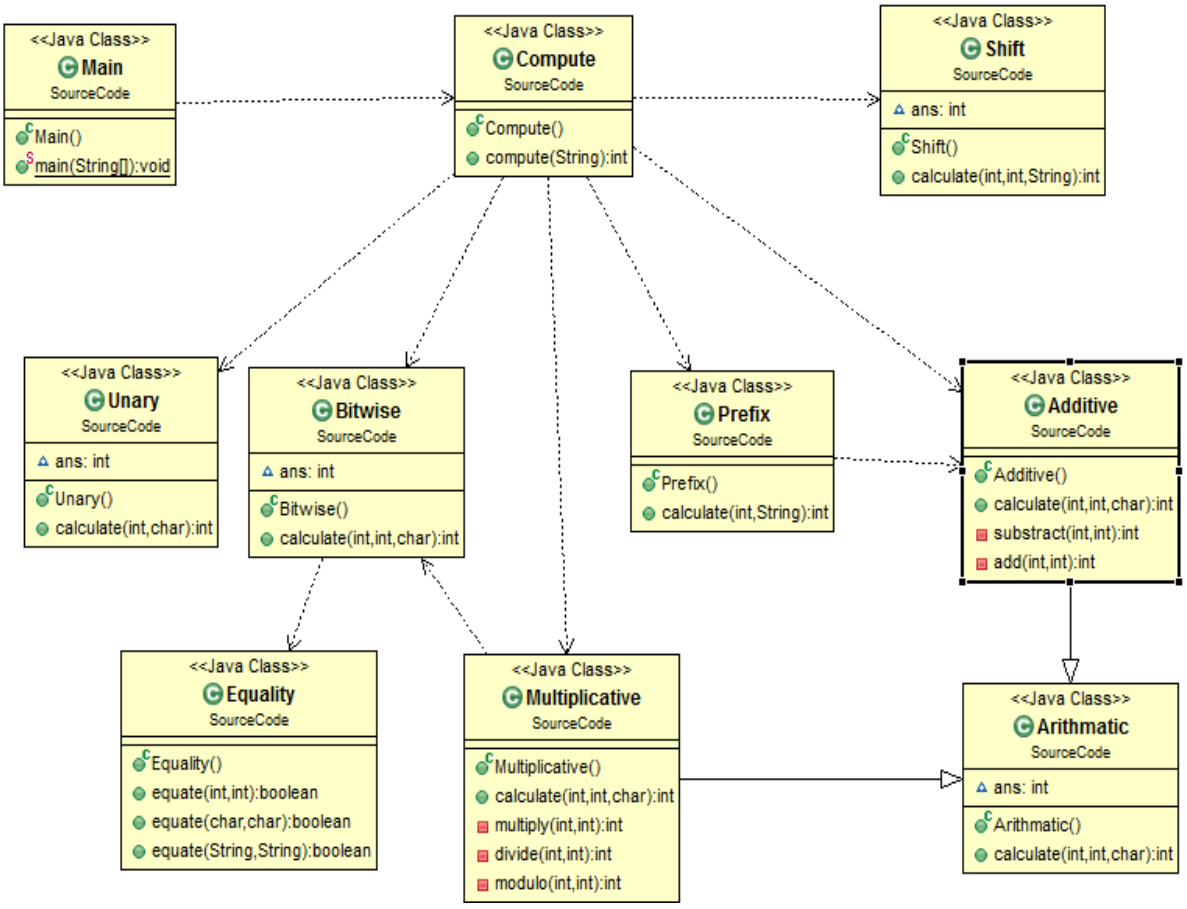

Figure: 2 Class Diagram for COT

V. RESULTS

For COT, we have following classes whose names are stored in an array A:
A[ ] = {" Additive", "Arithmetic", "Bitwise", "Compute", "Main", "Multiplicative", "Prefix", "Shift", "Unary", "Equality" }.
Now suppose we make a change to the ***Equality*** class for implementing the requested change, then the impact set (S) thus produced would be: Equality, Bitwise, Compute, Multiplicative and Main. This can be verified by applying Breadth First Search algorithm to find all connected components with the id of ***Equality*** class as source, on matrix DM shown in figure 4. The zero value of DM[i][j] represents the independency between class i and class j while the one shows that the class j depends on the class i. In fig. 3 we used the symbols to represent the class name i.e. 'A' Additive, 'A1' Arithmetic, 'B' Bitwise, 'C' Compute, 'M' Main, 'M1 ' Multiplicative, 'P' Prefix, 'S' Shift, 'U' Unary, 'E' Equality.

