WeMuTe – A Weak Mutation Testing Tool for WS-BPEL

Panya Boonyakulsrirung and Taratip Suwannasart

Abstract—**Web Service rapidly grows and is implemented by many famous enterprises. Web Service Business Process Execution Language (WS-BPEL) appears to solve and support more complex business processes. Mutation Testing has occurred for few decades to justify if the test data are sufficient to test the program or a component of code in the program. With fault-based testing method, mutation testing can help testers improve effectiveness of the test cases. Weak mutation testing is a kind of mutation testing which aims at reducing computational cost. Weak mutation considers only a component in the program, mutates it, and finally compares the results between the original program and the mutant. In our previous work [1], we analyzed a set of mutation operators, which can be used for weak mutation with WS-PEL and introduce a framework for weak mutation. This paper we continue our work by proposing a weak mutation tool for WS-BPEL, which supports mutant generation, running test cases against the mutants, as well as specifying live, killed, and equivalent mutants. The tool has been tested with six BPEL programs.**

Index Terms— **Web Service; WS-BPEL; Mutation Testing; Mutation Operators; Weak Mutation Testing**

I. INTRODUCTION

MANY organizations begin move from Object-Oriented paradigm to Service-Oriented paradigm (SOA) [2]. As a plenty of systems and applications increase, those organizations encounter challenges ranging from integration, and maintaining legacy systems. With loosely coupled and easy to reusable benefits make SOA implemented in various software industries.

SOA is made more concrete by executing web services, which are XML (Extensible Mark-up Language) [3] -based syntax and are able to integrate various applications in different platforms. Web Services communicate each other by using SOAP (Simple Object Access Protocol) [4] and WSDL (Web Service Description Language) [5] explaining web service characteristics for example; which partners this web service calls, which operations this web service provide.

Manuscript received January 09, 2012; revised January 16, 2012.

Panya Boonyakulsrirung is with Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok Thailand (e-mail: panya.javamania@gmail.com).

Taratip Suwannasart is with Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok Thailand (e-mail: taratip.s@chula.ac.th).

Nowadays various enterprises are developing more complicate, therefore, web services no longer meet requirements. Thus, Web Service Execution Business Process Execution Language (WS-BPEL) [6], which is also an XML document, are proposed to support more structure and more complex business processes. These advantages allow many applications integrate and cooperate with each other effectively.

Mutation testing has become a prominent testing strategy for thirty years ago and trends to increase in the future [7]. Mutation testing is performed by feeding a fault into a program. Then the program becomes a mutated program or a mutant. Test data or test cases are run against the mutant and the results between the original program and the mutant are compared. If the results of the original program and the mutant are the same, the mutant is dead, otherwise it is live. We must create new test cases or test data to kill the mutant. If the mutant cannot be killed by any test cases, the mutant is called the equivalent mutant. Weak Mutation Testing is another category of mutation testing which considers only a component in the program in order to decrease the computational cost.

There are numbers of researches has focused on testing web services. Natthapol and et al [8] defined expression mutation operators for WS-BPEL in selective mutation testing and developed an environment to support selective mutation testing technique for WS-BPEL. Antonia Estero-Botaro and et al [9] defined more three categories of mutation operators for WS-BPEL. Those are Identifier Mutation Operator, Activity Mutation Operators, as well as Exceptional and Event Mutation Operators.

Both two researches that mentioned above has focused on strong mutation testing, but Boonyakulsrirung P. and et al [1] proposed a weak mutation testing framework for WS-BPEL and also analyzed all categories of mutation operators proposed by [9], [10]. We concluded that there are twenty-six mutation operators that can be used in weak mutation technique for WS-BPEL.

We continue our on-going research by developing a weak mutation tool for WS-BPEL. In this paper, we propose the tool called WeMuTe that allows testers uploading WS-BPEL as an original program, selecting mutation operators, generating mutants from the uploaded original WS-BPEL program, uploading test cases, running test cases against the original program and mutants, as well as making live, killed, and equivalent mutants.
In section II, we give a brief description of WS-BPEL. Section III and IV illustrate mutation testing and weak mutation testing, respectively. We propose our weak

mutation tool in section V and the implementation in section VI. Section VII describes our experiments and results. Finally, our conclusions and future work are presented in Section VIII.

II. WS-BPEL LANGUAGE

Web Service Business Process Execution Language (WS-BPEL) is used for defining business processes, orchestrating web services to work together to produce more complex and structure software. WS-BPEL consists of two types of activity as describe below:

First, the basic activity includes *assign*, *invoke*, *receive*, and *reply*. The *assign* activity is used for copying from a variable to another. The *invoke* activity invokes with web service partner with ant operation. The receive activity receives the message from the outside process. The *reply* activity replies a message to outside process.

