2014

Test Cases Selection Based on Source Code Features Extraction

I. Alazzam

Izzat M. Alsmadi
Texas A&M University-San Antonio, ialsmadi@tamusa.edu

M. Akour

International Journal of Software Engineering and Its Applications
Vol.8, No.1 (2014), pp.203-214
http://dx.doi.org/10.14257/ijseia.2014.8.1.18

Test Cases Selection Based on Source Code Features Extraction

Iyad Alazzam[1], Izzat Alsmadi[2] and Mohammed Akour[3]

[1]*Yarmouk University*
[2]*Prince Sultan University*
[3]*Yarmouk University*
[1]*eyadh@yu.edu.jo*, [2]*ialsmadi@cis.psu.edu.sa*, [3]*mohammed.akour@yu.edu.jo*

Abstract

Extracting valuable information from source code automatically was the subject of many research papers. Such information can be used for document traceability, concept or feature extraction, etc. In this paper, we used an Information Retrieval (IR) technique: Latent Semantic Indexing (LSI) for the automatic extraction of source code concepts for the purpose of test cases' reduction. We used and updated the open source FLAT Eclipse add on to try several code stemming approaches. The goal is to check the best approach to extract code concepts that can improve the process of test cases' selection or reduction.

1. Introduction

In many cases, it is necessary to evaluate automatically software source codes with the goal of finding some relevant information for a particular task. For example, traceability between source code and related documents such as: requirements, manual, help, designs, *etc.*, is required to check that those documents reflect the source code or vice versa. Such process can be very complex and time consuming to conduct manually.

Concept or feature extraction is also widely used in information retrieval (IR) and natural language processing (NLP) fields. Search engines for example, response to user queries and try to retrieve information that is most relevant to the searched for queries. In NLP, concept extraction is used for example to categorize or classify documents, books, articles, etc. based on general pre-defined lists.

The main research aspect in the subject of features or concepts extraction from software source code is related to: What to extract or based on what to extract. This can be subjective and user defined based on the type and the nature of the source code. This can be also generalized based on some generic aspects that can be applied to all software applications given a particular context.

Source code feature or concept extraction approaches use or develop tools to evaluate coupling or cohesion aspects between the different elements of the software. Coupling refers to the external connections. For example, for a particular method in a program, it can be coupled with all methods that it calls or all methods that use or call it. Further, it can be coupled with external variables that are used in the method or any other type of software components.

Cohesion refers to the internal relatedness or connection between the components where for example several methods in a particular class are expected to be related to each other. Such semantic relatedness is translated practically through their call or use of each other.

Cohesion and coupling metrics are used as code and design quality evaluators where a good software is expected to have low coupling and high cohesion. Software metrics and metric tools are used to gather metrics related to coupling and cohesion where several metrics

ISSN: 1738-9984 IJSEIA

are proposed in this area (*e.g.*, Briand *et al.*, [20]). However, in the scope of this paper, coupling and cohesion metrics are used feature extraction and similarity evaluation.

Similar to search in search engines, feature or concept extraction from source code can start from concepts or keywords defined by users as an input. Feature extraction tools are then used to map code elements that reflect or response to such input concepts or keywords.

The rest of the paper is organized as the following: Section two presents a literature review for papers relevant to the subject of this paper. Section three presents methodology and approaches, section four presents experiments and analysis and the paper is concluded with conclusion section.

2. Literature Review

The subject of this paper is related to several categories. First, the paper discusses the use of the Information Retrieval (IR) technique: Latent Semantic Indexing (LSI). Test cases reduction is used in this paper to direct the source code feature or concept extraction method. Perhaps the combination of those three techniques is new and hence we will present papers to the combination of those research concepts or areas.

To the best of our knowledge, the approach presented in this paper is the first attempt that utilizing IR-LSI for developing test case reduction approach based on feature extraction.

Identifying the parts of the source code that correspond to a specific functionality is known as feature or concept location [1]. This activity is commonly considered in software maintenance and evolution.