	A	*A1*	*B*	*C*	*M*	*M 1*	*P*	*S*	*U*	*E*
A	0	0	0	1	0	0	1	0	0	0
A1	1	0	0	0	0	1	0	0	0	0
B	0	0	0	1	0	1	0	0	0	0
C	0	0	0	0	1	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0
M 1	0	0	0	1	0	0	0	0	0	0
P	0	0	0	1	0	0	0	0	0	0
S	0	0	0	1	0	0	0	0	0	0
U	0	0	0	1	0	0	0	0	0	0
E	0	0	1	0	0	0	0	0	0	0

Dependency Matrix for Equality class (E) of COT

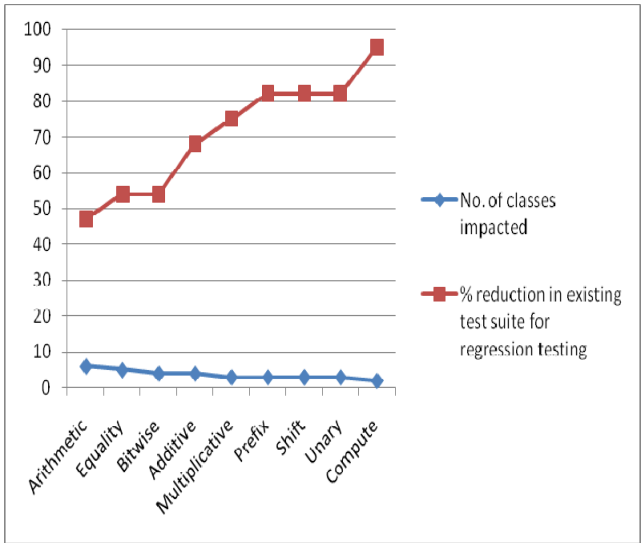

Figure: 3 No. of impacted class and % reduction in Existing test suite for regression testing

Table: 1 Classes and their impact area along with percentage reduction in existing test suite for regression testing

Classes	No of classes impacted	No. of test cases to be run after reduction	% reduction in existing test suit for regression testing
Arithmetic	6	99	47
Equality	5	86	54
Bitwise	4	86	54
Additive	4	60	68
Multiplicative	3	47	75
Prefix	3	34	82
Shift	3	34	82
Unary	3	34	82
Compute	2	8	95

For COT, we have following classes whose names are stored in an array A: A[ ] = {" Additive", "Arithmetic", "Bitwise", "Compute", "Main", "Multiplicative", "Prefix", "Shift", "Unary", "Equality" }.
The requested change can impact the system in two ways: directly or indirectly, as has been depicted in the dependency matrix shown above. When the equality class is considered as the element of change set, the directly impacted class will be bitwise, while compute, multiplicative and main will be indirectly impacted classes. This can be easily depicted from the figure 3. The brown colored class is the initial class to which the change is requested. The green colored classes represent the directly impacted classes. Blue colored classes are the indirectly impacted classes, and the black colored classes are those that are unaffected by the change that has been requested.
For a change requested in Equality classes, in the existing test suite there are 190 test cases, out of which we require to re-run only 86 test cases, thus we can conclude that to test the requested change in Equality class, we need to run only 46% test cases of the existing test suite and the % of reduction in testing will be 54% of the existing system testing.
The fig. 5 and table 1 shows the relation between Number of impacted classes and the percentage reduction in existing test suite of the existing test suite. From fig 5 it is cleared that as the number of impacted classes raises for a class, more the test cases are required for regression testing.

VI. CONCLUSION AND FUTURE WORK

As discussed in the beginning of the paper, change decision depends on the change analysis to identify impacted elements of the existing system for requested

change. The paper looks for a solution for finding the required test suite for regression testing with respect to the type of change requested by user or client. We are finding test suite required for regression testing based on the impact set that is the sub set of the existing test suite of the system. During work, we reached at a very interesting conclusion that the size of the test suite for regression testing is not only depends on the number of the classes impacted while it also depends on the size of the impacted classes for the requested change. And this fact is shown in the results. Currently the plug-in that we use ObjectAid specifically creates class diagrams for java classes only. In future, we aim to use a self-designed approach to create class diagrams independent of the platform used. Currently we are finding the impact set with the assumption that we have the class name, where the change will be implemented against the requested change. In future we will include function and data in the change set. Since there may be scenarios where only a few dependent functions are impacted due to the requested change instead of the whole class and we have the class name as well as the function where the change is made. So, in future we aim to analyze the impact at function level to further reduce the test suite by a much greater margin and more efficient impact set having impact elements in terms of class name with their function name.

