Automated Software Defect Detection and Identification in Vehicular Embedded Systems

Kyle Foss, Ivo Couckuyt, Adrian Baruta, and Corentin Mossoux

Abstract—**Trends in the automotive industry confirm that the demand for testing of embedded systems, especially advanced driver assistance systems (ADAS), will grow dramatically in the near future. This paper proposes a new solution that automates the detection of software defects in embedded systems. The solution consists of a data-driven sampling algorithm to intelligently sample the testing space by sequentially generating test cases. Moreover, it segregates unique defects from each other and identifies the signals that trigger each. The results are compared against other automated methods for defect identification and analysis, and it is found that this novel solution is able to identify defects more rapidly. In addition, it correctly separates defects and reliably reproduces each distinct defect.**

Index Terms— **Advanced driver assistance systems, automatic test pattern generation, automotive electronics, embedded software, machine learning, metamodeling, pattern clustering, system verification**

I. Introduction

RECENT studies have shown that prototype advanced driver assistance systems (ADAS) are nearing or exceeding human level performance [1]–[4]. As these ADAS systems enter production vehicles, a great deal of new hardware and software components are being introduced into customer hands.

Due to the safety sensitive nature of these new components, test procedures must be developed to guarantee that a system is free from defects. Traditional automotive verification ensures that all functional requirements are met. However, this testing is insufficient to validate the safe operation of embedded systems in unforeseen, but possible, real world situations.

Software and electronic integration defects contribute to recalls of millions of vehicles each year [5]. While various laws and industry standards mandate testing to verify functionality [6], [7], unexpected defects are still triggered when unforeseen or untested input signals are received by an Electronic Control Module (ECM). A huge time commitment is required in order to obtain even minimal coverage of an embedded system's input space. As a result, full factorial testing becomes unfeasible, and OEMs are forced to evaluate a subset of possible test cases before production. This inevitably requires a trade-off between test coverage and testing duration.

This paper introduces a novel data-driven method to automate testing and analysis of such systems. This reduces the test time and input required from OEM expert engineers. The method consists of three phases: (1) defect detection, (2) defect segregation, and (3) defect causal analysis. Phase 1 aims to detect all distinct software defects within minimal testing time. Phase 2 determines how many different defects were detected and segregates test cases based on the defects detected in each. Phase 3 determines the input signals that trigger each defect and provides additional information to enable rapid repair. These separate phases combine to form a cohesive path to eliminate defects. The overall flow, and output from each phase, is summarized in Figure 1.

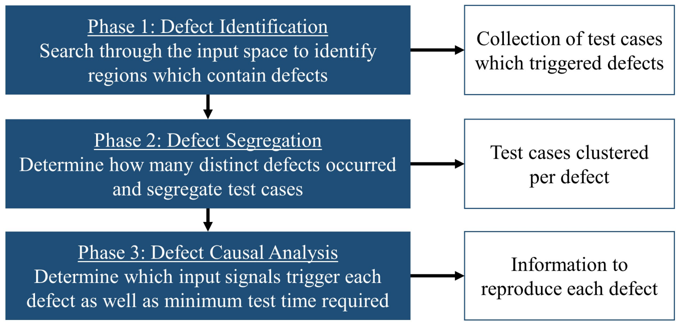

Phase 1: Defect Identification
Search through the input space to identify regions which contain defects
Collection of test cases which triggered defects
Phase 2: Defect Segregation
Determine how many distinct defects occurred and segregate test cases
Test cases clustered per defect
Phase 3: Defect Causal Analysis
Determine which input signals trigger each defect as well as minimum test time required
Information to reproduce each defect

Figure 1: Overall Solution Flow

These methods are validated on an isolated Electronic Control Module (ECM) and its embedded software. Testing was performed on several representative models at Toyota Motor Europe's Technical Center on a prototype road sign recognition ADAS software. The techniques detailed in this paper go beyond improving coverage of the input space. They intelligently generate test cases based upon the system behavior from past tests. These techniques have not been applied in the automotive domain before and are shown to significantly outperform the benchmark testing practices described later in this paper.

In the forthcoming parts of this paper, the word *defect* will be used to describe an unwanted behavior from the embedded system. A correct identification of a vehicle problem which triggers a diagnostic entry or Diagnostic Trouble Code (DTC) will be referred to as an indicator.

This work has been conducted within the ENABLE-S3 project that has received funding from the ECSEL joint undertaking under grant agreement no 692455. This joint undertaking receives support from the European Union's Horizon 2020 Research and Innovation Program and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia, Norway.

K. Foss and I. Couckuyt are with IDLab, Department of Information Technology at Ghent University – imec. UGent- IDLab, iGent, Technologiepark Zwijnaarde 15, 9052 Gent, Belgium (e-mail: kyle.foss@ugent.be, ivo.couckuyt@ugent.be, dirk.deschrijver@ugent.be, tom.dhaene@ugent.be).

A. Baruta and C. Moussoux are with Toyota Motor Europe, Technical Centre. Hoge Wei 33, 1930 Zaventem, Belgium (e-mail adrian.baruta@toyota-europe.com, corentin.mossoux@toyota-europe.com).

II. Related Work

In the past decade, automotive development and testing has come to rely heavily on model-based systems engineering and model-based verification and validation (V&V) [8]–[11]. A key part of V&V is test case generation, a task that is labor intensive and prone to human error. A plethora of methods have been proposed to automatically generate test cases [12]. Test cases can be generated utilizing SysML/UML requirements [13], [14]. Similarly, temporal logic [15] can be used to test boundary regions based on operational requirements. These methods are useful during early stage development, such as model-in-the-loop (MIL) testing, and are necessary to ensure all requirements are fulfilled [16]. These methods are based upon expert knowledge of the internal structure of the System Under Test (SUT), or are hand designed for requirement verification [17]–[19]. However, these testing strategies do not specifically address the problem of finding unknown defects.

In [20]–[22], statistics based machine learning techniques are used to search for automotive software defects. Specifically, the system is modeled as a Markov chain, and test cases are generated using either Monte Carlo or Gibbs samplers. These methods have been integrated into commercial tools and are able to successfully detect defects. The downside to Markov chains is that the system states and transitions must be predefined by development engineers before any tests can begin. These state and transition definitions are created solely for testing and add significant work for development engineers. Therefore, such a white box approach is not time efficient.

Mutation-based test case generation is a relatively new method for model based testing [23], [24]. This approach utilizes a validated system model, mutates it in some way, and then generates test cases to exploit this mutation. The method is valuable as long as the mutations are representative of real system defects, but provides no functionality during hardware-in-the-loop (HIL) testing. The approach described in this paper also utilizes a validated system model, but intelligently generates test cases during HIL testing instead of beforehand.

Other approaches treat the SUT as a black box by only evaluating its input and output signals [10], [25]. Black-box testing has proven valuable in many fields of software testing, and is particularly well suited for an automotive application because it requires a minimal amount of input from the test practitioner. By reducing the time required to set up test cases, a much greater amount of time can be devoted to actual testing. Although promising, little has been published in the last decade to advance black box testing in the automotive domain.

The work discussed so far has focused primarily on defect detection, under the assumption that engineers can determine the root cause given one test case that triggered the defect. While this is true in some cases, root cause identification is separate from defect detection, and can be a complex and time-consuming task. Existing works have treated identification and causal analysis as separate topics; this paper combines them within a streamlined process.