Chen and Rajlich [2] developed a semi-automated approach for locating features based on the search of program dependence graph. Other work that tackled the issue of concept or feature location include [3, 4], where they utilized reverse engineering approaches and visualization.

2.1. IR-LSI Technique for Code Features Extraction

Liu *et al.*, [5] presented a semi-automated hybrid feature location technique SIngle Trace and Information Retrieval (SITIR). They assumed that a single execution trace of a scenario, exercising a feature of interest, includes all the essential information to find the most important parts of the source code that are implementing this feature. The source code is indexed using Latent Semantic Indexing, they asked the users to write queries relevant to the desired feature and rank all the executed methods based on their textual similarity to the query. To address the accuracy of their approach two open source software (JEdit and Eclipse) were used. The result showed that the new technique has high accuracy in comparison with, comparable with previously published approaches.

Marcus *et al*., [6] Demonstrated a new approach for finding the location of desired concept in the source code by utilizing Latent Semantic Indexing (LSI). In their previous work, they used LSI to recover traceability links between external documentation and source code [7]. As the important difference is that in this application LSI is used to map domain concepts formulated as user queries to software components (*i.e.*, query to source). They evaluated the two ways of the feature location using LSI (*i.e.*, based on user formulated queries and based on partially automated generated queries).

Moreover, they compared the results of using their approach based on LSI other known methods of concept location which are based on static code analysis: a search of the program dependency graph and the traditional grep based method. As a case study they tried to locate concepts in version 2.7 of the NCSA Mosaic web browser.

2.2. Test cases' Reduction Techniques

John Regehr *et al.*, [8] started with an existing algorithms called delta debugging in order to reduce test cases in C programs that trigger compiler bugs. They designed and implemented three new domain specific test case reducers (*i.e.*, Seq-Reduce, Fast-Reduce, and C-Reduce). They compared their reducers against each other and against Berkeley delta by using 98 random generated C programs that trigger bugs in production compilers. The experiments showed how their reducer achieves the goal of producing reportable and valid test cases automatically delta debugging reducers were unsuccessful in generating sufficient small test cases.

Mahapatra and Singh [9] presented a new technique for improving the efficiency of software testing by reducing the number of test cases. They summarized their approach in 4 mains steps and assumed that their reduction steps will lead to less time to test run, and generate test cases automatically. They evaluate their technique by comparing it with Get Split algorithm technique. The result revealed that the proposed technique achieved greater reduction percentage of the test cases and kept test cases generation to a single run.

Dan Hao *et al.*, [10] Proposed on-demand test suite reduction approach that aimed to satisfy the same test requirements as an initial test suite. In order to decrease the losses in fault-detection capability after subset selection, they allow the engineer to specify upper limits on loss in fault-detection capability and confidence level. They applied their approach into eight C programs and one Java program in three scenarios. Their study showed that the proposed approach can be effective when it is applied to program versions and sets of similar programs. There was not any comparison with an existing test suite reduction approaches.

Heimdahl and Devaraj [11] addressed fault detection capability of test suite reduction for formal models of software systems. They generated reduced test-suites for a large case example of a Flight Guidance System (FGS) that seeded with faults. Their algorithm generates reduced test suites for a variety of structural coverage criteria while preserving coverage faults. Although their study results emphasize that test-suite reduction of test-suites providing structural coverage may not be effective in term of fault, still they need additional experiment to generalize their results and hypothesis.

Several techniques for test suite reduction include the heuristic algorithms, 0-1 integral programming are located in the literature [12-14]. These techniques reduce test suites by analyzing the harmony between testing requirements and test cases.

Chen *et al.*, [15] assumed that optimizing testing requirements might lead to solve the problem of test suite reduction. To achieve test suite reduction, a graph requirement relation contraction method is proposed. They conclude that the result of testing requirement optimization is no better than, but close to the result of test case reduction.

Raamesh and Uma [16] developed an algorithm that reduces test cases and produced manageable size of test suit. She addressed the potential shortcoming of existing test suites reduction approaches as they might cause high decrease in fault detection effectiveness of the reduced suite. The proposed algorithm clusters test cases based on the similarity of their execution profiles and produces some representatives to form the reduced test suite.

