EINDHOVEN UNIVERSITY OF TECHNOLOGY

# Integration and test sequencing for complex systems

*Citation for published version (APA):*
Boumen, R., Jong, de, I. S. M., Mestrom, J. M. G., Mortel - Fronczak, van de, J. M., & Rooda, J. E. (2007). *Integration and test sequencing for complex systems*. (SE report; Vol. 2007-07). Technische Universiteit Eindhoven.

*Document status and date:*
Published: 01/01/2007

*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: 16. May. 2025

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. 2007-07

# Integration and test sequencing for complex systems

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

ISSN: 1872-1567

SE Report: Nr. 2007-07
Eindhoven, April 2007
SE Reports are available via http://se.wtb.tue.nl/sereports

**Abstract**

The integration and test phase of complex manufacturing machines, like an ASML [1] lithographic manufacturing system, is expensive and time consuming. The tests that can be performed at a certain point in time during the integration phase depend on the modules that are integrated and therefore on the integration sequence. In this paper, we introduce a mathematical model to describe an overall integration and test sequencing problem and we propose an algorithm to solve this problem. The method is a combination of integration sequencing and test sequencing. Furthermore, we introduce several strategies that determine when test phases should start. With a case study within the development of a software release that is used to control a lithographic machine, we show that the described method and strategies can be used to solve real-life problems.

# 1 Introduction

In today's industry, time-to-market is increasingly important. Therefore, the development of systems is done concurrently. During the integration phase of a system, the different subsystems are assembled or integrated into a system and tested. This integration and test phase typically takes more than 45% of the total development time of a complex manufacturing system. Reducing this time reduces the time-to-market of a new system.

An integration and test plan describes the integration actions and the test cases that are performed in the integration and test phase of a system. For new ASML machines, this integration and test plan is currently made by hand which costs a lot of effort and often results in a plan that is suboptimal with respect to time. Creating an optimal integration and test plan automatically can decrease integration and test time and planning effort.

In our previous work [2], we developed a method that optimizes an integration sequence. This method is based on de Mello *et. al* [3, 4] where a method is introduced for optimizing mechanical assembly sequences and on Hahn *et. al* [5] where optimal integration strategies for object-oriented systems are determined. In [2], we extended this method with tests and the relation between tests and modules that defines which modules need to be integrated before a certain test may be performed. One of the main assumptions of this integration sequencing method is that every test is applied once, at the moment that it can be performed. However, in complex integration plans, certain tests are applied more than once, for example to ensure the quality of a system during integration. Additionally, certain tests are never performed because other tests cover the same requirements. Therefore, the integration method as proposed in [2] is only suited for a subset of integration sequencing problems.

In this paper, we introduce an integration and test sequencing method that is able to cope with more complex test phases. The method creates an optimal integration sequence and for each test phases within this integration sequence an optimal test sequence. This is done by incorporating test sequencing in the integration sequencing method. The test sequencing method creates the optimal test sequence based on a test model description of the test problem, see [6, 7]. This method is based on sequential diagnosis methods as described by Pattipati *et. al* [8]. The combined method is able to create the optimal complete integration and test sequence towards time or cost.

Furthermore, four strategies are introduced that can be used to define the start moment of test phases during integration. Every strategy has its own properties and is therefore suitable for a different type of integration problems.

The paper is constructed as follows. Section 2 explains in short the separate integration sequencing and test sequencing methods. Section 3 explains the combined integration and test sequencing method. Section 4 describes the strategies that can be used to determine the start moment of test phases. Section 5 shows the algorithm used to solve integration and test sequencing problems. In Section 6, a case study within the development of a lithographic machine software system is performed to show the benefits of applying this method. Finally, Section 7 discusses the conclusions.

# 2 Background

This section gives the necessary background information of the integration sequencing method, as presented in [2], and the test sequencing method, as presented in [6, 7], needed for the re-

mainder of this paper.

### 2.1 Integration sequencing

To illustrate an integration sequencing problem we use a simplified description of an ASML lithographic system or wafer scanner (see [2] for more details). This system consists of 7 modules ($m_1$ through $m_8$) that are connected through interfaces ($i_1$ through $i_8$). For each of the modules, a development time is known and for each integration of two modules into a subsystem the creation time (assembly time) of every interface is known.

The objective is to find a sequence of integration actions for each module such that the total integration time is minimized. The complete integration sequencing problem can be expressed in terms of the integration model $(M, I, T, C^m, C^i, C^{tm}, R^{im}, R^{tm})$, consisting of the following elements:

- $M, I, T$ are finite sets of modules, interfaces and tests.
- $C^m : M \to \mathbb{R}^+$ gives for each module in $M$ the associated cost in time units of developing that module.
- $C^i : I \to \mathbb{R}^+$ gives for each interface in $I$ the associated cost in time units of creating that interface.
- $C^t : T \to \mathbb{R}^+$ gives for each test in $T$ the associated cost in time units of performing that test.
- $R^{im} : I \to M \times M$ gives for each interface in $I$ the modules the interface is constructed in between.
- $R^{tm} : T \to \mathcal{P}(\mathcal{P}(M))$ gives for each test in $T$ its essential assemblies; an essential assembly describes the modules that should be integrated with each other before the test can be performed.

