# Systematic Design and Automated Execution of Embedded System Tests

Peter M Kruse, Jörg Reiner

▶ **To cite this version:**

Peter M Kruse, Jörg Reiner. Systematic Design and Automated Execution of Embedded System Tests. Embedded Real Time Software and Systems (ERTS2014), Feb 2014, Toulouse, France. hal-02272353

**HAL Id: hal-02272353**
**https://hal.science/hal-02272353**

Submitted on 27 Aug 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Systematic Design and Automated Execution of Embedded System Tests

Peter M. Kruse
Berner & Mattner Systemtechnik GmbH
Gutenbergstr. 15
Berlin, Germany
peter.kruse@berner-mattner.com

Jörg Reiner
Berner & Mattner Systemtechnik GmbH
Erwin-von-Kreibig-Str. 3
Munich, Germany
joerg.reiner@berner-mattner.com

## ABSTRACT

**During the development of embedded systems, tests are performed on various test platforms, such as hardware-in-the-loop platforms in order to find faults. Test cases must be specified to verify the properties demanded of the system on these test platforms. The main challenge is to find relevant test cases, since in most cases not all possible test cases can be found and executed. Combinatorial testing can solve this task systematically.**

**In this paper, we describe the integrated test design and test automation using the industrial testing tools CTE XL Professional and MESSINA. In a case study with an antilock braking system, we demonstrate the operation of the system.**

## 1 INTRODUCTION

A great number of today's products are based on the deployment of embedded systems. In industrial applications, embedded systems are predominantly used for controlling and monitoring technical processes.

In order to be able to find faults in the embedded system under development before its deployment into the target environment, tests are usually carried out on various in-the-loop test platforms. In-the-loop means that there is bidirectional interaction between the embedded system and its environment: the environment stimulates the sensors of the system, and in turn the system affects the environment using its actuators. Depending on the artifacts that are modeled and simulated, different in-the-loop test approaches are distinguished.

While a Model-in-the-loop (MiL) testing allows early tests with a model based approach; Software-in-the-loop (SiL) testing aims at testing executable artifacts of software components. With Hardware-in-the-loop (HiL) tests, the software integrated into the target hardware (e.g. an embedded controller) is examined. In all these cases the environment of the system under test is simulated.

Test cases must be created to verify the specified properties of the developed system on the different test platforms. Exhaustive testing is usually not possible due to the enormous amount of possible input situations and combinations of input signals. Care must therefore be taken to select relevant test cases, as the quality of the overall test directly depends on it. It is therefore necessary to follow a systematically approach, e.g. in terms of combinatorial testing.

In this paper, we describe the integration of the industrial testing tools MESSINA and CTE XL Professional.

### 1.1 MESSINA

MESSINA [1] is a testing environment for the test of embedded systems. It allows the connection to the embedded devices with their various interfaces, like electrical connections (e.g. analogue and digital signals) or bus systems (e.g. CAN) as well as sensor interfaces (e.g. cameras). MESSINA allows the implementation of hardware- and software-independent test sequences specified using different notations such as UML or Java. Using abstraction layers allows universal test execution on MiL, SiL and HiL platforms. One of these layers is the signal pool containing all system signals provided by the connected devices or software. The signal pool allows easy read and write access to all the signals used by the system under test (SUT) or the simulation environment.

Several approaches are available in MESSINA for the creation of test cases. Search-based testing approaches for example using evolutionary algorithms are described in [2]. Another approach is the application of Combinatorial Testing provided by CTE XL Professional.