Test prioritizing play an important role in test suite reduction, Pravin and Srinivasan [17] presented an approach that assign priority for each test case. The priority is given depends upon the code coverage, higher priority test case value are selected to be in the reduced test suites. To demonstrate the effectiveness of their algorithm, the approach applied on two applications.

3. Methodology

In order to evaluate using IR approaches in source codes concepts extraction, we selected the open source code: http://marc4j.tigris.org/. The qualifications for such selection include: First as this is an open source Java source code and second is that it includes test cases created and provided as well by the same team, company or developers. Feature extraction are then going to be conducted based on the test cases to check the code or the part of the code that responds to the test cases and then prioritize test cases based on that. For example, test cases that have no significant code to respond to will be eliminated. MARC4j is an Application Programming Interface (API) for working with MARC (Machine Readable Cataloging) and MARCXML

Open source FLAT3 Eclipse add-on will be customized and used. FLAT3 (http://www.cs.wm.edu/semeru/flat3/) [18] uses textual static and dynamic techniques for source code features' location techniques based on users' input queries or keywords. Textual extraction is used with the assistant of Lucene library: (http://lucene.apache.org/java/docs). Dynamic feature extraction is used with the assistant of MUTT library: (http://sourceforge.net/projects/muttracer).It also includes feature annotation capabilities and uses such annotations for calculating coupling features. The tool itself uses and extends earlier tools such as ConcernMapper [19]. LSI is conducted part of the tool for evaluating textual similarities. Figure 1 shows the overall architecture of FLAT3 [18].

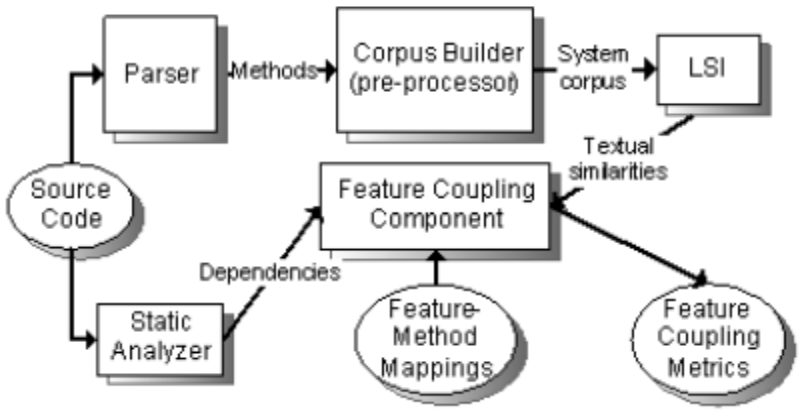

Figure 1. FLAT3 Overall Architecture [18, 22]

As the tool is open source, we modified stemming algorithms and details to evaluate the value of textual similarities based on different selections. For example, we evaluate the effect of keeping or eliminating programming reserved keywords (*e.g.*, public, private, int, *etc.*,) on the effectiveness of the stemming and textual similarity processes. Original implementation of stemming in FLAT3 includes stemming all keywords. It also include splitting compound names (*e.g.*, calculateAverage into calculate and average).

4. Case study: Experiment and Analysis

We have conducted extensive experiments using Eclipse plugin called FLAT3. This plugin is an open source, and we have made changes on it. As mentioned in the mythology we selected the open source code: http://marc4j.tigris.org/ as the testing case study. This is largely as the open source includes test cases written and published. We will evaluate

similarity between test cases and source code. We ran four different experiments. In the first one we use the plug-in as is, the second one we changed the stemming and pre processing techniques into excluding stop words and keeping the splitting identifiers, the third one including updated stop words and splitting identifiers and the fourth one including the updated stop words and excluding the splitting identifiers.

A similarity index value ranges between 0 and 1. One means identical match between the query and the retrieved text. Percentage below this then reflects the level of similarity between both elements.

The following tables show the weights (similarity) of each test class name towards the application or system under test.

