Change Impact Analysis with a Software Traceability Approach to Support Software Maintenance

Muhammad Shahid[*], Suhaimi Ibrahim[**]
[*]CESAT, Islamabad, Pakistan, [**]UniversitiTeknologi Malaysia, Kuala Lumpur, Malaysia

Abstract—**Change impact is an important issue in software maintenance phase. As retesting is required over a software change, there is a need to keep track of software impact associated with changes. Even a small software change can ripple through to cause a large unintended impact elsewhere in the system that makes it difficult to identify the affected functionalities. The impact after changes demands for a special traceability approach. This paper presents a new approach and prototype tool,Hybrid Coverage Analysis Tool (HYCAT), as a proof of concept to support the software manager or maintainers to manage impact analysis and its related traceability before and after a change in any software artifact. The proposed approach was then evaluated using a case study, On-Board Automobile (OBA), and experimentation. The traceability output before and after changes were produced and analyzed to capture impact analysis. The results of the evaluation show that the proposed approach has achieved some promising output and remarkable understanding as compared to existing approaches.**

Keywords-component; software traceability, impact analysis, software change, software maintenance

I. INTRODUCTION

Software undergoes changes to code and associated documents due to a problem or need for improvement in functionalitiesby the users [1]. These changes are done for different reasons, such as adding new features, correcting some errors or for performance improvement. Two fundamental tasks that are performed in maintenance phase by software engineer are impact analysis and regression testing [2].A maintainer himself needs to examine how much and which part of the program modules will be affected after a change and what is required to test it for impact analysis[3]. What will be its complexity and what types of test cases and requirements will be involved? Cost and time issues calculations are very important before the implementation of change. Change Control Board (CCB) needs to evaluate change requests before the actual change is implemented. Impact analysis needs complete traceability relations within software artifacts for the whole scenario.

Change impact analysis (CIA) plays an important role in software development, maintenance, and regression testing [5–7]. CIA can be used before or after a change implementation. Before doing changes, CIA can be employed for programme understanding, change impact prediction and cost estimation [4, 5]. After changes have been implemented, impact analysiscan be applied to trace ripple effects, select test cases and perform change propagation.

Impact analysis is normally used to know the consequences of change requests by end user on the existing software artifacts, i.e. code, requirements and test cases. These changes may be in the form of an addition, deletion or modification in existing software artifacts.It offers insight into the potential effects of the proposed changes before the real changes are applied. This can help in estimation of cost before implementation. The capability to identify the change impact helps a software maintainer to figure out relevant measures to take with regard to maintenance tasks. We may significantly reducethe risks of starting a costly change by identifying potential impacts before modification of source code.

The rest of this paper is organized as follows. Section IIdescribes the overview of software traceability and its types. Section III reports theimpact analysis and its techniques. Section IV provides details about new approach and Section V explains about case study. Results are discussed in Section VI. Conclusion and future work is mentioned in section VII.

II. SOFTWARE TRACEABILITY

Software traceability is useful to detect the affected software artifacts during change. Software change not only affects source code but also other artifacts like design, test artifacts and requirements. Requirements traceability offers support for numerous software engineering tasks like requirements verification and validation, coverage analysis, impact analysis, and regression test selection.Traceability is concerned with documenting the life of a requirement and to provide bi-directional traceability between various associated requirements. It enables individuals to discover the origin of each requirement and follow any change which was made to this requirement with passage of time.

Researchers [8] relate traceability as the ability to trace the dependent items within a model and the ability to trace the corresponding items in other models.Generally traceability relates requirements with other software artefacts that are produced during the software development life cycle [9]. Traceability information provides assistance to software developers or software maintainers to understand dependencies among software artefacts in all software phases.

978-1-4673-9127-6/16/$31.00 ©2016 IEEE

There are two major types of traceability, horizontal and vertical.