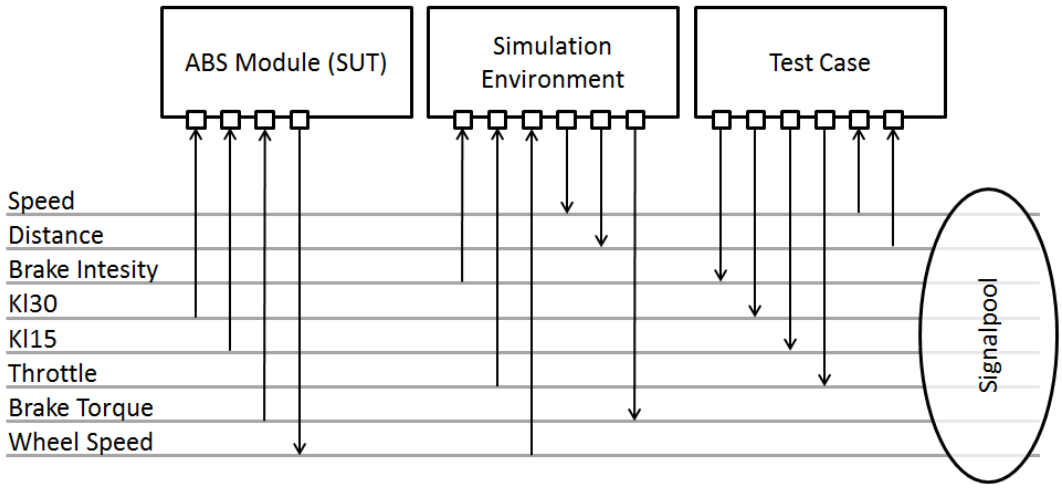


**Figure 1: MESSINA MiL test environment configuration**

### 1.2 CTE XL Professional

CTE XL Professional [3] is a popular tool for systematic combinatorial test design. It implements and supports the test design technique called Classification Tree Method [4].

Basically the method consists of classifying the combinatorial aspects relevant for the SUT into *classifications* and then decomposing each into disjoint *classes*. Found classes can again be classified in different ways, so called *refinements*. This creates a tree made of classes and classifications, a *Classification Tree*. By means of the classification tree, the CTE XL Professional then generates test cases by combining the individual classes of the various classifications. For a test generation with pairwise combination coverage, a test suite is obtained, that uses every class pair from disjoint classifications at least once in a test case. Since large trees can generate many test cases due to the combinatorial explosion, CTE XL Professional offers prioritization of classes in a tree and hence the generated test cases will have a corresponding prioritization [5].

## 2 APPROACH

We are illustrating, how a classification tree structure with test cases can be created and exported to MESSINA for execution.

The general idea is to implement a general test case in MESSINA, which is then parameterized several times with test cases from CTE XL Professional.

### 2.1 Test Object: Anti-lock Braking System

The anti-lock braking system (ABS) is a system which prevents the wheels of a vehicle from locking while braking, in order to allow the driver to maintain steering control under heavy braking conditions and, in most situations, to shorten braking distances. Sensors of the ABS measure the speed at which the wheels are turning. If the speed decreases rapidly, the electronic control system reports blocking danger. The pressure of the brake hydraulics is reduced immediately and then raised again to a point slightly below the blocking threshold. This process is repeated several times per second. The goal of the anti-locking control system is to maintain the slip of the wheels at a level which guarantees the highest braking power and the highest steerability of the vehicle.

### 2.2 Test Implementation

The first step is the creation of a system setup in MESSINA. The runtime environment consists of the SUT as well as environment simulation models. In our configuration, we start with a MiL test, since actual hardware is not yet available. Sensors and actors are also simulated in our configuration. For simulating driving and braking maneuvers, a car simulation model is connected to the SUT using the signal pool. Test cases also connect to the signal pool, so they are able to read, trigger, manipulate and assess all data flow between the SUT and the simulation environment. The complete setup is given in Figure 1.

A MESSINA test case implementation typically consists of three sections, a pre-condition part (used for initialization), the actual test action and a post-condition part (used for cleanup).

```
public class CheckBrakingDistance {

  public boolean preCondition() {
    throttle.setValue(0);
    brakeIntensity.setValue(100);
    ASSERT(speed.waitValue(0, 60000),
        "Timeout-preCondition");
    brakeIntensity.setValue(0);
    return true;
  }
```

**Listing 1: Initialization of the system**

In the pre-condition part, the system is brought to a meaningful starting point, e.g. the speed is reduced to zero.

```
public int run() {
    kl15.setValue(true)
    kl30.setValue(true)
    throttle.setValue(50);
    ASSERT(speed.waitValue(50, 60000),
        "Timeout-acceleration");
    throttle.setValue(0);
    int startPos = distance.getValue();
    brakeIntensity.setValue(100);
    ASSERT(speed.waitValue(0, 60000),
       "Timeout-brake");
    int stopPos = distance.getValue();
    int total = stopPos - startPos;
    ASSERT(total > 27,
       "FAIL: Distance too long with "
       + total + "m.");

    return 0; // 0 means PASS
  }
```

