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

Software Change Validation before Change Implementation

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 maintenance is the most demanding and most expensive task of the software development. Quality of the regression testing decides the quality of the maintained software. Several approaches, code-based and model- based are recommended in literature to minimize the regression test suit. This paper proposes the model based technique to validate the change before implementation. The approach considers the SRS and UML class diagram for earlier change validation. We update the SRS for including the requested change and prepare difference_SRS file that contain the differences of old and new SRS. After imposing these changes in UML class diagram, dependency matrix for both old and new class diagram and a difference dependency matrix from these two dependency matrices is generated we map the entire changes one by one form difference_SRS file to difference dependency matrix. Using the approach, we have concluded a case study and observed impressive gains in terms of less requirement of regression testing effort. The result shows that the approach allows the implementation of change only after validating the requested change after the design phase.

Keywords: Software Change Validation, SRS, Regression Testing, Class Diagram, UML and Dependency Matrix.

I. INTRODAUCTION

Software maintenance is basically a post development activity but most of the times it consumes 40-70% of the overall development costs [4]. To achieve confidence, currently organizations re-execute the entire system test suite on the entire software. Re-executing complete system test suite is an expensive and time consuming activity. To reduce such costs, execution of smaller regression test suite to validate the changed software is suggested. It is a good practice to validate changes required by the user before implementation. As defined by IEEE standards [14], Validation is the process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements [IEEE-STD-610]. In this work we are proposing an approach to validate the change after design phase. As the changes are requested by user or client, the Change Request Form (CRF) is filled by the user. After the approval of the change the Software Requirement Specification (SRS) is updated for incorporating all the requested change. In general practices, the regression testing comes in picture after implementing the change to achieve adequate confidence in changed software.

This paper proposes a method for validating the change at design phase. The rest of the paper is organized as follows. Section II summarises the related work of 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

Several techniques, both code-based and model-based that recommend smaller regression test suites have been proposed in the literature. Li [1] describes the major challenges in coping with requirement changes in the software verification and validation processes and indicates how those challenges are being addressed at Research In Motion (RIM.) Fluri [2] presents an approach that uses the structure compare services shipped with the Eclipse IDE to obtain the corresponding fine grained changes between two subsequent versions of any Java class. This information supports filtering those change couplings which result from structural changes. So we can distill the causes for change couplings along releases and filter out those that are structurally relevant. Briand [3] focuses on automating regression test selection based on architecture and design information represented with the Unified Modeling Language (UML) and traceability information linking the design to test cases. The approach considers few assumptions like UML diagrams are consistent with each other. Gorthi [5] presents a novel approach for regression test suite selection that utilizes Unified Modeling Language (UML) based Use Case Activity Diagrams (UCAD). In literature, a number of regression test suit selection techniques are recommended

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

to minimize the time and cost involved in regression testing, to validate modified software. Typically regression test selection techniques are either code-based or model-based [6, 7, 8, 9, 10]. Code-based techniques [11, 12, 13] use the information obtained from two different versions of the code to analyze the change impact and select the tests. Farooq [16] proposes an approach for regression test in this paper based on selective state machine. For this purpose class diagram and state diagrams are used and classes are defined the changes as class driven and the state driven.

III. PRAPOSED APPROACH

The aim is to validate the changes in the design phase itself. For this, we are using the SRS and class diagrams of the

