Evolutionary Functional Testing of an Automated Parking System

Oliver BUEHLER
STZ Softwaretechnik
Im Gaugenmaier 20, D-73730 Esslingen-Zell, Germany

and

Joachim WEGENER
DaimlerChrysler AG, Research and Technology
Alt-Moabit 96a, D-10559 Berlin, Germany

ABSTRACT

Evolutionary Testing is a promising approach for automating the testing of software-based systems. A number of papers have been published in the last years which have successfully applied evolutionary algorithms for test data generation. However, none of these papers address functional testing - the testing of the system's logical behavior on the basis of the system specification - which is, in practice, the most important and most common class of the methods. In this work we present the application of evolutionary testing to the functional testing of an automatic parking system which could automate the parking procedure in future cars. A test environment is described which automatically generates interesting driving maneuvers, performs a simulation of the system with the generated maneuvers and continuously improves the test quality.

Keywords: Evolutionary Functional Testing, Testautomation

1. INTRODUCTION

A considerable number of today's products is based on the deployment of embedded systems. There are examples of their use in nearly all industrial areas. The complexity of embedded systems is steadily increasing since they are more and more frequently developed to control very complex tasks, such as vehicle dynamics or engine electronics applications. Applications of embedded systems are often safety relevant. Therefore, the occurrence of errors in embedded systems can involve high risks and could endanger human life. In the case of non-safety relevant systems, errors could still result in enormous costs, e.g. in the automotive industry: Since updates for error correction are usually performed by replacing the entire embedded system the cars have to be called back to the service stations. For products produced in large quantities high costs arise. Accordingly, the development of embedded systems must comply with the highest quality standards.

In practice, the most important analytical quality assurance method is dynamic testing, namely the execution of the system under test with a set of carefully selected test data. The most significant weakness of the test is that the postulated functioning of the system under test can only be verified for those input situations selected as test data. Testing can show the existence but not the non-existence of errors [1]. A proof of correctness can only be produced by a complete test with all possible input values, input value sequences, and input value combination under all practically possible constraints. Usually a complete test is not feasible because of the huge amount of possible input situations and environmental conditions. Accordingly, an essential part of testing is the selection of the most error-sensitive test data during test case design.

Test case design can be automated for a set of test goals with the help of evolutionary algorithms. The precondition for such an evolutionary test is that the test goal can be expressed numerically and transformed into an optimization problem. Up to now no works have been published, in which functional tests have been automated with the help of evolutionary tests. This is, on the one hand, remarkable, since functional tests represent the most important test procedure - they are used to check the correct functioning of a system without analyzing the internal system structures (black box test) - on the other hand, it is also understandable, since it is very difficult to record the functional behavior of most systems and the effort required to transform the functional test into an optimization problem could be significantly greater than the test effort saved by automating the test.

In this paper, functional tests will be automated using evolutionary computation. This will be supported by the application field selected, which is becoming increasing important in vehicle development: control systems which introduce novel comfort and safety functions on the basis of measuring the distance of the vehicle to other objects. Examples of such applications are: Distronic (an adaptive cruise control), with which the vehicle keeps at a constant distance from a vehicle ahead; automatic vehicle parking, which shall be examined in more detail in this paper; or emergency brake systems, which are intended to prevent a vehicle form running into the tail end of a traffic jam. These are all very complex systems with a multitude of components, whose function is, however, narrowly focussed, making preconditions favorable for transforming the test of the systems and their components into an optimization problem.

In order to test the automatic parking system, we implemented an evolutionary functional test and integrated it into the simulation environment for the system. It was already possible to demonstrate the effectiveness of the evolutionary functional test by means of a set of initial experiments. Several input situations which lead to an error in the functional behavior of the system were found fully automatically.

This paper is arranged as follows: After a short introduction to evolutionary tests in the second chapter, the third chapter provides an overview of work already published on the evolutionary test. In the fourth chapter, the automatic parking system will be introduced. In chapter five, the idea of the evolutionary functional test for the automatic parking system is developed. Various possibilities for forming the fitness function are explained. The technical details of the implemented test environment are explained in chapter six. In the seventh chapter first experiments and results obtained are described. The paper concludes with a short summary and an outlook for future work.