Second, the structure activity includes *if-Else*, *flow*, *forEach*, *pick*, *repeatUntil*, *sequence*, *while*. The *if-Else* activity provides conditional behavior. The *flow* activity provides concurrency and synchronization. The *flow* activity is completed when all child activities within the *flow* are executed. The *forEach* activity provides a scenario that needs to interact with a set of partners in parallel, and the partners are dynamically examined at runtime. The *pick* activity waits for the occurrence of precisely one event from a group of events, and then executes the activity related with that event. The *repeatUntil* activity provides repeated execution of a group of activities. The child activity is executed until the Boolean expression or statement becomes true. The sequence activity provides a sequential activity that contains child activity with a specific order. The *while* activity is the same as *repeatUntil* activity but it checks Boolean statement at first before executing the child activity.

III. MUTATION TESTING

Mutation Testing [11], [12], [13], [14], [15] is fault-seeding method to generate a mutated program from an original program. Then, both original and mutated programs are executed against test cases or test data. The original program is mutated based on mutation operators. There are many categories of mutation operators, for example arithmetic expression mutation operators, and relational expression mutation operators. The arithmetic expression mutation operators is used by replacing an arithmetic operator (+, -, *, /, %) in an expression or a statement with another one. The relational expression mutation operator create mutants by replacing a relational operator (=, !=, <, >, <=, >=) in expression or statement by another one.

We consider if the mutant is killed by comparing its results with the original program. If the results of the mutant are different from the original program with same input data, the mutant will killed. Otherwise, the mutant is live. There is another type of mutant called equivalent mutant which produces outputs that are the same as the original program.

Mutation testing is not used only for generating mutants, but is also used for assisting testers to improve their test cases by considering a measurable value called mutation score (MS) which represents a ratio of dead mutants (D) divided by a difference total mutants (T) and equivalent mutants (E) as shown in Equation (1) belows:

$$MS = \frac{D}{T - E} \tag{1}$$

IV. WEAK MUTATION TESTING

This analysis method is another kind of mutation testing [16], [17], [18] with the same concept by feeding fault in a program and considering only the results of the component around the fault. Offut and et al [19], [20] proposed four types of components which can be considered when we want to create a mutant. The components can be considered as locations of a program that the fault can be seeded and the results can be compared. These components or locations are illustrated as follows:

1) *EX-WEAK/1 (Expression-WEAK/1 execution) Mutation*

The result of the expression that is surrounded with the mutation operator is compared with the result of the expression of the original program. For example, the original innermost expression is A = (B - C) * D and the expression of the mutant is A = (B + C) * D. Subsequently, the execution of innermost expression value of (B - C) and (B + C) must be compared.

2) *ST-WEAK/1 (Statement-WEAK/1 execution) Mutation*

The result of the first execution of the mutated statement is compared with the result of the statement in the original program. For instance, the original statement is ((X != Y) && (Y == Z) && (X != 0)) and the mutant is ((X == Y) && (Y==Z) && (X != 0)). The value of both statements would be compared after the first execution.

3) *BB-WEAK/1 (Basic-Block-WEAK/1 execution) Mutation*

The third type focuses on a basic-block as a maximal sequence of instructions with one entry and one exit in a program which is the biggest component. Basic block in WS-BPEL language can consider as activities including *forEach, repeatUntil,* and *while.* After the first execution of basic block is finished, values or some variables in the block would be compared between the original and the mutated block.

4) *BB-WEAK/N (Basic-Block-WEAK/N execution) Mutation*

Last type of weak mutation technique is BB-WEAK/N which is similar to BB/WEAK/1 except that it allows multiple executions of mutated basic-block. The BB-WEAK/N compares results of executing of each iteration between the original and the mutated block.

V. WEAK MUTATION TESTING TOOL

In our previous work, we proposed the framework for weak mutation testing for WS-BPEL. Figure 1 illustrates our framework, which is composed of seven components: BPEL Validator, Mutant Generator, Mutant Controller, State

Comparator, Execution timer, Mutation Score Calculator, and Test Cases Effectiveness Calculator. Functionalities of each component are described in details in [1].