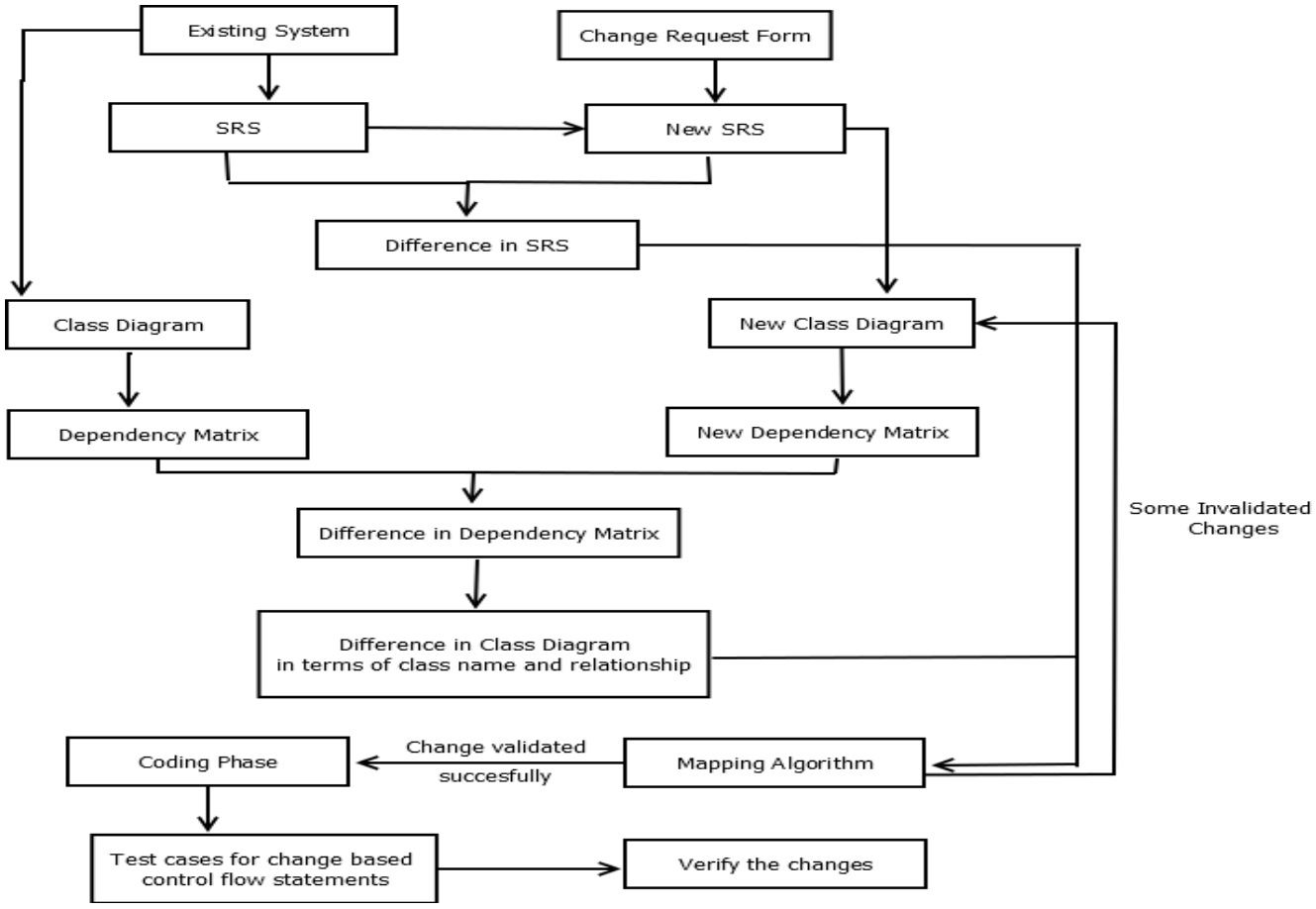

Figure: 1 Framework of Proposed Approach

Old system as well as the new system. For this we followed the following steps:

a. Finding difference between the two SRS..
b. Generation of XML representation of class diagrams
c. Finding dependencies between classes for old and new design
d. Finding changes in old and new dependency matrix
e. Mapping the changes of SRS and class diagram

The figure 1 represents our proposed approach. Here we have considered the SRS of the existing system as the input. Using the CRF given by the user, we update the existing SRS and make the new SRS. Using an open source JAVA code we compare old and new SRS. The differences between old and new SRS are stored in a document file: diff.doc. After this, we draw the class diagram of the existing system and hence the Dependency Matrix (DM1) by generating the XML representation of the class diagram. Similarly we draw the class diagram of the new changed system and its corresponding DM2. After this, we find the difference between these DM1 and DM2 that represents the difference in class diagram in terms of class name and relationship. We use a mapping algorithm to map that changes made in SRS and class diagram. Software changes are valid if for all the changes in the SRS have corresponding change in the class diagram. If the changes have been validated successfully, we proceed towards implementation of accepted changes otherwise we go back to the design phase for modifying the new class diagram. This process is repeated until all the changes are validated successfully.

