Test sequencing in a complex manufacturing system

Citation for published version (APA):
Boumen, R., Jong, de, I. S. M., Vermunt, J. W. H., Mortel - Fronczak, van de, J. M., & Rooda, J. E. (2005). Test sequencing in a complex manufacturing system. *XOOTIC Magazine*, *11*(2), 9-16.

Document status and date:
Published: 01/01/2005

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

Test sequencing in a complex manufacturing system[1]

R. Boumen, I.S.M. de Jong, J.W.H. Vermunt, J.M. van de Mortel-Fronczak and J.E. Rooda

Testing complex manufacturing systems, like the ASML TWINSCAN [2] lithographic machine, takes a lot of time and costs. Within the Tangram project, methods are investigated to reduce this test costs. In this article, we describe a method which is used to optimize a test sequence such that it takes the least amount of costs, or time. With several cases we demonstrate that this method can be used to optimize test sequences within the manufacturing of a TWINSCAN lithographic machine such that cycle time is reduced.

Introduction

In today's industry, time to market is extremely important. In their drive to reduce systems time-to-market, many companies develop their systems concurrently. The final phase within concurrent development of systems is integration and test. This phase is on the critical path, and therefore has great influence on time-to-market (see [3]). The goal of the Tangram project is to reduce the time and cost spent on testing and integrating, and by that reduce time-to-market and cycle time of a system. Within the Tangram project, we look at test and integration strategy. A test and integration strategy defines a test and integration phase which is optimal in terms of time, costs and/or quality. In our work we are looking at methods that select or optimize test and integration strategies, taking into account time, costs and quality.

In this article, we describe a method to create optimal or near-optimal test sequences. A test sequence is a key element of the test and integration strategy. The basis of this method is described as Sequential Diagnosis by Pattipati [4], who used this method for the diagnosis of electronics. This method can also be used for test sequencing problems related to the manufacturing of complex systems.

System test problems are multidisciplinair (e.g. electronics, software and mechanics), large (hundreds of tests) and take a long time (up to several weeks/months). A test and integration strategy for systems is traditionally created by experts which have a good knowledge of the systems architecture, the risks and the test costs. Test sequencing and selection is traditionally a risk-based decision. That is, the elements with the highest risk are tested, until time is up and the system is shipped. At that moment, the quality of the system is often unknown.

The semiconductor industry is a typical example of a time-to-market driven industry. For companies such as ASML, shipping your system before competition is wanted, and thus dominates the test and integration phase. Several cases within the manufacturing process of a TWINSCAN machine are presented in this article.

The structure of the article is as follows: first an example test problem is introduced, then the test problem is formally described, then different solving algorithms are mentioned, then the results of the different cases are shown, and finally conclusions and future work are mentioned.

[1]This work has been carried out as part of the TANGRAM project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under grant TSIT2026.

Example test problem

To illustrate a system test problem, a telephone is taken as system under test. This telephone consists of three modules: the device, the receiver and the cable connecting the receiver and the device. The system is shown in Figure 1. There are two interfaces between the modules: one between the device and the cable and one between the cable and the receiver.

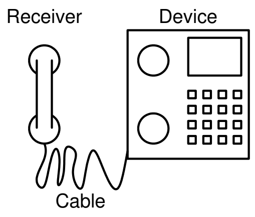


Figure 1: Telephone example

In this system under test, we can identify 5 possible faults:

1. The device is broken.
2. The cable is broken.
3. The receiver is broken.
4. The cable cannot be connected to the device.
5. The cable cannot be connected to the receiver.

The first three faults are logical, the last two may be less obvious. These two faults are interface faults, which are typical system faults that occur through concurrent engineering. All modules have been developed in parallel using interface specifications. If these specifications are ambiguous, the assembled system may not work as the specifications are interpreted differently for each module, which results in interface faults. Each fault has a certain probability that it exists. It is assumed that this fault probability is 10% for each fault.

The goal of testing the system is to find out which of the possible faults exists. 6 tests are available to test this system:

0. Test the complete telephone
1. Test the device
2. Test the cable
3. Test the receiver
4. Test the device and cable
5. Test the cable and receiver

The costs of each test are defined in uniform *cost units*. In real life, these costs can for example be defined in money or in time. Test 0 costs 3, while tests 1,2 and 3 each cost 1 and test 4 and 5 cost 2.