In [26], [27], genetic algorithms are implemented for root cause analysis to determine input signal thresholds at which a defect occurs. While useful, this method can only be applied to a single defect that depends on a small number of input signals. This is due to the slow convergence of genetic algorithms in high dimensions. Often, multiple independent defects are identified which depend on different input signals. Before using a genetic algorithm, manual steps must be taken to separate the unique defects as well as determine which signals trigger each.

This paper describes a new methodology for end-to-end automotive black box testing. By automating a majority of work required to test for defects, a thorough evaluation of the software can be performed in a shorter time. The methodology in this paper does not replace requirement verification, but is a complementary procedure for defect elimination.

III. Black Box Representation

The individual techniques utilized in this paper have proven effective in other fields[28]–[31]. They are introduced here specifically for automotive defect detection. In order to successfully describe these techniques, the System Under Test (SUT) will be discussed first.

A. Real-Time Model Based Testing

Testing of the SUT should be performed in real-time in order to provide a realistic evaluation for the automotive hardware [32]. In this paper, requirements are incorporated into a verified Simulink model. This verified model is used to define a defect. Therefore, this paper does not specifically evaluate system requirements as these are assumed to have been verified during model-in-the-loop (MIL) testing of the Simulink model. This approach has been shown to work well in practical industry applications [32]–[34].

B. Input Signal Specification

All bus messages which the test ECM reads will be referred to as the system input. Each input signal is a time series discretized at the bus refresh rate. The system output is defined as the set of all bus messages sent by the ECM under test. The algorithm independently generates a test case for each test, as shown in Figure 2. A test case is defined as the set of all input signals evaluated on one test.

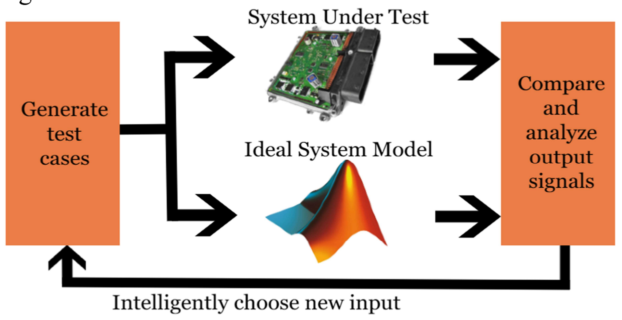

System Under Test
Generate test cases
Ideal System Model
Compare and analyze output signals
Intelligently choose new input

Figure 2: The SUT and ideal model are evaluated together simultaneously

C. Input Signal Generation

In the example software tested in this paper, there are 2 x 10^{13} possible input value combinations at any instant. With each signal refreshing at 100 Hz, it is impossible to exhaustively search the input space. This full input space is also not representative of ECM behavior, since different locations in the input space do not necessarily correspond to evaluating different ECM functions. Therefore, the dimensionality of the input space is reduced so that the algorithm has a lower-dimensional, more meaningful, feature space to analyze. This is achieved by describing the behavior of each signal with qualitative features as opposed to strict quantitative definitions. These features are shared by all input signals during phase 1.

Each multi-dimensional input signal is translated to a new point in a 2D feature space. The two qualitative features implemented are normalized to the interval [0, 1]:

- *Complexity: the rate at which the signal changes. A zero complexity signal is constant and a signal with complexity of one will change at that signal's bus refresh rate. The timing of each change is sampled uniformly at random.*
- *Proportionality: defines which values the signal can take. The possible values of each parameter are first divided into two sets that correspond to changes in the system state. The higher the proportionality of the signal, the more of its values are chosen from the second set as opposed to the first set.*

The division into separate sets for proportionality is done by OEM software experts based on domain knowledge of the signal values; an example is shown in Appendix A. When a signal transitions between set 1 and set 2 values, the system state can be changed without directly defining it. By similarly varying the complexity of each signal, the system state transitions are controlled indirectly. Defining the signal behavior with these features significantly reduces the input space but still allows for control of the system states.

The qualitative feature extraction satisfies two properties:

1. Given fixed features, the algorithm should generate input signals that evaluate similar ECM functions.
2. Given fixed features, each test case should be unique.

As seen in Figure 3, changing the features leads to the generation of signals with different characteristics as desired by property two. Property one is demonstrated in Figure 4; showing different signals for a fixed complexity and proportionality. The timing of changes, as well as the instantaneous values, are random. However, the number of change-points as well as the percentage of time spent in each set are approximately constant in all plots of Figure 4.

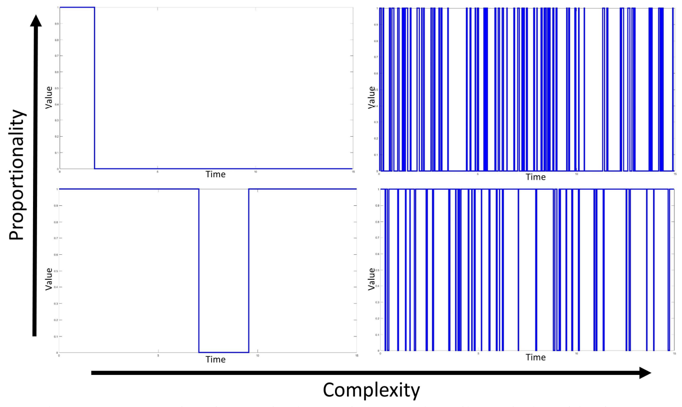


Figure 3: Example of complexity and proportionality relationship for a Boolean input signal

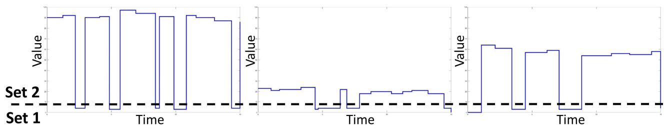


Figure 4: Example of three input signals generated with the same complexity and proportionality with this algorithm

D. Output Signal Representation

The instantaneous outputs from both the system under test and the ideal Simulink model are in the form of equal length binary vectors. Each element of the vector is a Boolean indicator corresponding to the status of a diagnostic entry. This instantaneous output is sampled at 100 Hz, with the vectors stacked in order to form two binary matrices of equal size, one for the SUT and one for the ideal Simulink model. All phase 1 tests are run at a constant duration that is defined before testing.

The output from the SUT is subtracted from the ideal model to yield one matrix which defines defects. A is the output from the validated model, B is the output from the SUT, and C is the combined representation; each of these is a matrix of size (T x n) where T is the number of time steps and n is the number of indicators in the output. For every time step t:

$$A = [A_{t0}\, A_{t1}\, A_{t2}\, \dots\, A_{tn}]$$
$$B = [B_{t0}\, B_{t1}\, B_{t2}\, \dots\, B_{tn}]$$

$$C_{ti} = \begin{cases} 1 & if\ A_{ti} = 1\ and\ B_{ti} = 0 \\ -1 & if\ A_{ti} = 0\ and\ B_{ti} = 1 \\ 0 & if\ A_{ti} = B_{ti} \end{cases} \quad (1)$$

Given this representation, there is a distinction between defects with a false positive (C_{ti} = -1) and a false negative (C_{ti} = 1), but no defect (C_{ti} = 0) if the software and ideal model agree. This output matrix C is saved for all test cases. Any deviation between the ideal model and SUT constitutes a defect, regardless of duration. In other words, a test case detects a defect if matrix C contains any nonzero elements.