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

A. Finding difference between the two SRS:

We have the SRS of the existing system. After obtaining the new user requirements we make the new SRS. Now we use an open source JAVA code to find the differences between the old and the new SRS.

B. 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 [15] 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.

C. Finding Dependencies between Classes for old and new system :

We can generate the class diagram for both old and new software. We now have the XML representation of both 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. The parsing algorithm XML_PARSER is developed for the XML file to find the classes defined in the software and the dependency among those classes as shown in Figure 2.

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 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 find out the differences between old and new dependency matrix.

Algorithm 1: XML_PARSER

Input: XML file (X), the representation of class diagram

Output: M the list stores the class id corresponding to the class name.

Declare: Dependency Matrix (DM) to preserve the dependency among all classes

Array (A) with indices as class id and value as class name.

a. Initialize a new instance of Document Builder Factory.
b. Extract the root element of document X (getDocumentElement).
c. Let NodeList (NL) is an initial empty NodeList where NodeList is a data structure defined in org.w3c.dom package.
d. Let initially DM is a matrix having all elements as 0.
e. NL = Get ElementsbyTagName ("class")
f. For i = 0 to NL.length
    a) Id = NL[i].getAttribute("id")
    b) ClassName = NL[i].getAttribute("name")
    c) A [Id] = ClassName
    d) M [ClassName] = Id
g. temp[] ={ " dependency ", " generalization " " association "}
h. For (int k= 0 to 2)
    a) NL = Get ElementsbyTagName (temp[k])
    b) For i =0 to NL.length
        i. Source = NL[i].getAttribute("source")
        ii. Target = NL[i].getAttribute("target")
        iii. DM[source][target] = 1

i. Return M, A, DM

Figure: 2 Algorithm XML-Parser

D. Finding Differences between old and new dependency matrix :

Now we have dependency matrix based on both initial and the final requirements. We find the difference between them which gives us the classes which have been added or deleted and also the relationships that have been added or deleted. In class dependency we have string array "classes", which stores all the classes that have been used in the source code. Now when we get a value 1 in the difference matrix we call the relevant function in class "dependency" and pass the row and column indices corresponding to that value 1 as arguments. This function then prints both the classes concerned with that row and column indices. Thus these classes represent the classes between which a new relationship has been added. In the next section we will map the difference in the two SRS obtained in step A with the difference in their class diagram for validating the change requested by the user.

.

E. Mapping the changes of SRS and class diagram :

To verify that the changes in the SRS have been successfully implemented in the design phase, we parse the differnce_SRS file to get the new nouns and verbs and store it in a SRS_Difference list. And the name of newly added class with their function and variables name are fetched and stored in other list named as Class_Difference. Then we will compare the SRS_Difference and Class_Difference tables, only after getting all the words of SRS_Difference list in Class_Difference list, we will conclude that all the requested change that are incorporated in the new SRS are also designed in the class diagram, and now we can move towards implementation for including the requested change in the existing system.