Table 1. Weights of Tested Classes using FLAT^3 using Default Tool Settings

Test Class Name	Weight
DataFieldTest	3.611
ControlFieldTest	0.354
RecordTest	11.34
LeaderTest	3.898
ReaderTest	11.461
WriterTest	31.461
RoundtripTest	12.544
Total	74.669

Table 1 shows the weight of each test class towards the system under test using the FLAT3. The results show that the WriterTest class has the highest weight of 74.67. The RecordTest and ReaderTest classes have almost the same weight 11.34, 11.461 accordingly. The results also show that the ControlFieldTest has the lowest weight. Those weights are evaluated based on the test cases. For example, a class with the highest weight in this table means that it is getting the highest percentage of test cases in comparison with other classes. Table 2 shows the weight of each test class towards the system under testing after modifying the stemming and pre processing.

Table 2. Weights of Tested Classes Excluding Stop Words and Including the Splitting Identifiers

Test Class Name	Weight
DataFieldTest	3.862
ControlFieldTest	0.354
RecordTest	12.36
LeaderTest	4.024
ReaderTest	11.0544
WriterTest	29.583
RoundtripTest	13.26
Total	74.4974

Here the stops words are removed and the splitting identifiers are kept. The results show that the weights are just a bit less than the weights in Table 1. Table 3 shows the weight of each test class towards the system under testing after modifying the stemming and pre processing.

Table 3. Class Weights with Updating Stop Words and including the Splitting Identifiers

Test Class Name	Weight
DataFieldTest	3.611
ControlFieldTest	0.354
RecordTest	11.833
LeaderTest	3.898
ReaderTest	10.222
WriterTest	32.095
RoundtripTest	10.53
Total	72.543

In this third case, the stops words are updated by adding new stop words and the splitting identifiers are kept. The results show that the weight of RecordTest class is greater than the weight of the RoundtripTest class where in the previous results the weight of the RoundtripTest class is always greater than the weight of RecordTest class. Table 4 shows the weight of each test class towards the system under testing after modifying the stemming and pre processing activities.

Table 4. Class Weights with Modifying Stop Words List and Excluding the Splitting Identifiers

Test Class Name	Weight
DataFieldTest	4.485
ControlFieldTest	0.283
RecordTest	3.163
LeaderTest	1.672
ReaderTest	8.349
WriterTest	29.082
RoundtripTest	6.444
Total	53.478

In the fourth case, the stops words are updated by adding new stop words and the splitting identifiers are excluded. The results show that there is a significant change according the previous results. The weights are decreased clearly. This leads to the conclusion that the splitting identifiers process has a high impact on the results. Table 5 shows the weight of each test case towards the system under test using the FLAT3.

Table 5. Test Cases' Weight

Test Case Name	Weight	Test Case Name	Weight
testConstructor	1.655	testMarcStreamReader	3.462
testSetData	0.354	testMarcStreamWriter	11.593
testComparable	0.7	testMarcXmlReader	7.999
testGetFields	0.533	testWriteRead	6.428
testFind	9.593	testWriteReadUtf8	6.116
testAddSubfield	0.599	testMarcXmlWriter	10.356
testMarshal	1.424	testWriteAndRead	9.512
testUnmarshal	2.474	testSetSubfield	0.657
testCreateRecord	1.214		
sum		74.669	

The results show that the test case testMarcStreamWriter has the highest weight which means that this test case is related more to the system under test. The second highest weight is for the testMarcXmlWriter test case. Whereas the testSetData test case has the lowest weight which is 0.354.

Table 6. Excluding Stop Words and Including the Splitting Identifiers

Test Case Name	Weight	Test Case Name	Weight
testConstructor	1.724	testMarcStreamReader	2.7104
testSetData	0.354	testMarcStreamWriter	11.038
testComparable	0.7	testMarcXmlReader	8.344
testGetFields	0.627	testWriteRead	6.63
testFind	9.85	testWriteReadUtf8	6.63
testAddSubfield	0.678	testMarcXmlWriter	9.343
testMarshal	2.53	testWriteAndRead	9.202
testUnmarshal	1.494	testSetSubfield	0.76
testCreateRecord	1.883		
sum		74.4974	

Table 6 shows the weight of each test case towards the system under testing after modifying the stemming and pre processing. Here the stops words are removed and the splitting identifiers are kept. The results show that the weights are just a bit less than the weights in Table 5.

