Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2007-12
Test set improvement using a next-best-test-case algorithm

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

ISSN: 1872-1567

SE Report: Nr. 2007-12
Eindhoven, February 2008
SE Reports are available via http://se.wtb.tue.nl/sereports
**Abstract**

The development of a new semi-conductor manufacturing system, like the ASML wafer scanner, is mainly driven by time-to-market. The final test phases during the development phase of a wafer scanner can consist of many (100+) test cases. Families of wafer scanners are developed to spread out the development effort and maintain the time-to-market requirements. Test cases from previous system types in the product family are used as the basis for the definition of new test cases. Experts investigate the changes in the system and define which additional test cases are required. The quality of the test cases depends on the expert knowledge. One of the problems is that this knowledge is not easily transferred and used by new hires to develop new test cases. This paper presents an algorithm that is able to define which next test case is most optimal, given the risk in the system and the coverage of the available test cases. This algorithm uses a simple model of the test cases and the system under test and an information gain-based method to determine the next-best-test-case. Furthermore, a clustering technique is developed such that this method can be used in an industrial setting, where large sets of test cases are common. Two cases have been performed using this method to identify where new test cases could be beneficial.
1 Introduction

The complexity of systems like the ASML wafer scanner increases with each new system type. The number of components and the complexity of the individual components increases. This increase in complexity results in an increase of the number integration tasks and an increase of the number and duration of test-diagnose-fix tasks. New system types are developed using a platform strategy. The first new system type contains basic functionality and is delivered with initial performance levels. The performance levels are increased with each new system type developed in the platform. The components in the platform are gradually upgraded such that each new type meets this higher system performance. A set of test cases, a test set, for an existing component is the basis for the test set for the new version of the component. Changes in the new version of the component require that new test cases are developed. Developing new test cases is a process where component experts and test experts are involved. First, the high-risk areas are defined. Second, the available test cases are reviewed. Third, missing test cases are identified. Finally, these missing test cases are developed and can be executed. High-risk areas result in new test cases that have a specific coverage on that high-risk area. The areas that are of high-risk change for subsequent versions of the component. A specific test case with high coverage on a high-risk area is not optimal in the next version of the component when this risk is reduced, because this test case does not reduce much risk in the next version of the component. A test case covering many low risk areas is much more beneficial for the new version of the component. Often, test cases are developed when the risk in a specific area is high and then these test cases are not reviewed and changed anymore. One of the characteristics of platform development is that the majority of the components, used in a version of the system, remain unchanged in the next system version. In other words, the risk in these unchanged components remains at the same low level as the previous component version. One of the characteristics of platform development is that the majority of the components remain unchanged between two system versions. In other words, the risk in these unchanged components remains at the same low level as the previous component version. The selected set of test cases executed in test-diagnose-fix tasks for two versions of a system type should therefore be adjusted based on the risk in the system. This paper presents a method that defines the so-called next-best-test-case, using the coverage of the existing test cases and the failure probability of possible faulty areas in the system. The problem of defining the next-best-test-case is defined in Section 2. A basic algorithm is presented in Section 3. This algorithm is able to determine optimal next-best-test-cases by selecting the coverage of the test case such that the highest information gain is obtained. This algorithm is able to determine the next-best-test-case for small systems. Industrial-size systems contain many possible faulty areas (fault states) and large test sets. Therefore, Section 4 presents an algorithm to select the next-best-test-case using a combination of a clustering technique and the optimal next-best-test-case algorithm. The performance of the optimal next-best-test-case-algorithm and the clustered next-best-test-case algorithm are illustrated in Section 5. Section 6 describes two cases that have been performed using this method to improve the test set of large test phases at ASML.

2 Problem definition

The next-best-test-case problem is a selection problem, where the best test case, out of all possible test cases, needs to be selected according to some objective function. The test signature describing the coverage of the test case is what defines a test case. The coverage is described in terms of faulty areas in the system or fault states. The next-best-test-case algorithm determines the best signature according to an objective function. Available test cases are taken into account in the objective function assuming that these test cases are executed first. Consequently, the coverage of the available test cases on the fault states is taken into account. The available test cases, the fault states in the system and the coverage of the test cases on these fault states are modeled in a system test model. Sections 3 and 4 describe
the next-best-test-case algorithm that uses the system test model and objective function to calculate the next-best-test-case.

2.1 System test model
This model of the system, the system test model, is represented using a five-tuple \( D = (S, T, R_{ts}, P, I) \), where:

- \( S \) represents the set of fault states,
- \( T \) represents the set of test cases,
- \( R_{ts} \) represents the coverage relation between test and fault state,
- \( P \) represents the a-priori failure probability of the fault state and
- \( I \) represents the impact of a fault state on the next phase in the process.

This model is based on a system test model used for diagnosis, test sequencing and test planning [1–4].

Table 1 depicts a system test model for version \( v_1.0 \) of an example component. Six test cases are defined and five fault states. The failure probabilities and impact are defined per fault state. The IG row represents the information gain per test case that is derived from the elements of the model. The information gain per test case is described later in this section. The coverage of each test case on a fault state is represented by a value between 0 and 1, where 0 means that the test case does not cover the fault state and 1 means that the fault state is 100% covered by the test case. A value between 0 and 1 represents partial coverage.