A. Horizontal Traceability

There are at least five areas of traceability that belong to horizontal traceability. These can be explained as horizontal traceability between requirement artifact and coding artifacts [10], requirement artifact and testing artifacts [11], requirement artifact and design artifacts [12], requirement artifact and defect reports [13] and design artifact and coding artifacts [14].

B. Vertical Traceability

Vertical traceability not only traces dependencies between different software artifacts in a software phase, but also dependencies within a software artifact itself, such as dependencies among requirements in a use case specification [15].

Data dependency, control dependency and component dependency use different vertical traceability techniques.

III. IMPACT ANALYSIS

Impact analysis provides a sound basis to judge whether a change is worth the effort, or if inevitable, which features should be changed as a consequence [16]. A complete impact analysis is achieved by the traceability links retrieved during recovery, allowing us to calculate all feature dependencies among different artifacts and across time.

There are two fundamental aspects of impact analysis, which are dependency analysis and traceability analysis as stated by researchers [17]. Dependency analysis or program analysis is the analysis of relations only between source codes. This analysis explores only the interior structure of the code. Whereas the traceability analysis uses the relationships between different software artifacts, e.g. requirements, design artifacts, source codes and test artifacts across different phases.

A. Impact analysis Techniques

Impact analysis techniques are divided into two types [18]:

- Static analysis technique
- Dynamic analysis technique

Static Impact Analysis Technique:

This techniques analyses program static information from software artifacts such as requirements, design, code and test cases. Outcome of this is a set of potential impacted classes. Static information of high level artifacts and low level artifacts is used to perform this type of analysis. The low level artifact model uses source code model from the existing source code to identify the set of potential impacted classes. On the other hand, high level artifact model needs the design and requirements artifacts to describe the set of potential impacted classes.

Dynamic Impact Analysis Technique:

These techniques use dynamic information by executing the code to create a potential impacted class. In dynamic analysis, source code is parsed through to generate the dynamic information.

IV. APPROACH

We have designed a new approach which deals impact analysis before and after a change in any artifact. The block diagram of this new approach is shown in Figure 1.

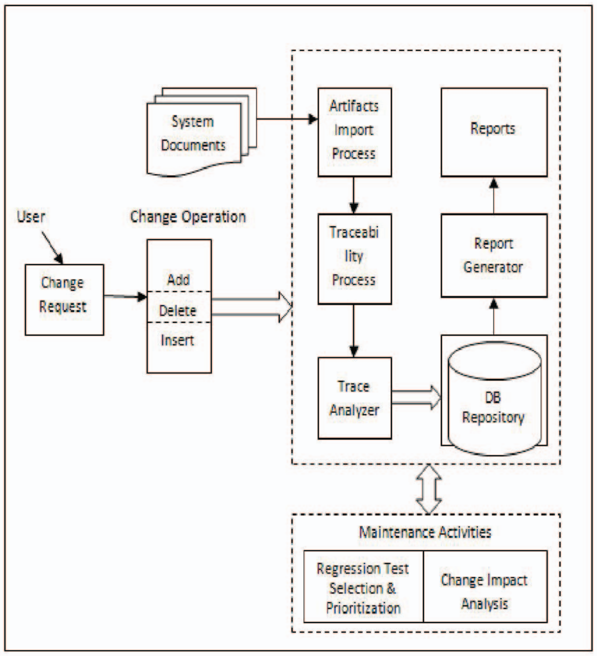


Figure 1. *Approach block diagram*

It is quite difficult to implement and maintain total traceability during software life cycle as the system is composed of different management, policies and development levels. The proposed approach implements automated traceability to measure impact analysis when there is a change in any artifact. The different processes involved in this approach are discussed below.

A. Traceability Generation

There are some mapping relationships among the different software artifacts in a software system. We need to trace and capture their relationships somehow not only within the same level but also across different levels of artifacts before an impact analysis can be implemented.

Traceability matrix is generated first between requirements and test cases. Then source code is instrumented with adding markers before each method. When a test case is executed, the information from the application is collected and saved in a log file. Then trace analyzer analyzes the collected information (log file) to the meaningful information. That info is transferred to a table in the data base.