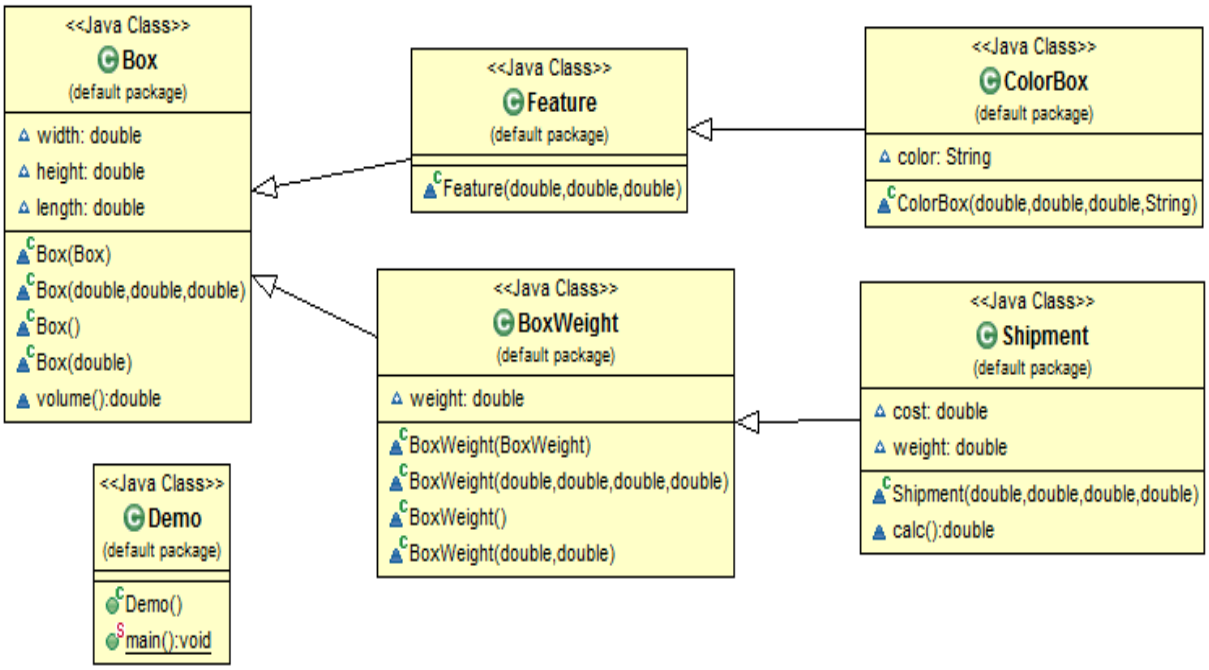

Figure: 3 Class Diagram Before Change

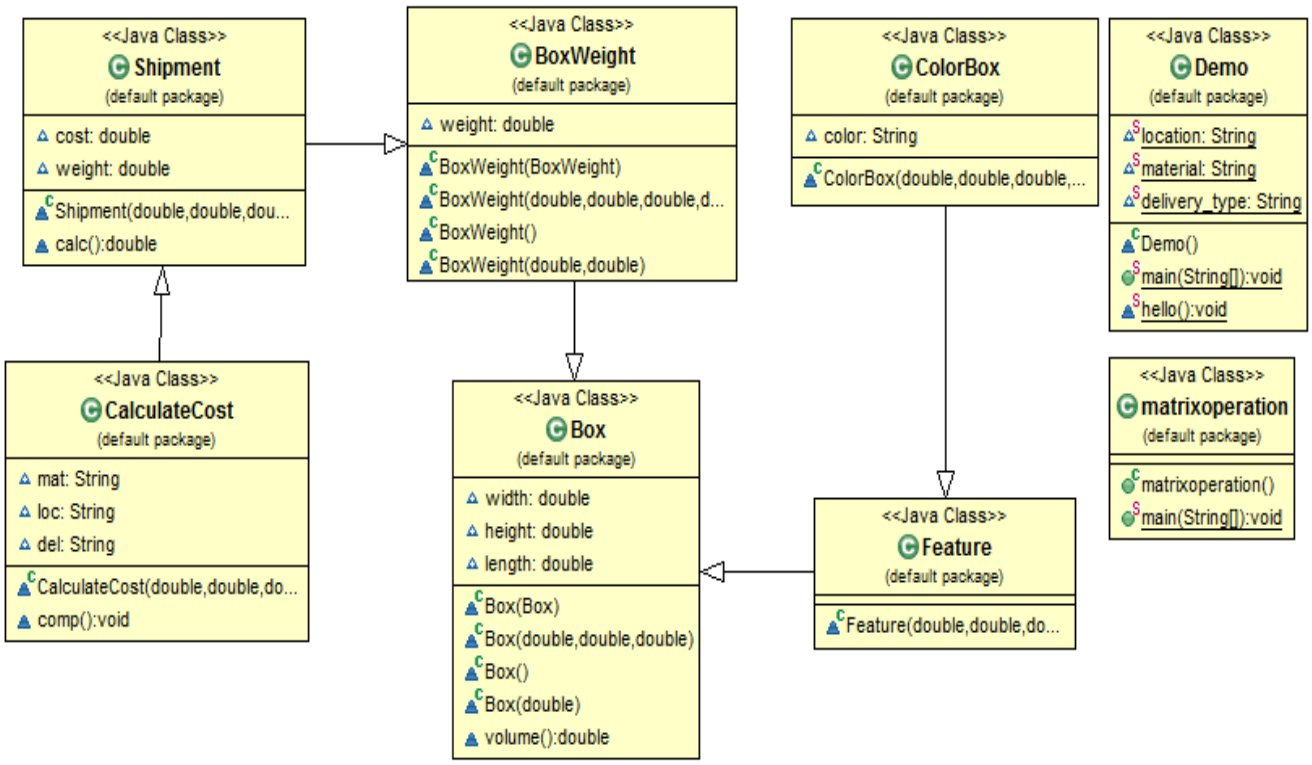

Figure: 4 Class Diagram Matrix After Change

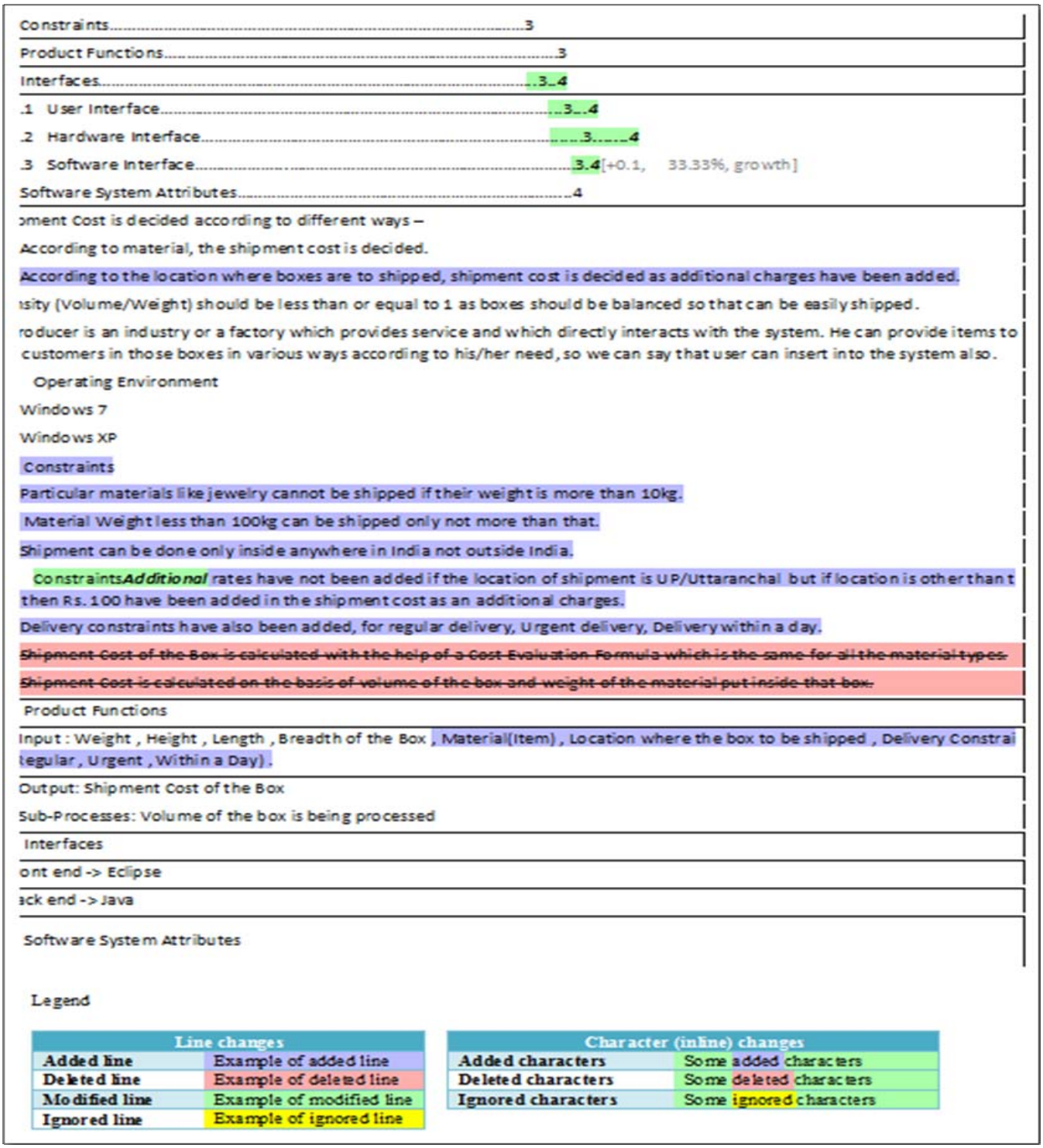

Figure: 5 Differences in SRS Before and After Change

IV. CASE STUDY

We demonstrate our work using a small self-made mini-application Shipment Retail Management System (SRMS 1.0) on JAVA. We assumed that we are only shipping the material if it is packed in a box. For computing shipping cost, we input the dimensions and weight of the box. In the process to incorporate the changes demanded by user and clients, we incorporated these changes in the existing SRMS 1.0 and developed SRMS 1.1. The requested changes are: an additional cost is charged, if the material inside the box is Jewellery, Clothes or Grocery. Also there is an additional cost if the consignment delivery place is outside Uttar Pradesh or Uttaranchal. Also there is an additional cost if delivery type is “urgent” or “within a day”. The final

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 ,

cost is the sum of basic cost and additional cost. The figure 3 and 4 are showing the class diagrams of the SRMS 1.0 and SRMS 1.1. The difference of old and new SRS are captured by a tool WinMerge and