The objective is to create a test sequence with minimal expected test costs. This optimal sequence logically depends on the outcomes of tests applied, as illustrated in Figure 2. According to this test sequence, a tester starts with test 0. If this test passes, the tester knows no fault exists in the system and the system works. If this test fails, the tester knows that at least one fault exists and the tester has to perform more tests to identify this fault. This way of working results in a test tree, which contains several test sequences depending on the outcomes of tests. The objective of calculating the optimal test sequence actually means calculating the optimal test tree with minimal expected test costs, identifying each possible fault.

The test costs of a test tree can be calculated as described in the sequel for the example test tree of Figure 2.To start with, test 0 is performed. This test fails with a certain probability and if so test 3 is performed next. This probability depends on the covered faults and their probabilities. The expected test costs are therefore the test costs of test 0 plus the test costs of test 3 multiplied by the chance that test 0 fails, and so on. An optimal solution is a tree with the least expected test costs. An optimal solution for the telephone example is shown later in this article. We continue in the next section with a formal description of the test problem.

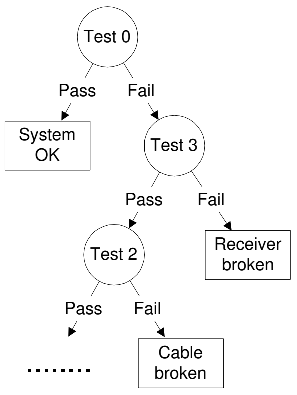


Figure 2: Test tree with multiple test sequences

Test problem formulation

Formally, a test problem $\mathcal{D}$ can be defined as a five-tuple: $\mathcal{D} = (\mathcal{T}, \mathcal{S}, \mathcal{T}_c, \mathcal{S}_p, \mathcal{R}_{ts})$, where:

- $\mathcal{T}$ is a finite set of *k* elements, called tests.
- $\mathcal{S}$ is a finite set of *l* elements, called fault states.
- $\mathcal{T}_c : \mathcal{T} \rightarrow \mathbb{R}$ gives for each test in $\mathcal{T}$ the associated costs of performing that test
- $\mathcal{S}_p : \mathcal{S} \rightarrow \mathbb{R}$ gives for each fault state in $\mathcal{S}$ the *a priori* probability that the fault state is present.
- $\mathcal{R}_{ts} : \mathcal{T} \rightarrow \mathcal{P}(\mathcal{S})$ gives the subset of fault states that are covered by a test.

The *a priori* probability is the absolute probability that a certain fault is present. The test problem can also be represented as a matrix A of dimensions $l \times k$, where $A_{ij} = 1$ if test t_j covers fault state s_i, otherwise $A_{ij} = 0$. The formal description is a model of the test problem and is therefore called the *system test model*. In Table 1, the system test model of the telephone example is shown, represented as a matrix.

Table 1: Telephone example system test model

$\mathcal{S}$ / $\mathcal{T}$	t_0	t_1	t_2	t_3	t_4	t_5	$\mathcal{S}_p$
s_1	1	1	0	0	1	0	10 %
s_2	1	0	1	0	1	1	10 %
s_3	1	0	0	1	0	1	10 %
s_4	1	0	0	0	1	0	10 %
s_5	1	0	0	0	0	1	10 %
$\mathcal{T}_c$	3	1	1	1	2	2	

In the following sections, different algorithms are discussed to solve the test problem and hence calculate the optimal test tree with minimal expected test costs.

Solving algorithms

Continuing on the work of Pattipati, many different solving algorithms using different heuristics have been developed. A good overview is given by Shakeri *et al* in [1]. The assumptions of the test problem solving algorithms are:

- binary outcome tests (only pass or fail),
- the fault states are independent of each other,
- the tests are 100% reliable,
- the tests are 100% sensitive and specific,
- a repair action 100% fixes the fault state.

The test problem solving algorithms consists of two types: single and multiple-fault algorithms. The single-fault algorithms have the assumption that at most one fault state is present. The multiple-fault algorithms do not have that assumption. Both types of algorithms are explained in the sequel.