The software create traceability matrix for “Test case to Package”, “Test case to Class” and “Test case to Method” by using this information. The source code file in C++ is then transformed into XML format. A supporting tool ‘Code Parser’ is developed to work further on this XML generated file. The functionally of tool is to parse the source code and save the information for each method, class, package and file inside the database. The information for methods consist of id name and line number, file parent id, package parent id and class package id.All the information regarding requirements, test cases, methods, classes and packages are stored in a relational database. Traceability matrixes are produced between requirements and methods, requirements and classes and requirements and packages.

B. *Impact Analysis Process*

This process is meant to handle the effect of one primary artifact on the other secondary artifacts. A complete impact analysis is achieved here by the traceability links retrieved during recovery, allowing us to calculate all feature dependencies among different artifacts and across change.

V. CASE STUDY

To implement and verify my approach, I applied it to a case study, On-Board Automobile (OBA). OBA is a development project task assigned to last year post graduate students of Software Engineering at the Advanced Informatics School (AIS), at UniversitiTeknologi Malaysia (UTM). The software system was about 5K lines of code (LOC) with a complete documentation standard supporting MIL-STD-498.

The OBA project itself is an interface system made to enable the driver to engage with his car while on auto cruise or non-auto cruise mode. Some interactions like accelerating speed, suspending speed, resuming speed and applying brakes to car should be technically balanced with the current speed of the car while vehicle is in auto-cruise state. On non-auto cruise, the manual interactions such as pressing or depressing pedals to change speed, braking a car, mileage maintenance and filling fuel need to be taken into consideration.

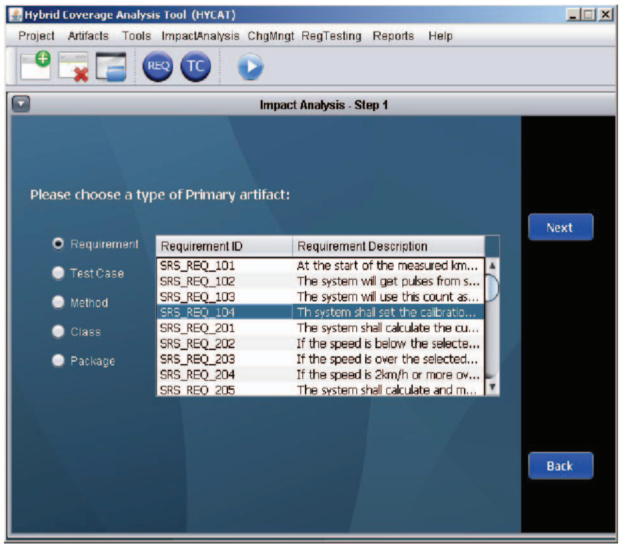
The code was analyzed thoroughly to obtain the program dependencies of methods, classes and packages before starting laboratory experiment by using code parser tool. All the test cases were executed to find the relationship between test cases and code. These linkages of artifacts were further extended to requirements to code and requirements to test cases. The objective here was to obtain the traceability groundwork for vertical and horizontal levels before the start of controlled experiment.

VI. RESULTS

Our case study OBA has 55 requirements, 95 test cases, 8 packages, 14 classes and 121 methods. My prototype tool, HYCAT assumes that a change request by end user has already been translated by domain expert and expressed in terms of the acceptable software artifacts i.e. requirements, test cases, methods and classes. HYCAT was designed to manage the potential effect of one type of artifacts at a time.

The system works such that given an artifact as a primary impact;HYCAT can determine its effects on other artifacts (secondary artifacts) in either top-down or bottom-up tracing. By top-down, it means that this approach can identify the traceability from the higher level artifacts down to its lower levels, e.g. from a requirement we can identify its associated impacted code. By bottom-up means it permits us to identify the impacted artifacts from the lower to a higher level, e.g. from a method we can identify its associated test cases and requirements.