The assumptions for this integration model are:

- All modules in $M$ must be connected with each other, so there exists a path of interfaces that connects every module in $M$ with every other module in $M$. This makes sure that there is exactly one system.
- For every test in $T$, there exists at least one module that is present in all essential assemblies of this test. Otherwise, the same test may be performed twice in parallel on two different subsystems which increases cost.
- Each test only needs to be performed exactly once. The test is performed at the moment that one of the essential assemblies of this test is integrated. This makes sure that the test is only performed once. This assumption is only valid for the integration sequencing method. For the combined integration and test sequencing methods, this assumption is not needed.

Furthermore, we define an assembly as a collection of modules which are integrated. Assemblies are represented by elements of $\mathcal{P}(M)$ (except $\emptyset$). An integration action is defined as instantiating all interfaces between exactly two assemblies. Integration actions are therefore represented by elements of $\mathcal{P}(I)$ (except $\emptyset$). A test phase consists of the set of tests that are performed on a subassembly. Test phases are therefore represented by elements of $\mathcal{P}(T)$.

## 3 Background

Table 1: $M$, $I$, $C^m$, $C^i$ and $R^{im}$ of the integration model

| $I$ / $M$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $C^i$ |
|---|---|---|---|---|---|---|---|---|
| $i_1$ | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $i_2$ | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| $i_3$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| $i_4$ | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| $i_5$ | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| $i_6$ | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| $C^m$ | 10 | 15 | 10 | 10 | 15 | 20 | 25 | |

Table 2: $T$, $C^t$ and $R^{tm}$ of the integration model

| $T$ | $R^{tm}$ | $C^t$ |
|---|---|---|
| $t_1$ through $t_6$ | $\{\{m_1\}\}$ | 1 |
| $t_7$ through $t_8$ | $\{\{m_1, m_2\}\}$ | 2 |
| $t_9$ through $t_{11}$ | $\{\{m_4\}\}$ | 1 |
| $t_{12}$ through $t_{13}$ | $\{\{m_3, m_4\}\}$ | 2 |
| $t_{14}$ through $t_{17}$ | $\{\{m_5, m_6, m_7\}\}$ | 3 |
| $t_{18}$ through $t_{20}$ | $\{\{m_2, m_4, m_5, m_6, m_7\}\}$ | 3 |
| $t_{21}$ through $t_{25}$ | $\{\{m_1, m_2, m_3, m_4, m_5, m_6, m_7\}\}$ | 5 |

Elements $M$, $I$, $C^m$, $C^i$ and $R^{im}$ are shown in Table 1 for the scanner example. Elements $T$, $C^t$ and $R^{tm}$ are shown in Table 2.

The solution to an integration sequencing problem can be represented by a function $G : M \to (\mathcal{P}(T) \cup \mathcal{P}(I))^*$, ($^*$ denotes a set of sequences), that gives for a single element of $M$, a sequence of integration actions (sets of interfaces $\mathcal{P}(I)$) and test phases (sets of tests $\mathcal{P}(T)$) that integrates this single module into the completely integrated and tested system. Such a solution can be represented as a tree of integration sequences. The cost of such a solution in terms of duration is:

$$J(G) = \max_{m \in M} \left( C^m(m) + \sum_{t \in G^t_m} C^t(t) + \sum_{i \in G^i_m} C^i(i) \right) \tag{1}$$

Where $G^t_m$ is the set of all tests that are present in all test phases of a solution $G(m)$, and $G^i_m$ is the set of all interfaces that are present in all integration actions of a solution $G(m)$.

A solving algorithm performs an AND/OR graph search to find the optimal solution to this problem. The optimal solution for the scanner example is shown in Figure 1(a). This figure shows for each module (square node), the integration steps (hexagonal nodes) which consist of creating interfaces, and the test steps (oval nodes) that should be done. The edges from one node to another node denote a precedence relation between these two nodes. The longest paths in this tree is the path of module $m_7$ which is 73 time units. This cost is the sum of the development of $m_7$ (25), the creation of interface $i_4$ (1), performing tests $t_{14}$ through $t_{17}$ (12), the creation of interface $i_5$ (1) and performing tests $t_{18}$ through $t_{25}$ (34). The cost of this optimal solution is therefore 73.