Table 7. Updated Stop Words and Including the Splitting Identifiers

Test Case Name	Weight	Test Case Name	Weight
testConstructor	1.655	testMarcStreamReader	2.533
testSetData	0.354	testMarcStreamWriter	11.283
testComparable	0.7	testMarcXmlReader	7.689

testGetFields	0.533	testWriteRead	5.265
testFind	9.593	testWriteReadUtf8	5.265
testAddSubfield	0.599	testMarcXmlWriter	9.427
testMarshal	2.424	testWriteAndRead	11.385
testUnmarshal	1.474	testSetSubfield	0.657
testCreateRecord	1.707		
sum		72.543	

Table 7 shows the weight of each test case towards the system under testing after modifying the stemming and pre processing. Here the stops words are updated by adding new stop words and the splitting identifiers are kept. The results show that the weight of testWriteAndRead test case is greater than the weight of the testMarcStreamWriter test case in just a bit.

Table 8. Updated Stop Words and Excluding the Splitting Identifiers

Test Case Name	Weight	Test Case Name	Weight
testConstructor	1.926	testMarcStreamReader	1.349
testSetData	0.283	testMarcStreamWriter	11.112
testComparable	0.65	testMarcXmlReader	7
testGetFields	0.391	testWriteRead	3.222
testFind	1.054	testWriteReadUtf8	3.222
testAddSubfield	0.849	testMarcXmlWriter	8.461
testMarshal	1.006	testWriteAndRead	9.509
testUnmarshal	0.666	testSetSubfield	1.06
testCreateRecord	1.718		
Total		53.478	

Table 8 shows the weight of each test case towards the system under testing after modifying the stemming and pre processing. Here the stops words are updated by adding new stop words and the splitting identifiers are excluded. The results show that there is an obvious change according the previous results. The weights are decreased clearly. This leads that the splitting identifiers has high impact on the results.

5. A comparison Study

We will compare experiment in our paper with two case studies:case1. Revelle *et al.*, [23] which describes the usage of FLAT3 tool. Case 2.Liu *et al.*, [24] which shows how to locate feature by using information retrieval based on filtering of a single scenario execution trace

5.1. Case 1:

The authors of this paper propose and define feature coupling metrics derived from two different sources of information: 1.Structural Feature Coupling (SFC) which gets and obtains the association among two features according to the structure information. 2. Textual Feature

Coupling (TFC) captures the relationships among two features according to the textual information in source code by using Latent Semantic Indexing technique (LSI).

5.1.1 Case Study and Methodology

They have done three case studies; the first case study investigates the association among fault proneness and feature coupling by computing the relationship among bugs and the metric values for every single pairs of features in dvViz nad Rhino applications. The second case study explores the impact analysis in the feature coupling metrics to determine if other features are affected by another feature which is currently modified. The third case study is a survey where done among 31 programmers to evaluate the power of coupling among 16 chosen pairs randomly of features from dbViz, Rhino, and iBatis applications. dbViz7 is an open source code for database visualization written in Java, it includes ninety three classes implemented in five hundred and fifty four methods with twelve thousands and seven hundreds lines of code. Rhino includes one hundred and thirty eight classes implemented in one thousand and eight hundreds methods with thirty two thousands line of code in java script. iBatis is an object-relational mapping tool consists of two hundreds and twelve classes implemented in more than one thousand and eight hundreds methods with thirteen thousands and three hundreds of lines of code.

5.1.2. Results

The results of all three studies show that the feature coupling metrics are certainly practical and useful in evaluating the impact of change, directing and guiding the testing process, locating and finding bugs.

5.2. Case 2:

The authors of this paper propose a hybrid approach for feature location called SIngle Trace and Information Retrieval (SITIR). The approach is based on execution just a single scenario that employs a specific feature and then traces that execution, then using the Latent Semantic Indexing (LSI) technique to do textual analysis on the traces in order to obtain the related source code regarding to the executed scenario.