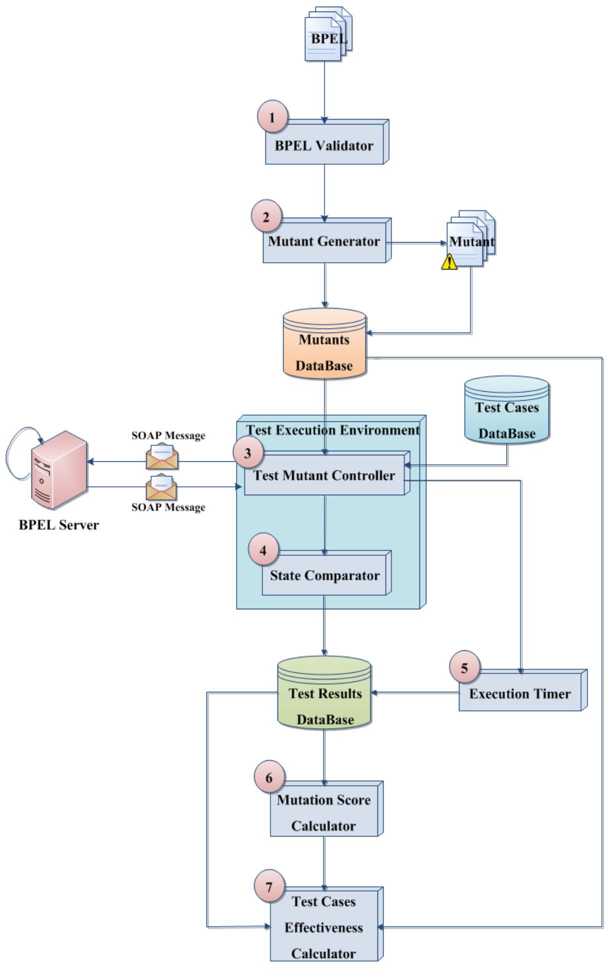


Fig. 1 Overview our framework

In this paper, we elaborate an important component: ***"State Comparator"***, that is responsible for finding the results of an original program and the mutants generated using four types of mutation operators mentioned above for WS-BPEL. The state comparator consists of two sub-components shown is figure 2 and described as follows:

a) *WeMuTe Expression Parser (WEP):* This component considers only an expression and a statement of an original program and mutants which are from Test Mutant Controller. For instance, if the original expression is $input.A + $input.B and a mutant expression is $input.A - $input.B. The state comparator will retrieve test data to evaluate the original and the mutated expression and. From the example the test data for A is 5 and for B is 2. The original expression result is 5 + 2 = 7 and the mutated result is 5 – 2 = 3. After that, the results are sent to the next sub-component to be compared.

b) *Result Comparison*: After WeMuTe Expression Parser component has already evaluated results of the original and the mutants, the results are sent to this component to be compared. If the results are same, it is implicit that the mutants are live. On the other hand, if the results are different the mutants are killed.

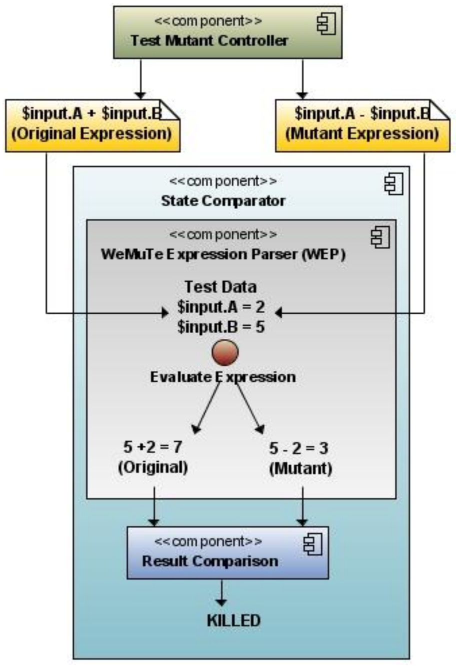


Fig. 2 Details State Comparator

VI. WeMuTe Implementation

The proposed weak mutation tool for WS-BPEL has been implemented as a Web-based tool called WeMuTe. WeMuTe allows testers to perform weak mutation testing technique with WS-BPEL by uploading a WS-BPEL program, choosing a type of weak mutation analysis, selecting mutation operators, uploading test cases or test data, and then WeMuTe will display the results of the testing. WeMuTe features can be described as follows:

1) Uploading BPEL file UI: This feature allows testers to browse for upload a zip file which contains BPEL file, WSDL file, and other optional files such as XML Schema or XML file as shown in Figure 3.