<table>
<thead>
<tr>
<th>S / T</th>
<th>( t_0 )</th>
<th>( t_1 )</th>
<th>( t_2 )</th>
<th>( t_3 )</th>
<th>( t_4 )</th>
<th>( t_5 )</th>
<th>( P )</th>
<th>( I )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1 )</td>
<td>0.5</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
<td>80%</td>
<td>1</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td>80%</td>
<td>1</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
<td>10%</td>
<td>1</td>
</tr>
<tr>
<td>( s_4 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>10%</td>
<td>1</td>
</tr>
<tr>
<td>( s_5 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>10%</td>
<td>1</td>
</tr>
<tr>
<td>IG</td>
<td>0.89</td>
<td>0.97</td>
<td>0.97</td>
<td>0.29</td>
<td>0.93</td>
<td>0.99</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: An example system test model of component version 1.0, \( D_{v1.0} \)

Table 2 represents a system test model for the second version of the same component. The differences between both models are the failure probabilities and the additional fault state \( s_6 \). The objective function, used to evaluate possible next-best-test-cases, is based on the information gain of a test case. The information gain for a test case is maximal if a test case has a failure probability of 50%. The failure probabilities are reduced due to the test cases, diagnosis and fix tasks that are executed for version \( v1.0 \) or increased because of changes in the system. Fault state \( s_6 \) represents the specific changes for version \( v2.0 \) of the component.

<table>
<thead>
<tr>
<th>S / T</th>
<th>( t_0 )</th>
<th>( t_1 )</th>
<th>( t_2 )</th>
<th>( t_3 )</th>
<th>( t_4 )</th>
<th>( t_5 )</th>
<th>( P )</th>
<th>( I )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1 )</td>
<td>0.5</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
<td>8%</td>
<td>1</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td>8%</td>
<td>1</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
<td>8%</td>
<td>1</td>
</tr>
<tr>
<td>( s_4 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>8%</td>
<td>1</td>
</tr>
<tr>
<td>( s_5 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>70%</td>
<td>1</td>
</tr>
<tr>
<td>( s_6 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>80%</td>
<td>1</td>
</tr>
<tr>
<td>IG</td>
<td>0.92</td>
<td>0.24</td>
<td>0.24</td>
<td>0.24</td>
<td>0.52</td>
<td>0.94</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: An example system test model of component version 2.0, \( D_{v2.0} \)

3 Problem definition
The next-best-test-cases for model version v1.0 and v2.0 are expected to be different because of the failure probabilities of the fault states in the two versions of the model and the coverage of the test cases on the fault states \((s_i)\).

2.2 Objective function

The objective function, used to evaluate possible next-best-test-cases, is based on the information gain of a test case. The information gain for a test case, defined in Equation 1, is maximal if a test case has a failure probability of 50%. A test case with a failure probability of 50% has a 50% probability that a fault is found. Fault detection is one of the purposes of a test case. This test case also has a 50% probability that it passes. A passed test case means that the covered fault states do not exist in the system. A failing test case is an opportunity to improve the system (and reduce the failure probability). A passing test case reduces the failure probability of the covered fault states. A test case is represented by its test signature. A test signature represents the set of fault states that is covered by the test case. The information gain for a single test case with signature \(\text{sig}\) is calculated using Equation 1.

\[
IG(\text{sig}) = -\left( p_p(\text{sig}) \log_2 p_p(\text{sig}) + p_f(\text{sig}) \log_2 p_f(\text{sig}) \right)
\]

(1)

where \(p_p(\text{sig})\) is the pass probability of a test with signature \(\text{sig}\) and \(p_f(\text{sig})\) is the fail probability. The pass probability is defined by

\[
p_p(\text{sig}) = \prod_{s \in \text{sig}} \left( 1 - P(s) R(s) \right)
\]

(2)

and the corresponding fail probability as

\[
p_f(\text{sig}) = 1 - p_p(\text{sig})
\]

(3)

The objective function used in the next-best-test-case algorithm takes the coverage of the previously executed test cases into account. For this purpose, it is assumed that the test cases, currently present in the test model, are executed before the new next-best-test-case. The exclusion of fault states by previous test cases results in a change of the failure probability of the covered fault states. Therefore, the pass and failure probability of a possible new test case is affected. This means that Equation (2) to determine the pass probability for a test case needs to be revised such that the previously covered test cases are taken into account.