IV. Phase 1: Defect Detection Algorithm

In order to evaluate the SUT in an efficient manner, this paper introduces an algorithm to identify regions of the input space containing defects. Test cases are generated using the adaptive sampling algorithm FLOLA-Voronoi (FV), applied to the 2-dimensional feature space [28], [29].

A. FLOLA-Voronoi

The FV strategy is based upon two separate components: Fuzzy LOcal Linear Approximation (FLOLA) and Voronoi tessellation. It balances exploitation and exploration by scoring all locations in the feature space according to the number of samples nearby, as well as the defect detection rate.

Exploitation assigns high scores to regions where the defect detection rate behaves nonlinearly. These regions have already identified a defect. Adding additional samples in nonlinear regions will help define boundaries in the 2D feature space where a defect stops occurring. Exploration assigns higher scores to regions which have been sampled sparsely, calculated with a Voronoi tessellation where a cell is drawn around each previously sampled point.

These two components of the FV algorithm are combined to assign a score to each location in the input space, and the coordinates with the best score are selected as the next sample point for testing. FV analyzes the results from all past test cases on each iteration, then chooses the next optimal point in the feature space. The balance between exploitation and exploration ensures that multiple examples of each defect are obtained, while also continuing to search for new defects.

B. Sequential Sampling Loop

Sample points are chosen sequentially to maximize the efficiency of each point. This drastically reduces the number of test cases, as well as test time, required to identify the same number of defects. Since this method seeks to identify an unknown number of defects that depend on unknown input parameters, no region of the input space is excluded by FV. Each chosen location in the feature space is evaluated according to the following pseudo code:

```
Generate 20 test cases using highest scored feature values
Evaluate SUT and Simulink model once for each test case
Compare SUT and Simulink outputs to determine defect rate
Score all regions of the feature space according to FV
```

FV measures the ratio of test cases at the selected location that detected a defect, yielding a result between 0 and 1. After all tests are performed at the location, FV reevaluates the feature space and the location with the best score is chosen next.

Since the algorithm has no knowledge of the system before the sequential sampling loop begins, it requires a small number of initial points. Eleven initial points are chosen using a Latin hypercube space-filling design [35].

V. Phase 2: Defect Segregation Algorithm

After phase 1, numerous test cases have been identified that contain defects. Phase 2 determines the number of unique defects. Clustering is performed in order to group test cases caused by the same defect into homogeneous clusters.

For each test case, the output C is a $(T \text{ x } n)$ matrix which contains many repeated rows. Each C matrix is reduced to a set of n dimensional vectors, where n is the number of indicators. Rows of all zeros are removed since they represent instances of no defect detected. Second, all rows which are identical to the previous row are removed since they represent the same output sequence occurring for an extended time. This results in a set of vectors for each C matrix. Due to interaction noise, the number of unique vectors is typically much greater than the number of true system defects. A novel clustering approach is implemented to determine the true number of unique defects.

The standard algorithm for clustering categorical vectors is k-modes [30]; a modified version is utilized here. After clustering, each cluster is assumed to correspond to a unique defect. The phase 2 result is a set of test cases for each cluster.

A. Weighted dissimilarity metric

The goal of a clustering algorithm is to minimize intra-cluster distances while maximizing inter-cluster distances. A dissimilarity metric must be used which appropriately fits the unique structure of automotive bus signals.

In the standard k-modes categorical clustering application, the value of each attribute is compared to the corresponding cluster centroid to determine equality. In other words, the dissimilarity metric does not distinguish among values for each attribute. This metric is shown below, where X and Y are two categorical vectors [30]:

$$d(X,Y) = \sum_{i=1}^{n} \delta(x_i, y_i) \tag{2}$$

where

$$\delta(x_i, y_i) = \begin{cases} 0 & if\ x_i = y_i \\ 1 & if\ x_i \neq y_i \end{cases} \tag{3}$$

Given the representation in matrix C, this metric does not fully capture the information contained in the vectors. For each categorical attribute from phase 1, there are three possible categories [-1, 0, 1]. Category 0 can be viewed as a special default case since it represents "no detected defect". Since most defects manifest on only a few output attributes, the majority of vectors will be a sparse representation of primarily zeros. If all attributes are equally weighted, there are many situations where a vector is equidistant from several cluster centroids.

This paper introduces a modified dissimilarity metric which adjusts the weight on specific attributes when calculating distance. This dissimilarity metric adds an additional weight where both the centroid and sample are default (value 0). The new metric serves to distance examples which agree on default indicators (correct operation) from those that agree on defect indicators.

This can be seen as a special case of the solution proposed in [36]. The new metric corresponds to replacing $\delta(x_i, y_i)$ in (3) with that shown in (4), where ϵ is a value between 0 and 1.

$$\delta(x_i, y_i) = \begin{cases} 0 & if\ x_i = y_i \neq 0 \\ \epsilon & if\ x_i = y_i = 0 \\ 1 & if\ x_i \neq y_i \end{cases} \tag{4}$$

In this implementation, the value of ϵ was made dependent on vector length, $\epsilon = 3 / n$. The value of 3 is the median of the number of nonzero elements from all vectors from phase 1. This new dissimilarity metric is used within the standard k-modes clustering algorithm.

B. Initial cluster centroid selection

A key input to clustering algorithms is the initial cluster centroids. A heuristic is introduced to determine the initial centroids by making use of the specific structure of data from phase 1. First, the test cases where the C matrix contains only one vector are retained. It is assumed these vectors correspond to a single defect. From these vectors, those which deviated on one indicator become the initial categorical cluster centroids.

This heuristic tends to overestimate the number of defects, which is preferred instead of an underestimate. If two separate defects are clustered together and reported as one, then most likely one defect will not be repaired. If, however, one defect is reported as multiple, it will merely take some additional time to determine they are the same defect, at a small cost to efficiency.

VI. Phase 3: Defect Reproduction Algorithm

Taking the clustered results from phase 2, the goal of phase 3 is to determine the cause of each unique defect. Treating the embedded system as a black box renders true root cause analysis unfeasible. The goal of causal analysis is defined in this paper as the ability of the algorithm to reliably reproduce a defect using different test cases. In other words, it should be able to reproduce each defect by specifying the behavior of only the input signals upon which the defect depends.

Almost all software implementation errors in embedded systems can be detected by evaluating the interactions among 4-6 parameters [31], [37]. This technique, called combinatorial or t-way testing, creates test cases that include all possible combinations among a subset of the n input parameters. Testing only these t-way interactions aims to provide coverage similar to that of full factorial design with only a small fraction of the test cases. For this paper, t was set at 5 to cover the most interactions without an excessive number of tests.

For each cluster, a small number of additional tests will be performed in order to isolate the signals relevant to each defect. The values for complexity and proportionality define the behavior of one signal. This will be utilized in order to "switch off" signals by setting each to a baseline. Baseline for complexity is 0, meaning the parameter will remain constant during the entire simulation. For proportionality, since the boundary values 0 and 1 actually choose between two groups of values, the baseline value will be the boundary further away from the feature value used in the original test case.

