2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops

Evaluation of t-wise Approach for Testing Logical Expressions in Software

Sergiy Vilkomir, Oleksii Starov, Ranjan Bhambroo
Department of Computer Science
East Carolina University
Greenville, NC 27858
{vilkomirs, starovo12, bhambroor11}@ecu.edu

Abstract— **Pair-wise and, more generally, t-wise testing are the most common and powerful combinatorial testing approaches. This paper investigates the effectiveness of the t-wise approach for testing logical expressions in software in terms of its fault-detecting capabilities. Effectiveness is evaluated experimentally using special software tools for generating logical expressions and t-wise test cases, simulating faults in expressions, testing faulty expressions, and evaluating effectiveness of the testing. T-wise testing effectiveness is measured in its totality and for specific types of faults; it is then compared with random testing. A detailed analysis of the experimental results is also provided.**

Keywords-testing; pair-wise; t-wise; effectiveness; logical expressions; experimental evaluation

I. Introduction

Pair-wise [1, 2] and t-wise (t-way) [3] testing are the most common and powerful combinatorial testing approaches [4]. According to the t-wise testing approach, for each subset of *t* input parameters of a system, every combination of valid values of these parameters should be covered by at least one test case. In pair-wise testing, which is a case of t-wise testing, *t* equals 2. The idea behind the t-wise approach is that faults in software are more likely triggered by a small number of input parameters with benefits being that t-wise testing provides reasonable coverage of software input space while using a small number of test cases. For example, if there are 15 Boolean input variables, the total number of various input combinations is 2^{15} or 32,768. However, it takes only ten input combinations (as pair-wise test cases) to cover all different values for each pair of input variables.

This paper examines the effectiveness (i.e., the ability of test cases to detect faults) of a specific application of the t-wise approach for testing logical expressions in software. There are many special methods to test logical expressions: the Modified Condition/Decision Coverage (MC/DC) criterion, for instance, has shown high effectiveness for revealing software faults [5]. However, for most logical expression test methods, test cases have to be created separately for each individual expression in the software. When the number of expressions is large, this becomes a time-consuming task. On the other hand, for the t-wise approach, test cases are created only based on the set of input variables, and each test case is applied for all logical expressions with the same inputs. This makes investigation of t-wise effectiveness for testing logical expressions attractive from both the practical and theoretical point of views.

This paper provides the results of an experimental evaluation of the effectiveness of t-wise testing to detect faults in logical expressions, and is structured as follows: Section 2 contains a review of related works in the field. Section 3 describes the general organization of the experiments for t-wise approach assessment. Section 4 provide experimental results of t-wise effectiveness in total and for specific types of faults. A comparison of the effectiveness of the t-wise approach and random testing is also considered. Conclusions and directions for future work are addressed in Section 5.

II. Related Works

There are many different approaches to testing logical expressions in software, from simple approaches such as random testing [6, 7] and decision and condition coverage [8] to more sophisticated methods such as MC/DC [9], RC/DC [10], BOR [11], MUMCUT [12], etc. Empirical and experimental investigations of the effectiveness of such methods have been provided for 1) the MC/DC criterion [5, 13-15]; 2) the set of strategies for testing logical expressions (Weyuker et al. [16]); 3) BOR testing (Vouk et al. [17]); 4) MUMCUT criterion (Yu and Lau [18]), etc.

Experimental studies of the effectiveness of testing logical expressions are often carried out for specific types of faults in logical expressions. In this paper we use the following five fault classes that have been previously considered in other studies [16, 19-21]:

- Variable Negation Fault (VNF), where a Boolean variable is replaced by its negation.
- Operator Reference Fault (ORF), where a Boolean operator is incorrectly used instead of another Boolean operator.
- Variable Reference Fault (VRF), where a Boolean variable is replaced by another variable.
- Expression Negation Fault (ENF), where a Boolean expression is replaced by its negation.
- Associative Shift Fault (ASF), where parentheses in a Boolean expression are misused.

The purpose of pair-wise and t-wise testing is to cover various combinations of values from the input domain.

These approaches use orthogonal arrays and similar mathematical techniques to create test cases with required levels of coverage. The idea of applying orthogonal arrays for software testing can be found in Mandl's 1985 paper [22], but only recently it has gained recognition in practical testing methods. Several experiments have compared the t-wise approach with random testing and have shown efficacy using the t-wise approach [23, 24]. The practical suitability of t-wise testing has been also confirmed by applying it to various software programs [25-28].

When input variables are Boolean, the t-wise approach can be used for testing logical expressions in software. With the exception of two of our previous papers [29, 30], we know only one more[1] by Kobayashi et al. [31] that is devoted to this specific topic. Kobayashi et al. [31] investigated effectiveness of t-wise testing for *t* equals 2, 3, and 4. However, their experiments were based only on a small set of logical expressions (20 logical specifications of the TCAS II system [32]), which is considered insufficient for making statistically valid conclusions. In [29, 30], we increased the total number of expressions under investigation to 200 and compared our results with [31], but only for *t* equals 2 (pair-wise testing). A special Fault Evaluator tool was created for these experiments [33]. We also used the Allpairs tool [34] and the TConfig tool [35] to generate pair-wise test cases. The originality of the current paper is the following:

- The effectiveness of t-wise approaches for testing logical expressions is evaluated for *t* from 2 to 6.
- Logical expressions are generated automatically; they are increased to 2000 to assure validity of experimental results.
- The National Institute of Standards and Technology (NIST) ACTS tool [36] is used to generate t-wise test cases.