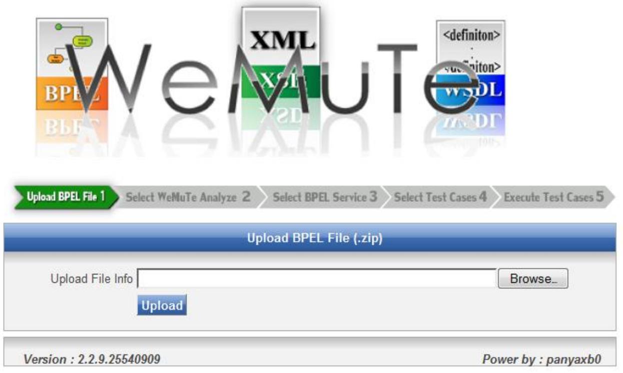


Fig.3 Uploading BPEL zip file UI

2) Select Weak Mutation Analysis Type UI: This page allows testers to choose a type of weak mutation analysis and select mutation operators under the chosen weak mutation analysis type as shown in figure 4. Weak mutation analysis types that WeMuTe provides are described below

a) Expression Weak – 1: This type considers identifier and expression operators, which consist of ISV, EAA, ERR, EEU, ELL, ECC, ECN, EMD, and EMF.

b) Statement Weak – 1: This type focuses activity operators and exceptional operators that include AIE, AJC, APA, APM, XMF, XMC, XMT, XTF, XER, and XEE.

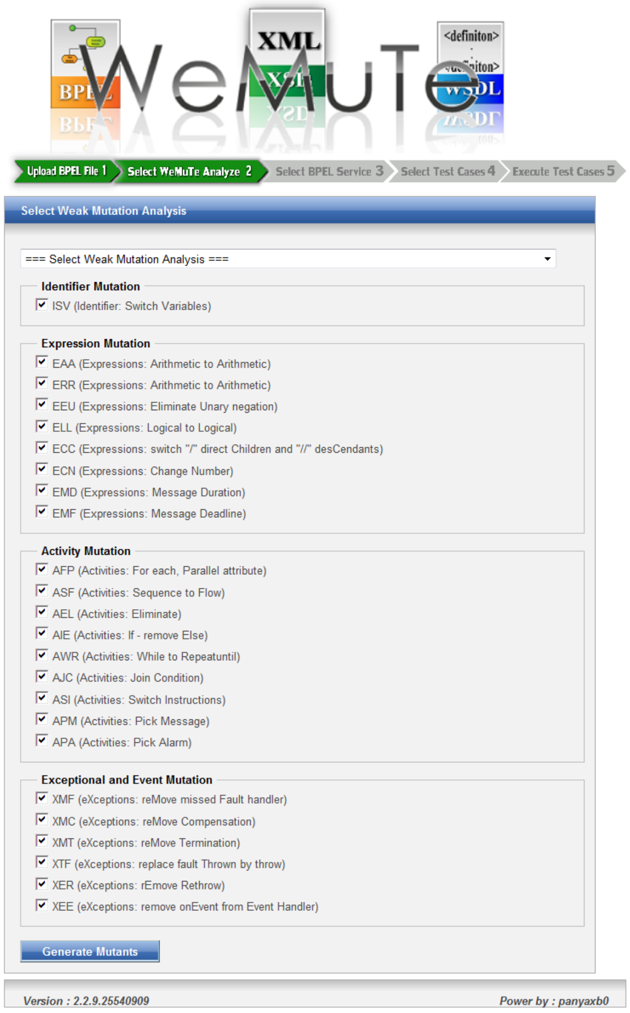
c) Basic-Block Weak – 1: This type considers additional activity operators that related to iteration such as AWR, AEL, AFP, ASF, and ASI.

d) Basic-Block Weak – N: This type focuses on the same mutation operators as Basic-Block Weak – 1.

3) Selecting Test Cases UI: This part permits testers to load prepared test cases for testing an original program, and mutants respectively as shown in figure 5.

4) Executing Test Mutant UI: This feature is performed after weak mutation testing mutants task is finished. A testing result is displayed as a table shown in table 1 with several columns that are mutant names, mutant killed, original expression, mutant expression, result in original expression, and result mutant in expression.