A controlled experiment was done by different groups to know the impact of artifacts on each other before doing any change. Figure 2 shows an initial user entry into the HYCAT system by selecting a type of primary artifact. The user had selected requirement as the primary artifact. Upon selection of primary artifact, HYCAT prompts another window that requires again the user to select the types of secondary artifacts to view the impact. Secondary artifacts can be chosen from requirements, test cases, methods, classes and packages. Figure 3 shows the selected secondary artifacts by the user. Finally all secondary impacted artifacts are displayed in a new window when user press 'Coverage' button as shown in the Figure 4. Coverage result shows that REQ_104 has impacted 2 test cases, 16 methods, 5 classes and 3 packages. It also provides information about total number of lines and percentage covered by this requirement for each artifact

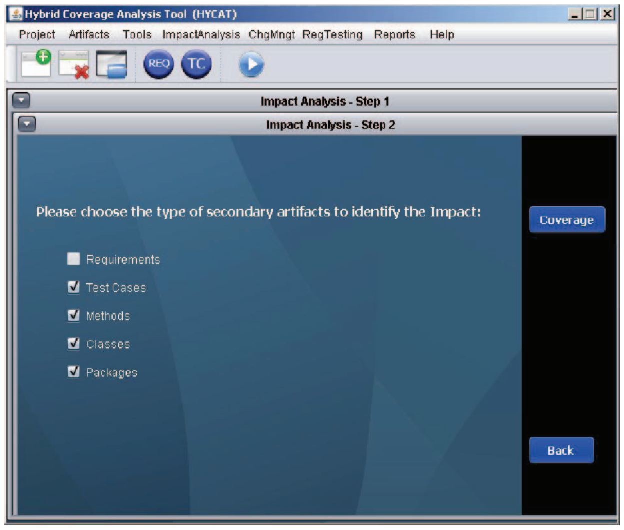

Hybrid Coverage Analysis Tool (HYCAT)
Project Artifacts Tools ImpactAnalysis ChgMngt RegTesting Reports Help
REQ
TC
Impact Analysis - Step 1
Next
Back

Please choose a type of Primary artifact:

(● Requirement) Test Case Method Class Package

Requirement ID	Requirement Description
SRS_REQ_101	At the start of the measured km...
SRS_REQ_102	The system will get pulses from s...
SRS_REQ_103	The system will use this count as...
SRS_REQ_104	Th system shall set the calibratio...
SRS_REQ_201	The system shall calculate the cu...
SRS_REQ_202	If the speed is below the selecte...
SRS_REQ_203	If the speed is over the selected...
SRS_REQ_204	If the speed is 2km/h or more ov...
SRS_REQ_205	The system shall calculate and m...

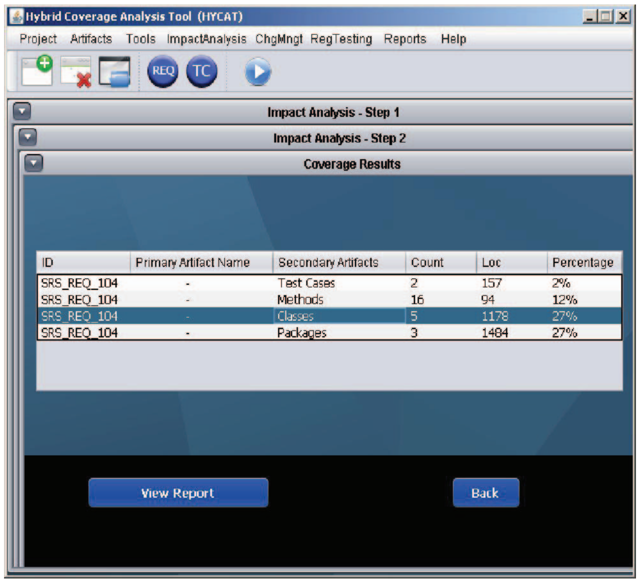
Figure 2. *Primary artifacts selection window*