III. ORGANIZATION OF EXPERIMENTAL EVALUATION OF T-WISE APPROACH

As a measure of t-wise testing effectiveness, we consider probability (in percentage) of detecting a fault in an expression using a t-wise test set. The experimental evaluation of t-wise effectiveness contains six steps:

1. Generation of sets of initial (correct) logical expressions.
2. Generation of sets of t-wise test cases.
3. Generations of faults in initial logical expressions (i.e., generation of faulty expressions).
4. Testing faulty expressions from step 3 using t-wise test sets from step 2.
5. Detection of each fault whether the fault is revealed or not. A fault is considered as revealed by some test set, if values of the correct expression and faulty expression are different for some test case from this test set.
6. Evaluation of effectiveness, i.e., evaluation of percentage of revealed faults.

Three software tools have been used in our experiments. The main one is Fault Evaluator [33], developed in the Software Teasing Research Group (STRG) at East Carolina University (ECU). Two additional tools prepare input data for Fault Evaluator: the ACTS tool by NIST generates t-wise test cases, and the Boolean Expression Generator by ECU STRG generates sets of initial expressions.

The ACTS tool uses an In-Parameter-Order-General (IPOG) algorithm [37] to generate t-wise test cases. It supports t-wise test generation for *t* from 1 to 6. Generation of t-wise test cases depends on the number of Boolean variables in an expression but does not depend on the specific expression. The number of unique Boolean variables in one expression was random and varied from 5 to 15 in our experiments. Altogether, 55 different test sets were generated (five test sets for *t* from 2 to 6 combined with different 11 numbers of variables). The sizes (numbers of test cases) of t-wise test sets developed by ACTS are presented in Table I. The output results of ACTS (i.e., t-wise test cases) are presented in Comma Separated Values (CSV) format and can be directly used as input data for Fault Evaluator.

TABLE I. SIZES OF TEST SETS

Number of variables	2-wise	3-wise	4-wise	5-wise	6-wise
5	6	12	16	32	32
6	8	12	28	32	64
7	8	16	35	61	64
8	8	18	38	73	124
9	8	20	41	85	154
10	10	20	42	98	181
11	10	22	46	107	203
12	10	22	48	115	229
13	10	22	51	120	254
14	10	24	54	124	273
15	10	24	58	131	296

The Boolean Expression Generator tool generates random logical expressions with the different numbers of unique variables as well as number of appearances of each variable. The tool allows users to choose from three options for a number of unique variables in each expression: random, fixed, and from the given interval. The output result of Boolean Expression Generator (i.e., list of expressions) is presented in a text file (one expression per line) and can be directly used as input data for Fault Evaluator. During this experiment we set up the Boolean Expression Generator to create expressions with lengths from 5 to 30 in variable-operator pairs. Such lengths should reflect sizes of logical expressions in real-life software.

[1] Application of the t-wise approach to testing logical expressions is also considered in [39] but the object of investigation is different. Instead of effectiveness of testing, various types of coverage provided by t-wise test sets are analyzed.

In total, 2000 initial logical expressions (10 sets with 200 expressions in each set) were generated during this investigation. The amount of 200 expressions in one set was chosen with the intent to provide stability of obtained experimental results. To investigate the stability of effectiveness evaluations, we generated sets of expressions of different sizes (20, 50, 100, and 200 expressions in a set) and estimated variations of effectiveness between different sets of the same size (see Fig. 1 for pair-wise testing). For sets with 20 expressions the range of pair-wise effectiveness values was from 24.4% to 33.4% with mean equaling 30.0 and standard deviation equaling 2.7. For sets with 200 expressions, our results were more stable: the range of effectiveness value was from 24.3% to 29.8%, with mean equaling 27.7 and standard deviation equal to1.7. According to these data, testing 200 expressions can provide a good representation of the effectiveness level. As mentioned, ten times more expressions (2000 in total) were used in our new experiments, which increase the trustworthiness of the results.

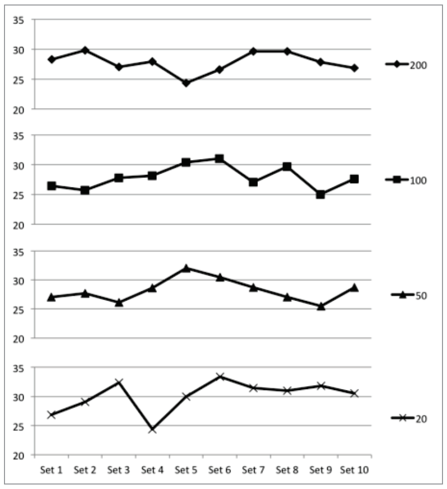

Figure 1. Stability of effectiveness for expressions of different sizes.

For each initial expression, Fault Evaluator generated all possible faulty expressions for five types of faults, mentioned in Section 2. In the current study, only faulty expressions with a single fault were considered. The numbers of generated faults for each set of expressions, each type of fault, and their totals, are presented in Table II.

The general organization of experimental evaluation is presented in Fig. 2. Fault Evaluator applies each t-wise test case to each faulty expression and evaluates the effectiveness of the t-wise testing. The values of effectiveness are provided in total (average for all test sets, average for all types of fault) and for any subset of data (effectiveness for some specific set of expressions, specific type of fault, etc.).