VII. REFERENCES

[1] Software Engineering - K. K. Aggarawal & Yogesh Singh, 2nd Ed., New Age International

[2] Lefteris Angelis and Claes Wohlin. 2008. An Empirical Study on Views of Importance of Change Impact Analysis Issues. IEEE Trans. Softw. Eng. 34, 4 (July 2008), 516-530.

[3] Ali, H.O.; Rozan, M.Z.A.; Sharif, A.M.; , "Identifying challenges of change impact analysis for software projects," Innovation Management and Technology Research (ICIMTR), 2012 International Conference on , vol., no., pp.407-411, 21-22 May 2012

[4] Kobayashi, K.; Matsuo, A.; Inoue, K.; Hayase, Y.; Kamimura, M.; Yoshino, T.; , "ImpactScale: Quantifying change impact to predict faults in large software systems," Software Maintenance (ICSM), 2011 27th IEEE International Conference on , vol., no., pp.43-52, 25-30 Sept. 2011

[5] Zhou Xiao-Bo, Jiang Ying, and Wang Hai-Tao. 2011. Method on Change Impact Analysis for Object-Oriented Program. In Proceedings of the 2011 4th International Conference on Intelligent Networks and Intelligent Systems (ICINIS '11). IEEE Computer Society, Washington, DC, USA, 161-164.

[6] Michele Ceccarelli, Luigi Cerulo, Gerardo Canfora, and Massimiliano Di Penta. 2010. An eclectic approach for change impact analysis. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2 (ICSE '10), Vol. 2. ACM, New York, NY, USA, 163-166

[7] Mithun Acharya and Brian Robinson. 2011. Practical change impact analysis based on static program slicing for industrial software systems. In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11). ACM, New York, NY, USA, 746-755.

[8] Xiaobing Sun, Bixin Li, Sai Zhang, Chuanqi Tao, Xiang Chen, and Wanzhi Wen. 2011. Using lattice of class and method dependence for change impact analysis of object oriented programs. In Proceedings of the 2011 ACM Symposium on Applied Computing (SAC '11). ACM, New York, NY, USA, 1439-1444.

[9] Li and A. Jefferson Offutt. 1996. Algorithmic Analysis of the Impact of Changes to Object-Oriented Software. In Proceedings of the 1996 International Conference on Software Maintenance (ICSM '96). IEEE Computer Society, Washington, DC, USA, 171-184.

[10] Chien-Hung Liu, Shu-Ling Chen, and Wei-Lun Jhu. 2011. Change impact analysis for object-oriented programs evolved to aspect-oriented programs. In Proceedings of the 2011 ACM Symposium on Applied Computing (SAC '11). ACM, New York, NY, USA, 59-65.

[11] Sun Yingying, Zhang Yikun, Yang Kaifeng, Zhang Baowei, Xia Hui .Software test approach based on analyzing program association[J]. Application Research of Computers, 2008, 12.

[12] Guo Xuepin, Chen Huandong, Wu Shulei, Zhan Jinmei, Zhong Sheng, A Method of Fast Module Location Test Based on Requirements Changes, Procedia Environmental Sciences, Volume 11, Part A, 2011, Pages 372-379, ISSN 1878-0296, 10.1016/j.proenv.2011.12.060.

[13] Gorthi, R.P.; Pasala, A.; Chanduka, K.K.P.; Leong, B.; , "Specification-Based Approach to Select Regression Test Suite to Validate Changed Software," Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific , vol., no., pp.153-160, 3-5 Dec. 2008.

[14] Xiang Chen, Lijiu Zhang, Qing Gu, Haigang Zhao, Ziyuan Wang, Xiaobing Sun, and Daoxu Chen. 2011. A test suite reduction approach based on pairwise interaction of requirements. In Proceedings of the 2011 ACM Symposium on Applied Computing (SAC '11). ACM, New York, NY, USA, 1390-1397.

[15] L. C. Briand, Y. Labiche, and L. O'Sullivan. 2003. Impact Analysis and Change Management of UML Models. In Proceedings of the International Conference on Software Maintenance (ICSM '03). IEEE Computer Society, Washington, DC, USA, 256- 265.

[16] www.objectaid.com/ accessed on 01.11.2012.

CONFERENCE PAPER
II International Conference on
"Advance Computing and Creating Entrepreneurs (ACCE2013)"
On 19-20 Feb 2013
Organized by
2nd SIG-WNs, Div IV & Udaipur Chapter , CSI , IEEE Computer Society Chapter India Council ,
IEEE Student Chapter Geetanjali Institute of Technical Studies, Udaipur, Rajasthan, India