Single-fault algorithms

The single-fault algorithm has the assumption that at most one fault state exists. This assumption results in some changes to the original test problem. The possibility that no fault state exists (the system is OK) must be modelled explicitly because the algorithm assumes that at least one fault is present. This is done by adding an extra state to $\mathcal{S}$, named s_0 which represents the system OK state. Element $\mathcal{S}$ of the basic test problem is denoted by $\underline{\mathcal{S}}$ for the single-fault problem. Also, because at most one fault state can be present, the sum of the fault state probabilities must be 100%. Therefore, the *a priori* fault probabilities $\mathcal{S}_p$ are converted to *conditional* fault probabilities $\underline{\mathcal{S}}_p$ using,

$$\underline{\mathcal{S}}_p(s_0) = \frac{1}{1 + \sum\limits_{s \in \mathcal{S}} \frac{\mathcal{S}_p(s)}{1 - \mathcal{S}_p(s)}} \quad (1)$$

and

$$\underline{\mathcal{S}}_p(s_i) = \frac{\frac{\mathcal{S}_p(s_i)}{1 - \mathcal{S}_p(s_i)}}{1 + \sum\limits_{s \in \mathcal{S}} \frac{\mathcal{S}_p(s)}{1 - \mathcal{S}_p(s)}} \text{for } i = 1, \cdots, l. \quad (2)$$

The single-fault system model of the telephone example is shown in Table 2.

Table 2: Telephone example single-fault system test model

$\underline{\mathcal{S}}$ / $\mathcal{T}$	t_0	t_1	t_2	t_3	t_4	t_5	$\mathcal{S}_p$	$\underline{\mathcal{S}}_p$
s_0	0	0	0	0	0	0	-	64.28%
s_1	1	1	0	0	1	0	10%	7.14%
s_2	1	0	1	0	1	1	10%	7.14%
s_3	1	0	0	1	0	1	10%	7.14%
s_4	1	0	0	0	1	0	10%	7.14%
s_5	1	0	0	0	0	1	10%	7.14%
$\mathcal{T}_c$	3	1	1	1	2	2	-	100%

A solution to the single-fault test problem is an AND/OR decision tree as shown in Figure 3. This tree consists of three types of nodes: AND, OR and leaf nodes. The OR nodes represent the suspected set of fault states, the AND nodes represent tests applied to the OR nodes and the leaf nodes represent isolated faults states.

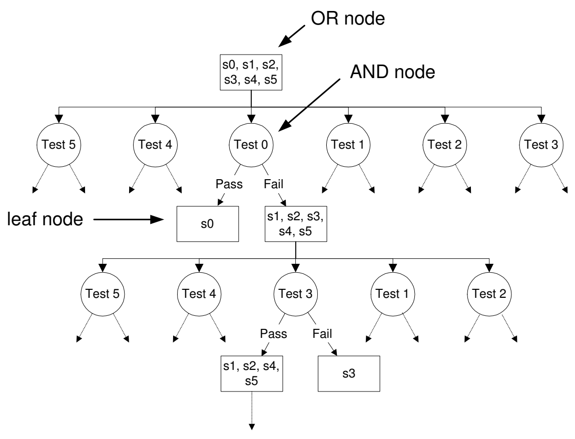


Figure 3: An AND/OR graph

Calculating an optimal AND/OR tree is NP-Hard [5]. Therefore in literature, two types of solving algorithms are described: optimal algorithms for small and near-optimal algorithms for large test problems.

To calculate an optimal AND/OR tree, two optimal algorithms can be used: Dynamic Programming and AND/OR graph search [5]. The Dynamic Programming technique is a recursive algorithm that constructs an optimal tree from the leave nodes up by identifying larger subtrees until the optimal tree is generated. The Dynamic Programming technique has storage and computational complexity of $\mathcal{O}(k^3)$ for the basic test problem. Therefore in this article, we use the more efficient top-down algorithm based on AND/OR graph search (AO^*).

The AO^* algorithm constructs an AND/OR graph as a directed graph with a root (or initial) node and a nonempty set of terminal leaf nodes. The initial node represents the given problem to be solved, while the terminal leaf nodes correspond to the subproblems with known solutions. An OR node is solved if any one of its successor nodes is solved, and an AND node is solved only when all of its immediately successors are solved. During the search within the AND/OR graph, the expected test costs of visited OR nodes are saved to reduce computational effort: these costs do not have to be calculated again.

For larger problems near-optimal algorithms are necessary. Several near-optimal search algorithms are known from literature [5], for example: the AO^*_ϵ algorithm, the limited search AO^*, and the AO^* algorithm combined with a multi-step information gain heuristics. The near-optimal one-step information gain heuristics can be used during the AO^* algorithm to solve the larger cases presented in this article.