TABLE II. NUMBER OF GENERATED FAULTS

Set	Number of generated faults					
	VNF	*ORF*	*VRF*	*ENF*	*ASF*	*Total*
Set 1	1823	2039	20157	1028	830	25877
Set 2	1701	1919	18785	1004	855	24264
Set 3	1794	1960	19737	1010	840	25341
Set 4	1744	1926	19480	965	837	24952
Set 5	1914	2115	21400	1068	912	27409
Set 6	1881	2038	19969	1014	794	25696
Set 7	1795	1980	19873	1029	839	25516
Set 8	1720	1917	18538	1014	858	24047
Set 9	1928	2170	21974	1099	877	28048
Set 10	1851	1994	20554	1020	878	26297
Total	18151	20058	200467	10251	8520	257447

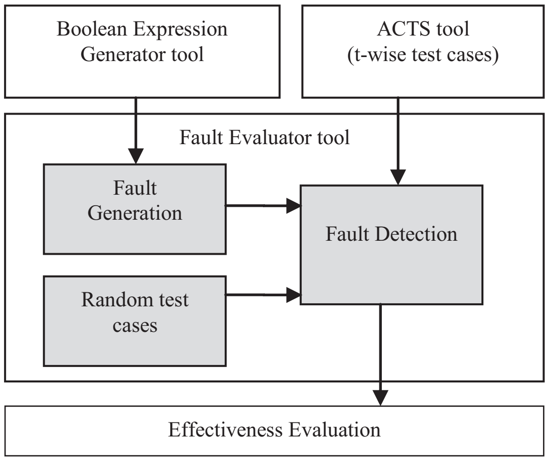

Figure 2. Organization of experimental evaluation of testing effectiveness.

The Fault Evaluator tool is described in [33]. The tool has been modernized with several new features, including:

- Checking for false-faulty expressions: During generation of faulty expressions, Fault Evaluator now compares the truth tables of each faulty expression with the truth table of the initial correct expression. The truth tables have to be different; otherwise the expression is not truly faulty. Without this check, the results can be highly distorted.
- Ability to reset only test cases but keeping faulty expressions: The most time-consuming operation during experiments is generation of faulty expressions. Currently, Fault Evaluator does not regenerate faulty expressions for experiments that differ only with the type of testing. This decreases the time of the entire experiment by almost 10 times.

IV. Results of Experimental Evaluation of Effectiveness of t-wise Testing

A. *Evaluation of Overall Effectivenes*

Values of t-wise testing effectiveness (t = 2...6) for different sets of expressions are shown in Fig. 3. The range of values for the same t but different sets is relatively small. This shows stability of t-wise testing effectiveness for different logical expressions and confirms validity of our estimations of effectiveness. For t-wise testing, the bigger the t, the more the test cases that are required, and the more effective the testing becomes. However, this also increases the amount of resources needed for testing.

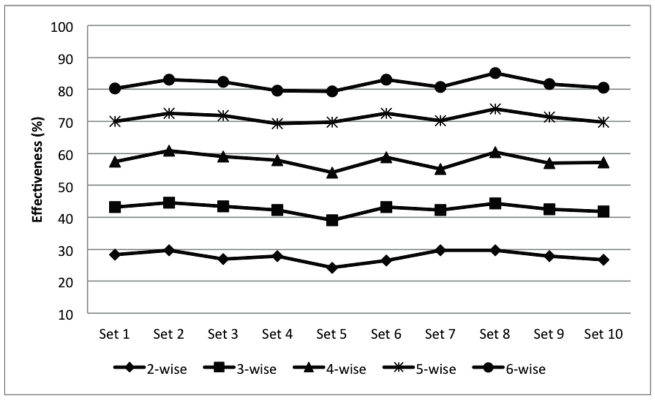

Figure 3. Effectiveness of t-wise testing for different sets of expressions.

Fig. 3 demonstrates that the growth of effectiveness of t-wise testing logical expressions is close to linear: When t is incremented by one, the effectiveness is increased by 10-15%. These data can help to find a trade-off between effectiveness and resource intensity of t-wise testing for specific software.

In practical testing, the t-wise approach is rarely used for t more than 4. Some researches show that the fault detection rate reaches "100% detection with 4 to 6-way interactions" [38]. This is true for some types of software but not a case for testing logical expressions. According to our experimental data, the effectiveness of the 6-wise approach for testing logical expressions is slightly higher than 80%. This is not enough for certain important applications, such as safety-critical software.

In the current experiment, we assessed that pair-wise testing effectiveness was slightly smaller when compared to our previous investigations [30] and the results obtained by Kobayashi et al. [31] for the TCAS II system.

Fig. 4 illustrates the effectiveness of t-wise testing for different types of faults. It was calculated as weighted effectiveness based on results for all 2000 expressions.

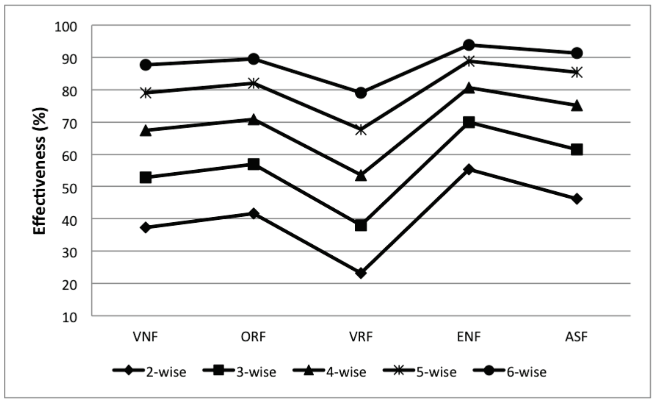

Figure 4. Effectiveness of t-wise testing for different types of faults.