The minimum test time is determined from all test cases in each cluster from phase 2. This will be a different value for each cluster. The minimum test time is the earliest time at which the defect occurred, and will be used for the duration of the phase 3 tests for each cluster. Since all tests are performed in real-time, this significantly reduces the time required for phase 3 as well as ensures simple test cases.

For each cluster, the centroid is the most common output vector. From all test cases which generated this output vector, the one which triggered the defect quickest is used as the initial input for phase 3. The algorithm will now evaluate the possible input parameter interactions using this test case. This is accomplished with binary combinatorial test design, using the 5-way interactions.

For each parameter, the signal will be either the baseline or identical to the initial test case. The baseline will be a constant value selected from the proportionality set opposite to that used in the initial test case. In this sense, t-way testing can be seen as switching on and off specific input parameters, and evaluating every t-way combination of switches. Based on the switch combinations that still trigger the defect, the causal signals are determined.

VII. Experimental Setup and Results

The methods outlined in this paper are validated by tests on a portion of the On Board Diagnostic functions of Toyota's Road Sign Assistance (RSA), a component of the Toyota Safety Sense 2.0 system currently under development [38]. The control module is treated as a black box which receives and sends messages over the vehicle bus. This test software has 32 input parameters, as shown in Appendix A, and 43 output indicators. A fixed test time of 15 seconds was used for phase 1, resulting in a C matrix of size 1500 x 43 before post processing. Each of the input parameters has between 2 and 94 possible discrete values.

The methods described in this paper were developed in MATLAB and integrated with a Simulink system model, the industry standard used during development of control module software. The experimental set up communicated with the ECM via Vector hardware and software (CANoe). Although tests were performed with Controller Area Network (CAN) protocol signals, the solution presented is designed to interface readily with standard communications tools, allowing for communication via more advanced protocols (Ethernet, FlexRay, etc.). The ECM under test controls the Road Sign Assistance feature, which functions to identify traffic signs adjacent to roads and display them to the driver. This system will be utilized for logical decision making in future generations of driver assistance systems. This same technique can readily be applied to model-based software-in-the-loop (SIL), processor-in-the-loop (PIL), and hardware-in-the-loop (HIL) testing for any embedded system. All tests were performed at the Toyota Technical Center in Zaventem, Belgium.

For validation, the algorithm was evaluated on known software defects that were injected into the validated Simulink model. These defective models replaced the SUT in the test setup shown in Figure 2. Eight separate representative defects were introduced, and various scenarios are covered where defects may occur either individually or simultaneously (e.g. 2 and 3 at a time). These injected defects were created by Toyota Motor Europe based upon real defects in similar systems. A summary of the 8 defects are shown in Table 1.

Table 1: Defect Descriptions

Defect Model	Defect Description
1	Operation cycle incorrect
2	Malfunction recovery condition edited (AND to OR)
3	CAN Invalid judgment / Diagnostic mask edited (AND to OR)
4	Wrong diagnostic entry for speed
5	Battery voltage input wrong (pulsing AND)
6	CAN invalid judgment wrong
7	CAN invalid judgment wrong
8	Diagnostic logic edited for camera
10	Combination of 1 & 7
11	Combination of 6, 7, & 8

A. Defect detection

Phase 1, defect detection, was performed until 100 points in the complexity and proportionality feature space had been evaluated. At each point, 20 stochastic test cases were generated, resulting in a total number of 2000 evaluations.

Different approaches are benchmarked against each other for generating these test cases:

- **FLOLA:** the feature space transform followed by FLOLA-Voronoi, as detailed in Section IV. This forms the base of the method described in this paper.
- **Random Parameter (RP)**: values are chosen at uniformly random for the 2D feature space transform of

complexity and proportionality. These values are then used to generate new input signals for each test.

- **Random**: generates input signals from randomly chosen values (from all values for each parameter). The time and number of transitions are also chosen uniformly at random. 2000 random test cases were generated.

These two randomized benchmarks allow for comparison of both aspects of the sequential sampling algorithm: the input signal parameterization and FLOLA-Voronoi sampling. The Random approach chooses from all possible signal values and therefore does not utilize the signal parameterization method. The RP generation does utilize the signal parameterization techniques, but does not intelligently sample the 2D design space. With these benchmarks, it is expected for the random parameter generation to outperform the random generation, and both of these to be outperformed by the FLOLA algorithm.

1) Single Defects

The algorithm developed in this paper was tested on 8 synthetic defects with the percentage of defect detections recorded. Initially, only one defect at a time was considered. The results are summarized below in Table 2 and Figure 5.

Table 2: Phase 1 percentage detection results for defects occurring individually

	Percentage Defect Detections							
Defect:	1	2	3	4	5	6	7	8
FLOLA	**0.653**	**0.036**	**0.032**	**0.736**	**0.622**	**0.181**	**0.507**	0.690
RP	0.500	0.020	0.015	0.511	0.435	0.060	0.325	0.741
Random	0.454	0.001	0.002	0.212	0.120	0.019	0.100	**0.906**

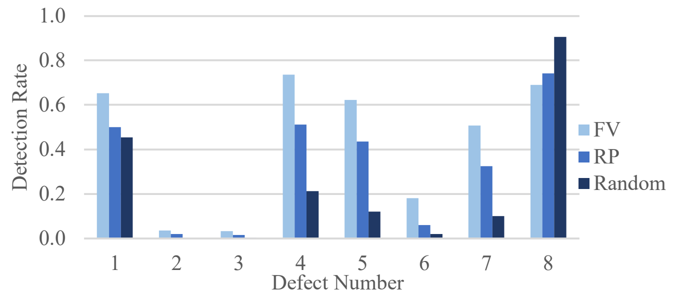


Figure 5: Detection percentage for the three generation methods - single defects

Table 2 shows that FLOLA sampling performs best on seven out of the eight defects. On these seven defects, the RP sampling also outperformed the completely random generation, as expected. On defect #8, FLOLA performs worse than RP sampling, and Random generation significantly outperforms both of the more advanced methods. Since this defect is easy to trigger (it is the only one with a detection rate above 80%), the signal parameterization and feature space exploration reduce detections slightly. Regardless, FLOLA still captures the defect a significant portion of the time at 69%. In cases with multiple defects appearing simultaneously, it will be shown that having a lower detection rate is beneficial on common defects once the algorithm needs to separate one defect from another.

In addition to the detection rate, it is also beneficial to evaluate which method detects each defect first. To measure this, twelve tests were performed with each method for each defect. The test case to first detect each defect was recorded. The mean and standard deviation of these first identifications have been calculated. A standard deviation of 0 means that defect was first detected at the same time on all 12 tests. This only occurs using FLOLA due to the structured generation of the initial Latin hypercube test cases. Recorded in Table 3 are the mean and standard deviation values, as well as 1-way ANOVA results. If the ANOVA p-value is less than 0.05, it can be said with 95% confidence that the results for the different methods come from different distributions.