In this work, we propose the complete WeMuTe tool and try five more BPEL programs and display summary results of total mutant for each mutation operators, and summarized testing results



Identifier Mutation
- [✓] ISV (Identifier: Switch Variables)

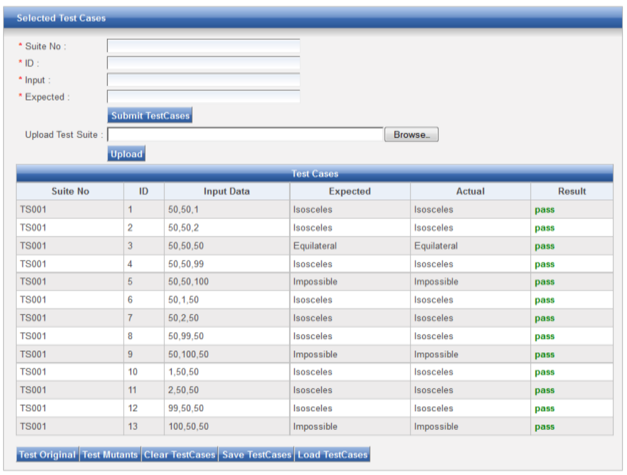
Expression Mutation
- [✓] EAA (Expressions: Arithmetic to Arithmetic)
- [✓] ERR (Expressions: Arithmetic to Arithmetic)
- [✓] EEU (Expressions: Eliminate Unary negation)
- [✓] ELL (Expressions: Logical to Logical)
- [✓] ECC (Expressions: switch "/" direct Children and "//" desCendants)
- [✓] ECN (Expressions: Change Number)
- [✓] EMD (Expressions: Message Duration)
- [✓] EMF (Expressions: Message Deadline)

Activity Mutation
- [✓] AFP (Activities: For each, Parallel attribute)
- [✓] ASF (Activities: Sequence to Flow)
- [✓] AEL (Activities: Eliminate)
- [✓] AIE (Activities: If - remove Else)
- [✓] AWR (Activities: While to Repeatuntil)
- [✓] AJC (Activities: Join Condition)
- [✓] ASI (Activities: Switch Instructions)
- [✓] APM (Activities: Pick Message)
- [✓] APA (Activities: Pick Alarm)

Exceptional and Event Mutation
- [✓] XMF (eXceptions: reMove missed Fault handler)
- [✓] XMC (eXceptions: reMove Compensation)
- [✓] XMT (eXceptions: reMove Termination)
- [✓] XTF (eXceptions: replace fault Thrown by throw)
- [✓] XER (eXceptions: rEmove Rethrow)
- [✓] XEE (eXceptions: remove onEvent from Event Handler)

Version : 2.2.9.25540909 *Power by : panyaxb0*

Fig. 4 Selecting Weak Mutation Analysis Type UI



Suite No	ID	Input Data	Expected	Actual	Result
TS001	1	50,50,1	Isosceles	Isosceles	pass
TS001	2	50,50,2	Isosceles	Isosceles	pass
TS001	3	50,50,50	Equilateral	Equilateral	pass
TS001	4	50,50,99	Isosceles	Isosceles	pass
TS001	5	50,50,100	Impossible	Impossible	pass
TS001	6	50,1,50	Isosceles	Isosceles	pass
TS001	7	50,2,50	Isosceles	Isosceles	pass
TS001	8	50,99,50	Isosceles	Isosceles	pass
TS001	9	50,100,50	Impossible	Impossible	pass
TS001	10	1,50,50	Isosceles	Isosceles	pass
TS001	11	2,50,50	Isosceles	Isosceles	pass
TS001	12	99,50,50	Isosceles	Isosceles	pass
TS001	13	100,50,50	Impossible	Impossible	pass

Fig. 5 Selecting Test Cases UI

VII. Experiments and Results

After we have implemented WeMuTe tool with six WS-BPEL programs which are Triangle, SimpleCalculator, LoanApproval, ATM, ShopProduct, and TravelReservation. Figure 6 demonstrates the characteristics of WS-BPEL programs. There are three values for each program which are Line of Codes, Total Mutants, and Killed Mutants. Details are described as follow:

- Triangle program generated the most numbers of mutants than others because Triangle program contains much more statements and expressions than other programs.
- On the other hand, the programs except Triangle are not much emphasized on logic checking. Logic checking activities in WS-BPEL includes *if*, *while*, and *repeatUntil*. Therefore, numbers of produced mutants in SimpleCalculator, LoanApproval, ATM, ShopProduct, and TravelReservation programs are significantly less than Triangle.

Figure 7 shows another results of testing, there are Execution Time, Mutation Score, and Test Cases Effectiveness. Explanations are described below:

- Obviously, Triangle program has more computational cost than others because the program produces many expression and statement mutants.

Accidentally, mutation score and test cases effectiveness are in the same line due to we use one test suite in our testing tool.