The growth of the testing effectiveness with increasing t is also stable for each type of fault separately. For ENF and VRF types of fault the benefit of incrementing t becomes slightly less after each time. The VRF type of fault is the hardest to detect not only with pair-wise, but also with all kinds of t-wise testing.

B. *Comparison with Random Testing*

Understanding the benefits of t-wise testing requires matching its effectiveness with results for other approaches, especially with random testing. Figures 5-9 illustrate a comparison of effectiveness between t-wise testing and the same amount of randomly generated test cases. Results are shown per set of expressions and for different values of t.

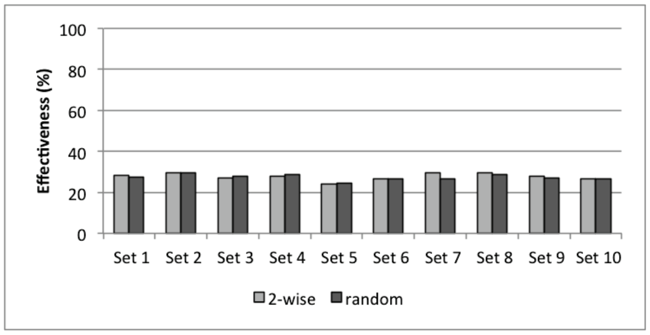

Figure 5. Comparison of the effectiveness of 2-wise and random testing for different sets of expressions.

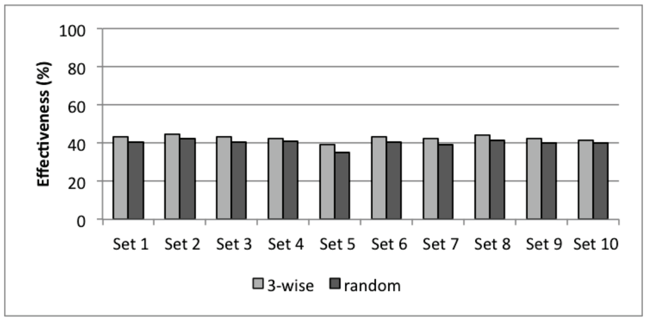

Figure 6. Comparison of the effectiveness of 3-wise and random testing for different sets of expressions.

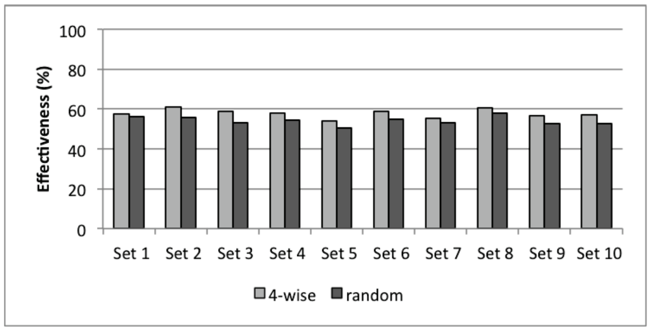

Figure 7. Comparison of the effectiveness of 4-wise and random testing for different sets of expressions.

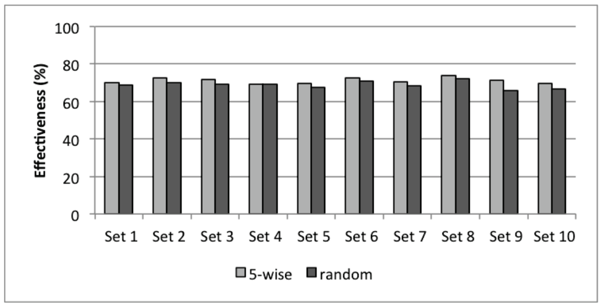

Figure 8. Comparison of the effectiveness of 5-wise and random testing for different sets of expressions.

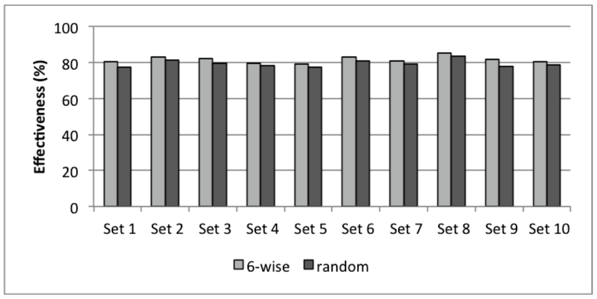

Figure 9. Comparison of the effectiveness of 6-wise and random testing for different sets of expressions.

The t-wise testing approach is observed as always more effective than random testing. This is true for each set of expressions under investigation. At the same time, results show that the benefit of t-wise approach is relatively small.

Detailed values of overall effectiveness and per each expression set for t-wise and random testing are summarized in Table III. The absolute difference between t-wise and random testing effectiveness is in the range 0.4 (for pair-wise testing) to 3.6 (for 4-wise testing). The relative benefit of t-wise testing (taking the value for random testing as 100%) is maximum 6.7% (for 4-wise testing).

Figures 10-14 illustrate comparison of effectiveness between t-wise and random testing per each type of fault. The average difference between t-wise and random testing effectiveness does not depend on a particular fault type. There are no anomalies or peaks on the charts for each type of fault comparison; the benefit of the t-wise approach is stable but small.