Table 3: First test case to detect each individual defect, based on 12 tests each

		Test Case that First Detected Each Defect							
	Defect:	1	2	3	4	5	6	7	8
FLOLA									
	Mean	**2.00**	**44.42**	**68.33**	**2.17**	**2.00**	**5.75**	**2.00**	1.83
	Stdev	**0.00**	**38.36**	**66.16**	**0.58**	**0.00**	**4.85**	**0.00**	0.39
RP									
	Mean	3.75	72.25	94.17	4.17	2.58	19.58	9.08	1.83
	Stdev	5.55	72.65	65.04	4.45	1.98	23.09	12.99	1.59
Random									
	Mean	2.00	705.75	218.67	5.42	10.92	50.33	10.25	**1.08**
	Stdev	0.95	517.87	116.49	5.65	7.30	35.61	6.22	**0.29**
1-way ANOVA p-value		0.326	**0.000**	**0.000**	0.172	**0.000**	**0.000**	**0.043**	0.101

FLOLA outperforms RP and Random generation with statistical significance on five of the eight defects. These five defects are those where the RP and Random methods struggle to locate the detect. On the other three defects, FLOLA still performs very well, but it is also the case that the RP and Random methods quickly detect these common defects. The performance of FLOLA is especially visible on defects #2 and #3, which occur more rarely and are difficult for all methods to detect. Defects #2 and #3 are both related to the diagnosis of the ambient air sensor, which requires a very particular signal sequence to execute.

2) Multiple Simultaneous Defects

In order to evaluate more complex scenarios, the same procedure was performed where multiple defects were simultaneously present in the same model. Two defect scenarios were tested: defects #1 and #7 (scenario 10) and defects #6, #7, and #8 (scenario 11). Segregation for scenario 11 will be especially challenging due to the presence of a rare defect (#6) mixed with a common defect (#8). The total percentage of defects detected is recorded in Table 3, disregarding which defect is occurring.

Table 4: Phase 1 percentage detection results for multiple simultaneous defects

	Percentage Defect Detections	
	Scenario 10	Scenario 11
FLOLA	**0.714**	0.779
RP	0.621	0.781
Random	0.478	**0.925**

The results show that FLOLA outperformed RP and Random for scenario 10, but not scenario 11. The Random generation result from scenario 11 is nearly identical to that from defect 8 (while FLOLA's has increased), hinting that the majority of Random detections are of defect 8. When evaluating detections with multiple defects, it is insufficient to view the total detection rate. Since the goal is to detect all defects, it is much more important to evaluate which defects were detected. The number of test cases which triggered each unique defect are summarized in Table 5 below. For each scenario, 2000 tests were performed; however, the results do not sum to 2000 due to the fact that many test cases triggered more than 1 defect.

Table 5: Number of test cases to detection for multiple simultaneous defects

	Number of Test Cases Detecting Each Defect				
	Scenario 10		Scenario 11		
	1	7	6	7	8
FLOLA	**1245**	**942**	**161**	**1008**	1413
RP	1031	711	134	701	1389
Random	872	234	33	223	**1822**

Viewing these detailed results clarifies that FLOLA outperforms both other methods in detecting defects. The FLOLA result provides a much more balanced detection of defects due to the exploitation – exploration tradeoff. It is also clear that the feature transform aids in balancing the defect detections, as RP outperforms the pure random generation. The result from scenario 11 using random generation is especially skewed toward only detecting defect 8. Although its overall detection rate was highest, this skew would prove problematic during phase 2. The random detection of defect 6 is only 33, the majority of which occurred simultaneously with defects 7 or 8. Segregating defect 6 from the 1822 detections of defect 8 is analogous to finding a needle in a haystack. With FLOLA-Voronoi, defect 8 is still detected at a higher rate but the results are closer in magnitude, which will allow clustering to successfully identify these defects as separate.

Again, twelve tests were performed for each scenario in order to record how quickly each method first detected each defect. The results are shown below in Table 5, with each defect listed individually for both scenarios, as well as the overall results for each scenario. As expected, FLOLA outperforms the other two methods on all defects except #8, where it performs slightly worse than Random. This minor difference is strongly outweighed by the statistically significant improvement in detection of defects #6 and #7, even when occurring simultaneously with #8. Defect #7 is also detected at a significantly faster pace when occurring alongside defect #1 in scenario 10.

Table 6: First test case to detect each simultaneous defect, based on 12 tests

		Test Case that First Detected Each Defect						
Scenario/Defect:		10	11	1	7 (10)	6	7 (11)	8
FLOLA								
	Mean	**2.00**	1.33	**2.00**	**2.00**	**4.25**	**2.00**	1.33
	Stdev	**0.00**	0.49	**0.00**	**0.00**	**4.71**	**0.00**	0.49
RP								
	Mean	3.08	1.33	3.25	4.83	13.25	4.42	1.33
	Stdev	5.66	0.49	5.64	5.34	22.36	6.22	0.49
Random								
	Mean	2.50	**1.25**	3.00	9.58	57.83	10.25	**1.25**
	Stdev	1.00	**0.62**	1.86	7.33	53.91	12.34	**0.62**
1-way ANOVA p-value		0.728	0.909	0.644	**0.004**	**0.001**	**0.046**	1.000

B. Defect segregation

The results from phase 1 are all randomized and grouped together to form the inputs to phase 2, where the test cases are segregated into unique defects. As discussed earlier, this clustering must be performed to eliminate interference noise from multiple examples of the same defect. For example, in testing the injection of one defect (defect 5) was expected to impact 4 different indicators; actual results yielded created 400 unique output vectors with an average of 13 output vectors on each 20 second simulation. Using the heuristic described to choose initial cluster centroids, this method will tend to overestimate the number of unique defects.

1) Single Defects

The clustering is initially performed on the results from all tests where only one defect was active. There are known to be 8 unique defects, which are automatically grouped into 19 clusters using the methods described in Section V. The composition of all 19 clusters is summarized in the Figure 6 pie charts below. These can be interpreted as the purity of each cluster; the percentage of the largest component is shown next to each chart and the number of vectors in each is shown below.

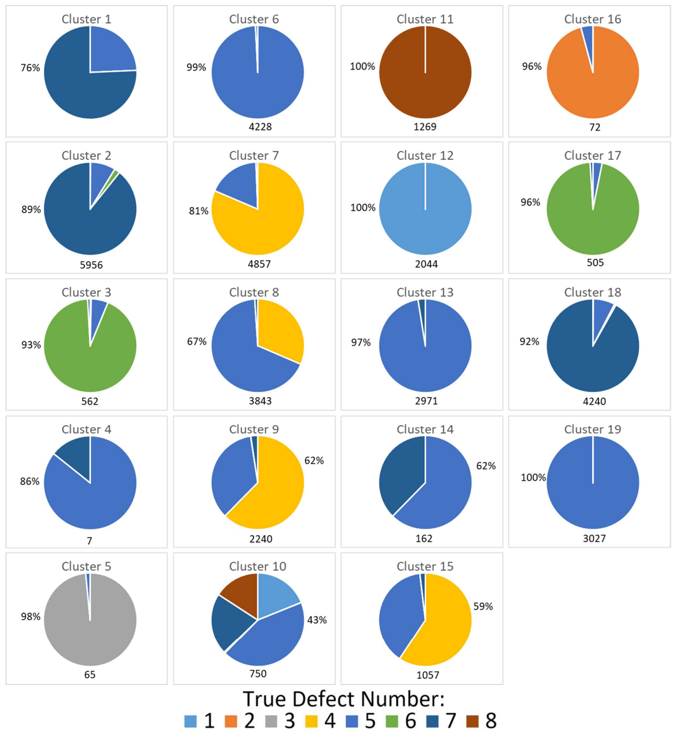


Figure 6: Distribution of single defects within clusters. The percentage of the largest constituent is shown, plus the total number of vectors in the cluster

The goal is to obtain one cluster composed primarily of each separate defect, knowing that there will be additional clusters containing heterogeneous mixtures of defects. This goal is clearly accomplished as there is a cluster with at least 81% of each unique defect; in fact, 7 of the 8 defects have clusters with concentration greater than 90%. Having one defect constitute the significant majority of a cluster guarantees its identification by OEM engineers.

