Reasonability of MC/DC for Safety-Relevant Software Implemented in Programming Languages with Short-Circuit Evaluation

Susanne Kandl
Institute of Computer Engineering
Vienna University of Technology
Austria
Email: susanne@vmars.tuwien.ac.at

Sandeep Chandrashekar
Infineon Technologies India Pvt. Ltd.
India
Email: Sandeep.Chandrashekar@infineon.com

***Abstract*—MC/DC (modified condition/decision coverage) is a structural code coverage metric, originally defined in the standard DO-178B [1], intended to be an efficient coverage metric for the evaluation of the testing process of software incorporating decisions with complex Boolean expressions. The upcoming standard ISO 26262 [2] for safety-relevant automotive systems prescribes MC/DC for ASIL D as a highly recommended coverage metric. One assumed benefit of MC/DC is that it requires a much smaller number of test cases in comparison to MCC (multiple condition coverage), while sustaining a quite high error detection probability [3].**

Programming languages like C, commonly used for implementing software for the automotive domain, are using short-circuit evaluation. For short-circuit evaluation the number of test cases for MCC is much smaller than in a non-short-circuit environment because many redundant test cases occur. We evaluated the trade-off between the number of test cases for MCC and MC/DC for a case study from the automotive domain and observed an overhead of only approximately 5% for the number of test cases necessary for MCC compared to MC/DC. This motivated an analysis of programs containing decisions where the number and structure of the referring Boolean expressions vary. Our results show that the overhead for a test suite for MCC is on the average only about 35% compared MC/DC (for decisions with up to 5 conditions). We conclude with the strong recommendation to use MCC as a coverage metric for testing safety-relevant software implemented in programming languages with short-circuit evaluation.

I. Introduction

Safety-relevant software, causing crucial damage to people or the environment when malfunctioning, has to be tested exhaustively to guarantee a high reliability. Different methods for evaluating the maturity of the testing process are applied: review processes, assessment processes, coverage metrics.

MC/DC (modified condition/decision coverage) is a structural code coverage metric, originally defined in the standard DO-178B Software Considerations in Airborne Systems and Equipment Certification [1]. Besides using MC/DC as a qualitative measure to ensure the compliance of the implementation with the specification, MC/DC is intended to be an efficient coverage metric for the evaluation of the testing process of software incorporating decisions with complex Boolean expressions containing multiple conditions, like `if`$(A\&\&(B||C))$ `statement_1 else statement_2`.

The original definition of MC/DC addressed mainly programming languages used at that time for safety-critical applications in the avionics domain, like Ada. The Boolean operators AND and OR of Ada are using *eager evaluation* (aka: *greedy evaluation*), i.e. the whole Boolean expression is evaluated to determine the resulting outcome. Complete testing of such an expression is given by the metric MCC (multiple condition coverage). This coverage requires all possible Boolean assignments to the input variables of the decision. For a decision containing *N* Boolean conditions ($N = 3$ for the example above) we would need to generate 2^N inputs to test all possible combinations. This means the testing effort grows *exponentially* with an increasing complexity of the decision. MC/DC requires a much smaller number of test cases in comparison to MCC. For a Boolean expression with *N* conditions MC/DC requires only $N + 1$ test cases, so the number of test cases grows only *linearly* with the number of conditions. Although the number of test cases is much smaller than for complete testing, MC/DC sustains a quite high error detection probability [3].

MC/DC is also defined as a highly recommended metric for ASIL D systems in the upcoming standard ISO 26262 Road Vehicles - Functional Safety [2] for safety-relevant automotive systems. Commonly used programming languages for software applications in the automotive domain are Assembler and C. In contrast to programming languages like Ada, C uses *short-circuit evaluation*. That means if the result of a Boolean expression is already determined by the first part of an expression, the remaining part of this expression is not executed anymore. For short-circuit evaluation the number of test cases for MCC is much smaller than 2^N because many redundant test cases occur.