The optimal method to calculate the pass probability for a test case takes all combinations of covered fault states for each test case into account. This method is computationally intensive, because of the calculation of all possible fault state combinations for each test. Therefore, an estimator of the pass probability is required. This probability estimator estimates the pass probability for a test case with a high accuracy. The optimal method to determine the pass probability of a test case and the probability estimator are investigated in [5]. The probability estimator, defined in Equation (4), is used to calculate the pass probability of a test case.

\[
p_p(\text{sig}) = \prod_{s \in \text{sig}} \left( 1 - P(s) \sum_{s' \subseteq S \setminus S} \left( 1 - P(s') \right) \right)
\]

(4)

3 Optimal next-best-test-case algorithm

The next-best-test-case function \(\text{NBTC} : P(S) \times P(P(S)) \rightarrow P(S)\) takes a set of fault states as input and the set of candidate sets \([S_c]\) of fault states. The function returns the set of fault states covered by the next-best-test-case, the signature of the next-best-test-case. The set of fault states describes what could be covered by the next-best-test-case. The candidate set \([S_c]\) describes what is already covered by the test cases that are previously executed. The candidate set contains the signature of all test cases that are currently in the test model.
The candidate set $S_c$ is the set of test signatures for all test cases that are currently present in test set $T$, using the coverage relation $R(t) : \mathcal{P}(T) \rightarrow S$: $R(t) = \{ s \mid R_{nt}(t, s) > 0 \}$.

$$S_c = \{ R(t) \mid t \in T \}$$ (5)

The optimal next-best-test-case algorithm is defined as follows:

$$\text{NBTC}(S, S_c) = \{ \text{sig}^* \mid \forall \text{sig} \in \Theta(S, S_c) : IG(\text{sig}^*) \geq IG(\text{sig}) \}$$ (6)

The idea behind the optimal next-best-test-case algorithm is that the signature of a test case is a set of fault states. The idea behind the optimal next-best-test-case algorithm is that the signature of a test case is a set of fault states. Generating all possible combinations of fault states, therefore results in all possible test cases that can be evaluated. A system test model containing $l$ fault states results in $2^l$ fault state combinations. This includes the empty set $\emptyset$ that is removed, because a test case that does not cover any fault state is not useful. The set of fault states in the candidate set are also removed, because these ‘test cases’ are already present in the model (and have been executed). Calculating the information gain for each test signature and choosing the test signature with the highest information gain results in the next-best-test-case.

The function $\Theta : \mathcal{P}(S) \times \mathcal{P}(\mathcal{P}(S)) \rightarrow \mathcal{P}(\mathcal{P}(S))$ calculates all combinations of fault states of the fault states in $S$, excluding the empty set and the sets of fault states $S_c$ that are already covered in previous test cases $S_c$.

$$\Theta(S, S_c) = \text{setcombine}(S) \setminus (\emptyset \cup S_c)$$ (7)

The optimal next-best-test-case algorithm is able to determine the optimal test case for a given system test model, because all possible test cases are examined. The problem with the optimal next-best-test-case algorithm is that determining all possible test cases using Equation (7) is computationally intensive. Algorithms to generate all possible combinations are fairly simple, but the memory usage or duration of the calculation increases exponentially with the number of fault states in the system. Therefore, the optimal next-best-test-case algorithm cannot be used to determine the next-best-test-case for large system test models. The current maximum number of fault states lies around 17. The clustered next-best-test-case algorithm has been developed to overcome this disadvantage. The clustered next-best-test-case algorithm is explained in Section 4.

4 Clustered next-best-test-case algorithm

The clustered next-best-test-case algorithm replaces fault states with a new fault state, such that the information of the replaced fault states is represented by the new fault state. Clustering of fault states continues until $|S| < maxS$, where $3 \leq maxS \leq 15$. The next-best-test-case for the clustered system test model can now be calculated using the optimal next-best-test-case algorithm. This process is repeated on the resulting signature such that the result is refined.

A flowchart of the clustered next-best-test-case algorithm $\text{NBTC}_c$ is depicted in Figure 1.

In a formal definition the clustered next-best-test-case algorithm is defined as a function $\text{NBTC}_c : D \times \Pi^d \times N \rightarrow S$ that takes a system test model $D$, a clustering strategy $\Pi^d = \{ \text{minmin}, \text{minmax}, \text{maxmin}, \text{maxmax} \}$ and a maximum number of fault states in the cluster $maxS$ as input and outputs the signature of the next-best-test-case. The corresponding algorithm is described in Equation (8).

The system test model, used in the clustering algorithm, is based on the system test model defined in Section 2.1: $D = (S, T, R_a, P, I, R_c, S_c)$. The elements $R_a$ and $S_c$ are specific for the clustered next-best-test-case algorithm and used to maintain additional information required in the algorithm. The relation $R_c$ defines for a fault state in which cluster the fault state is clustered or $\{ \emptyset \}$ if the fault state is not clustered. This clustering relation is defined as

$$\text{The maximum number of fault states depends on the available computer processing power. A safe choice of 15 fault states is chosen here.}$$
Figure 1: Flowchart of the clustered next-best-test-case algorithm

The clustering relations defined for the set of all possible fault states $S$ are empty initially. Note that the domain of clustering relation is larger than the domain of the fault state set $S$ for the purpose of clustering and unclustering using Equations 23,31, and 32. The candidate set $S_c$ is used to maintain the signatures of the executed test cases.

$$R_c : S \rightarrow \mathcal{P}(S)$$

$$\text{NBTC}_c(d, \Pi^{cl}, maxS) =
\begin{cases}
\text{NBTC}(D.o, D.6) & \text{if } |D.o| \leq maxS \\
|D.o| \leq maxS \lor |t'| = |D.o| \lor |t'| = 1 & \text{if } |D.o| > maxS \land |t'| \neq |D.o| \land |t'| > 1
\text{NBTC}_c(\text{remove}(D, S \setminus t')) & \text{if } |D.o| > maxS \land |t'| \neq |D.o| \land |t'| > 1
\end{cases}$$