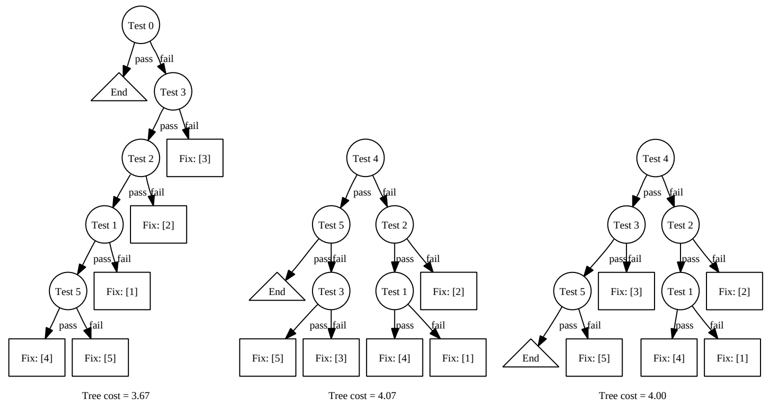
The test tree shown in Figure 4(b) is an optimal tree for the telephone example. The expected test cost of this tree are 4.07. This means that on average, 4.07 test cost are necessary to identify one fault state in the system.

To illustrate the different test sequences that can be found for different fault probabilities, we reduce the *a priori* fault chance of each fault state from 10% to 5%. The resulting tree can be seen in Figure 4(a). In the third situation, the *a priori* fault chance of each fault state is 50%. The resulting tree can be seen in Figure 4(c). In the 5% situation, only test 0 is necessary to check whether the system is OK, in the 10% situation both tests 4 and 5 are necessary to check whether the system is OK, while in the 50% situation tests 4, 3 and 5 are necessary to check whether the system is ok.

Multiple-fault algorithms

When fault probabilities are high, the assumption that at most one fault state is present in the system is questionable. In these cases, it is still possible to use the solution tree of the single-fault algorithm, over and over again until all fault states have been identified, but it is certainly not optimal. Therefore multiple-fault algorithms are necessary.

Multiple-fault algorithms construct AND/OR graphs in the same way as the single-fault algorithms. However, instead of considering one possible fault state, they consider all possible combinations of fault states. The OR node in an AND/OR graph represents all possible subsets of suspected fault states. Multiple-fault problems have a exponential complexity of $\mathcal{O}(2^l)$ (see [1]). The AO^* multiple-fault algorithm used in this article, is derived from the AO^* single-fault algorithm. Compared to the single-fault algorithm, the multiple-fault algorithm considers fix actions of fault states. If a fault state is isolated, it can be fixed immedi-



(a) 5% fault probability (b) 10% fault probability (c) 50% fault probability

Figure 4: Telephone example optimal single-fault test trees

ately. After the fix action, isolated fault states are removed and more tests are applied to find other faults. The algorithm terminates when all faults are excluded and the system is ok. The resulting graph has one root node and one leaf node. An example multiple fault tree is shown in Figure 5.

Besides test and fix actions, the algorithm also has diagnosis actions. If a number of fault states is under suspicion, but none have been isolated and additional testing does not give more information, a diagnosis action removes the suspected fault states. This diagnosis action has high costs, but is necessary to terminate the algorithm and solve the test problem.

To reduce computational complexity, the same information gain heuristic is implemented as in the single-fault algorithm. Most computational costs are spent during the calculation of the pass and fail probabilities of a test, as all subsets of fault states must be taken into account. Therefore, estimators are used to estimate the pass and fail probabilities and reduce this computational complexity. If a problem is still to large, it can be divided into subproblems that can be solved optimal or near-optimal. The subproblems by itself can then be sequenced with the same algorithm, or by hand. To reduce storage complexity of saved OR nodes, the implemented multiple-fault algorithm uses the compact set notation (see [1]). The compact set notation is a shorter notation for all possible subsets of fault states.

An optimal multiple-fault tree of the telephone example, shown in Figure 5, has been calculated with the optimal multiple-fault algorithm.

Tree simulation

Both single and multiple-fault algorithms can be used for system test problems. The advantage of a single-fault algorithm is that the resulting tree is smaller and better understandable. Also, the computational effort is less. However, the resulting test costs may be higher than in case of using solution from a multiple-fault algorithm. By using a simulation model of the test process, called the testFactory, the difference between the average test costs can be made clear. The testFactory is not discussed in this article. The testFactory simulates the testing of a number of predefined faulty systems either using a single-fault tree over and over again until all faults are found, or using the multiple-fault tree. In Figures 6(a) and 6(b) two histograms are shown of the simulation of the single and multiple-fault 10% fault probability trees. After 5000 simulation runs (number of systems tested), the average test costs of the single-fault tree were 5.7, while the average test costs of the multiple-fault tree were 5.3. If all tests would be performed, the test costs would be 10.