2. EVOLUTIONARY TESTING

Testing is aimed at finding errors in the system under test and giving confidence in its correct behavior by executing the system with selected input situations. A systematic test is divided into the core activities of test case design, test execution, monitoring and test evaluation as well as the activities of test planning, test organization and test documentation, which prepare for the test and accompany it (compare Fig 1).

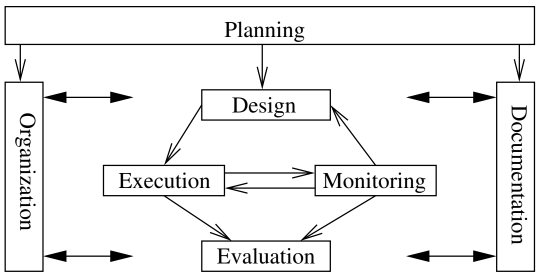

Fig 1: Evolutionary Test Case Design Embedded in the Testing Process

Of all the test activities, test case design is assigned decisive importance. Test case design determines the type and scope and thus the quality of the test. If test cases relevant to the practical application of the system are omitted or forgotten, the probability of detecting errors which may exist within the system sinks. Due to the central importance of test case design for testing, a number of testing methods have been developed over the last decades designed to help the tester with the selection of appropriate test data. One important weakness of the testing methods currently available is that they are not easily automatable. Manual test case design, however, is time-intensive and error-prone. The test quality is dependent on the performance of the single tester. In order to increase the effectiveness and efficiency of the test and thus to reduce the overall development and maintenance costs for embedded systems, we require a test that is systematic and extensively automatable. Both objectives are addressed by the evolutionary test.

Evolutionary testing is characterized by the use of meta-heuristic search techniques for test case generation. The test aim considered is transformed into an optimization problem. The test object's input domain forms the search space in which test data that fulfils the respective test aim is searched for. Due to the non-linearity of software (if-statements, loops etc.) the conversion of test problems into optimization tasks results in complex, discontinuous, and non-linear search spaces. Therefore, meta-heuristic search methods are employed, e.g. evolutionary algorithms, simulated annealing or taboo search.

In order to transform a test aim into an optimization task a numeric representation of the test aim is necessary, from which a suitable fitness function for the evaluation of the generated test data can be derived. Depending on which test aim is pursued, different fitness functions emerge for test data evaluation. If, for example, the temporal behavior of an application is being tested, the fitness evaluation of the individuals is based on the execution times measured for the test data. For safety tests, the fitness values are derived from pre- and post-conditions of modules, and for robustness tests of fault-tolerance mechanisms, the number of controlled errors can form the starting point for the fitness evaluation. Applications of evolutionary testing to structural testing result in different fitness functions again.

If an appropriate fitness function can be defined, then the evolutionary test proceeds as follows. The initial population is usually generated at random. If the test data has been obtained by a previous systematic test, this could also be seeded into the initial population [8]. The evolutionary test could thus benefit from the tester's knowledge of the system under test. Each individual within the population represents a test datum with which the system under test is executed. For each test datum the execution is monitored and the fitness value is determined for the corresponding individual with respect to the defined test aim. Next, population members are selected with regard to their fitness and subjected to combination and mutation processes to generate new offspring. Offspring individuals are then also evaluated by executing the system under test with the corresponding test data. A new population is formed by combining offspring and parent individuals, according to the survival procedures laid down. From here on, the process repeats itself, starting with selection, until the test objective is fulfilled or another given stopping condition is reached.

3. RELATED WORK

A number of papers have been published in the last years which have successfully applied evolutionary algorithms for test data generation. They have pursued various test goals and different test methods.

Work on the automation of test data generation for structure tests is most widespread. The aim of the work is to determine a quantity of test data for various structure test criteria, such as the statement test (the execution of all a program's statements), the branch test (the execution of all program branches) or the condition test (each of a program's conditions is at least evaluated as true once and as false once), which achieve the highest possible coverage of the respective internal program structures considered. Structure tests are based on the assumption that a system can only be tested thoroughly if all its parts are executed at least once during the test. The main weakness of structure tests is that they are unable to detect a lack of functions specified for the system due to their one-sided orientation to the program code. For structural testing, the fitness functions are typically based on the computation of a distance for each individual indicating how far it is away from executing the program in the desired way required to reach a high program code coverage [4], [6], [7], [9], [10].