Although there are over double the number of clusters as defects, 19 is still a relatively manageable number. There were a total of 39,085 vectors used as input to the clustering algorithm, of which there were 537 unique configurations. Therefore, the clustering reduced the number of potential defects from 537 to 19. Although it is possible to further reduce the number of clusters, the heuristic developed was capable of regularly guaranteeing that each defect contained a cluster where it was a significant majority. The challenge of this problem is clearly seen in the defects that occurred rarely. Defect 2 only generated 79 vectors, of which 69 were clustered together; similarly, 64 out of the 69 vectors generated by defect 3 were clustered correctly. For comparison, defect 5 generated 16,403 vectors with 400 unique vectors.

2) Multiple Simultaneous Defects

The same clustering algorithm is also applied to the test cases where multiple defects were active at the same time. These are the same test cases (scenarios 10 and 11) from phase 1, randomized and grouped together. The clustering is initiated from the beginning; the clusters and centroids determined during clustering of single defects are not used on the multiple defect data.

Using scenarios 10 and 11, there are 4 distinct defects. They were clustered by the algorithm into 9 groups as shown in Figure 9. In these tests, there were actually a greater number of unique vector combinations (703) than there were with single defects. This is due to interaction noise among defects when they occur simultaneously. The fact that this method clustered the results into a smaller number of clusters reinforces the heuristic used to determine initial cluster centroids. The heuristic was able to determine that, despite having a larger number of unique vectors, the number of true defects is less. The number of clusters is approximately twice the number of true defects, but is two orders of magnitude less than the number of unique vectors. This result gives phase 3 a reasonable number of clusters to evaluate. Again, it can be clearly seen that each defect has one cluster where it comprises a strong majority (greater than 80%), even with rare defects such as defect #6. Clustering was performed with all vectors, but the plots were generated using only the vectors where one defect was active at a time to prevent double or triple counting some vectors.

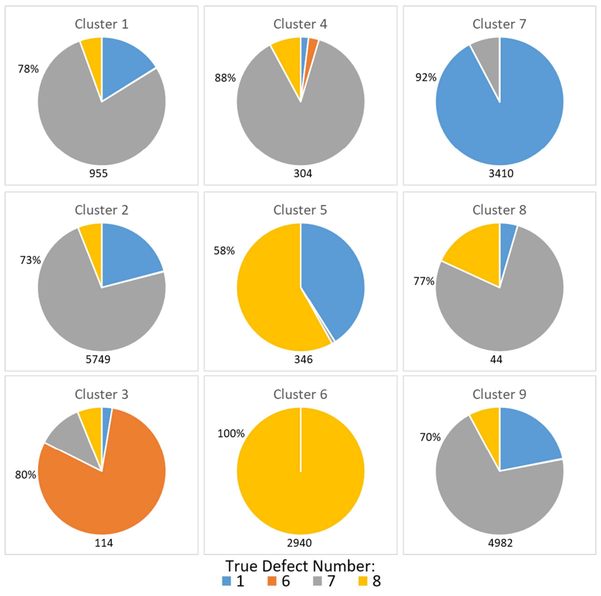


Figure 7: Distribution of multiple defects within clusters

C. *Defect reproduction*

The algorithm extracts the minimum test time at which each cluster occurred. For most clusters, this time is significantly less than the 15 seconds used in phase 1. The clusters with majority of the same defect share the same minimum time.

Table 7: Minimum test time for individual defect clusters

Cluster	Min Time (s)	Cluster	Min Time (s)	Cluster	Min Time (s)
1	3.05	8	5.05	15	3.05
2	3.05	9	3.05	16	7.01
3	3.13	10	3.28	17	3.13
4	7.83	11	0.03	18	3.05
5	7.01	12	0.03	19	5.04
6	5.01	13	3.24		
7	3.05	14	3.3		

The 5-way interactions for 32 input parameters result in an additional 214 tests to be performed at minimum test time for each cluster. The t-way switches for each parameter are recorded on each test, as well as if the defect was triggered. It is expected that a defect will only trigger for each cluster if several signals are in a specific switch position. To determine this, the standard deviation and mean of the proportionality for each input signal are calculated for the test cases which triggered the defect and those which did not trigger it.

The results identify the parameters that trigger each defect, as well as what value each should take. Any parameter with a standard deviation value below a threshold of 0.25 is determined to trigger the defect. A low standard deviation on that particular parameter means it favors one value set whenever a defect triggered. From these selected parameters, the mean proportionality value is measured in order to find the values it should take; this is based upon the two sets of values supplied by development engineers before testing begins. This information, combined with the minimum test time, places constraints on certain input parameters so that each defect can be easily reproduced. The output information from phase 3, and the overall method, is a list of defects where each is defined by its minimum test time, a set of input signals that trigger it, and the value set that each trigger input should be drawn from.

A graphical representation of phase 3 results for two different clusters is shown in Appendix B. These two defects both depend upon three input parameters, two of which are the same. This is determined using the standard deviation, where three parameters have a standard deviation value of 0. Every time these defects triggered, those parameters were chosen from the same value set. The 0.25 threshold on standard deviation cleanly separates these trigger signals from the signals that are independent of the defect. Viewing the plots of mean proportionality, it can be seen that the signals take different values for the different defects. This provides a different set of criteria for the two different defects. This allows for the two defects to be confirmed by engineers as distinct, and also provides the information needed to reliably reproduce each.

VIII. Conclusion

It has been shown that this method is able to detect defects much more rapidly and reliably than the benchmark methods. This method requires as input only the simply parameter definitions shown in Appendix A, plus the validated Simulink model (which is often already available to development engineers). In addition to identifying defects, it is able to determine how many unique defects and separate all test cases into clusters that correspond to these unique defects. This information is invaluable to engineers while they try to determine the root cause of each defect. In addition to providing these examples, the solution also analyzes the input signals in order to identify the causal signals; furthermore, it places constraints on the input values as well as the test time.

The overall testing approach described in this paper has been tailored for the automotive use case and specifically testing of embedded ADAS system software. Multiple components of the method have been previously proven in different fields. Due to this, as well as general similarities among embedded software, it is expected that this solution could be easily generalized to various other software testing scenarios within and outside the automotive domain.