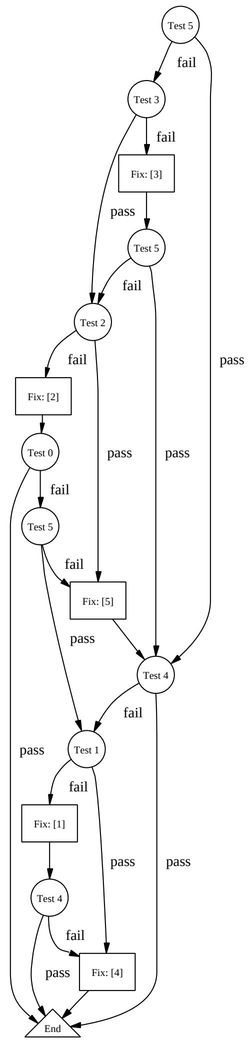


Figure 5: Telephone example optimal multiple-fault test tree

Cases

Within the manufacturing department at ASML, several test steps are performed during the production of a TWINSCAN lithographic machine. These test steps consist of performance, measurement and fault-detection tests, and calibrations. The presented test sequencing method is applied to three test steps, called job-steps, of different modules to reduce the cycle time of manufacturing a TWINSCAN machine.

The approach of the case is as follows:

1. Three models are created for 3 different job-steps.
2. For each model the optimal single and multiple-fault test trees are calculated.
3. The resulting test sequences are simulated using a test factory simulation model to show the expected test time.

In Table 3 the properties of the 3 created models are shown. The first column denotes the size of the matrices for job-steps A, B and C. The second column denotes the sum of all test costs, denoting the current situation. The cost of a test is for this case defined in time units. The third column shows average fault probability. The fourth column indicates the density of the A matrices, that is, how well 'filled' these matrices are.

Table 3: Case system test model properties

Case	$k \times l$	$\sum_{t \in \mathcal{T}} \mathcal{T}_c(t)$	Aver.($\mathcal{S}_p$)	Dens.(A)
A	15×15	815	71.3%	38.2%
B	33×60	33	46.0%	15.2%
C	39×73	730	15.8%	10.4%

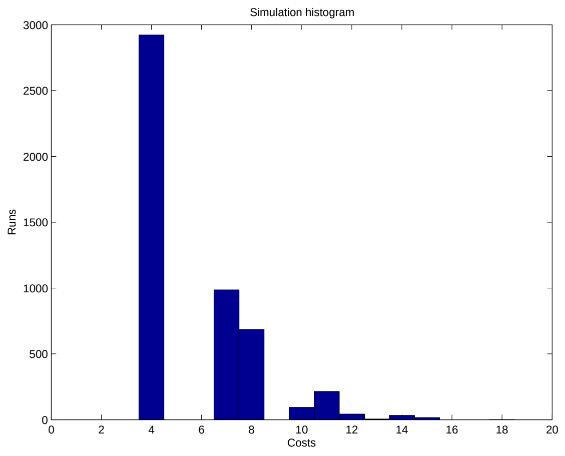
Now, the single and multiple-fault trees can be calculated. In Table 4, the properties of the trees and algorithms used are shown. The first single-fault column denotes which methods have been used to solve the single-fault problem: either the optimal calculation or using the information gain (IG) heuristic or by dividing (div) the problem in multiple problems. The second column shows the expected tree costs. The same columns are shown for the multiple-fault algorithm.

The costs of the single-fault trees are much lower then the multiple-fault trees. This due to the single-fault assumption and the conditional probabilities which are much lower in these cases then the *a priori* fault probabilities.

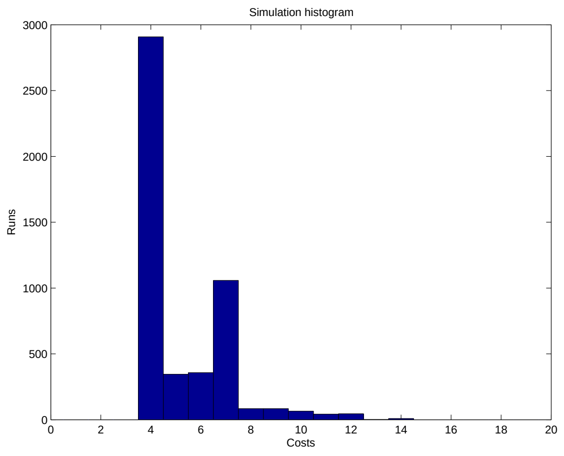
Table 4: Case test tree properties

	Single-fault		Multiple-fault	
Case	Method	Costs	Method	Costs
A	Optimal	202.9	IG	690.0
B	IG	5.0	div(4)	25.8
C	Optimal	144	div(4)	504

After the trees have been calculated, they are simulated in the simulation environment, as mentioned previously. In Table 5, the simulation results are shown. The first column shows the average single-fault tree costs and the second column shows the gain or loss in cycle time compared to the current situation. The third and fourth columns show the same for the multiple-fault test trees.