Based on a study of the code structure of typical code for programs of the MCAL (microcontroller abstraction layer, part of the AUTOSAR concept) to gain an overview of the attributes for an eligible case study, we selected a representative case study for our experiment and evaluated the testing effort for both, MCC and MC/DC. The initial aim was to show the

This work has been partially funded by the ARTEMIS Joint Undertaking and the National Funding Agency of Austria for the project VeTeSS under the funding ID ARTEMIS-2011-1-295311 and was supported by Infineon Technologies AG (Munich, Germany).

benefit of using MC/DC instead of MCC for the evaluation of the testing process, expecting a significant decrease of the testing effort (the number of required test cases, respectively). Indeed the number of test cases for MCC was only about 5% higher than the number of test cases for MC/DC. This result motivated us to systematically examine programs with decisions depending on different complex Boolean expressions up to the number of 5 conditions. Even for 5 conditions we observed only an *average overhead* of around 35% compared to the number of test cases for MC/DC, which is still feasible for testing. As the MC/DC-test set is only a subset of the MCC-test set, it may happen that errors are not detected by the MC/DC-test set although covered by the MCC-test set. Regarding this enhanced confidence in the testing process using the MCC-criterion instead of the MC/DC-criterion, and considering the acceptable overhead for the increased number of test cases to achieve full MCC, we question the use of the MC/DC-criterion for safety-relevant software implemented in a programming language with short-circuit evaluation.

The paper is organized as follows: First we give a short overview of the test environment for executing the test runs and a rough description of the use case for our experiment. In the following section we recapitulate the coverage metric MC/DC, addressing also the expected and the real error detection probability. In an example we compare an MCC-test set with an MC/DC-test set for a programming language with short-circuit evaluation. In Section IV we describe first in detail the set-up to determine the estimated overhead for programs containing multiple decisions of different structures, then we show our experimental results for the estimated overhead and discuss what these results mean for practice. After presenting some related work, we conclude with a summary.

II. Test Environment

The recent test environment within Infineon is realized in the UVP (universal validation platform). The main idea is to generate for each test the application running on the target and then test it with the appropriate set of stimuli and testing algorithms. This approach is feasible by automating the code generation path and including in the test application the necessary code under the control of a master test plan. Each version of generated code has also a corresponding test set-up, test stimuli sets, and testing algorithms which are automatically loaded into the test bench. The concept is based on reusable code (and test set-ups) building blocks. The generation of the hardware-dependent software in the current configuration is done automatically with a small parametric application on the top. Also the compilation, download, launching, and execution of the test application are automated. Result logging and determination of different coverage criteria is supported [4], [5].

A. Use Case

We defined some attributes for the case study on which we want to apply our analysis. The case study should provide a complexity in the control flow justifying the use of MC/DC. We studied the code structure of typical code for programs part of the MCAL (microcontroller abstraction layer) to gain some information on the nature of the programs to choose an appropriate case study. Based on an analysis of the code structure of the programs we decided for a case study with following properties: The case study is a program for pulse width modulation, part of a set of hardware drivers for the Tricore processor. It has about 6100 lines of code and it includes 24 decisions, with a different number of conditions. Moreover it contains further decisions in the preprocessor directives. This program was chosen because it showed the highest complexity regarding the control flow graph. It is remarkable that the considered case studies have a limited number of conditions in the decisions (usually less than or equal 5).

B. Test Runs

The case study cannot be executed as a standalone application, but only as part of a driver package consisting of multiple files. So in the overall a system is executed with approximately 15.000 lines of code, including the observed case study with 6100 LOC. The first test run was executed by a test set provided by Infineon. This test set contains test cases generated manually based on the requirements from the system specification. The initial test run for the case study yields the following coverage results:

The %-value gives the achieved percentage of achieved coverage, calculated by the quotient: number of executed test cases/number of required test cases (given in the second row).

Decision Coverage	*MCC*	*MC/DC*
63 %	63 %	62 %
795/1253	937/1499	887/1424

TABLE I. Results of the initial test run for the case study

Observation A: The value for the achieved MCC-coverage is similar to the value of the achieved decision coverage (DC). For MCC in comparison to DC only 246 (=1499 - 1253) additional test cases are needed. This means that many of the decisions occurring in the tested system are simple decisions (depending on a singular condition like `if(A)`), i.e., for these decisions the number of test cases needed for DC equals the number of test cases needed for MCC [and equals the number of test cases needed for MC/DC]. Only a small part of decisions contain a complex Boolean expression with more than one condition (additional test cases to achieve maximum MCC are required).