Further work deploys evolutionary algorithms for testing non-functional properties such as safety constraints [4] or timing constraints [2], [11], [12], [13]. Here, the aim is to check non-functional properties of the system under test using evolutionary algorithms, by searching for test data for which the system violates the specified safety or timing constraints. The fitness functions are based on the calculation of a distance to the violation of the safety conditions or, for the temporal behavior test, on the measurement of the execution times for the test data generated.

Further work describes the deployment of evolutionary algorithms for robustness testing [5] and mutation testing [4],[14]. The application of evolutionary tests to functional testing is not widespread. Singular work by researchers such as [15] applies evolutionary algorithms to generate test data for formally specified test cases.

In the following, we present an application area for evolutionary functional testing, for which a complete automation of the functional test can be achieved without using formal specification techniques.

4. AUTOMATIC PARKING SYSTEM

As leading automobile supplier, DaimlerChrysler is constantly developing new systems in order to improve vehicle safety, quality, and comfort. Within this context, prototypical vehicle systems are developed, which support automatic vehicle parking - a function that might be introduced to the market in some years time.

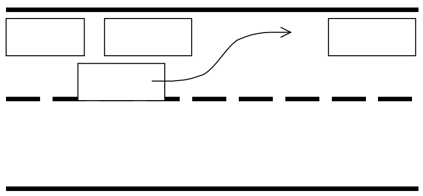

Fig. 2: Functionality of the Automatic Parking System

The automatic parking systems examined in this paper is intended to automate parking lengthways into a parking space (Fig. 2). To this end, the vehicle is equipped with environment sensors, which register objects surrounding the vehicle. On passing, the system can recognize sufficiently large parking spaces and signals to the driver that a parking space has been found. If the driver decides to park in the parking space the vehicle does this automatically.

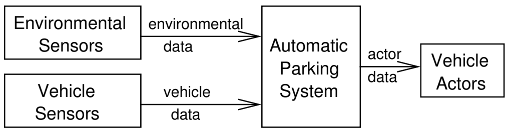

Fig. 3: System Environment for the Automatic Parking System

The system environment for the automatic parking system is illustrated in Fig. 3. System inputs are sensor signals, which receive information on the state of the vehicle, e.g. vehicle speed or steering position, and information from the environmental sensors, which register objects on the left and right hand side of the vehicle. For output the system possesses an interface to the vehicle actors, where the vehicle's velocity and steering angle can be preset. The internal structure of the automatic parking controller (Fig. 4). The parking space detection processes the data from the environmental sensor systems and delivers the recognized geometry of a parking space if it has been recognized as being sufficiently large. The parking controller component uses the geometry data on the parking space together with the data from the vehicle sensors to steer the vehicle through the parking procedure. For this purpose, velocity and steering angle are preset for the vehicle actors.

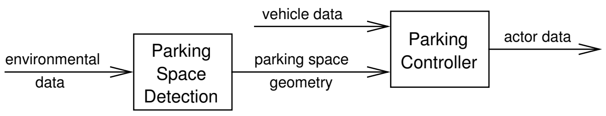

Fig. 4: Sub-Components of the Automatic Parking System

5. EVOLUTIONARY FUNCTIONAL TEST OF THE AUTOMATIC PARKING SYSTEM

The automated parking system is a complex application. One reason for its complexity is the parking space detection, which has to clearly distinguish between drivable area and collision area. Another source of complexity is the navigation of the vehicle itself into the parking space. This has to be managed by controlling speed and steering angle and without touching the collision area. Entering the collision area means with a high probability to cause damage at adjacent parking vehicles or objects. So, a fundamental requirement of the automated parking application is, that the vehicle must not damage or collide with other objects or vehicles during the parking procedure. A system fault might cause considerable material and commercial damage. For that reason exhaustive and efficient testing is essential before the release of such a system, which means to perform as much as possible test cases in a systematic way.