Table 3: System test model of the test-phase for module $m_1$

| $S$ / $T$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $P$ |
|---|---|---|---|---|---|---|---|
| $s_1$ | 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 % |
| $C^t$ | 3 | 1 | 1 | 1 | 2 | 2 | |

## 2.2 Test sequencing

The test sequencing method uses a system test model to derive the cost-optimal test tree. To illustrate this, we derive a test sequence for the first test phase of module $m_1$ of the integration sequence shown in Figure 1(a). This test phase consists of 6 tests, $t_1$ through $t_6$, that should be performed. With the test sequencing method, we are able to derive the optimal test selection and sequence for that test phase.

Module $m_1$ contains 5 possible faults that could be present and that can be detected by the tests. These possible faults are modeled as fault states. Each fault state has an associated fault probability that it is present.

The system test model is defined as a 5-tuple $(T, S, C^t, P, R^{ts})$, where:

- $T$ and $S$ are finite sets of tests and fault states.
- $C^t : T \rightarrow \mathbb{R}^+$ gives for each test in $T$, the associated cost of performing that test (the same as defined in the integration sequencing model).
- $P : S \rightarrow \mathbb{R}^+$ gives for each fault state in $S$, the *a priori* (absolute) probability that the fault state is present.
- $R^{ts} : T \rightarrow \mathcal{P}(S)$ gives for each test in $T$, a subset of fault states that is covered by that test and is also known as the test signature. This element can be represented as a matrix $A$ where $A_{ij} = 1$ if test $t_j$ covers fault state $s_i$, otherwise $A_{ij} = 0$.

The system test model for the test phase of module $m_1$ is shown in Table 3, where $R^{ts}$ is represented by a matrix $A$ shown as the relation between $T$ and $S$.

The following assumptions hold:

- Tests can only have two outcomes (pass or fail).
- Fault states are independent.
- Fault states each have a unique test signature such that they can be distinguished.
- Tests are 100% reliable, meaning a fault state is certainly present if a test fails.
- Tests are 100% sensitive, meaning a test always fails if a covered fault state is present.
- Each fault state has a fix action.
- A fix action certainly repairs a fault state.

- The time (cost) of a fix action is not taken into account.

A solution $G^t$ to the system test sequencing problem is a function $G^t : \mathcal{P}(S) \rightarrow T^*$, which gives for each set $S_S \subseteq S$ of fault states that could be present, a test sequence $G^t(S_S)$ with tests from $T$ that isolates and fixes every fault state in $S_S$. The cost of such a solution is [8]:

$$J^t(G^t) = \sum_{S_S \subseteq S} \sum_{t \in G^t(S_S)} \left( C(t) \prod_{s \in S_S} P(s) \prod_{s' \in (S \setminus S_S)} (1 - P(s')) \right) \tag{2}$$