5.2.1. Case Study and Methodology

They have done two cases studies to assess the performance of their approach (SITIR). The first case study includes the locating three features (“Search, Add marker and Show whitespace”) in JEdit application related with change requests using two techniques LSI and SITR. JEdit is an open source code for text editor; it includes five hundreds classes implemented in five thousand methods and eighty eight thousands lines of code written in Java. The second case study includes locating three features (“Select, Add files and Search”) in Eclipse related to the bugs using three techniques LSI, PROMESIR, SITIR and SPR. Eclipse is an open source code for integrated development environment; it includes seven thousands classes implemented in eighty nine thousands methods and more than eight thousand source code file and more than two millions and four hundred thousand line of code written mainly in Java with a bit of C and C++.

5.2.2. Results

In the first case study the results indicate that the SITIR is notably less sensitive to the weak user quires than LSI merely. In the second case the results show that SITIR exceeds

SPR and LSI in finding bugs associated features in Eclipse, also the results show that SITIR and PROMESIR are similar in locating bugs.

6. Conclusion

In this paper, we propose a new technique for test case selection and reduction based on feature or concept testing by using information retrieval, latent semantic indexing. Our approach uses LSI to find the semantic similarity among source code and test cases in order to find the test case that has the highest weight. A case study of finding the semantic analysis among test cases and source code in MARC4j is analyzed and presented. We have tried and compared the results of four different steaming approaches; the first one we use the FLAT3 plug-in as is, the second one we changed the stemming and pre processing techniques into excluding stop words and keeping the splitting identifiers, the third one including updated stop words and splitting identifiers and the fourth one including the updated stop words and excluding the splitting identifiers. The results show that splitting identifiers has high impact on the semantic similarity.

References

[1] N. Wilde and M. Scully, "Software Reconnaissance: Mapping Program Features to Code", Software Maintenance: Research and Practice, vol. 7, **(1995)**, pp. 49-62.
[2] K. Chen and V. Rajlich, "Case Study of Feature Location Using Dependency Graph", Proceedings of Intern. Workshop on Program Comprehension (IWPC'00), **(2000)**, pp. 241-249.
[3] R. Fiutem, P. Tonella, G. Antoniol and E. Merlo, "A Cliche'-Based Environment to Support Architectural Reverse Engineering", Proceedings of Intern Conference on Software Maintenance (ICSM '96), **(1996)** November 04-08, pp. 319-328.
[4] K. Lukoit, N. Wilde, S. Stowell and T. Hennessey, "TraceGraph: Immediate Visual Location of Software Features", Proceedings of International Conference on Software Maintenance (ICSM'00), San Jose, **(2000)** October 11-14, pp. 33-39.
[5] D. Liu, A. Marcus, D. Poshyvanyk and V. Rajlich, "Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace", Proceedings of 22nd IEEE/ACM International Conference on Automated Software Engineering ASE, Atlanta, Georgia, **(2007)**.
[6] A. Marcus, A. Sergeyev, V. Rajlich and J. I. Maletic, "An Information Retrieval Approach to Concept Location in Source Code", Proceedings of the 11th Working Conference on Reverse Engineering, WCRE, isbn0-7695-2243-2, IEEE Computer SocietyWashington, DC, USA, **(2004)**, pp. 214-223.
[7] A. Marcus and J. I. Maletic, "Recovering Documentationto- Source-Code Traceability Links using Latent Semantic Indexing", Proc Intern Conference on Software Engineering (ICSE'03), Portland, OR, **(2003)** May 3-10, pp. 125-137.
[8] J. Regehr, Y. Chen, P. Cuoq, E. Eide, C. Ellison and X. Yang, "Test-Case Reduction for C Compiler Bugs", Proceedings of 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation PLDI, Beijing, China, PLDI, ACM isbn978-1-4503-1205-9, **(2012)** June.
[9] R. P. Mahapatra and J. Singh, "Improving the Effectiveness of Software Testing through Test Case Reduction", **(2008)**.
[10] D. Hao, L. Zhang, X. Wu, H. Mei and G. Rothermel, "On-Demand Test Suite Reduction", Proceedings of the International Conference on Software Engineering, **(2012)** June, pp. 738-748.
[11] M. P. E. Heimdahl and D. George, "Test Suite Reduction for Model Based Tests: Effects on Test Quality and Implications for testing", Proc. 19th Intl. Conf. on Automated Software Engineering, **(2004)**, pp. 176-185.
[12] C. and L., "A new heuristic for test suite reduction", Information and Software Technology, vol. 40, no. 5, **(1998)**, pp. 347-354.
[13] H., G. and S., "A methodology for controlling the size of a test suite", ACM Trans. on Soft. Eng. and Meth, vol. 2, no. 3, **(1993)**, pp. 270-285.
[14] L. and C., "An optimal representative set selection method", Information and Software Technology, vol. 42, no. 1, **(2000)**, pp. 17 25.
[15] Z. Chen, B. Xu, X. Zhang and C. Nie, "A novel approach for test suite reduction based on requirement relation contraction", Proceedings of the 2008 ACM symposium on Applied computing, isbn978-1-59593-753-7,Fortaleza, Ceara, Brazil}, ACM, **(2008)**, pp. 390-394.