Manual testing of the complete system is costly and time consuming, because every test case comprises building up a park scenario with real cars and manual driving of each maneuver. Furthermore, performing a test in this way is difficult to reproduce, because the details of the test execution vary. In contrast, automated tests can perform a great number of test cases with less effort. Therefore automated functional tests performed in a controlled simulation environment in addition to manual tests could form an important quality assurance measure.

Evolutionary functional testing provides a way to automate functional tests as a complete process. Instead of selecting the test cases manually, a search for interesting test cases is performed automatically. This is done by translating the test case selection into an optimization problem. This requires the solution of two problems. First, how to generate the test data and second how to evaluate the test results.

For the test data generation the possible input situations of the system under test are mapped to the search space. On one hand the mapping should keep the size of the search space as small as possible, on the other hand the mapping should be able to produce all possible input data for the system. If one considers the whole input range during design of the test data generator does not mean that all test cases in this range are actually tested, but it provides the possibility for the search to come into this range. An appropriate model has to be designed for this purpose.

The evaluation of the test cases is carried out by the fitness function. In the automatic parking system, the fitness function calculates a numerical fitness value for the parking maneuver driven by the automatic parking system for the parking scenario generated. This fitness value represents the quality of the corresponding test case and intends to lead the evolutionary search into a direction of faulty input situations. The aim of the test is to find system faults and for that reason the fitness function is designed to assign good fitness values to parking scenarios which lead the system to enter the collision area or end up in an inadequate parking situation. Bad fitness values are assigned to scenarios which reach a good parking position with enough clearance to the collision area. Different strategies for the implementation of a fitness function are possible, e.g. measuring the minimum distance between the vehicles' surface and the borders of the collision area during the parking maneuver. Entering the collision area would lead to a negative distance. Alternatively, the time to contact can be calculated - taking into account not only the distance to a collision but also the speed of the vehicle. The shortest time to contact measured for the parking maneuver could form the fitness value of the parking scenario generated.

6. TEST ENVIRONMENT

The test environment of the automatic parking system comprises the simulation environment, an evolutionary computation toolbox, the calculation of the fitness values and the test data generator which translates individuals into concrete parking scenarios (Fig. 5). The test object is the control unit of the vehicle with the implementation of the automated parking system inside.

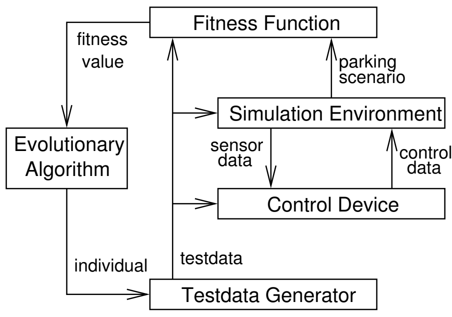

Fig. 5: Design of the Test Environment

The simulation environment (built up on a Matlab R12.1 platform) simulates the properties of the vehicle and the surrounding environment. It runs with the control unit “in-the-loop” meaning that the simulation environment calculates the sensor data of the vehicle and presents it to the control unit. The control unit processes this sensor data and reacts on it with control data for the simulation environment. This loop simulates a complete parking scenario. The parameters necessary for a simulation of a parking scenario, such as positions of close-by cars forming the paring space or the driving maneuver when passing the parking space are outputs of the test data generator. After the simulation of a parking maneuver the fitness value is calculated for the parking scenario, by simulation the calculated parking maneuver. The fitness value determined is assigned to the generated individual.

In this way for every individual of the generation a simulation of its corresponding parking maneuver is performed determining the fitness value of the individual. When all individuals of a generation have a fitness-value assigned, the evolutionary algorithm makes another iteration and calculates the next generation of individuals.

7. EXPERIMENT

In an initial experiment the parking control unit of the automatic parking system was tested (Fig. 6). The parking control unit calculates the necessary speed and steering angle for the parking maneuver from the presented vehicle data and the geometry data of the parking space. The test bypasses the parking space detection unit and calls the parking control unit with the generated parking space geometry directly.