(a) Single-fault simulation (5.7 average)



(b) Multiple-fault simulation (5.3 average)

Figure 6: Telephone example tree simulation histograms

The average test costs of the single-fault tree are much higher then the test costs of the current situation. This results from the assumption that only one fault exists, while the average number of faults present is large (larger then 10). The single-fault tree is therefore only suitable when the average number of faults is small, in the range of 1 through 5. For a larger number of average faults, the multiple-fault algorithm performs better. The resulting test trees are even better then the test trees that are currently used.

Table 5: Case simulation results

	Single-fault		Multiple-fault	
Case	Sim.	Delta	Sim.	Delta
A	1848	+126%	689.0	−15.5%
B	60.8	+84.4%	25.9	−21.5%
C	1608	+120%	504.2	−30.9%

Conclusions

The presented method describes the test problem in a system test model. A single-fault algorithm calculates an optimal, with the least test costs, test tree, consisting of multiple test sequences, based on this system test model. This algorithm has the assumption that at most one fault exists. Besides this algorithm, a multiple-fault algorithm is described that creates a test tree with the assumption that multiple-faults can exist. This algorithm needs to take fixing and diagnosis of faults into account. The single-fault algorithm needs few computation to give an optimal solution, however it is recommended that this solution may only be used with test problems that have no more then 5 faults on average present in the system. The multiple-fault algorithm takes more computation effort, but the calculated solutions can also be used with problems that have more then 5 faults present.

We can conclude that the presented method is suitable for system test problems, as seen within ASML. There are two main benefits for using this method in the test and integration phase of systems. First, the test cost can be reduced by calculating the optimal test sequence as is shown in this case. Even test sequences that are judged to be quite good by experts, can be improved and cycle time can therefore be reduced.

Second, more insight in the test coverage of faults is gained when creating system test models. For large systems little knowledge exists about the relation between faults and tests: if a test fails it is difficult to indicate why. The presented system test model is a summary of these relations. Furthermore, the available test set can be made more explicit. New tests can be developed that cover faults which are not covered by the current test set. Also new tests can be developed that replace multiple other tests but cover the same or even more faults.

Future work

In the sequel of this project we will continue developing methods to optimize test and integration strategies. Test and integration sequences depend on each other. For example, the telephone consists of three modules. If the development of a certain module is delayed, tests using this module cannot be performed, while tests concerning the other two modules can be performed. Also, if the modules are separated, parallel testing would be possible, which probably reduces test time. In other words, the integration sequence of modules must be taken into account to determine the optimal test sequences. Or even further: the integration sequence must be optimized regarding time, costs and/or quality. Other aspects of the test and integration strategy relevant to our project are: scheduling tests over resources, strategy decisions regarding cost, time and quality and as already mentioned in this article, test and integration process simulations to determine the difference between certain test and integration strategies.

References

[1] M. Shakeri, V. Raghavan, K. R. Pattipati and A. Patterson-Hine, *Sequential Testing Algorithms for Multiple Fault Diagnosis* In *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, Volume 30, Number 1: 1-14, 2000

[2] ASML, `http://www.asml.com`

[3] M. Prins, *Testing Industrial Embedded Systems - An Overview* In *Proceedings of the 14th Annual International Symposium of INCOSE*, 2004

[4] K. R. Pattipati, S. Deb, R. W. Dontamsetty and A. Maitra, *START: System Testability Analysis and Research Tool* In *IEEE Aerosp. Electron. Syst. Mag.*: 13-20, 1991

[5] K. R. Pattipati and M. G. Alexandridis, *Application of heuristic search and information theory to sequential diagnosis* In *IEEE Trans. Syst. Man, Cybern.*, Volume 20: 872–887, 1990

Contact Information

Roel Boumen

Technische Universiteit Eindhoven
Department of Mechanical Engineering
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
r.boumen@tue.nl

Ivo de Jong

Technische Universiteit Eindhoven
Department of Mechanical Engineering
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
i.s.m.d.jong@tue.nl