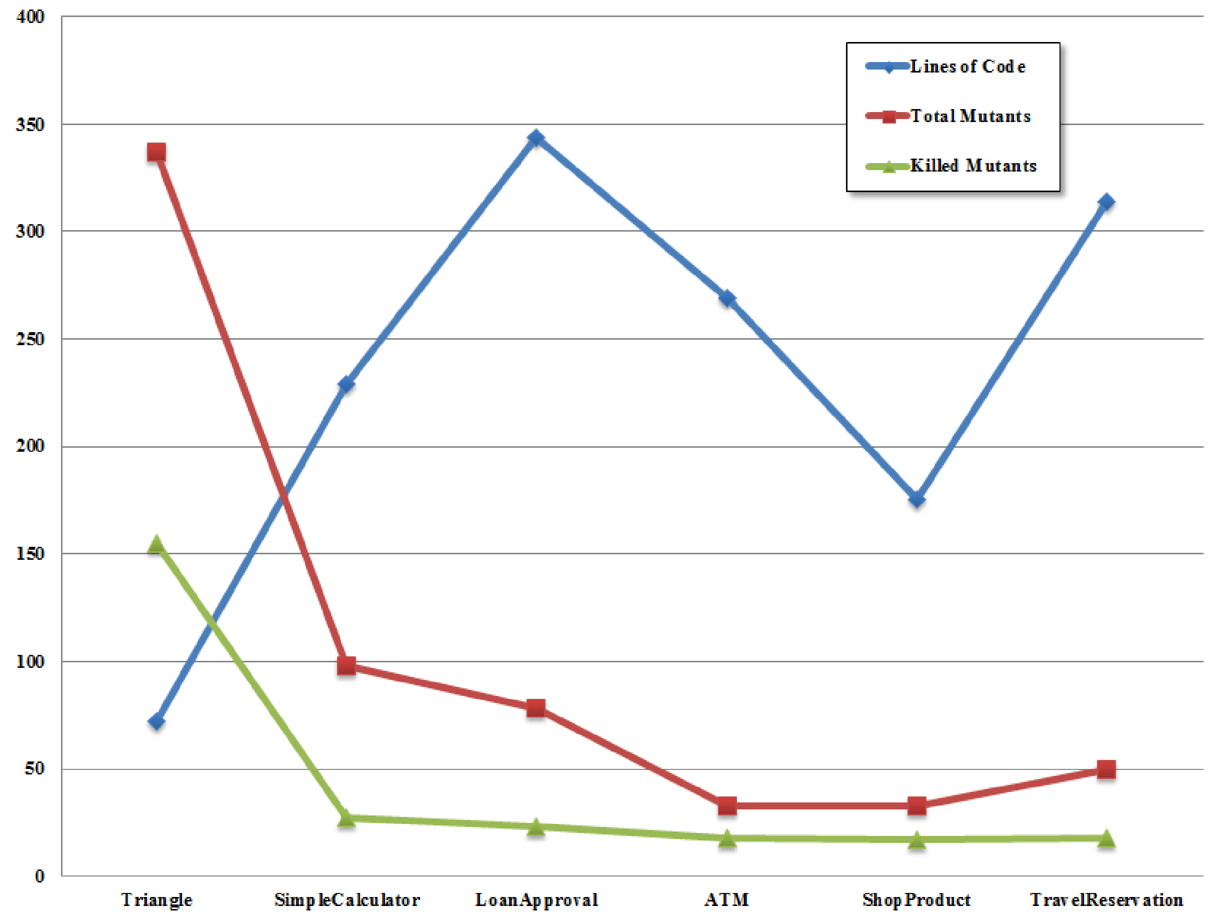


Fig. 6 Summary of Experiment I

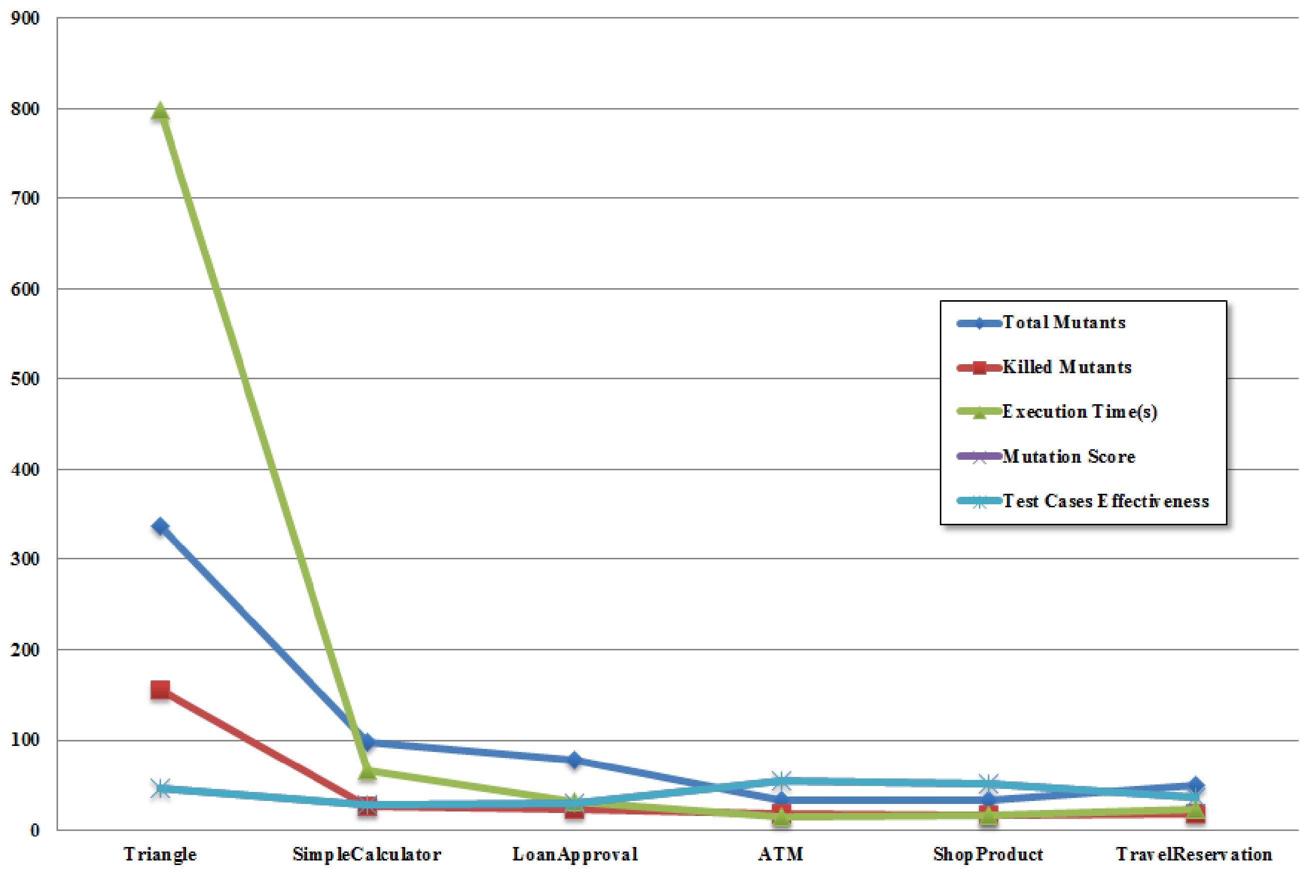


Fig. 7 Summary of Experiment II

VIII. LIMITATIONS

Although, the tool can automatically produce mutants and support all four type of weak mutation techniques. Our tool also has some restrictions below:

- The tool now generates test cases manually. Tester needs to create own test data and places it in a specified location.
- Even though WeMuTe can generate many mutants from WS-BPEL programs against four types of mutation analysis, it can now use only one test suite.
- WeMuTe now cannot identify equivalent mutants.

IX. CONCLUSIONS AND FUTURE WORKS

This paper proposes a weak mutation testing tool for WS-BPEL called WeMuTe. We have designed and developed WeMuTe from our proposed framework in the previous work. Our weak mutation tool covers all types of weak mutation analysis techniques. This tool is still our on-going work. We have experienced the tool with six WS-BPEL programs.

In section V, we re-introduce overall image of WeMuTe and deep down into WeMuTe Expression Parser functions.

In section VI, we demonstrate some user interfaces of WeMuTe tool and brief detail for each component one by one.

We give the results of weak mutation techniques and explain some different information among the WS-BPEL processes in section VII.

In section VIII, we explain about constraints of our tool and expect to improve the functions hereafter.

For our future works, we plan to continue solving some difficulties, improving tool performance, and to experience with more complex WS-BPEL programs.

REFERENCES

[1] Panya Boonyakulsrirung and Taratip Suwannasart, “A Weak Mutation Testing Framework for WS-BPEL,” Computer Science and Software Engineering (JCSSE), 2011 Eighth International Joint Conference on Date:11-13 May 2011, pp. 313-318.