Hybrid Coverage Analysis Tool (HYCAT)
Project Artifacts Tools ImpactAnalysis ChgMngt RegTesting Reports Help
REQ
TC
Impact Analysis - Step 1
Impact Analysis - Step 2
Coverage
Back

Please choose the type of secondary artifacts to identify the Impact:

- ☐ Requirements
- ☑ Test Cases
- ☑ Methods
- ☑ Classes
- ☑ Packages

Figure 3. *Secondary artifacts selection window*


Hybrid Coverage Analysis Tool (HYCAT)
Project Artifacts Tools ImpactAnalysis ChgMngt RegTesting Reports Help
REQ
TC
Impact Analysis - Step 1
Impact Analysis - Step 2
Coverage Results
View Report
Back

ID	Primary Artifact Name	Secondary Artifacts	Count	Loc	Percentage
SRS_REQ_104	-	Test Cases	2	157	2%
SRS_REQ_104	-	Methods	16	94	12%
SRS_REQ_104	-	Classes	5	1178	27%
SRS_REQ_104	-	Packages	3	1484	27%

Figure 4. *Coverage result of secondary artifacts*

The results were then tabulated for the analysis. Table I below shows the results of the top-down impact analysis for our case study OBA. We can see that the SRS_REQ_102 had caused impacts on 49 methods, 8 classes, 5 packages and 5 test cases.

TABLE I. TOP-DOWN IMPACT ANALYSIS

Group #	Primary Artifact	Secondary Artifacts	Count	LOC	% age
1	SRS_REQ_102	Methods	49	554	40%
		Classes	8	1910	57%
		Packages	5	2480	62%
		T. Cases	5	843	5%
		Reqs.	-	-	-
2	SRS_REQ_209	Methods	18	154	14%
		Classes	4	1080	28%
		Packages	2	1472	25%
		T. Cases	2	201	2%
		Reqs.	-	-	-
3	SRS_REQ_221	Methods	24	350	19%
		Classes	3	946	21%
		Packages	2	1686	25%
		T. Cases	1	350	1%
		Reqs.	-	-	-
4	SRS_REQ_403	Methods	31	277	25%
		Classes	4	562	28%
		Packages	2	2456	25%
		T. Cases	2	277	2%
		Reqs.	-	-	-

The impacts covered up LOC 554and 40 % on methods, LOC 1910 and 57% on classes, LOC 2480 and 62% on packages, LOC 843 and 5% on test cases.

For bottom-up impact analysis of OBA case study, results are shown in Table II. The result from the Table II reveal how Primary Impacted Set (PIS)generated at the lowest level was identified to have an effect on Secondary Impacted Set (SIS) and Actual Impacted Set (AIS) of greater degrees. Inclusiveness and S-Ratio (AIS / SIS) was then calculated for each artifact.

TABLE II. BOTTOM-UP IMPACT ANALYSIS

Group #	Levels	PIS#	SIS#	AIS#	Incl.	S-Ratio A/S
1	Method	3	6	4	1	0.67
	Class	-	2	2	1	1.0
	Package	-	1	1	1	1.0
	T. Case	-	22	15	1	0.68
	Reqt.	-	29	21	1	0.72
2	Method	3	12	8	1	0.67
	Class	-	4	3	1	0.75
	Package	-	2	2	1	1.0
	T. Case	-	34	18	1	0.52
	Reqt.	-	47	27	1	0.57
3	Method	2	11	7	1	0.63
	Class	-	5	4	1	0.80
	Package	-	3	2	1	0.67
	T. Case	-	29	18	1	0.62
	Reqt.	-	36	21	1	0.58
4	Method	3	13	9	1	0.69
	Class	-	4	3	1	0.75
	Package	-	2	2	1	1.0
	T. Case	-	34	19	1	0.54
	Reqt.	-	41	20	1	0.48
5	Method	2	7	5	1	0.71
	Class	-	3	2	1	0.67
	Package	-	2	2	1	1.0
	T. Case	-	32	18	1	0.56
	Reqt.	-	40	24	1	0.60

We can see from the above Table II that the *Inclusiveness* was 1 in all artifacts level (method, class, package, test case, and requirement). It meansthat the AIS were always a part of the SIS. This specifies that the real impacts are not beyond the estimated impacts which prove the accuracy of this approach.The *S-Ratio* expresses how fast the secondary impact (SIS) corresponds to the actual impact (AIS).