Observation B: The achieved coverage value for MCC is nearly the same as for MC/DC (63% vs. 62%). To achieve maximum MCC only 75 (= 1499 - 1424) additional test cases are needed in comparison to the MC/DC test set. These additional 75 test cases denote an overhead of approximately 5% compared to the number of necessary test cases for MC/DC.

Although we learned from Observation A that only a few decisions are of a complex structure, the value of 5% is surprisingly low. Usually MCC needs a significantly higher number of test cases than MC/DC, and thus we would have expected a bigger overhead of the MCC test set compared to the MC/DC test set. This insight motivated us to compare the test sets for MCC and MC/DC for programs with short-circuit evaluation in more detail (see Section IV).

Based on the results from the initial test run, we identified the missing test cases to achieve full MCC and MC/DC. After adding the missing test cases that can be executed we achieved maximum coverage.

III. UNIQUE-CAUSE MC/DC

MC/DC is introduced in DO-178B [1], discussed in detail in [6], and expanded with variations of the metric in [7], respectively. The metric is a structural coverage metric defined on the source code and is designed to test programs with decisions that depend on one or more conditions, like `if` $(A\&\&(B||C))$ `statement_1 else statement_2`.

For MC/DC we need a set of test cases to show that changing the value for each particular condition changes the outcome of the total decision independently from the values of the other conditions. (This works as long there is no coupling between different instances of conditions.)

A test suite conforming to MC/DC consists of test cases that guarantee that [1], [6]:

- every point of entry and exit in the model has been invoked at least once
- every basic condition in a decision in the model has been taken on all possible outcomes at least once, and
- each basic condition has been shown to *independently* affect the decision's outcome.

The independence of each condition has to be shown. If a variable occurs several times within a formula each instance of this variable has to be treated separately. Independence is defined via *Independence Pairs*.

Consider the example $A\&\&(B||C)$: The truth table is given in Table II (third column). In the following $\bar{0}$ represents the test case (F, F, F), $\bar{1}$ represents the test case (F, F, T), and so on. The independence pairs for the variable A are $(\bar{1}, \bar{5})$, $(\bar{2}, \bar{6})$ and $(\bar{3}, \bar{7})$, the independence pair for the variable B is $(\bar{4}, \bar{6})$ and the independence pair for the variable C is $(\bar{4}, \bar{5})$. Thus we have the test set for MC/DC consisting of $\{\bar{4}, \bar{5}, \bar{6}\}$ plus one test case of $\{\bar{1}, \bar{2}\}$ (remember that for N conditions we need $N + 1$ test cases).

Testcase	*A*	*B*	*C*	$A\&\&(B\|\|C)$
$\bar{0}$	F	F	F	F
$\bar{1}$	F	F	T	F
$\bar{2}$	F	T	F	F
$\bar{3}$	F	T	T	F
$\bar{4}$	T	F	F	F
$\bar{5}$	T	F	T	T
$\bar{6}$	T	T	F	T
$\bar{7}$	T	T	T	T

TABLE II. EXAMPLE MC/DC-TEST SET

A. Error-Detection Probability of MC/DC

In [3] different code coverage metrics are compared and a subsumption hierarchy for the most relevant code coverage metrics is given. It is stated that *"the modified condition/decision coverage criterion is more sensitive to errors in the encoding or compilation of a single operand than decision or condition/decision coverage"*, but MCC is still the strongest coverage metric. The probability of detecting an error is given as a function of tests executed. For a given set of M distinct tests, the probability $P_{(N,M)}$ of detecting an error in an incorrect implementation of a Boolean expression with N conditions is given by [3]

$$P_{(N,M)} = 1 - \left[\frac{2^{(2^N - M)} - 1}{2^{2^N}} \right].$$

This correlation is shown in Figure 1 for $N = 4$.