TABLE III. COMPARISON OF T-WISE AND RANDOM TESTING EFFECTIVENESS FOR DIFFERENT SETS OF EXPRESSIONS

Set #	**Type of testing**	t				
		2	**3**	**4**	**5**	**6**
Set 1	*t-wise*	28.3	43.3	57.5	70.0	80.2
	Random	27.5	40.8	56.1	68.9	77.4
Set 2	*t-wise*	29.8	44.6	60.9	72.4	83.2
	Random	29.6	42.3	55.8	70.2	81.5
Set 3	*t-wise*	27.0	43.5	59.0	71.8	82.3
	Random	27.9	40.5	53.3	69.0	79.4
Set 4	*t-wise*	27.9	42.2	57.9	69.2	79.6
	Random	28.7	41.0	54.4	69.2	78.2
Set 5	*t-wise*	24.3	39.1	54.1	69.8	79.3
	Random	24.5	35.2	50.6	67.4	77.3
Set 6	*t-wise*	26.6	43.2	58.8	72.6	83.0
	Random	26.8	40.5	54.9	71.1	80.9
Set 7	*t-wise*	29.6	42.2	55.2	70.3	80.9
	Random	26.6	39.0	53.3	68.5	79.2
Set 8	*t-wise*	29.6	44.4	60.4	73.8	85.2
	Random	28.6	41.4	57.9	72.2	83.5
Set 9	*t-wise*	27.8	42.6	56.9	71.3	81.7
	Random	27.1	40.4	52.9	65.9	78.0
Set 10	*t-wise*	26.8	41.7	57.2	69.7	80.4
	Random	26.4	40.3	52.7	66.6	78.6
Whgt. Avg.	*t-wise*	27.7	42.6	57.7	71.1	81.5
	Random	27.3	40.1	54.1	68.9	79.3

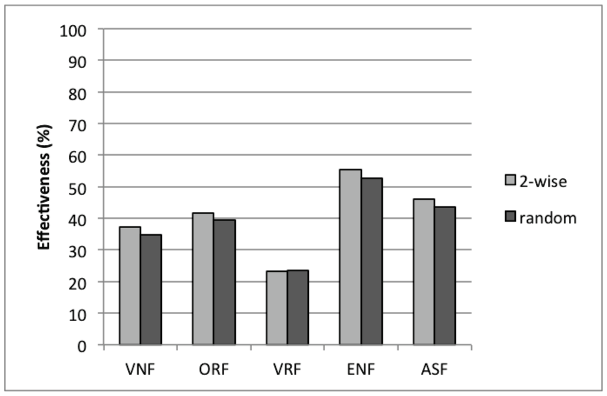

Figure 10. Comparison of the effectiveness of 2-wise and random testing for different types of faults.

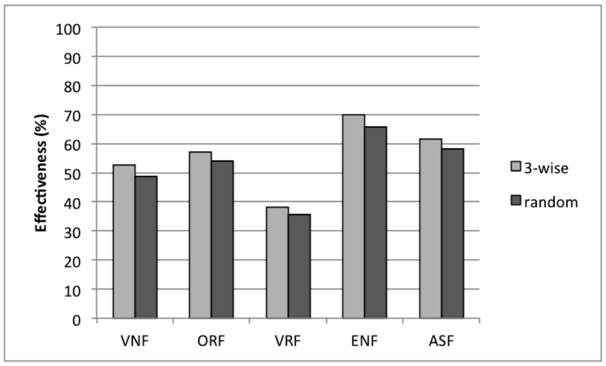

Figure 11. Comparison of the effectiveness of 3-wise and random testing for different types of faults.

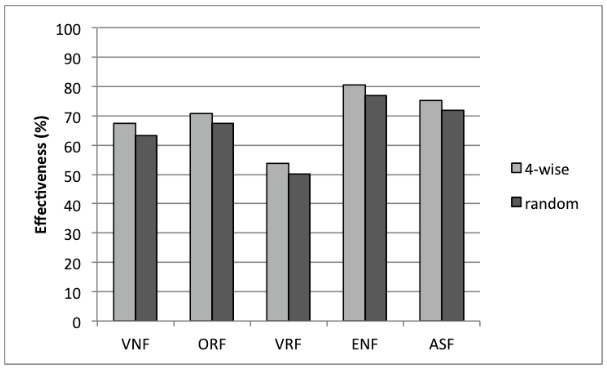

Figure 12. Comparison of the effectiveness of 4-wise and random testing for different types of faults.

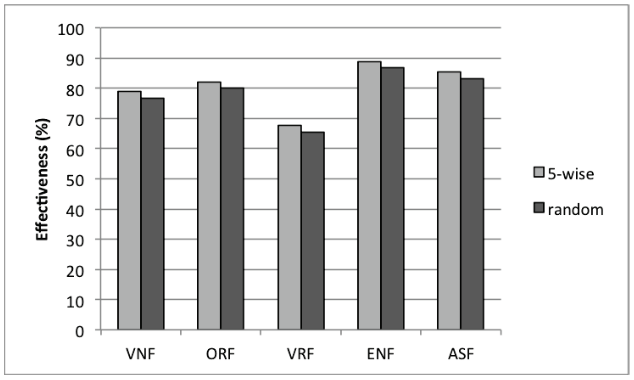

Figure 13. Comparison of the effectiveness of 5-wise and random testing for different types of faults.