[2] OASIS Standard, “Reference Model for Service Oriented Architecture 1.0”, 12 October 2006, http://docs.oasis-open.org/soa-rm/v1.0/

[3] W3C Recommendation, “Extensible Markup Language (XML) 1.0 (Fifth Edition)”, 26 November 2008, http://www.w3.org/TR/REC-xml/J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[4] W3C Recommendation, “SOAP Version 1.2 Part 0: Primer (Second Edition)”, 27 April 2007, http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[5] W3C Recommendation, “Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language”, 26 June 2007, http://www.w3.org/TR/2007/REC-wsdl20-20070626/.

[6] OASIS, “Web Services Business Process Execution Language 2.0”, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007, Organization for the Advancement of Structured Information Standards.

[7] Y. Jia and M. Harman, “An Analysis and Survey of the Development of Mutation Testing”, CREST Center. King’s College, London, Tech. Rep. TR-09-06, 2009.

[8] Natthapol Thaisakonpun and Taratip Suwannasart, “Mutation Testing for Expression Modification Operator of BPEL” Software Engineering Laboratory, Center of Excellence in Software Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand.

[9] Antonia Estero-Botaro, Francisco Palomo-Lozano, and Inmaculada Medina-Bulo, “Quantitative Evaluation of Mutation Operators for WS-BPEL Compositions” Department of Computer Languages and Systems, University of C?adiz, C?adiz, Spain.

[10] A. Estero-Botaro, F. Palomo-Lozano, and I. Medina-Bulo,“Mutation operators for WS-BPEL 2.0,” in ICSSEA 2008: 21th International Conference on Software & Systems Engineering and their Applications, Paris, France, 2008.

[11] H. Agrawal, R. Demillo, R. Hathaway, W. Hsu, W. Hsu, E. Krauser, R. J. Martin, A. Mathur, and E. Spafford, “Design of mutant operators for the C programming language,” Software

Engineering Research Center, Department of Computer Science, Purdue University, Indiana, Tech. Rep. SERC-TR-41-P, 1989.

[12] K. N. King and A. J. Offutt, “A FORTRAN language system for mutation-based software testing,” Software – Practice and Experience, vol. 21, no. 7, pp. 685–718, 1991.

[13] A. J. Offutt, J. Voas, and J. Payne, “Mutation operators for Ada,” Information and Software Systems Engineering, George Mason University, Tech. Rep. ISSE-TR-96-09, 1996.

[14] J. Tuya, M. J. Su?arez Cabal, and C. de la Riva, “Mutating database queries,” Information and Software Technology, vol. 49, no. 4, pp. 398–417, 2007.

[15] A. Derezi?nska, “Quality Assessment of Mutation Operators Dedicated for C# Programs,” in QSIC 2006: Sixth International Conference on Quality Software. Beijing, China: IEEE,Computer Society, 2006, pp. 227–234.

[16] W. E. Howden. “Weak Mutation Testing and Completeness of Test Sets”, IEEE Transactions on Software Engineering, 8(4):371-379, July 1982.

[17] M. R. Girgis and M. R. Woodward. 1985. “An integrated system for program testing using weak mutation and data flow analysis”. In Proceedings of the 8th international conference on Software engineering (ICSE '85). IEEE Computer Society Press, Los Alamitos, CA, USA, 313-319.

[18] Mike Papadakis and Nicos Malevris, “Metallaxis: An Automated Framework for Weak Mutation” , Department of Informatics, Athens University of Economics and Business Athens, Greece.

[19] A. Jefferson Offutt, and Stephen D. Lee, “An Empirical Evaluation of Weak Mutation”, Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030, Stephen D. Lee, IBM Corporation A00/062, P.O. Box 12195, Research Triangle Park, NC 27709, February 24, 1996.

[20] A. Jefferson Offutt and Stephen D. Lee. 1991, “How strong is weak mutation?”, In Proceedings of the symposium on Testing, analysis, and verification (TAV4). ACM, New York, NY, USA, 200-213. DOI=10.1145/120807.120826, http://doi.acm.org/10.1145/120807.120826.

Author Photographs

Taratip Suwannasart received Ph.D. degree in computer science at the Illinois Institute of Technology in 1996. She is working as an association professor in the Department of Computer Engineering, faculty of Computer Engineering at Chulalongkorn University. Her research interests are software engineering especially software testing and software quality assurance.

Panya Boonyakulsrirung graduated with Bachelor's degree in Electrical Communication Engineering at Kasetsart University in 2005. He is studying his Master degree in Software Engineering in the Department of Computer Engineering, faculty of Computer Engineering at Chulalongkorn University. His research interests are software engineering particularly software testing and software quality assurance.