stored in a text file named as Difference_SRS. Content of the Difference_SRS is shown in figure 5. For SRMS 1.0, we have six classes named as, Box, Feature, ColorBox, Demo, BoxWeight, and Shipment. After change in class diagram for incorporating the requested change, we have new class diagram in SRMS 1.1. In SRMS 1.1 we have eight classes, i.e. classes that are in SRMS 1.0 and two other new classes- Calculatecost and matrixoperation.

V. RESULTS

The dependency matrix represents the relation among classes. Figure 6 and 7 are showing the dependency matrix of SRMS 1.0 and SRMS 1.1 and the figure 8 is showing the difference between the two dependency matrixes. In fig. 6, 7 and 8 we used the symbols to represent the class name i.e. 'B' Box, 'F' Feature, 'C' ColorBox, 'D' Demo, 'W' BoxWeight, 'Co' Calculatecost and 'M' matrixoperation and 'S' Shipment

	B	*W*	*D*	*F*	*C*	*S*
B	0	1	0	1	0	0
W	0	0	0	0	0	1
D	0	0	0	0	0	0
F	0	0	0	0	1	0
C	0	0	0	0	0	0
S	0	0	0	0	0	0

Figure: 6 Class Dependency Matrix Before Change

	B	*W*	*D*	*F*	*C*	*S*	*Co*	*M*
B	0	1	0	1	0	0	0	0
W	0	0	0	0	0	1	0	0
D	0	0	0	0	0	0	0	0
F	0	0	0	0	1	0	0	0
C	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	1	0
Co	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0

Figure: 7 Class Dependency Matrix After Change

	B	*W*	*D*	*F*	*C*	*S*	*Co*	*M*
B	0	0	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	1	0
Co	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0

Figure: 8 Differences in Dependency Matrices Before and After Change

From the figure 8, we get the changes in terms of class names and class relationship. The newly added classes are stored in the table Class_difference. In this case two new classes, Co and M are added in the existing design. From figure 3 and 4 we can get that new added classes are: Matrixoperation and CalculateCost. Also a relationship between CalculateCost and Shipment is added in the new class diagram. In the CalculateCost class the addition cost function is added and this requirement is also available in Difference_SRS document. This Difference _SRS file is parsed and the nouns and verbs are filtered. These words are stored in a list named SRS_Difference. After comparing the list SRS_Difference and Class_Difference, we found that the words exist in the SRS_Difference list is also in Class_Difference. Thus finally we are validating that the user requested change has been implemented successfully in the design phase. From the difference matrix we can also visualize the least affected classes after incorporating the change in the design.