Detailed values of the effectiveness per each considered type of fault for t-wise and random testing are summarized in Table IV. The absolute difference between t-wise and random testing effectiveness for different types of faults is a maximum of 4.2 (for 4-wise testing and VNF type of fault). This gives a maximum relative benefit (taking the value for random testing as 100%) of 6.7%.

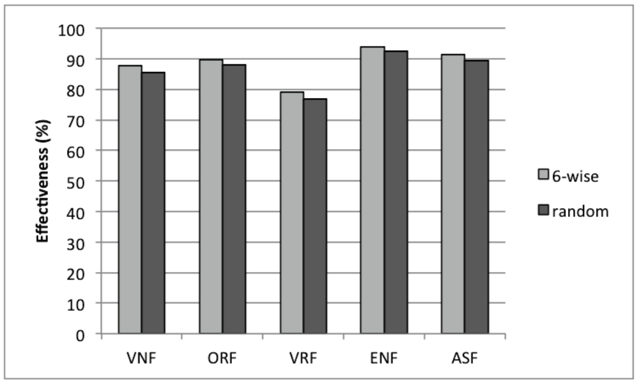

Figure 14. Comparison of the effectiveness of 6-wise and random testing for different types of faults.

TABLE IV. COMPARISON OF T-WISE AND RANDOM TESTING EFFECTIVENESS FOR DIFFERENT TYPES OF FAULTS

Type of fault	**Type of testing**	T				
		2	**3**	**4**	**5**	**6**
VNF	*t-wise*	37.3	52.8	67.3	79.1	87.7
	Random	34.9	48.9	63.1	76.7	85.4
ORF	*t-wise*	41.7	57.0	70.8	82.0	89.6
	Random	39.4	54.1	67.4	80.1	87.9
VRF	*t-wise*	23.2	38.1	53.6	67.8	79.1
	Random	23.4	35.8	50.1	65.5	76.8
ENF	*t-wise*	55.5	69.9	80.5	88.8	93.9
	Random	52.7	65.8	77.0	86.8	92.6
ASF	*t-wise*	46.2	61.6	75.1	85.4	91.3
	Random	43.7	58.2	71.9	83.0	89.4

Comparing the effectiveness values for different types of faults with average effectiveness values from Table III, it is possible to see that effectiveness for some types of faults is much higher than average. For example, the average effectiveness of 3-way testing is only 42.6 and at the same time the effectiveness for ENF type of fault and 3-way testing is 69.9. This can be explained by the fact that weighted average values are used, i.e., the average between all faults (not between types of faults). From the total amount of generated faults, 78% are VRF faults (see Table II) so this type of fault makes a major contribution to the weighted average value of effectiveness.

Fig. 15 shows the mean effectiveness trends for t-wise and random testing while summarizing the comparison.

V. CONCLUSIONS AND FUTURE WORK

This paper presents results of the experimental evaluation of effectiveness of t-wise technique for testing logical expressions in software. The main characteristics of the experiment are the following:

- Different types of t-wise testing for *t* from 2 to 6 were considered.

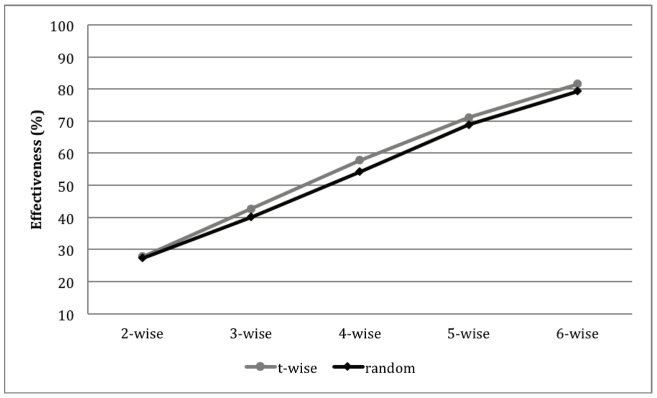

Figure 15. Comparison of t-wise and random testing effectiveness.

- 2000 initial logical expressions were generated as a basis of the experiment.
- More than 250,000 faults in the initial expressions of five different types were simulated.
- 55 t-wise test sets were used for experimental testing.
- Three tools were used: Boolean Expression Generator (ECU) to generate sets of initial expressions, ACTS (NIST) to generate t-wise test cases, and Fault Evaluator (ECU) to simulate faults and evaluate effectiveness of testing.
- Effectiveness of t-wise testing was evaluated in total, for separate sets of expressions, and for different types of faults.
- Results for t-wise testing were compared with similar results for random testing.

The main conclusions and results of our experiments are the following:

- Our experiments provide stable and trustworthy assessments of t-wise effectiveness for testing logical expressions.
- The evaluated average effectiveness is 27.7% for pair-wise, 42.6% for 3-wise, 57.7% for 4-wise, 71.1% for 5-wise, and 81.5% for 6-wise.
- The growth of t-wise effectiveness with the growth of *t* is close to linear with the 10-15% step.
- t-wise testing is the most effective for ENF faults and less effective for VRF faults. These results are the same for all values of *t* from 2 to 6.
- t-wise testing is more effective than random testing for all sets of expressions and all types of faults.
- However, the benefit of t-wise testing vs. random testing is not significant and is maximum 3.6 in absolute values and 6.7% in relative values.
- Because t-wise test cases depend only on the size of the expression and can be applied to the large group of the expression at the same time, t-wise testing can be considered as a simple and effective way to test logical expressions.
- However, even for 6-wise testing, the effectiveness is slightly higher than 80%, which is not sufficient for important applications, such safety-critical software.