When comparing the results with other works, CATIA [3] was the one which is determining impact analysis but only before change. At the class level, CATIA produced the averages results of *S-Ratio* = 0.75, while our results showed *S-Ratio* = 0.79. It can be observed that at class level, our result is more accurate. Results are also better for methods and packages where our *S-Ratio* = 0.67 for methods and *S-Ratio* = 0.93 for packages as compared to 0.59 and 0.87 respectively for CATIA.

Feature analysis was also done to compare our tool with the existing tools. The analysis of result is based on the DESMET method suggested by Kitchenham [19]. On the basis of this method, HYCAT got maximum mean value than other tools as shown in the Figure 5.Regarding particular feature of Traceability (Figure 6), 5% choose Moderate, 50% choose Useful, and 45% choose Very Useful.

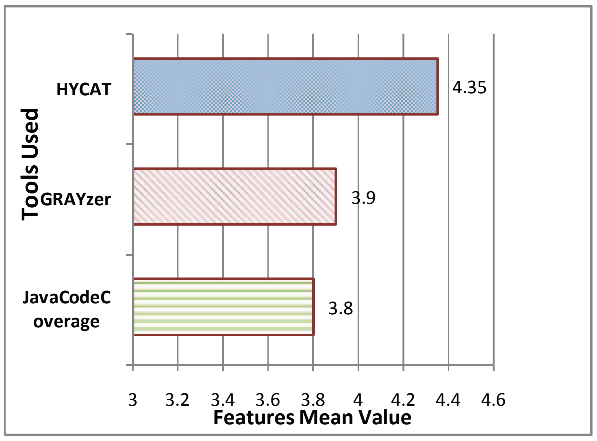


Figure 5. *Comparison with other tools*

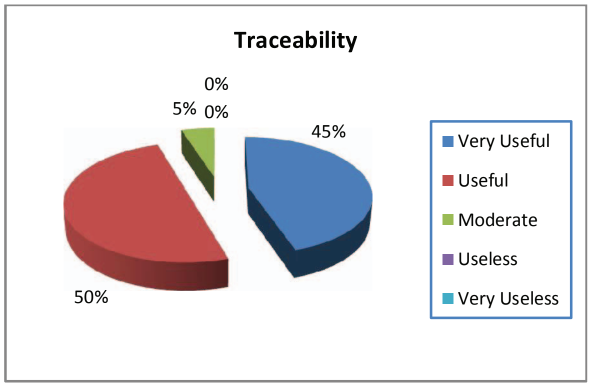


Figure 6. *Traceability support*

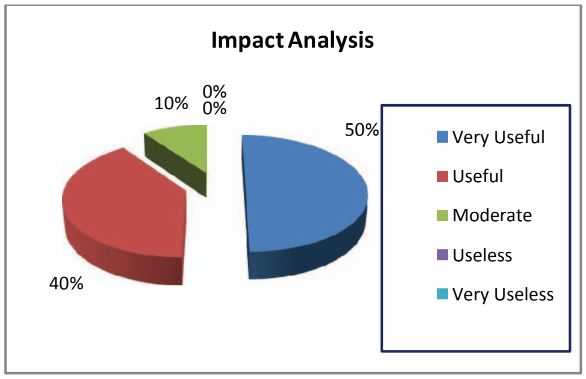


Figure 7. *Impact Analysis support*

In terms of Impact Analysis support (Figure 7) 10% choose Moderate, 40% choose Useful, and 50% choose Very Useful.