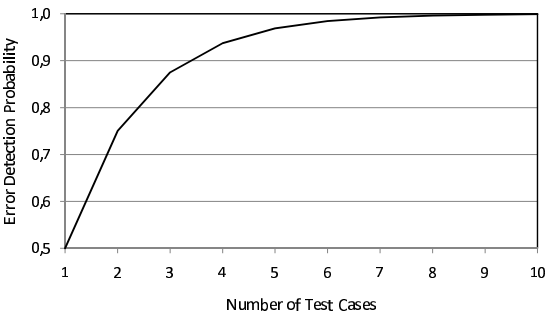

Fig. 1. Error Detection Probability of MC/DC [3]

For $N = 4$ and $M = 5$ (the minimal number of test cases) the resulting error detection probability is $0,96$.
For $N = 5$ and $M = 6$ the error detection probability is $0,98$.

As M increases there is a rapid increase in the error detection probability. As N grows, $P_{(N,M)}$ rapidly converges to $1 - 1/2^M$ and the sensitivity changes only marginally with N. That means for N increasing the likelihood of detecting an error in an expression of N conditions with $N + 1$ test cases increases also. This non-intuitive result occurs because the dominant factor (the number of tests) increases with N while the sensitivity to errors remains relatively stable [3].

Indeed the *real* error detection probability of MC/DC is less than expected, as we showed in previous works [8]: For a case study 22% of the mutated variable names and 8% of the mutated operators are not detected by the MC/DC test set. In contrast to that MCC covers all detectable errors.

B. MCC-Test Set vs. MC/DC-Test Set for Short-Circuit Evaluation

The surprisingly low overhead for our case study motivated an analysis in which extend short-circuit evaluation influences the number of test cases required for MCC. The required number of test cases depends on the structure of the Boolean expression, demonstrated by following examples, see Table III and Table IV:

Expression	*MCC*	*MC/DC*
$A\&\&B\&\&C$	F- -	F- -
	TF-	TF-
	TTF	TTF
	TTT	TTT

TABLE III. EXAMPLE_A

Expression	*MCC*	*MC/DC*
$A\&\&B\|\|C$	F-F	F-F
	F-T	F-T
	TFF	TFF
	TFT	
	TT-	TT-

TABLE IV. EXAMPLE_B

Both examples show an expression with 3 conditions, so full MCC without short-circuit evaluation would need $2^3 = 8$ test cases. In the column MCC the required test cases for MCC with short-circuit evaluation are given: For the expression in Example_A 4 test cases are required for MCC, so the number of test cases for MCC equals the number of test cases for MC/DC. For the expression in Example_B 5 test cases for MCC are required, whereas 4 test cases are required for MC/DC. For this Boolean expression the overhead is 25%.

IV. ANALYSIS AND RESULTS

We executed an analysis of the number of test cases needed for MCC, and MC/DC, respectively, for all possible Boolean expressions up to 5 conditions: For a Boolean expression with 2 conditions, 2 expressions are possible: `A && B`, and `A || B`. For a Boolean expression with 3 conditions, 4 expressions are possible: `A && B && C`, `A && B || C`, `A || B && C`, and `A || B || C`. For a Boolean expression with 4 conditions 8 different expressions are possible, and for a Boolean expression with 5 conditions 16 different expressions are possible.
Besides that we can integrate parentheses, this increases the number of possible Boolean expressions (OP...operand): For a Boolean expression with 4 conditions 6 different ways to apply parenthesis are possible (for each of the 8 expressions): `(A OP B) OP C OP D; A OP (B OP C) OP D; A OP B OP (C OP D); (A OP B) OP (C OP D); (A OP B OP C) OP D; A OP (B OP C OP D)`. For a Boolean expression with 5 conditions 14 different ways to apply parenthesis are possible (for each of the 16 expressions).
For the evaluation we only considered variants of the Boolean expressions for which the parentheses have an impact on the result of the expression: `(A && B && C` is the same as `(A && B) && C`, whereas `A || B && C` differs from `(A || B) && C`).

The idea of these permutations is to cover as many different Boolean expressions as possible.

Analyzing the number of test cases needed for MCC with short-circuit evaluation for all possible Boolean expressions with up to 5 conditions (without parenthesis *wp* and including parenthesis *ip*) shows following results, see Table V. The columns are described in the following.