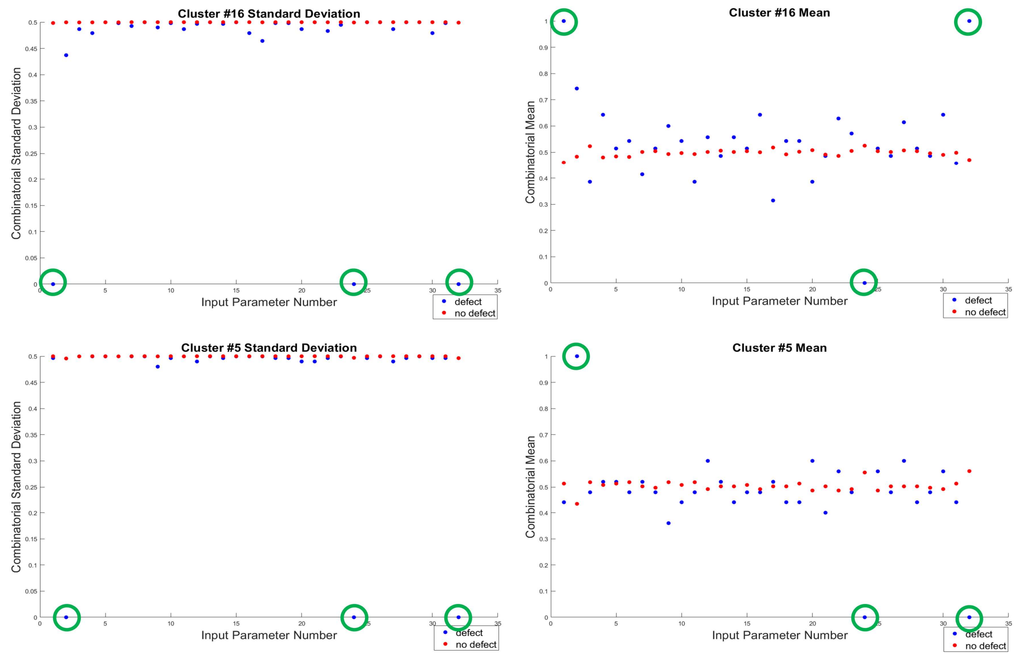
APPENDICES

Appendix A: Input parameters for example software

Parameter Name	Type	Refresh (ms)	ValueSet1	ValueSet2	Info1	Info2	Description
Parameter1	pulse_limited	1000	[12]	[0]	100		Ignition voltage
Parameter2	pulse_fixed	100	[0]	[1]	11	2	Diag request
Parameter3	pulse_variable	100	[0]	[1]	500	1	Diag request
Parameter4	pulse_fixed	100	[0]	[1]	11	2	Diag request
Parameter5	boolean	10	[0]	[1]	4		Camera ECU malfunction
Parameter6	boolean	10	[0]	[1]	4		Camera temperature malfunction
Parameter7	boolean	10	[0]	[1]	4		Camera voltage malfunction
Parameter8	boolean	10	[0]	[1]	4		Camera out of aim
Parameter9	boolean	10	[0]	[1]	4		Camera aim adjustment not finished
Parameter10	boolean	10	[0]	[1]	4		Camera blocked
Parameter11	boolean	10	[0]	[1]	4		Yaw-sensor invalid
Parameter12	boolean	10	[0]	[1]	4		Yaw-sensor malfunction
Parameter13	step	10000	(1:1:58)'	[0;59]	2		Meter speed tolerance A
Parameter14	boolean	10	[0]	[1]	4		Wheel speed FR invalid
Parameter15	boolean	10	[0]	[1]	4		Wheel speed FL invalid
Parameter16	boolean	10	[0]	[1]	4		Wheel speed RR invalid
Parameter17	boolean	10	[0]	[1]	4		Wheel speed RL invalid
Parameter18	boolean	30	[0]	[1]	4		Wheel speed malfunction
Parameter19	boolean	30	[0]	[1]	4		Wheel speed malfunction
Parameter20	boolean	30	[0]	[1]	4		Wheel speed malfunction
Parameter21	boolean	30	[0]	[1]	4		Wheel speed malfunction
Parameter22	boolean	20	[0]	[1]	4		Vehicle speed malfunction
Parameter23	step	1000	(1:1:5)'	[0]	1		Front wiper state
Parameter24	step	1000	(5:1:95)'	[0;3;4]	3		Ambient temperature sensor
Parameter25	boolean	10	[0]	[1]	4		CAN communication invalid
Parameter26	boolean	12	[0]	[1]	4		CAN communication invalid
Parameter27	boolean	24	[0]	[1]	4		CAN communication invalid
Parameter28	boolean	1000	[0]	[1]	1		CAN communication invalid
Parameter29	boolean	1000	[0]	[1]	1		CAN communication invalid
Parameter30	boolean	1000	[0]	[1]	1		CAN communication invalid
Parameter31	boolean	500	[0]	[1]	1		CAN communication invalid
Parameter32	boolean	1000	[0]	[1]	1		CAN communication invalid

The Type column defines the overall behavior of a signal, e.g. it will pulse from one value to be temporarily at another, or will step from one value to adjacent values. Refresh is the bus refresh rate for each signal. ValueSet 1 and 2 are the two sets of possible signal values.

Appendix B: Phase 3 example, proportionality standard deviation and mean plots for two defects



Standard deviation (left) and mean (right) plots for two separate defects. Trigger signals are circled in green.

ACKNOWLEDGMENT

This work has been conducted within the ENABLE-S3 project that has received funding from the ECSEL Joint Undertaking under grant agreement No 692455. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation program and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia, and Norway. The authors would also like to thank the contributions of Professors Tom Dhaene and Dirk Deschrijver from the Department of Information Technology at Ghent University – imec.

REFERENCES

[1] A. Broggi *et al.*, "Extensive tests of autonomous driving technologies," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 3, pp. 1403–1415, 2013.

[2] Waymo, "On the road to Fully Self-driving," 2017.

[3] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition," *Neural Networks*, vol. 32, pp. 323–332, 2012.

[4] H. Zhu, K. V. Yuen, L. Mihaylova, and H. Leung, "Overview of Environment Perception for Intelligent Vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 10, pp. 2584–2601, 2017.

[5] N. Steinkamp and R. Levine, "2017 Automotive Warranty & Recall Report," 2017.

[6] M. McCarthy, "Update on Light Duty OBD II," Pasadenca, CA, 2005.

[7] S.-H. Jeon, J.-H. Cho, Y. Jung, S. Park, and T.-M. Han, "Automotive hardware development according to ISO 26262," *13th Int. Conf. Adv. Commun. Technol.*, pp. 588–592, 2011.

[8] C. Wewetzer, K. Lamberg, and R. Otterbach, "Creating Test Patterns for Model-Based Development of Automotive Software," *SAE Tech. Pap.*, vol. 1598, no. 01, 2006.

[9] Y. Y. Wang, Y. Sun, C. F. Chang, and Y. Hu, "Model-based fault detection and fault-tolerant control of SCR urea injection systems," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 4645–4654, 2016.

[10] M. Conrad, I. Fey, and S. Sadeghipour, "Systematic Model-Based Testing of Embedded Automotive Software," *Electron. Notes Theor. Comput. Sci.*, vol. 111, no. SPEC. ISS., pp. 13–26, 2005.

[11] E. Bringmann and A. Krämer, "Model-Based Testing of Automotive Systems," *2008 1st Int. Conf. Softw. Testing, Verif. Valid.*, pp. 485–493, 2008.

[12] J. Kapinski, J. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-guided approaches for verification of automotive powertrain control systems," *Proc. Am. Control Conf.*, vol. 2015–July, pp. 4086–4095, 2015.

[13] J. L. Boulanger and V. Q. Dao, "Requirements engineering in a model-based methodology for embedded automotive software," *RIVF 2008 - 2008 IEEE Int. Conf. Res. Innov. Vis. Futur. Comput. Commun. Technol.*, vol. 00, no. c, pp. 263–268, 2008.