VII. CONCLUSION AND FUTURE WORK

The new approach integrates between the static and dynamic analysis in order to establish a rich set of artifact dependencies within the software system and establishes the links between different software artifacts. The correctness of these links helped to establish automated traceability. From the analysis, result shows that prototype toolbased on new approach is highly accurate and efficient.Tracing indirect links, second or third order, between different software artifacts especially in code may make the traces more accurate and understandable. It can help in deep analysis of the artifacts dependency. This area can be considered as a future research work.

REFERENCES

[1] Bennett, K. H., & Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. In Proceedings of the Conference on the Future of Software Engineering (pp. 73-87). ACM.

[2] Orso, A., Apiwattanapong, T., & Harrold, M. J. (2003). Leveraging field data for impact analysis and regression testing. ACM SIGSOFT Software Engineering Notes, 28(5), 128-137.

[3] Ibrahim, S. (2006). A document-based software traceability to support change impact analysis of object-oriented software (Doctoral dissertation, Universiti Teknologi Malaysia, Faculty of Computer Science and Information System).

[4] Bohner S, Arnold R. Software Change Impact Analysis, (1996) IEEE Computer Society Press: Los Alamitos, CA, USA.

[5] Bohner SA. Impact analysis in the software change process (1996): a year 2000 perspective. *Proceedings of the* International Conference on Software Maintenance, Monterey, CA, USA, pp. 42–51.

[6] Rovegard P, Angelis L, Wohlin C. (2008), An empirical study on views of importance of change impact analysis issues. IEEE Transactions on Software Engineering; 34(4):516–530.

[7] Turver RJ, Munro M. Early impact analysis technique for software maintenance. Journal of Software Maintenance:*Research and Practice* 1994; **6**(1):35–52.

[8] Ramesh, B. and Jarke, M. (2001). Toward Reference Models for Requirements Traceability, *IEEE Transactions on Software Engineering*. 27(1): 58-93.

[9] Dorfman, M. (1990). System and software requirements engineering. In *IEEE Computer Society Press Tutorial.*

[10] Grechanik, M., McKinley, K. S., & Perry, D. E. (2007, September). Recovering and using use-case-diagram-to-source-code traceability links. In *Proceedings of the the 6th joint meeting of the European*

software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 95-104). ACM.

[11] Lucia, A. D., Fasano, F., Oliveto, R., & Tortora, G. (2007). Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *16*(4), 13.

[12] Lormans, M., & Van Deursen, A. (2006, March). Can LSI help reconstructing requirements traceability in design and test?. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on* (pp. 10-pp). IEEE.

[13] Yadla, S., Hayes, J. H., & Dekhtyar, A. (2005). Tracing requirements to defect reports: an application of information retrieval techniques. *Innovations in Systems and Software Engineering*, *1*(2), 116-124.

[14] Nistor, E. C., Erenkrantz, J. R., Hendrickson, S. A., & Van Der Hoek, A. (2005, September). ArchEvol: versioning architectural-implementation relationships. In *Proceedings of the 12th international workshop on Software configuration management* (pp. 99-111). ACM.

[15] Spanoudakis, G., Zisman, A., Pérez-Minana, E., & Krause, P. (2004). Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, *72*(2), 105-127.

[16] Passos, L., Czarnecki, K., Apel, S., Wąsowski, A., Kästner, C., & Guo, J. (2013). Feature-oriented software evolution. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems* (p. 17). ACM.

[17] Ali, H. O., Rozan, M. Z. A., and Sharif, A. M. (2012), Identifying challenges of impact analysis for software projects. *Innovation Management and Technology Research (ICIMTR), 2012 International Conference on.* 21-22 May 2012, 407-411.

[18] Jashki, M. A., Zafarani, R., & Bagheri, E. (2008). Towards a more efficient static software change impact analysis method. In *Proceedings of the 8th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering* (pp. 84-90). ACM.

[19] Kitchenham, B., Linkman, S., & Law, D. (1997). DESMET: a methodology for evaluating software engineering methods and tools. Computing & Control Engineering Journal, 8(3), 120-126.