Where variable $t'$ stands for:

$$t' = \text{uncluster}(\text{NBTC}(D'.o, D'.6), D')$$

and

$$D' = \text{cluster}(D, \Pi^{cl}, maxS)$$

The clustered algorithm $\text{NBTC}_c$ calculates the next-best-test-case using the optimal next-best-test-case algorithm when $|S| \leq maxS$. Otherwise, a clustered next-best-test-case is calculated.
using the variables \( t' \) and \( D' \), which cluster the \( D \) first into \( D' \), calculate the clustered next-best-test-case and uncluster the clustered next-best-test-case into \( t' \). If the number of fault states in the signature is more or equal than \( \max S \) or if the signature of the clustered next-best-test-case \( t' \) is the same the original fault state set in the model \( D,o \) or the number of elements of the clustered next-best-test-case is 1, then the clustered next-best-test-case \( t' \) is the result of the algorithm. Otherwise, the fault states \( S \setminus t' \) are removed from the original model \( D \) by \( \text{removeS} \) and the algorithm restarts. The recursion stops when the size of the clustered next-best-test-case is 1 or if the clustered next-best-test-case is the same as the original set of fault states.

The function \( \text{removeS} : D \times \mathcal{P}(S) \rightarrow D \) removes a set of fault states \( \text{sig} \) from the system test model \( D \) and removes the relations between test cases that are not relevant anymore because of the removal of fault states.

\[
\text{removeS}(D, \text{sig}) = (S', T', D.2 \upharpoonright (S' \times T'), D.3 \upharpoonright S', D.4 \upharpoonright S', D.5 \upharpoonright S', \{x \mid \text{sig} \neq D.6 \setminus \{x\})
\] (11)

Where: \( S' = D.o \setminus \text{sig} \) and \( T' = D.1 \setminus \{t \mid \forall s : D.o \setminus \text{sig} \neq R_o(t, s) = 0 \} \). The other properties in the model are updated by projecting the changes in \( S \) and \( T \) on the properties using the \( \upharpoonright \) operator.

The function \( \text{uncluster} : \mathcal{P}(S) \times D \rightarrow \mathcal{P}(S) \times \mathcal{P}(S) \) translates the signature that could contain clustered fault states to a signature without clustered fault states. This function returns the unclustered signature by looking up all fault states that are in the model \( D' \) and related via \( D'5 \) with a clustered fault state in \( \text{sig} \). The uncluster function assumes that the clustering function \( \text{cluster} \) only adds clustered fault states, without reusing the properties of existing fault states for this purpose.

\[
\text{uncluster}(\text{sig}, D') = \{s \mid s \in D'.o \land (\exists s' : s' \in \text{sig} \land s' \in D'.5(s')) \lor (\exists s' : s' \in \text{sig} : s' \in D'.o))\}
\] (12)

The introduced clustered next-best-test-case algorithm can be used as a generic clustered next-best-test-case algorithm. The clustering technique determines what the performance is of the clustered next-best-test-case algorithm. The clustering technique determines what the performance is of the clustered next-best-test-case algorithm. The three clustering algorithms, used in this paper, are described next.

4.1 Clustering

The clustering technique used in the clustered next-best-test-case algorithm determines the performance of the algorithm, because a bad clustering algorithm results in next-best-test-cases that are not optimal at all. A general clustering algorithm is defined in Equation (13), and utilizes an objective function to cluster fault states. Three different clustering methods are defined. Additionally, a clustering strategy \( \Pi^d \) needs to be selected.

\[
\text{cluster}(D, \Pi^d, \max S) =
\begin{align*}
D & \quad \text{if } |D.o| \leq \max S \\
\text{cluster}(\text{mrg}(D, IG^<(x), IG^>(y))) & \quad \text{if } |D.o| > \max S \land \Pi^d = \text{minmin} \\
\text{cluster}(\text{mrg}(D, IG^<(x), IG^>(y))) & \quad \text{if } |D.o| > \max S \land \Pi^d = \text{minmax} \\
\text{cluster}(\text{mrg}(D, IG^<(x), IG^>(y))) & \quad \text{if } |D.o| > \max S \land \Pi^d = \text{maxmin} \\
\text{cluster}(\text{mrg}(D, IG^<(x), IG^>(y))) & \quad \text{if } |D.o| > \max S \land \Pi^d = \text{maxmax}
\end{align*}
\] (13)

Where, \( x \in IG^<(D.o) \) and \( y \in IG^<(D.o \setminus \{x\}) \). Four clustering strategies are defined: \text{minmin}, \text{minmax}, \text{maxmin} and \text{maxmax}. The \text{minmin} clustering strategy combines the two fault states with the two lowest values for the objective function. The \text{minmax} clustering strategies combine the two fault states with the lowest and highest value for the objective function and otherwise for the \text{maxmin} clustering strategy. Both strategies are equal. The \text{maxmax} clustering strategy selects two fault states with the two highest information gains.