**Listing 2: Actual test execution**

In the test implementation (the run-method), the car is then accelerated to a given speed (50m/s), then a full brake is performed (with 100% brake intensity). When the car has come to a full stop, the braking distance is evaluated. If it is below a given threshold (27m), the test case passes. Otherwise, the test case produces a failure.

```
public boolean postCondition() {
    brakeIntensity.setValue(0);
    return true;
  }
}
```

**Listing 3: Post-condition and cleanup**

In the post-condition part, the brake is released.

## 2.3 Test Generalization

The concrete values in the test case can also be extracted so that the test case can be parameterized. Thus, different *Speeds*, *Brake Intensities* and so on can be tested using the same test template. For the combination of different parameter values, we use the CTE XL Professional.

## 2.4 Test Variation

The CTE XL Professional project is used to illustrate the campaign test cases and their parameters as a tree structure. Figure 2 shows the classification tree structure as created in CTE XL Professional.

All parameters (*Speed*, *BrakeDistance*, *BrakeIntesity*, *KL30_Status*, *KL15_Status*, *ManeuverDurantion*, *ManeuverDataset*) and their respective values are graphically displayed to illustrate their connection to the test cases. The test case groups each contain test cases which use different parameter settings as defined by the tree structure. These parameter values will be assigned to the MESSINA test case for execution of the test.

The CTE XL Professional export creates campaign structures in the MESSINA project based on the classification tree. The parameters for each test case execution within each campaign are then also defined in the classification tree.

The test case group (*Braking Distance Tests*) will be exported as a campaign into MESSINA. The actual export then consists of specifying the MESSINA project folder and selecting the template test case.

After a refresh of the project folder has been done in MESSINA, the project structure is updated (Figure 3). The new campaigns have been added. The test case (selected during the export process) is added to each campaign with the corresponding parameter settings as defined in the CTE XL Professional project.

The parameter values for each test case (within each campaign) can be examined by double clicking on the parameter name. The values correspond to those defined in the CTE XL Professional project.