VI. CONCUSION AND FUTURE WORK

The regression testing assures the corrective implementation of the requested change. To test the system regressively in minimum effort the test suit selection and prioritization techniques are used. This paper proposes an approach for validating change before implementation. The usage of this approach significantly reduces the effort required in change implementation as well as in regression testing. It finds the invalid changes at design phase, so that effort required for coding the testing is consumed only for implementing the valid changes and during testing we only require to verify the requested changes. This significantly reduces the number of test cases that need to be run to verify the new source code. Currently we are using only the class diagram. in future we shall include more diagrams like sequence diagram and collaboration diagram to validate the change. Also currently we have used open source software to find the differences between the old and new SRS. In future we aim to develop a self designed code for the above purpose.

VII. REFERENCES

[1] Shimin Li, Ladan Tahvildari, Weining Liu, Mike Morrissey, and Gary Cort. 2008. Coping with Requirements Changes in Software Verification and Validation. In Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering (CSMR '08). IEEE Computer Society, Washington, DC, USA, 317-318.

[2] Beat Fluri, Harald C. Gall, and Martin Pinzger. 2005. Fine-Grained Analysis of Change Couplings. In Proceedings of the Fifth IEEE International

Workshop on Source Code Analysis and Manipulation (SCAM '05). IEEE Computer Society, Washington, DC, USA, 66-74.

[3] L. C. Briand, Y. Labiche, and S. He. 2009. Automating regression test selection based on UML designs. *Inf. Softw. Technol.* 51, 1 (January 2009), 16-30.

[4] Aggrawal K. K. and Singh Y., Software Engineering, 4th Edition, New Age International.

[5] Ravi Prakash Gorthi, Anjaneyulu Pasala, Kailash KP Chanduka, and Benny Leong. 2008. Specification-Based Approach to Select Regression Test Suite to Validate Changed Software. In Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference (APSEC '08). IEEE Computer Society, Washington, DC, USA, 153-160.

[6] Lihua Xu and Debra Richardson, "Generating Regression Tests Using Model Checking", http://gracehopper.org/2004/Proceedings/PDF/ni_Xu.pdf

[7] Zalewski, M. and Schupp, S., "Change Impact Analysis for Generic Libraries", 22nd IEEE International Conference on Software Maintenance, September 24-27, 2006, pp 35-44.

[8] Zheng, J., Robinson, B., Williams, L., and Smiley, K., Applying Regression Test Selection for COTS based Applications", roceedings of ICSE 2006, Shanghai, China, ay 20-28, 2006, pp 512-521.

[9] Sujith Kumar Chakrabarti and Y N Srikanth, "Specification based regression testing using explicit state space enumeration", International conference on software engineering advances, ctober 29 to November 3, 2006.

[10] Yanping Chen, Robert L. Probert and Hasan Ural, "Model-Based regression test suite generation using dependency analysis", 3rd International workshop on advances in model-based testing, London, July 9-12, 2007, pp 54-62.

[11] Anjaneyulu, P., Srinivasa, R., Srinivas, G. and Sinha, P., "An Approach Based on Modeling Dynamic Behavior of the System to Assess the Impact of COTS Upgrades", 13th Asia-Pacific Software Engineering Conference, Dec. 6-8, 2006, pp 19-26.

[12] Anjaneyulu Pasala, Yannick LH, Fady A, Appala Raju G and Ravi P Gorthi, "Selection of regression test suite to validate software applications upon deployment of upgrades", 19th Australian Software Engineering conference, 25-28 March 2008, pp 130- 138.

[13] Apiwattanapong, T., Orso, A., and Harrold, M.J., "JDiff: A Differencing Technique and Tool for Object--Oriented Programs", Journal of Automated Software Engineering, Vol 14, No. 1, March 2007, pp 3-36.

[14] Software Engineering Standards Committee of the IEEE Computer Society, "IEEE Recommended Practice for Software Requirements Specifications," IEEE Inc. NY, USA, 1998.

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

[16] Qurat-ul-ann Farooq, Muhammad Zohaib Z. Iqbal, Zafar I Malik, and Aamer Nadeem. 2007. An approach for selective state machine based regression testing. In Proceedings of the 3rd international workshop on Advances in model-based testing (A-MOST '07). ACM, New York, NY, USA, 44-52.