Three clustering objective functions are defined: \( IG^+/IG^- \) based on the information gain per test signature, \( P^+/P^- \) based on the failure probability of a test signature and \( IGR^+/IGR^- \)
based on the combination of information gain and risk for a test signature. Each of the three objective functions is defined in two forms, a function to determine the minimal value and a function to determine the maximal value. The minimal and maximal values are used in the different clustering strategies.

A fault state with the lowest information gain is selected using Equation (14). A fault state with the highest information gain is selected using Equation (15). Fault states are not selected twice for merging, by removing the first fault state from the set of fault states as in variable y. Clustering using information gain as objective function is referred to as CTMIG in the remainder of this paper.

\[
IG^r(S) = \{x \in s' \mid s' \in S \land \forall s \in S : IG^r(s') \leq IG^r(s)\}
\]

(14)

\[
IG^r(S) = \{x \in s' \mid s' \in S \land \forall s \in S : IG^r(s') \geq IG^r(s)\}
\]

(15)

Where:

\[
IG^r(s) = -(p^r_s(t)s \log_2 p^r_s(t) + p^r_s(t)s \log_2 p^r_s(t))
\]

(16)

where \(p^r_s(t)s\) is defined as:

\[
p^r_s(t) = \prod_{t \in T} \left(1 - P(s)R_o(t, s)\right)
\]

(17)

and:

\[
p^r_s(t) = 1 - p^r_s(t)
\]

(18)

The second, similar to the first, clustering objective function clusters two fault states using failure probability instead of the information gain. Equations (19) and (20) are used to combine two fault states for the different strategies. Clustering using failure probability as objective function is identified with CTMP in the remainder of this paper.

\[
P^r(S) = \{x \in s' \mid s' \in S \land \forall s \in S : P(s') \leq P(s)\}
\]

(19)

\[
P^r(S) = \{x \in s' \mid s' \in S \land \forall s \in S : P(s') \geq P(s)\}
\]

(20)

The third clustering objective function clusters two fault states using the information gain per fault state together with the risk involved with this fault state. The use of risk in addition to the information gain could result in test cases with less information gain in favor of the risk covered. The usage of an objective function that is a combination of two functions has an advantage when determining what fault states can be clustered. The usage of risk could lead to better solutions. The information gain risk is calculated using Equations (21) and (22). Clustering using information gain and risk as objective function is identified as CTMIGR in the remainder of this paper.

\[
IGR^r(S) = \{x \in s' \mid s' \in S \land \forall s \in S : IG(R(s')) \leq IG(R(s))\}
\]

(21)

\[
IGR^r(S) = \{x \in s' \mid s' \in S \land \forall s \in S : IG(R(s')) \geq IG(R(s))\}
\]

(22)

The information gain risk for a fault state is defined as the product of the risk and information gain: \(IGR(s) = IG^r(s)P(s)R_o\).

The actual combination, or merging, of fault states is performed by the \(mrg\) function, defined in Equation (23). This function merges \(s'\) and \(s''\) into a new fault state \(newS\) and removes \(s'\) and \(s''\) from the system test model \(D\). Note that, the new fault state \(newS\) can be uniquely identified. Next to that, the properties related to \(s'\) and \(s''\) are updated. The \(mrg\) function is defined as:

\[
mrg(D, s', s'') =
((D \cup \{\text{newS}\}) \setminus \{s', s''\}, D.1, R_m, P', P'\prime, R_o, S')
\]

(23)

Where:
The performance of the clustered next-best-test-case is compared with the optimal next-best-test-case algorithm in this illustration. For this purpose, a large number of system test models have been generated. These system test models have been used to calculate the next-best-test-case for a range of clustering methods and strategies. The calculated next-best-test-cases are compared with the next-best-test-cases calculated using the optimal algorithm.

The following system test model parameters have been varied: the number of test cases in the model [8 or 16], the number of fault states in the model [5, 7, 9, 11 and 13], the failure probability for all fault states in the models [0.1, 0.25, 0.5, 0.75 and 0.9] and the density of the model [0.1, 0.25, 0.5, 0.75 and 0.9]. All these parameters are self explanatory, except the density that is defined as: 

\[ \rho = \frac{1}{|T \times S|} \sum_{t \in T \in S} R_0(t, s) \]

The density of a system test model is a measure for the coverage of all test cases on all fault states.

The system test models used for this analysis are randomly generated resulting in 250 system test models with different settings. These models are used to measure the performance of the clustered next-best-test-case algorithm. The clustering method, the clustering strategy and the maximum number of fault states in a cluster are varied, such that the influence of these parameters on the information gain of the calculated next-best-test-case is measured. Furthermore, the optimal next-best-test-case algorithm has been applied, such that the results of the clustered algorithm and different settings can be compared with the optimal next-best-test-case. A total number of 23050 next-best-test-cases has been determined in this setup. A number of results can be obtained from this data. The 5 different settings: model density,
clustering method, clustering algorithm, failure probability of the model and whether refinement is used are varied for each of the 10 models generated with different modeling settings. This leads to 38 results for each model and combination of the modeling settings. No results are obtained when \( \text{max}^S > |S| \).