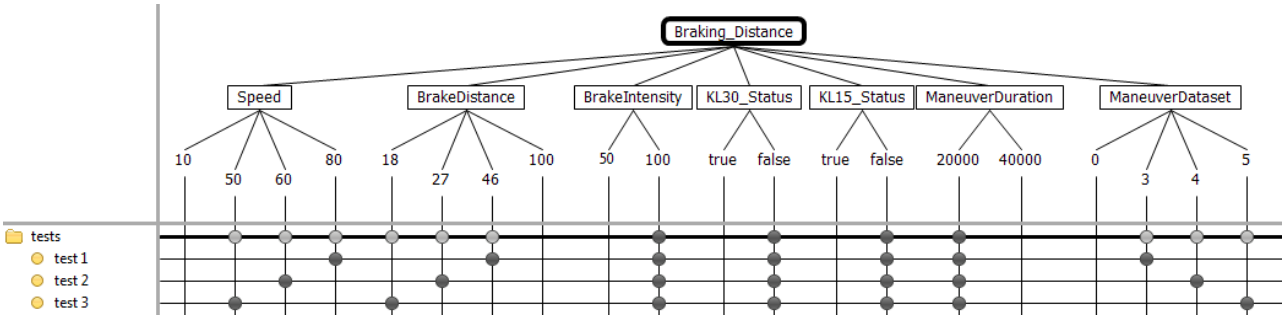


**Figure 2: Classification Tree**

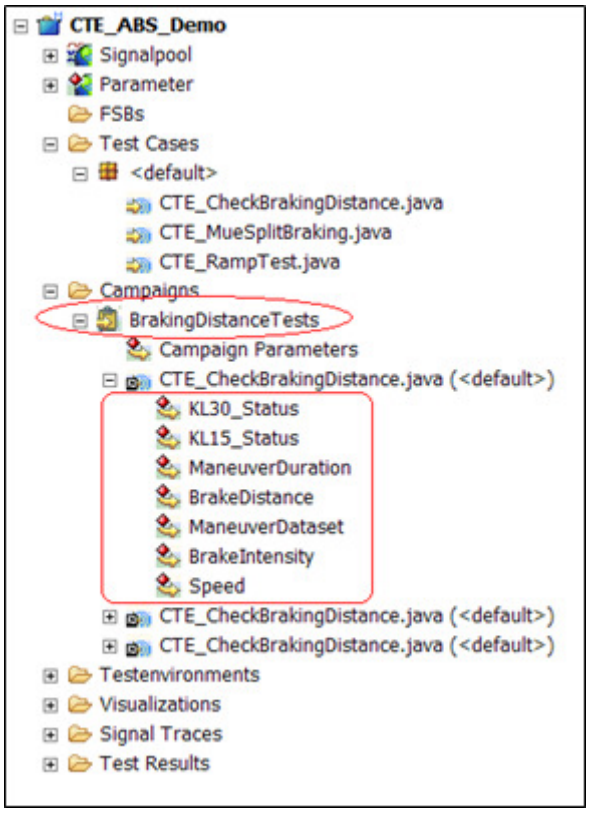


**Figure 3: Resulting MESSINA campaign**

## 2.5 Test Execution

The campaigns can then be executed in the usual way. All different configurations of the master test case are executed then one by one. In our example, this would be all three test cases. By the means of the MESSINA hardware abstraction layer the test cases can be run on all test configurations from MiL to HiL.

For the execution of test cases in a HiL setup, with actual hardware of the system under test, the MESSINA configuration would look slightly different (Figure 4). In the upper part, the ABS Module (SUT) is missing, as it is replaced by an ABS Electronic Control Unit (ECU). When the SUT evolves to an embedded system, as in our case, the access is done by hardware access (e.g. bus system or digital I/O or analogue I/O). MESSINA provides adapters between these interfaces and the signal pool. I.e. that environment models and test cases can be reused without modifications.

This combines both, the systematic test case generation and the consistent verification from early stages of the development process to the complete system test. The test reports of the executed test runs contain all relevant configuration information including the parameter set generated by CTE XL Professional. Reusable test cases on various integration levels (e.g. MiL, SiL, HiL) reduce creation and maintenance efforts and therefore cuts cost in embedded software development.