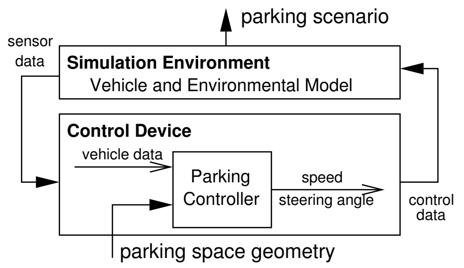

Fig. 6: Test Object Parking Controller

Generation of Test-Data

The geometry data are generated by the evolutionary algorithms and the test data generator, as shown in Fig. 7. When the geometry data is provided, the simulation of the parking maneuver begins. The vehicle data presented to and the speed and steering angle calculated by the parking control unit are inside the simulation loop. The generation of the geometry data is done in the experiment with a simplified parking space model. The borders of the parking space are always rectangular. The shape of the parking space can only vary in length and depth.

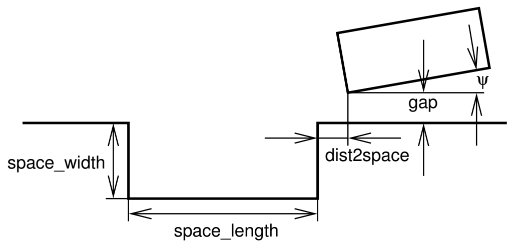

Fig 7: Test Data Generation for the Test of the Parking Control Unit

This simplified parking space model needs the values of five independent variables to calculate a parking space geometry. With the values of space_length and space_width the length and width of the parking space are defined. The value of dist2space defines the distance between vehicle and parking space at the beginning of the parking maneuver. The value gap defines the distance between the vehicle and the collision area. The angle ψ defines the yaw angle of the vehicle to the parking space at the starting position.

Fitness-Function

For fitness evaluation a simulation for the generated parking scenario is performed. The result of the simulation is a complete parking maneuver calculated by the automatic parking controller for the parking scenario represented by the individual. The parking maneuver is provided to the fitness function for its evaluation. The implementation of the fitness function in the experiment calculates the minimal distance between the collision area and the vehicle in a parking maneuver (Fig. 8). The distance is calculated for every single simulation step. The minimal value during all simulation steps represents the fitness value of the whole scenario.

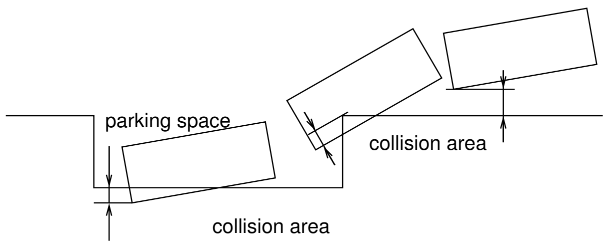

Fig 8: Fitness-Calculation of a Parking Scenario with the Distance Criteria

Parameters and Boundaries of the Evolutionary Test

The tbxmpga function of the GEA toolbox for Matlab [16] was used as implementation for the evolutionary algorithm in the experiment. This function implements a multi population genetic algorithm. The search was done with a local selection model, the fraction of the population to be reproduced every generation was set to less offspring than individuals in population, identical to elitest selection. A breeder genetic algorithm was used for mutation, discrete recombination was applied.

The experiment was performed in 20 generations with 44 individuals per generation divided to two sub-populations. This means a total of 880 parking scenarios were simulated. The test data generation model used, maps 6 independent variables to the input domain of the system. Each of the 6 variables was limited to a defined range for the experiment. The range for each variable was chosen by considering what is reasonable in the context of the application. The definition of a range for each variable limits the search space and prevents the search from ending with solutions which have a good fitness value but are not sensible for the application. The range of the length of the parking space was defined to be between 7 and 10 meter, the width was restricted to the range between 2.5 and 3.5 meter. The distance between vehicle and parking space was chosen to be between 10 cm and 3 m. The distance to the collision area on the right vehicle side was in the range of 10 cm and 1 m. The yaw angle was within –4° and +4°.

Results

During the simulations of the evolutionary search, various kinds of parking scenarios appeared. All these different parking scenarios were associated to one of three generalized categories. The correct parking scenarios are in the first category; they comply with the specified behavior. Either the vehicle arrives at a correct stop position (without touching the collision area) or the parking controller rejects the situation and refuses to park in. Another category are the critical parking maneuvers, their path nearly touches the collision area or the parking controller rejects the situation or discontinues the maneuver without obvious reason. The third category comprises scenarios, in which the vehicle enters the collision area. Either the vehicle reaches the collision area with its edge or the end-position is parallel in the collision area. Fig. 9 shows examples of each category.