The first result that is evaluated is the effect of the clustering method and clustering strategy versus the average failure probability of the fault states and the density of the system test model. The average information gain of each next-best-test-case is compared with information gain of the optimal next-best-test-case. Table 3 depicts the results of this analysis. The information gain for the clustered next-best-test-cases is an average over 38 next-best-test-cases calculated for each of the different settings. These 38 next-best-test-cases were calculated for a combination of different models and maximum fault states \( \text{max}^S \). The optimal information gain is an average of 10 optimal next-best-test-cases calculated for 10 models with different settings.

Table 3 contains 81 results. The effect of the clustering strategy is investigated. Therefore, the best clustering strategy is selected for the 27 combinations of clustering method, failure probability and density. The \text{minmin} clustering strategy performs best in 23 of the 27 cases. The \text{maxmax} clustering strategy performs better in two of the 27 cases. The \text{minmax} clustering strategy performs better in one of the 27 cases. The \text{minmin} and \text{minmax} clustering strategy lead to the same results in one of the 27 cases.

The results of the experiments for different clustering methods (CTMIG, CTMP or CTMIGR) are less conclusive. It can be seen that all three clustering methods are optimal in some case, for all combinations of clustering strategies, failure probabilities and model densities. This is also the case if only the best clustering strategy (\text{minmin}) is analyzed.

The second investigated clustering setting is the use of the refinement step, defined in Equation (8) as recursive call to the \( NBTGd \) algorithm. For this purpose, a number of experiments has been conducted with and without the use of the refinement step in the algorithm. The results of these experiments for the different models are depicted in Table 4. The table depicts the average information gain. The models are described as 16x11 if the model contained 16 test cases and 11 fault states. Three rows of results are presented: \text{don't refine} if no refinement step was used, \text{optimal} for the optimal result and \text{refine} if the refinement step was applied. An 19\% improvement of the information gain is obtained if the refinement step in the algorithm is used. The data in Table 4 includes all clustering strategies. If the results of the \text{maxmax} and \text{minmax} clustering strategies are removed from the dataset, then the average improvement due to refinement is 7\%. Both results justify the application of the refinement step in the clustered next-best-test-case algorithm.

The average information gain of the next-best-test-cases, which is derived using the clustered next-best-test-case algorithm, is on average 0.82, while the information gain derived using the optimal next-best-test-case algorithm is 1.0 on average. Additionally, the application of the refinement step is justified, because improvement of the information gain between 7\% and 19\% is observed.

Table 3: The average information gain for different clustering methods and strategies versus the average failure probability and model density

<table>
<thead>
<tr>
<th>Clustering Method</th>
<th>Clustering Algorithm</th>
<th>Failure Probability</th>
<th>Model Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTMIG</td>
<td>maxmax</td>
<td>0.703</td>
<td>0.705</td>
</tr>
<tr>
<td></td>
<td>minmax</td>
<td>0.713</td>
<td>0.717</td>
</tr>
<tr>
<td></td>
<td>minmin</td>
<td>0.892</td>
<td>0.845</td>
</tr>
<tr>
<td>CTMPIGR</td>
<td>maxmax</td>
<td>0.668</td>
<td>0.793</td>
</tr>
<tr>
<td></td>
<td>minmax</td>
<td>0.709</td>
<td>0.763</td>
</tr>
<tr>
<td></td>
<td>minmin</td>
<td>0.936</td>
<td>0.798</td>
</tr>
<tr>
<td>CTMP</td>
<td>maxmax</td>
<td>0.691</td>
<td>0.832</td>
</tr>
<tr>
<td></td>
<td>minmax</td>
<td>0.712</td>
<td>0.786</td>
</tr>
<tr>
<td></td>
<td>minmin</td>
<td>0.914</td>
<td>0.766</td>
</tr>
<tr>
<td>CTMPG</td>
<td>maxmax</td>
<td>0.770</td>
<td>0.755</td>
</tr>
<tr>
<td></td>
<td>minmax</td>
<td>0.709</td>
<td>0.763</td>
</tr>
<tr>
<td></td>
<td>minmin</td>
<td>0.914</td>
<td>0.766</td>
</tr>
</tbody>
</table>
Table 4: The average information gain for different system test models and the application of a refinement step in the algorithm.

6 Case

Two case studies have been performed with the clustered next-best-test-case algorithm. The first case study determines the next-best-test-case for the weekly validation test for software that controls an ASML wafer scanner. The second case study determines the next-best-test-case at the start of the alpha test, a test-diagnose-fix task executed to determine if a new software release operates according to the system specifications.

6.1 Case 1: Weekly validation test

At ASML, the weekly validation test for the software baseline is executed weekly to determine if the software still operates according to the specifications. The selection of test cases and the sequence in which the test cases are executed is determined every week, based on the software delivered to the baseline in that week. For this purpose, a system test model is maintained with 349 test cases and 156 fault states. The set of test cases available for selection is fixed, while new test cases could improve the selection or sequence. This case study applies the next-best-test-case algorithm to the system test model.