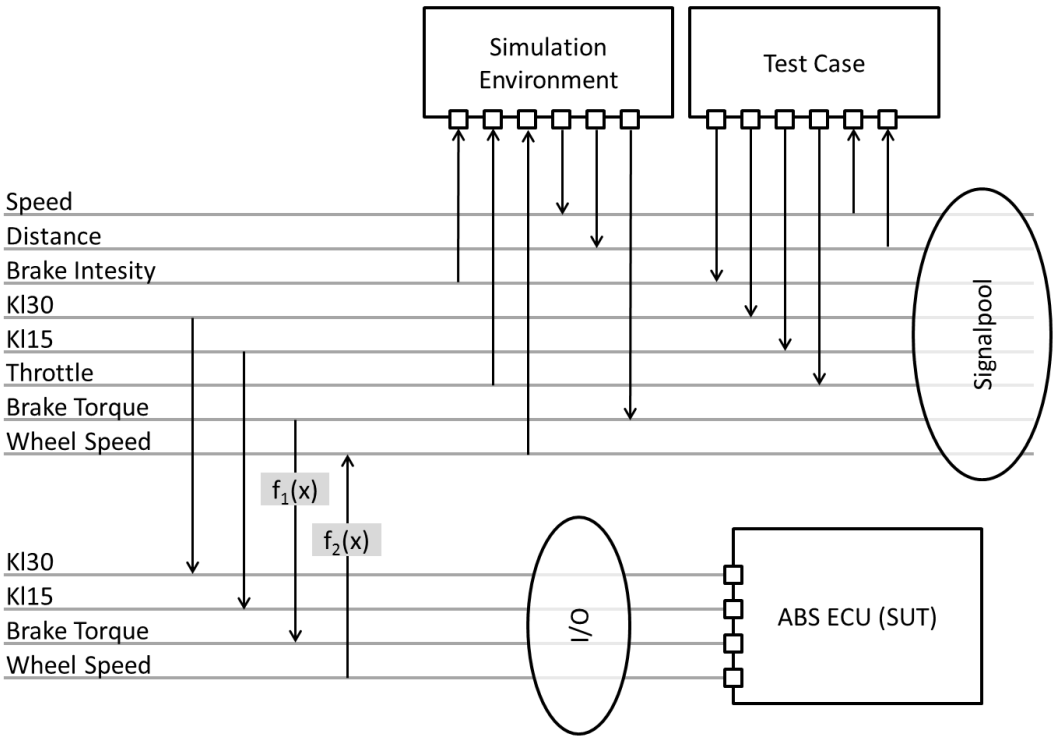


**Figure 4: MESSINA HiL test environment configuration**

## 3 BEYOND PARAMETERIZATION

In this work, a master test case is to be implemented in MESSINA by the test engineer which is then parameterized using the CTE XL Professional. While this approach is already of great help for the test engineer, it does not benefit from all available test generation facilities of CTE XL Professional, i.e. the generation of semantically correct test sequences [6].

For testing state-based systems with continuous actions, test cases must therefore reflect these properties of the SUT in a certain order. The outcome of one test case is used as the input for the next test case. The composition of several test cases into a larger test scenario, so called test sequence, and their generation remains a challenging task. Furthermore, test steps as part of test sequences cannot be composed in any arbitrary order as it is required for some configurations of the software that other things have been done first.

In Figure 5, an example tree for driving maneuvers of a car is given. Different steering wheel angles and speeds are available.

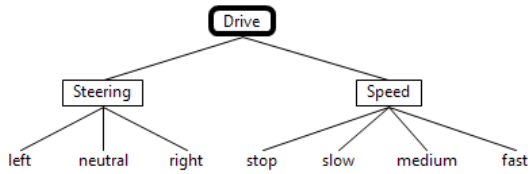


**Figure 5: Classification Tree for Driving Car**

Since the different configurations are only available depending on previous states of the car (e.g. steering to left is not directly reachable if steering is currently to right), not all possible tests can be performed in any arbitrary order. Therefore, the tester can assign allowed transitions to the classes of the classification tree.

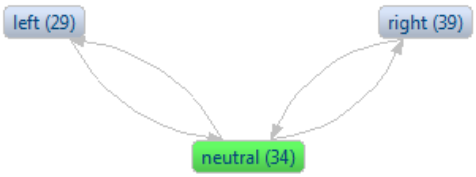


**Figure 6: Allowed Transitions for Steering**

The resulting valid transitions for Steering can be seen in Figure 6. Initially, the steering is neutral, straight forward (indicated by green background of the class). From neutral, the steering wheel can be turned to left and right, and also back from left and right, but there is no direct connection between left and right.


**Figure 7: Allowed Transitions for Speed**

The available transitions for the car's speed are given in Figure 7. Initially, the car is always standing still (stop, again indicated by green background). It can accelerate to slow, medium and fast, but only in that order, and decelerate in reverse order from fast to medium to slow.

The allowed transitions can then be used for test sequence generation in CTE XL Professional.

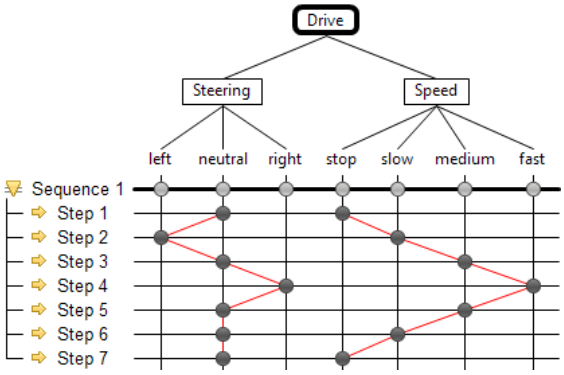


**Figure 8: Resulting Test Sequence**

The resulting test sequence is given in Figure 8. The sequence covers each transition at least once. The steering wheel is turned from neutral to the left, back to neutral, to the right and back to neutral, covering all four transitions for steering (as given in Figure 6). For the speed of the car, the sequence uses all possible transitions by accelerating from the stopped car to fast speed traversing low and medium speed and decelerates back to stop using medium and low speed again. The sequence containing of seven steps covers all possible six transitions of speed (as given in Figure 7).