During the evolutionary functional test more than 25 parking scenarios were detected for which the calculated parking maneuver resulted in entering the collision area. The scenarios revealed two important insights. First, the parking strategy of the parking control unit seems to have problems in situations when the vehicles starting position is near to the collision area on the right side and the distance from the parking space is large. An example of such a situation is shown in Fig. 10. Second, the parking maneuvers which entered the collision area also lead to detection of a fault in the simulation environment. The investigation revealed that the simulation environments calculations of the car positions were too imprecise for the parking application. This was not remarkable during development, but caused some of the parking scenarios which entered the collision area. The experiment has shown, independent from the source of the error, that the evolutionary functional testing method found test cases fully automatically, which lead the system to erroneous behavior.

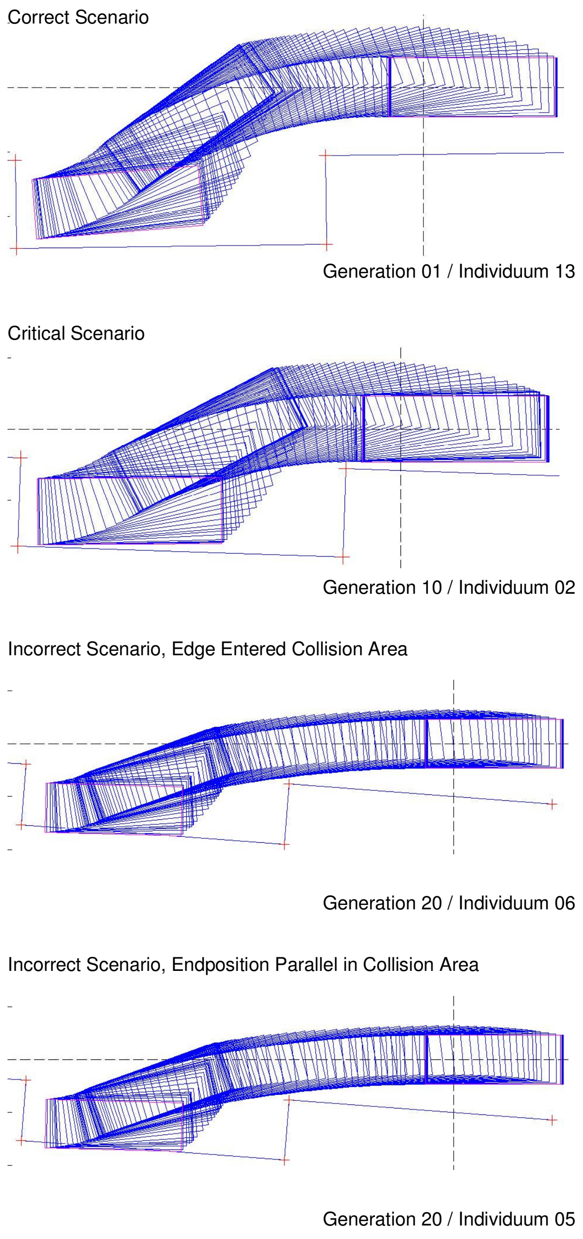

Fig 9: Different Parking Maneuvers Resulting from Different Test Data Sets Generated

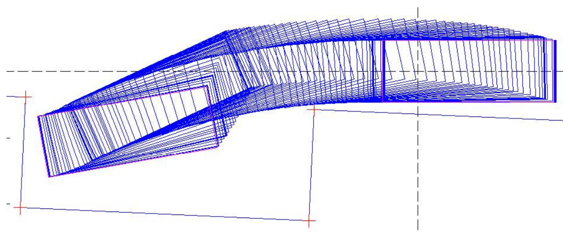

Fig 10: Inadequate Parking Situation

8. CONCLUSION AND FUTURE WORK

The results of the deployment of evolutionary functional tests are promising. Initial experiments were already able to find several parking scenarios automatically, for which an error in the system could be identified, allowing further improvement of the parking strategy. The aim of this work is to provide a comprehensive test environment for the entire system and its subsystems for the time when the automatic parking system goes into production-vehicle development, so that the series product can be tested fully automatically, thoroughly and in a target-oriented way with a multitude of very different generated test scenarios.