The properties of the system test model are:

- Number of test cases: 349
- Number of fault states: 156
- Average failure probability: 0.278
- Average impact: 0.449
- Average information gain of the test cases: 0.685
- Average failure probability of the test cases: 0.483
- Average coverage of the test cases on the fault states: 0.244

A total of 5 next-best-test-cases has been derived using the clustered next-best-test-case algorithm. The coverage of the fault states that are covered by newly generated next-best-test-case is set to 0.224, which is the average coverage of the test cases that are already present in the system test model. Only the first test case is considered for development, because the other four test cases depend on the coverage of the previous test cases, including the first next-best-test-case. The other four test cases are derived to check the performance of the algorithms and settings.

The maximum number of fault states in a cluster was set to 13, such that the largest possible clusters were used and the results were obtained quickly. The used clustering techniques are: CTMIG, CTMP and CTMIGR. The \textit{minmin} and \textit{maxmax} clustering strategies were both used for all three clustering techniques. The resulting information gain profiles for the three techniques are depicted in the Figures 2, 3 and Table 6.1 for the \textit{minmin} and \textit{maxmax} strategy respectively.

It can be concluded from the Table 6.1 and Figures 2 and 3 that the \textit{maxmax} clustering strategy performs worse than the \textit{minmin} clustering strategy. The first next-best-test-case
of the CTMIGR clustering method, with \textbf{minmin} clustering strategy, performs best, i.e., the information gain for this test case is $IG(t_{350}) = 1.0$.

The test experts analyzed the newly generated test case $t_{350}$. Test case $t_{350}$ covers a single fault state $s_{37}$ with a failure probability of 96.6%. Two other test cases $t_{320}$ and $t_{313}$ that were already present in the system test model and covered $s_{37}$ both with a coverage of 0.3. The failure probability of $s_{37}$ was reduced by the two existing test cases into a failure probability of: $P(s_{37})_{t_{320}},t_{313} = (1-R_{ts}(t_{320},s_{37}))(1-R_{ts}(t_{313},s_{37})) = 0.066(1-0.3)(1-0.3) = 0.473$. The information gain of a test case that only covers this fault state with a failure probability of $P(s_{37}) = 0.473$, leads to an information gain close to 1.

The second test case $t_{351}$ derived using the same clustering technique and strategy also leads to an information gain of approximately 1.0. This test case again covers fault state $s_{37}$, which has a failure probability of $P(s_{37}) = 0.378$, after executing all test cases in the model including $t_{350}$. A second fault state $s_{73}$ is covered by $t_{351}$, such that the information gain is improved.

### 6.2 Case 2: Software alpha test

ASML releases a software baseline to customers when a new type of wafer scanner is released. Additional ‘consolidation’ software releases are released to enable customers to upgrade all

![Figure 2: The information gain profile for the 5 test cases determined using the minmin strategy](image)

![Figure 3: The information gain profile for the 5 test cases determined using the maxmax strategy](image)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>CTM</th>
<th>CTMP</th>
<th>CTMIGR</th>
</tr>
</thead>
<tbody>
<tr>
<td>350</td>
<td>0.949</td>
<td>0.873</td>
<td>1.000</td>
</tr>
<tr>
<td>351</td>
<td>0.929</td>
<td>0.954</td>
<td>0.956</td>
</tr>
<tr>
<td>352</td>
<td>0.820</td>
<td>0.498</td>
<td>0.428</td>
</tr>
<tr>
<td>353</td>
<td>0.879</td>
<td>0.001</td>
<td>0.973</td>
</tr>
<tr>
<td>354</td>
<td>0.879</td>
<td>0.473</td>
<td>0.474</td>
</tr>
</tbody>
</table>

**Table 5:** Information gain of the 'new' weekly validation test cases for the different clustering techniques and strategies
wafer scanners in the IC factory to the same software release. The number of different software releases delivered to customers is four or more per year. Every software release is qualified in an alpha test and beta test before it is ready to be rolled out at customers world-wide. Alpha testing is performed on different types of wafer scanners at the ASML premises. Beta testing is typically performed at wafer scanners that are already running production at customers worldwide. The alpha test of a new software release has a number of goals. First, it needs to be tested if the performance of the wafer scanners is equal or better with the new software release. Second, the high-risk areas in the new software release are tested. Third, problems found during the alpha test period are analyzed, solved and retested. A standard set of performance test cases is available to test the performance of the new release. This set is always executed. Another set of test cases is selected to meet the second goal. This selection process is continued throughout the test execution and problem solving process to meet the third goal. A system test model has been created to support the selection process of the alpha test cases. The system test model had the following characteristics at the start of the alpha test:

- Number of test cases: 76
- Number of fault states: 15
- Average failure probability: 0.306
- Average impact: 0.393
- Average information gain of the test cases: 0.4535
- Average failure probability of the test cases: 0.6212
- Average coverage of the test cases on each fault state: 0.189

The same modeling parameters as used in the previous case study were applied: five ‘new’ test cases were defined using the three clustering methods and the minmin and maxmax clustering strategy. The information gain profile of all clustering methods and strategies is depicted in Figure 4. The detailed results are depicted in Table 6.2.