Useful directions for future work can include the following:

- Investigation of effectiveness of t-wise testing depending on the expression size. The size can be considered as a number of unique Boolean variables in the expression.
- Investigation of effectiveness of t-wise testing versus complexity of logical expressions. Complexity can be considered as a number of all instances of variables or as an expression length.
- Consideration of more types of fault and their possible multiple occurrences in one expression.
- Comparison of the effectiveness of t-wise testing versus MC/DC and RC/DC testing approaches.
- Evaluation of t-wise effectiveness for logical specifications of real software products.

REFERENCES

[1] Y. Lei and K.C. Tai, "In-parameter-order: a test generation strategy for pairwise testing," Proc. 3rd IEEE Int. High-Assurance Systems Engineering Symp., IEEE Press, Nov. 1998, pp. 254-261, doi: 10.1109/HASE.1998.731623.

[2] D.R. Kuhn, Y.Lei, and R. Kacker, "Practical Combinatorial Testing - Beyond Pairwise," IEEE IT Professional, Jun. 2008, pp. 19-23.

[3] J.R. Maximoff, M.D. Trela, D.R. Kuhn, and R. Kacker, "A Method for Analyzing System State-space Coverage within a t-Wise Testing Framework," IEEE International Systems Conference 2010, Apr. 4-11, 2010, San Diego.

[4] M. Grindal, J. Offutt, and S.F. Andler, "Combination testing strategies: a survey," Software Testing, Verification and Reliability, vol. 15, no. 3, Mar. 2005, pp. 167-199, doi: 10.1002/stvr.319.

[5] S.A. Vilkomir, K. Kapoor, and J.P. Bowen, "Tolerance of control-flow testing criteria," Proc. 27th Annu. Int. Computer Software and Applications Conf. (COMPSAC 2003), IEEE Press, Nov. 2003, pp. 182-187, doi: 10.1109/CMPSAC.2003.1245339.

[6] P. Thevenod-Fosse, H. Waeselynck, and Y. Crouzet, "An experimental study on software structural testing: deterministic versus random input generation," Proc. 21st Int. Symp. Fault-Tolerant Computing (FTCS 91), IEEE Press, Jun. 1991, pp. 410-417, doi: 10.1109/FTCS.1991.146694.

[7] I. Ciupa, A. Pretschner, A. Leitner, M. Oriol, and B. Meyer, "On the predictability of random tests for object-oriented software," Proc. 1st Int. Conf. Software Testing, Verification, and Validation (ICST 08), IEEE Press, Apr. 2008, pp. 72-81, doi: 10.1109/ICST.2008.20.

[8] Y.T. Yu and M.F. Lau, "Comparing several coverage criteria for detecting faults in logical decisions," Proc. 4th Int. Conf. Quality Software (QSIC 04), IEEE Press, Sep. 2004, pp. 14-21, doi: 10.1109/QSIC.2004.1357940.

[9] J.J. Chilenski and S.P. Miller, "Applicability of modified condition/decision coverage to software testing," Software Engineering Journal, vol. 9, no. 5, Sep. 1994, pp. 193-200.

[10] S.A. Vilkomir and J.P. Bowen, "From MC/DC to RC/DC: formalization and analysis of control-flow testing criteria," Formal Aspects of Computing, vol. 18, no. 1, Mar. 2006, pp. 42-62, doi: 10.1007/s00165-005-0084-7.

[11] K. C. Tai and H. K. Su, "Test generation for boolean expressions," Proc. COMPSAC '87, 1987, pp. 278-283.

[12] Y. Yu, M. Lau, and T. Chen, "Automatic generation of test cases from Boolean specifications using the MUMCUT strategy," Journal of Systems and Software, vol. 79, no. 6, Jun. 2006, pp. 820-840, doi:10.1016/j.jss.2005.08.016.

[13] A. Dupuy and N. Leveson, "An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software," Proc. 19th Digital Avionics Systems Conf. (DASC 00), IEEE Press, Oct. 2000, pp. 1B6/1-1B6/7, doi: 10.1109/DASC.2000.886883.

[14] M.P.E. Heimdahl, M.W. Whalen, A. Rajan, and M. Staats, "On MC/DC and implementation structure: An empirical study," Proc. 27th IEEE/AIAA Digital Avionics Systems Conference (DASC 2008), 26-30 October 2008, St. Paul, Minnesota, USA, pp. 5.B.3-1-5.B.3-13, doi: 10.1109/DASC.2008.4702848.

[15] J.J. Chilenski, "An Investigation of Three Forms of the Modified Condition Decision Coverage (MCDC) Criterion," DOT/FAA/AR-01/18 Technical Report, 2001.

[16] E. Weyuker, T. Goradia, and A. Singh, "Automatically generating test data from a Boolean specification," IEEE Trans. Software Engineering, vol. 20, no. 5, May 1994, pp. 353-363, doi: 10.1109/32.286420.

[17] M.A. Vouk, K.C. Tai, and A. Paradkar, "Empirical studies of predicate-based software testing,", Proc. 5th International Symposium on Software Reliability Engineering, Nov. 6-9, 1994, Monterey, California, USA, pp. 55-64, doi: 10.1109/ISSRE.1994.341348

[18] Y. Yu and M. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions," Journal of Systems and Software, vol. 79, no. 5, May 2006, pp. 577-590, doi: 10.1016/j.jss.2005.05.030.