[14] G. Bahig and A. El-Kadi, "Formal Verification of Automotive Design in Compliance with ISO 26262 Design Verification Guidelines," *IEEE Access*, vol. 5, pp. 4505–4516, 2017.

[15] G. E. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel, "Verification of automotive control applications using S-TaLiRo," *Am. Control Conf. (ACC), 2012*, pp. 3567–3572, 2012.

[16] T. Schmidt, S. Jin, J. Rogalli, T. Rogier, H. Pohlheim, and I. Stürmer, "Efficient Testing Framework for Simulink Models with MTCD and Automated Test Assessments in the Context of ISO 26262," *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.*, vol. 7, no. 1, pp. 2014-01–0306, 2014.

[17] I. Arsie, G. Betta, D. Capriglione, A. Pietrosanto, and P. Sommella, "Functional testing of measurement-based control systems: An application to automotive," *Meas. J. Int. Meas. Confed.*, vol. 54, pp. 222–233, 2014.

[18] M. Lochau and U. Goltz, "Feature interaction aware test case generation for embedded control systems," *Electron. Notes Theor. Comput. Sci.*, vol. 264, no. 3, pp. 37–52, 2010.

[19] R. Parker, A. Mouzakitis, S. Puthiyapurayil, and N. Muniyappa, "Advanced Automated Onboard Vehicle Diagnostics Testing," *UKACC Int. Conf. Control 2010*, vol. 44, no. 0, pp. 757–762, 2010.

[20] M. Tatar and J. Mauss, "Systematic Test and Validation of Complex Embedded Systems," *Embed. Real Time Softw. Syst.*, pp. 5–7, 2014.

[21] A. Feliachi and H. Le Guen, "Generating transition probabilities for automatic model-based test generation," *ICST 2010 - 3rd Int. Conf. Softw. Testing, Verif. Valid.*, no. X, pp. 99–102, 2010.

[22] A. L. Raffaëlli *et al.*, "Facing ADAS Validation Complexity with Usage Oriented Testing," in *Embedded Real Time Software and Systems*, 2016.

[23] K. G. Larsen, F. Lorber, B. Nielsen, and U. M. Nyman, "Mutation-Based Test-Case Generation with Ecdar," in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2017.

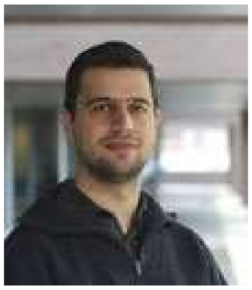
[24] A. Fellner, W. Krenn, R. Schlick, T. Tarrach, and G. Weissenbacher, "Model-based , mutation-driven test case generation via heuristic-guided branching search," in *MEMOCODE '17 Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, 2017, pp. 56–66.

[25] G. Betta, D. Capriglione, A. Pietrosanto, and P. Sommella, "A methodology to test instrument software: An automotive diagnostic system application," *IEEE Trans. Instrum. Meas.*, vol. 57, no. 12, pp. 2733–2741, 2008.

[26] H. Pohlheim, M. Conrad, and A. Griep, "Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences," *Analysis*, no. 724, pp. 804--814, 2005.

[27] T. E. J. Vos, F. F. Lindlar, B. Wilmes, H. Gross, and J. Wegener, “Evolutionary functional black-box testing in an industrial setting,” *Softw. Qual. J.*, no. 21, pp. 259–288, 2013.
[28] D. Gorissen DIRKGORISSEN, U. Ivo Couckuyt, P. Demeester, T. Dhaene TOMDHAENE, and K. Crombecq KARELCROMBECQ, “A Surrogate Modeling and Adaptive Sampling Toolbox for Computer Based Design,” *J. Mach. Learn. Res.*, vol. 11, pp. 2051–2055, 2010.
[29] J. van der Herten, I. Couckuyt, D. Deschrijver, and T. Dhaene, “A Fuzzy Hybrid Sequential Design Strategy for Global Surrogate Modeling of High-Dimensional Computer Experiments,” *SIAM J. Sci. Comput.*, vol. 37, no. 2, pp. A1020–A1039, 2015.
[30] Z. Huang, “Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values,” *Data Min. Knowl. Discov.*, vol. 2, no. 3, pp. 283–304, 1998.
[31] D. R. (NIST) Kuhn, R. N. (NIST) Kacker, and Y. (NIST) Lei, “Practical Combinatorial Testing,” 2010.
[32] P. Skruch and G. Buchala, “Model-Based Real-Time Testing of Embedded Automotive Systems,” *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.*, vol. 7, no. 2, 2014.
[33] H. Shokry and M. Hinchey, “Model-Based Verification of Embedded Software,” *Computer (Long. Beach. Calif).*, vol. 42, no. 4, pp. 53–59, 2009.
[34] B. Murphy, A. Wakefield, and J. Friedman, “Best Practices for Verification, Validation, and Test in Model- Based Design,” *SAE Tech. Pap.*, vol. 1469, no. 01, 2008.
[35] T. W. Simpson, D. K. J. Lin, and W. Chen, “Sampling Strategies for Computer Experiments: Design and Analysis,” *Int. J. Reliab. Appl.*, vol. 2, no. 3, pp. 209–240, 2001.
[36] L. Bai, J. Liang, C. Dang, and F. Cao, “A novel attribute weighting algorithm for clustering high-dimensional categorical data,” *Pattern Recognit.*, vol. 44, no. 12, pp. 2843–2861, 2011.
[37] A. Survey and J. Offutt, “Combination Testing Strategies :,” *Softw. Testing, Verif. Reliab.*, vol. 15, no. 3, pp. 1–32, 2004.
[38] Toyota Europe, “Toyota Safety Sense.” .

Kyle Foss obtained his M.S. degree in engineering from the University of California, Los Angeles in 2016. He received his B.S. in mechanical engineering from California State University, Long Beach in 2013. In 2017, he became a researcher with the IDLab, Department of Information Technology, Faculty of Engineering, Ghent University. He previously worked as a test engineer for Volkswagen Group of America.



Ivo Couckuyt received the M.Sc. degree in computer science from the University of Antwerp, Antwerp, Belgium, in 2007, and the Ph.D. degree from IDLab, Ghent University, Ghent, Belgium, in 2013. In 2007, he joined the research group Computer Modeling and Simulation (now merged with CoMP) supported by a research project of the Fund for Scientific Research Flanders (FWO-Vlaanderen). He is currently an FWO Post-Doctoral Research Fellow with the IDLab, Department of Information Technology, Faculty of Engineering, Ghent University.

Adrian Baruta is a - Dipl. Engineer in Automation and Computer Science (2001 – 2006 University “Politehnica” Timisoara). From 2005 until 2014 he worked as an engineer in the design, development and testing of safety critical automotive electronic control units for Continental Automotive. He joined Toyota Motor Europe in 2014 as a senior engineer working for the process development for the validation of Advanced Driving Assistant Systems and development of cybersecurity requirements for in vehicle networks, and is active member in AUTOSAR consortium for the development of standard security requirements.

Corentin Mossoux is a - Dipl. Engineer in Electromechanical Engineering (2014 – 2016 University of Antwerp, Belgium). Since September 2017 he joined Toyota Motor Europe as a junior engineer, working for the process development for the validation of Advanced Driving Assistant Systems and development of cybersecurity requirements for in vehicle networks.