<table>
<thead>
<tr>
<th></th>
<th>CTM/min-min</th>
<th>CTMP/min-min</th>
<th>CTMIGR/min-min</th>
</tr>
</thead>
<tbody>
<tr>
<td>77</td>
<td>0.450</td>
<td>0.410</td>
<td>0.460</td>
</tr>
<tr>
<td>78</td>
<td>0.460</td>
<td>0.400</td>
<td>0.460</td>
</tr>
<tr>
<td>79</td>
<td>0.460</td>
<td>0.460</td>
<td>0.460</td>
</tr>
<tr>
<td>80</td>
<td>0.460</td>
<td>0.460</td>
<td>0.460</td>
</tr>
<tr>
<td>81</td>
<td>0.460</td>
<td>0.460</td>
<td>0.460</td>
</tr>
</tbody>
</table>

Table 6: Information gain of the ‘new’ alpha test cases for the different clustering methods and strategies

Figure 4: Information gain profile for the different clustering methods and strategies
The test case resulting in the best information gain (1.0) is derived using the CTM clustering method with a \textbf{maxmax} clustering strategy. For this case study, the \textbf{maxmax} clustering strategy leads in general to better results. The fault states covered by the best test case are: \{s_5, s_{11}, s_{13}\}. The properties of these fault states are \(P(s_5) = 82.85\%\), \(P(s_{11}) = 0.1\%\) and \(P(s_{13}) = 30\%\). All three fault states are already covered by other test cases. Fault state \(s_5\) describes the possibility that the software operating system (OS) is faulty and is the lowest level fault state. Fault state \(s_{11}\) describes the possibility that the measurement sensors are faulty, a mid-level fault state. Fault state \(s_{13}\), a high-level fault state, models the possible fault that a complete system is not calibrated. The test graph for this model, depicting the relations between fault states and test cases, can be found in Figure [6].

A test case covering fault state \(s_{11}\) and \(s_5\) can be designed using available test means. However, a test case that covers sensors and the system calibration, fault state \(s_{13}\) and \(s_5\), and stresses the software OS in addition, is more difficult to design. Whether this test case is actually developed depends on the required and available development effort.

\section{Conclusions}

The complexity of manufacturing machines, like the ASML wafer scanner, increases as well as the number of test cases that are available to test the components and sub-systems. Over time, some test cases become irrelevant, while other test cases could be beneficial for the performance of a test-diagnose-fix task. An example of test cases that become less relevant over time are design qualification test cases that are executed once. An example of test cases that could be beneficial are very specific test cases covering high-risk areas.

A method to determine which new test cases are most beneficial has been presented. The gained information is used as objective function to determine these so called \textit{next-best-test-cases}. The method does not take into account if a next-best-test-case can actually be developed. The method guides the development. A system test model is used to model the test cases and the coverage of these test cases on possible fault states.

An optimal next-best-test-case algorithm is described that is able to determine the optimal next-best-test-case for systems of limited size. A clustered next-best-test-case algorithm is defined that is able to determine the next-best-test-case for larger systems. The performance of these two algorithms has been illustrated and two industrial case studies have been performed at ASML. The next-best-test-cases that are determined for the industrial case studies have been discussed with the test experts, test architects, at ASML. It is difficult for the experts to evaluate if the suggested next-best-test-cases are good test cases, because of the size of the model. Depicting the system test model as a test graph helps. The number of relations between the test cases and fault states is the reason why this next-best-test-case algorithm has been investigated.


Authors biography

I.S.M. de Jong has a B.Sc. in Laboratory Informatics and Automation from Breda Polytechnic. He has been a software engineer in various companies in the USA and The Netherlands. Since 1996 he has worked with ASML in systems testing, integration, release and reliability projects. His specialization is in the field of test strategy. Since 2003 he is an active member in the TANGRAM project and a Ph.D. student at the Eindhoven University of Technology. His research concerns integration and test strategies.

R. Boumen received his M.Sc. degree in Mechanical Engineering from the Eindhoven University of Technology, the Netherlands, in 2004. During his work as a master student he worked in the field of supervisory machine control of lithographic machines. Since 2004 he is a Ph.D. student at the Eindhoven University of Technology. His research, embedded in the TANGRAM project, concerns test strategies.

J.M. van de Mortel-Fronczak graduated in computer science at the AGH University of Science and Technology of Cracow, Poland, in 1982. In 1991, she received the Ph.D. degree in computer science from the Eindhoven University of Technology, the Netherlands. Since 1997 she has worked as assistant professor at the Department of Mechanical Engineering, Eindhoven University of Technology. Her research interests include specification, design, analysis and verification of supervisory machine control systems.

J.E. Rooda received the M.S. degree from Wageningen University of Agriculture Engineering and the Ph.D. degree from Twente University of Technology, The Netherlands. Since 1985 he is Professor of (Manufacturing) Systems Engineering at the Department of Mechanical Engineering of Eindhoven University of Technology, The Netherlands. His research fields of interest are modelling and analysis of manufacturing systems. His interest is especially in the control of manufacturing lines and in the supervisory control of manufacturing machines.