#Cond	Number of conditions.
#MCDC	Number of test cases for MC/DC.
#MCC_max_wp	Maximum number of test cases for MCC for all Boolean expressions without parenthesis.
#MCC_max_ip	Maximum number of test cases for MCC for all Boolean expressions including parenthesis.
OH_max	Considering that a system under test has decisions with Boolean expressions that require the maximum number of observed test cases for MCC, this value would be the maximum overhead for the number of test cases for MCC (compared to the number of required test cases for MC/DC).
OH_av_wp	Considering that a system under test contains all kinds of possible Boolean expressions with *N* conditions without parenthesis (uniformly distributed), this value describes the average overhead for the required test cases for MCC.
OH_av_ip	Considering that a system under test contains all kinds of possible Boolean expressions with *N* conditions also including parenthesis (uniformly distributed), this value describes the average overhead for the required test cases for MCC.

Observations from this survey:

- For 2 conditions the number of test cases for MCC is always equal to the number of test cases for MC/DC.
- For 3, or 4 conditions, respectively, the maximum overhead for MCC is 25%, or 60%, respectively. So in the worst case (many decisions with Boolean expressions containing 4 conditions), MCC testing means an overhead of 60% for the test cases (in comparison to MC/DC). This overhead is acceptable.
- For 3, or 4 conditions, respectively, the average overhead (including also expressions with parenthesis) for MCC is approx. 9%, or 20%, respectively, so almost negligible.
- Even for 5 conditions the average overhead is around 35% (compared to the number of test cases for MC/DC), which is still feasible for testing. Based on the experiences from our case studies, the number of conditions within software for the automotive domain is often limited by 5, so the resulting overhead is acceptable.

#Cond	*#MCDC*	*#MCC_max_wp*	*#MCC_max_ip*	*OH_max*	*OH_av_wp*	*OH_av_ip*
2	3	3	3	0%	0%	0%
3	4	5	5	25%	6,25%	9,38%
4	5	7	8	60%	17,50%	20,42%
5	6	11	13	116%	33,33%	35,82%

TABLE V. SUMMARY OF THE RESULTS

A. Discussion of the Results

Based on the observations from the case study, we question the reasonability of MC/DC instead of MCC for software from the automotive domain (with a manageable complexity) realized in a programming language with short-circuit evaluation. We learned that the overhead for the MCC-test set is almost negligible (regarding the number of test cases) in comparison to a MC/DC-test set. As we showed in the analysis the number of test cases required for MCC (for a system implemented in a language with short-circuit evaluation) causes only a small overhead (5% for our case study) for testing in comparison to MC/DC. This can be explained in following way: Many of the decisions with a complex Boolean expression contain only 2 conditions. For 2 conditions the number of test cases required for MCC is equal to the number of test cases required for MC/DC (for both 3 test cases), so the MCC-test set is the same as the MC/DC-test set for these decisions. Some decisions contain more conditions, even for these decisions the additional test cases for MCC are only a few (e.g., 5 test cases vs. 4 test cases for `A && B ||C`, see Example_B in Section III-B). In contrast to an MC/DC-test set, the MCC-test set covers the whole possible input-data space, so it guarantees that all possible occurring errors are identified. With the restricted MC/DC-test set, not all errors may be identified.

The usage of MC/DC makes sense as a *qualitative* means to assess the maturity of the software development process. The metric can be used to prove whether the requirements defined in the system specification map the implemented code (a poor value for MC/DC for a test set generated requirement-based indicates a lack in the specification, or not specified functionality in the implemented code). This kind of deviations indicate a gap between the specification and the implementation.

The use of MC/DC used as a *quantitative* measure is reasonable when it is used as an alternative coverage metric to stronger coverage metrics, like MCC, because it is not feasible to realize full testing (*stronger* in this context means that the test set of MCC is a superset of the test of MC/DC, i.e. the test cases of MCC cover a bigger part of the input data space, thus the ability to detect errors in the program is higher). But as far as the overhead for MCC is so low, MCC is better suitable as a quantitative measure for the evaluation of the testing process for safety-relevant programs implemented in a programming language with short-circuit evaluation.