For further use in MESSINA, the resulting sequence currently needs to be implemented in Java manually. By assigning Java statements to the corresponding elements of the classification tree (i.e. setter-/getter-methods to the classifications, concrete values to classes) the whole process can be further automatized. We are also evaluating the use of different coverage criteria for the actual generation of sequences.

## 4 SEARCH-BASED TESTING

The use of search-based techniques for functional testing of embedded systems has been described in [2].

Evolutionary testing is based on meta-heuristic search techniques such as evolutionary algorithms. An evolutionary algorithm is an optimization technique based on the principles of the Darwinian theory of evolution. It implements an iteratively process of evaluation, selection, crossover, mutation and population update, which is repeated unless a termination criterion applies, such as that the ideal solution is found. With evolutionary testing, the test objective considered is transformed into an optimization problem. The input domain of the test object forms the search space in which an evolutionary algorithm searches for test data that fulfils the respective test objective.

Through the EvoTest project, an extensible and open automated evolutionary testing architecture and framework was developed [7] that provides general components and interfaces to facilitate the automatic generation, execution, monitoring and evaluation of test cases using evolutionary computation.

Empirical evaluation in an industrial context using MESSINA has been evaluated in [8]. In summery the study identifies the creation of appropriate fitness functions the most challenging task, although results indicate that evolutionary functional testing using MESSINA is scalable and applicable.

## 5 CONCLUSSION

In this paper, we presented an approach for systematic test design and test automation of MiL, SiL and HiL tests. We have laid out a simple system configuration for MESSINA consisting of a SUT and an environment simulation. An existing single test case is then parameterized allowing the use of CTE XL Professional for systematic test case design. Resulting test suites are exported from CTE XL Professional to MESSINA for execution and evaluation. The repeated instantiation of the master test cases reduces the manual effort of test case creation and maintenance.

For future work, we see a more tightened integration of both tools, to ease the systematic specification of embedded system tests and their automated execution. This might include actual Java test case generation from CTE XL Professional instead of parameterizing existing test cases. The distinction between pre-condition, test action and post-condition requires special attention, which might be handled more naturally with the test sequence generation facilities [6] of CTE XL Professional.

Post evaluation is another topic we would investigate further. CTE XL Professional supports the analysis of test results for specified test case to assess the influence of individual parameter values (the class) on test success or failure. Having a test suite with test results available allows for an in-depth root-cause-analysis.

We will also continue on the Search-based Software Testing track, as already described in [2].

## REFERENCES


[1] Berner & Mattner Systemtechnik GmbH MESSINA. www.berner-mattner.com/en/berner-mattner-home/products/messina

[2] P.M. Kruse, J. Wegener, and S. Wappler. A highly configurable test system for evolutionary black-box testing of embedded systems. GECCO 2009, Montreal, Canada, 2009.

[3] Berner & Mattner Systemtechnik GmbH, CTE XL Professional. http://www.cte-xl-professional.com/

[4] M. Grochtmann and K. Grimm. Classification trees for partition testing. Softw. Test., Verif. Reliab., 3(2):63--82, 1993.

[5] P.M. Kruse and M. Luniak. Automated test case generation using classification trees. Software Quality Professional, 13(1):4--12, 2010.

[6] P.M. Kruse and J. Wegener. Test Sequence Generation from Classification Trees. A-MOST 2012/ICST 2012, Montreal, Canada, April 2012.

[7] M. Dimitar, I. M. Dimitrov, and I. Spasov. Evotest-Framework for customizable implementation of Evolutionary Testing. International Workshop on Software and Services. 2008.

[8] T. E. J. Vos, F.F. Lindlar, B. Wilmes, A. Windisch, A. I. Baars, P. M. Kruse, H. Gross, and J. Wegener. Evolutionary functional black-box testing in an industrial setting. Software Quality Journal (2013): 1-30.