[16] R. and U., "An Efficient Reduction Method for Test Cases", International Journal of Engineering Science and Technology, vol. 2, no. 11, **(2010)**, pp. 6611-6616.
[17] P. and S., "An Efficient Algorithm for Reducing the Test Cases which is Used for Performing Regression Testing", 2nd International Conference on Computational Techniques and Artificial Intelligence (ICCTAI'2013), Dubai (UAE), **(2013)** March 17-18.
[18] M. Revelle, "Supporting Feature-Level Software Maintenance", WCRE, **(2009)**, pp. 287-290.
[19] M. P. Robillard and F. Weigand-Warr, "Concernmapper: Simple view-based separation of scattered concerns", Proceedings of the Eclipse Technology Exchange at OOPSLA (ETX'05), **(2005)**, pp. 65-69.
[20] L. C. Briand, J. Daly and J. Wust, "A unified framework for coupling measurement in object oriented systems", IEEE Transactions on Software Engineering, vol. 25, no. 1, **(1999)**, pp. 91-121.
[21] A. Ohno and H. Murao, "Measuring Source Code Similarity Using Reference Vectors", ICICIC, vol. 2, **(2006)**, pp. 92-95.
[22] T. Savage, M. Revelle and D. Poshyvanyk, "FLAT3: feature location and textual tracing tool", ICSE, vol. 2, **(2010)**, pp. 255-258.
[23] M. Revelle, M. Gethers and D. Poshyvanyk, "Using structural and textual information to capture feature coupling in object-oriented software", Empirical Software Engineering, vol. 16, no. 6, pp. 773-811, **(2011)**.
[24] D. Liu, A. Marcus, D. Poshyvanyk and V. Rajlich, "Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace", Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, **(2007)**, pp. 234-243.

Authors

Iyad M Alazzam, is an assistant professor in the department of computer information systems at Yarmouk University in Jordan, he has received his Ph.D degree in software engineering from NDSU (USA). His master from LMU (UK) in electronic Commerce and his B.Sc in computer science and information systems from Jordan University of Science and Technology in Jordan. His research interests lays in software engineering and software testing.

Izzat Alsmadi, is an associate professor in the department of information systems at Prince Sultan University in Saudi Arabia. He obtained his Ph.D degree in software engineering from NDSU (USA). His second master in software engineering from NDSU (USA) and his first master in CIS from University of Phoenix (USA). He had a B.sc degree in telecommunication engineering from Mutah university in Jordan. He has several published books, journals and conference articles largely in software engineering different fields.

Mohammed Akour, is an assistant professor in the department of computer information systems at Yarmouk University in Jordan, he has received his Ph.D degree in software engineering from NDSU (USA). His master and B.Sc in computer information systems from Yarmouk University in Jordan. His research interests focuses in software engineering and adaptive software testing.