Regarding the guidelines of the standard ISO 26262 [2] and addressing the aim of high reliability required for safety-relevant programs it would be desirable to combine the benefits of both metrics: As deriving the MC/DC-test set is a non-trivial issue this process assumes a detailed analysis of the structure (the control flow) of the program. So building a test set to achieve maximum MC/DC for a system under test forces the test engineers to study both, the specification and the implementation, in a very precise way. This activity by itself enforces the quality of the testing process. On the other side by achieving full MCC it is guaranteed that all *detectable* errors are identified (not detectable errors are *latent* faults or errors, deviations in the program that have no impact on the resulting output; these errors are not detectable, no matter which test cases are applied).

Besides that, the test engineer should always be aware of that a *structural* code coverage metric is only evaluated based on the implementation. Achieving a specific coverage goal by incremental test case-generation until $x\%$ coverage is achieved may increase the part of the tested code. But in the sense of a structured verification process, i.e., checking whether the system is conform with the specification, or not, this is by far not sufficient, see also [9]. Structural code coverage metrics should only be a supplement to approaches like *requirement-based* testing, in which the *requirements* guide the testing process (and not the test data), see [10] and [11].

V. Related Work

The most important discussion on the *applicability of MC/DC* for testing (safety-critical) software is [3]. [12] addresses also the topic short-circuit evaluation. A list of all statements that are covered by MC/DC for the programming languages C, C++, and Ada83 is given in [13]. General comparisons of code coverage metrics are [14] for structural based metrics and [15] for data-flow based metrics. Jones and Harrold [16] introduce a test-suite reduction with the focus on the MC/DC-criterion. Staas et.al. [17] discuss general considerations for the usage of code coverage metrics only as a quantitative measure. The authors conclude with two simple statements: 1. Coverage criteria satisfaction alone is a poor indication of test suite effectiveness. 2. The use of structural coverage as a supplement -not a target- for test generation can have a positive impact. Our results emphasize especially the second point. Also Büchner [18] presents in his collection of eight misapprehensions about coverage that in general *code coverage is* not *a sufficient criterion to assess the quality of code*. In [19] different code coverage metrics (decision coverage, full predicate coverage, and MC/DC) are compared regarding their effectiveness in finding errors. Yu and Lau compare several structural coverage criteria, including MC/DC, with the result that MC/DC is cost effective in relation to other criteria [20]. In [21] the testing effort between decision coverage and MC/DC is compared. A case study for structural testing applied to safety-critical embedded software is [22]. An empirical evaluation of MC/DC for satellite software is [23]. To our knowledge there are no empirical works on the comparison of MCC and MC/DC, especially not focusing on short-circuit evaluation.

VI. Summary and Conclusion

A satisfying test process should realize the ideal trade-off between effort and confidence in the test result. Structural code coverage metrics are one means to determine the maturity of the testing process, both in a qualitative and in a quantitative way. A structural code coverage metric used in a quantitative way only determines the part of the program executed during testing. MCC covers all possible input values of a decision depending on a complex Boolean expression. Considering *all* input values is, in general, not possible as the number of test cases increases exponentially (assuming a non-short circuit evaluation). MC/DC requires only a subset of the MCC-test set, the number of required test cases grows linearly with the number of conditions in the Boolean expression of the decision. MC/DC used as a qualitative measure can help to identify deviations of the implemented system from the original specification. The usage of MC/DC as a quantitative

measure instead of a stronger coverage metric, like MCC, would only be reasonable if full testing is not feasible.

For the use case we showed that the overhead of the number of test cases for MCC is only approx. 5% compared to the number of test cases for MC/DC. This is caused, on the one hand, by the restricted complexity of the underlying system, on the other hand, by the given property of the programming language C using short-circuit evaluation. In our detailed analysis we considered C-programs with all possible Boolean expressions with up to 5 conditions, without parenthesis and including parenthesis. As we showed in our analysis the overhead of test cases for MCC in comparison to MC/DC for short-circuit evaluation is in the worst case (only complex Boolean expressions of a type with maximum overhead for the number of test cases) 116% for 5 conditions. Considering all possible variants of Boolean expressions the expected average overhead for the number of test cases for MCC in comparison to MC/DC is around 35% for 5 conditions (and even less for a smaller number of conditions). So the overhead for testing is reasonable.