[19] D.R. Kuhn, "Fault classes and error detection capability of specification-based testing," ACM Trans. Software Engineering and Methodology (TOSEM), vol. 8, no. 4, Oct. 1999, pp. 411-424, doi: 10.1145/322993.322996.

[20] D.J. Richardson and M.C. Thompson, "The RELAY model of error detection and its application," Proc. 2nd Workshop Software Testing, Verification, and Analysis, IEEE Press, Jul. 1988, pp. 223-230, doi: 10.1109/WST.1988.5378.

[21] D.J. Richardson and M.C. Thompson, "An analysis of test data selection criteria using the RELAY model of fault detection," IEEE Trans. Software Engineering, vol. 19, no. 6, Jun 1993, pp. 533-553, doi: 10.1109/32.232020.

[22] R. Mandl, "Orthogonal Latin Squares: an Application of Experiment Design to Compiler Testing," Commun. ACM 28(10), 1985, pp. 1054–1058.

[23] P. Schroeder, P. Bolaki, and V. Gopu, "Comparing the fault detection effectiveness of n-way and random test suites," Proc. 2004 Int. Symp. Empirical Software Engineering (ISESE 04), IEEE Press, Aug. 2004, pp. 49-59, doi: 10.1109/ISESE.2004.1334893.

[24] D.R. Kuhn, R. Kacker, and Y.Lei, "Random vs. Combinatorial Methods for Discrete Event Simulation of a Grid Computer Network," Proc. Mod Sim World 2009, NASA CP-2010-216205, Oct. 14-17 2009, Virginia Beach, USA, pp. 83-88.

[25] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton, "The AETG system: an approach to testing based on combinatorial design," IEEE Trans. Software Engineering, vol. 23, no. 7, Jul. 1997, pp. 437-444, doi: 10.1109/32.605761.

[26] D.M. Cohen, S.R. Dalal, J. Parelius, and G.C. Patton, "The combinatorial design approach to automatic test generation," IEEE Software, vol. 13, no. 5, Sep. 1996, pp. 83-88, 10.1109/52.536462.

[27] M. Ellims, D. Ince, and M. Petre, "The Effectiveness of T-Way Test Data Generation," Proc. SAFECOMP 2008, 22-25 September 2008, Newcastle upon Tyne, UK, Springer LNCS 5219, pp.16–29.

[28] V. Hu, D.R. Kuhn, and T. Xie, "Property Verification for Generic Access Control Models," Proc. IEEE/IFIP International Symposium on Trust, Security, and Privacy for Pervasive Applications, Shanghai, China, Dec. 17-20, 2008.

[29] W. Ballance, W. Jenkins, and S. Vilkomir, "Probabilistic assessment of effectiveness of software testing for safety-critical systems," Proc. 10th Int. Probabilistic Safety Assessment and Management Conf. (PSAM 10), 7-11 June 2010, Seattle, Washington, USA.

[30] W. Ballance, S. Vilkomir, and W. Jenkins, "Effectiveness of Pair-wise Testing for Software with Boolean Inputs," Proc. Fifth International Conference on Software Testing, Verification and Validation (ICST 2012), April 17-21, 2012, Workshop on Combinatorial Testing (CT-2012), Montreal, Canada, pp. 580-585.

[31] N. Kobayashi, T. Tsuchiya, and T. Kikuno, "Non-specification-based approaches to logic testing for software," Information and Software Technology, vol. 44, no. 2, Feb. 2002, pp. 113-121, doi: 10.1016/S0950-5849(01)00222-1.

[32] N.G. Leveson, M.P.E. Heimdahl, H. Hildreth, and J.D. Reese, "Requirements specification for process-control systems," IEEE Trans. Software Engineering, vol. 20, no. 9, Sep. 1994, pp. 684-707, doi: 10.1109/32.317428.

[33] W. Jenkins, S. Vilkomir, and W. Ballance, "Fault Evaluator: a tool for experimental investigation of effectiveness in software testing," Proc. 2010 IEEE Int. Conf. Progress in Informatics and Computing (PIC 2010), IEEE Press, Dec. 2010, pp. 1077-1083, doi: 10.1109/PIC.2010.5688000.

[34] J. Bach, "ALLPAIRS Test Case Generation Tool (Version 1.2.1)," Accessed on Jan. 2013, http://www.satisfice.com/tools/pairs.zip.

[35] A. Williams, J.H. Lo, and A. Lareau, "TConfig," Accessed on Jan. 2013, http://www.site.uottawa.ca/~awilliam/TConfig.jar.

[36] NIST, "ACTS tool," Accessed on Jan. 2013, http://csrc.nist.gov/groups/SNS/acts/.

[37] Y.Lei, R. Kacker, D.R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A general strategy for t-way software testing," Proc. 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '07), Tucson, AZ, USA, March 26-29, 2007, pp. 549-556.

[38] R. Bryce, Y. Lei, D.R. Kuhn, and R. Kacker, "Combinatorial Testing," Chap. 14, Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization, Ramachandran, ed. , IGI Global, 2009, pp. 196-208.

[39] R.C. Bryce, A. Rajan, and M.P.E. Heimdahl, "Interaction Testing in Model-Based Development: Effect on Model-Coverage," Proc. 13th Asia-Pacific Software Engineering Conference (APSEC 2006), 6-8 Dec. 2006, Bangalore, India, pp. 259-268, doi: 10.1109/APSEC.2006.42.