Furthermore, we intend to expand the application of evolutionary functional tests to further vehicle systems such as intelligent speed control or emergency brake systems. We also intend to research the interaction between evolutionary functional tests and structure tests more intensively. This should answer questions such as: Which coverage is achieved with functional tests? Does the seeding of functionally determined test data prove useful for an evolutionary structure test and, on the other hand, does the seeding of structure-oriented test data increase the test quality of the evolutionary functional test?

9. REFERENCES

[1] Dijkstra, E. (1972): Notes on Structured Programming. In: O.-J. Dahl, E. Dijkstra and C. Hoare. Structured Programming. Academic Press, pp. 114 - 137

[2] Wegener, J., and Grochtmann, M. (1998): Verifying Timing Constraints of Real-Time Systems by means of Evolutionary Testing. Real-Time, Systems, vol. 15, no. 3, Kluwer Academic Publishers, pp. 275 - 298

[3] Sthamer, H.-H. (1996): The Automatic Generation of Software Test Data Using Genetic Algorithms. PhD Thesis, University of Glamorgan, Pontyprid, Wales, Great Britain

[4] Tracey, N., Clark, J., Mander, K., and McDermid, J. (1998): An Automated Framework for Structural Test-Data Generation. Proceedings of the 13th IEEE Conference on Automated Software Engineering, Hawaii, USA

[5] Schultz, A. C., and Grefenstette, J. J. and De Jong, K. A. (1993): Test and Evaluation by Genetic Algorithms. IEEE Expert 8(5), pp. 9 – 14

[6] Pargas, R., Harrold, M., Peck, R. (1999): Test-Data Generation Using Genetic Algorithms. Software Testing, Verification & Reliability, vol. 9, no. 4 pp. 263 - 282

[7] Michael, C., McGraw, G., Schatz, M. (2001): Generating Software Test Data by Evolution. IEEE Transactions on Software Engineering, vol. 27, no.12 pp. 1085 - 1100

[8] Wegener, J., Grimm, K., Grochtmann, M., Sthamer, H., and Jones, B. (1996): Systematic Testing of Real-Time Systems. Proceedings of the Fourth European International Conference on Software Testing, Analysis & Review, Amsterdam, Netherlands

[9] Jones, B., Sthamer, H., and Eyres, D. (1996): Automatic Structural Testing Using Genetic Algorithms. Software Engineering Journal, vol. 11, no. 5, pp. 299 - 306

[10] Wegener, J., Baresel, A., and Sthamer, H. (2001): Evolutionary Test Environment for Automatic Structural Testing. Information and Software Technology, Special Issue devoted to the Application of Metaheuristic Algorithms to Problems in Software Engineering, vol. 43, pp. 841 - 854

[11] Puschner, P., and Nossal, R. (1998): Testing the Results of Static Worst-Case Execution-Time Analysis. Proc of the 19th Real-Time Systems Symposium, pp. 134 - 143

[12] Wegener, J., and Mueller, F. (2001): A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints. Real-Time Systems, Vol. 21, No. 3, Kluwer Academic Publishers, pp. 241 - 268

[13] Gross, H.-G., Jones, B., and Eyres, D. (2000): Structural performance measure of Evolutionary Testing applied to worst-case timing of real-time systems. IEE Proc.-Softw., vol. 147, no. 2, pp. 25 - 30

[14] Bottaci, L. (2001): A Genetic Algorithm Fitness Function for Mutation Testing. Proceedings of the SEMINALL-Workshop at the 23rd International Conference on Software Engineering, Toronto, Canada

[15] Jones, B., Sthamer, H.-H., Yang, X., Eyres, D. (1995): The Automatic Generation of Software Test Data Sets using Adaptive Search Techniques. Proceedings of the 3rd International Conference on Software Quality Management (SQM'95), Seville, Spain, pp. 435 - 444

[17] Pohlheim, H.: Genetic and Evolutionary Algorithm Toolbox for Use with Matlab - Documentation. http://www.geatbx.com/