Regarding the increased confidence in an MCC-test set (covering all detectable errors) and the observed overhead we conclude with the strong recommendation to use the MCC-criterion for safety-relevant programs with short-circuit evaluation.

REFERENCES

[1] RTCA Inc., "DO-178B: Software Considerations in Airborne Systems and Equipment Certification," Requirements and Technical Concepts for Aviation, Washington, DC, December 1992.

[2] ISO: International Organization for Standardization, "ISO 26262: Functional safety road vehicles, draft," 2009.

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

[4] Infineon - Automotive & Industrial System & Software Engineering SCE5, "Universal validation platform - common validation platform for safety-related projects," April 2004, Internal document of IFX.

[5] Infineon - AIM MC D SCE5 System & Software Engineering, "UVP - concept," 2004, Internal document of IFX.

[6] RTCA Inc., "DO-248B: Final Report for Clarification of DO-178B: Software Considerations in Airborne Systems and Equipment Certification," Requirements and Technical Concepts for Aviation, Washington, DC, October 2001.

[7] John Joseph Chilenski, "An investigation of three forms of the modified condition decision coverage (MCDC) criterion," U.S.Department of Transportation, Federal Aviation Administration, DOT/FAA/AR-01/18, April 2001.

[8] S. Kandl and R. Kirner, "Error detection rate of MC/DC for a case study from the automotive domain," *LNCS 6399: S.L.Min et al.(Eds.): Proceedings of the 8th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2010)*, pp. 131–142, Oct. 2010.

[9] F. Büchner, "White paper: Is 100% code coverage enough?" *Tessy, Hitex*, 08/2012.

[10] P. Ammann and P. E. Black, "A Specification-Based Coverage Metric to Evaluate Test Sets." in *HASE*, 1999, pp. 239–248.

[11] A. Abdurazik, P. Ammann, W. Ding, and J. Offutt, "Evaluation of three specification-based testing criteria," in *Engineering of Complex Computer Systems, 2000. ICECCS 2000. Proceedings. Sixth IEEE International Conference on*, pp. 179–187.

[12] H. Kelly J., V. Dan S., C. John J., and R. Leanna K., "A practical tutorial on modified condition/decision coverage," Tech. Rep., 2001.

[13] J. Chilenski and L. A. Richey, "Definition for a masking form of modified condition decision coverage (mcdc)," *Technical report, Boeing, Seattle, WA,*, December, 1997.

[14] S. Ntafos, "A comparison of some structural testing strategies," *Software Engineering, IEEE Transactions on*, vol. 14, no. 6, pp. 868 –874, June 1988.

[15] L. A. Clarke, A. Podgurski, D. J. Richardson, and S. J. Zeil, "A comparison of data flow path selection criteria," in *ICSE '85: Proceedings of the 8th international conference on Software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1985, pp. 244–251.

[16] J. Jones and M. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," in *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, pp. 92–101.

[17] M. Staats, G. Gay, M. Whalen, and M. Heimdahl, "On the danger of coverage directed test case generation," in *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*, ser. FASE'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 409–424. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28872-2_28

[18] F. Büchner, "Acht Irrtümer über Code Coverage," *Elektronik Praxis*, 2010.

[19] K. Kapoor and J. Bowen, "Experimental evaluation of the variation in effectiveness for DC, FPC and MC/DC test criteria," in *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*, Sept.-1 Oct. 2003, pp. 185–194.

[20] Y. T. Yu and M. L. Laub, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions," *Journal of Systems and Software*, vol. 79, no. Issue 5, pp. 577–590, May 2006.

[21] Z. Szűgyi and Z. Porkoláb, "Necessary test cases for decision coverage and modified condition / decision coverage," *Department of Programming Languages and Compilers, Eötvös Loránd University*, 2002.

[22] J. Guan, J. Offutt, and P. Ammann, "An industrial case study of structural testing applied to safety-critical embedded software," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, ser. ISESE '06. New York, NY, USA: ACM, 2006, pp. 272–277. [Online]. Available: http://doi.acm.org/10.1145/1159733.1159774

[23] A. Dupuy and A. Leveson, "An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software," *Digital Aviation Systems Conference*, October 2000.