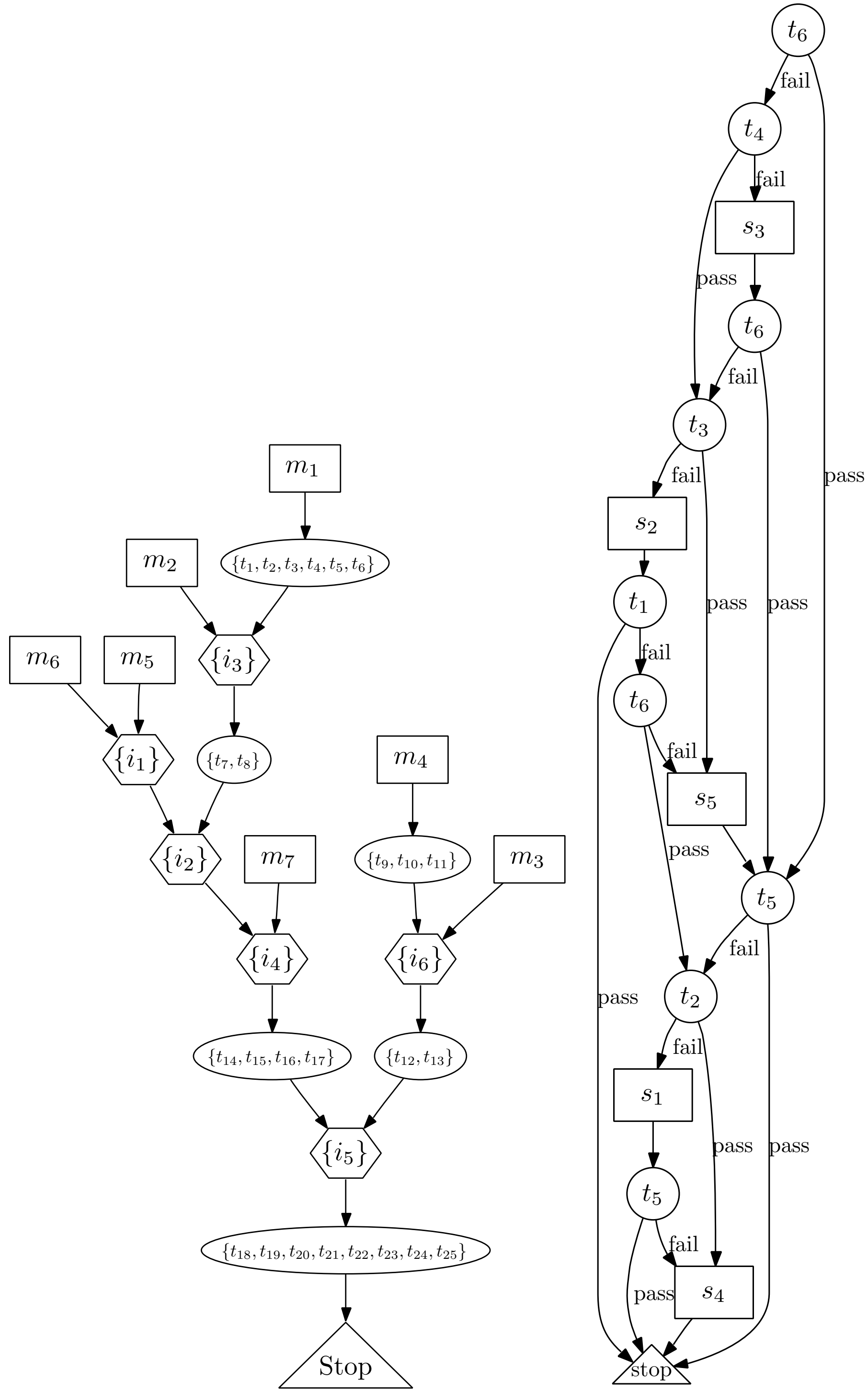
The multiple-fault $AO^*_\sigma$ algorithm proposed in [6] determines the optimal solution $G^t$ with respect to the cost. This is done by constructing an AND/OR graph, where AND nodes denote the tests that are applied to an OR node. Every OR node denotes the system ambiguity, that is all possible fault state combinations that could be present in the system considering previous test outcomes. After the construction of the complete or only the most promising part (determined by heuristics) of the AND/OR graph, the cheapest solution is selected. The cost-optimal test tree for the module $m_1$ is shown in Figure 1(b). The nodes in this tree denote actions that have to be performed: round nodes denote tests, square nodes denote fix actions of fault states and the triangle node is the leaf node that denotes when to stop testing. The directed edges denote a precedence relation between actions, for tests the edges denote the precedence relation given the outcome of that test. The expected test cost $J^t$ of this solution is 5.3 cost units.

# 3 Integration and test sequencing

The integration sequencing method described in the previous section has the assumption that each test is performed exactly once as soon as it can be performed. This may create not-optimal test phases because some tests may be redundant and therefore do not need to be performed. To create optimal test phases for each test phase, the described test sequencing method can be used. Using this method makes the assumption during integration sequencing method unnecessary. Therefore, we propose a new method that combines the two previously mentioned models and algorithms.

The basic idea of integration and test sequencing is that during integration sequencing, the test sequences and therefore the cost of the test phases are calculated using the test sequencing algorithm. To do so, the following additional actions are required:

- During integration sequencing, the possible fault states need to be known. Fault states can be introduced by the development of a module or by creating an interface between modules (integration of two modules).
- Test phases are started depending on the chosen test strategy. Every time a test phase is started, the corresponding test model should be available. This test model consists of the present fault states that can be tested and the tests that may be performed. If the test model is empty (no fault states or no tests), the test phase is also empty and no tests are performed. There are 4 different strategies that each construct test models according to some rules. Each of them is explained in more detail in the following section.
- The test phase cost are calculated using the test sequencing algorithm. Therefore, the objective function of the integration sequencing algorithm is changed.




(a) Integration sequence where $J^i = 73$

(b) Test sequence where $J^t = 5.3$

Figure 1: Solutions of the scanner illustration

## 7 Integration and test sequencing

### 3.1 Model

The complete integration and test sequencing problem called $D$ can be formulated in terms of the 12-tuple $(M, I, T, S, C^m, C^i, C^t, R^{im}, R^{tm}, R^{ts}, R^{is}, R^{ms})$, where:

- $M, I, T, S, C^m, C^i, C^t, R^{im}, R^{tm}$ and $R^{ts}$ are already defined in the test or integration model.
- $R^{is} : I \rightarrow \mathcal{P}(S \times \mathbb{R}^+)$ gives for each interface in $I$ the fault states that are introduced when creating this interface and the probabilities of each introduced fault state.
- $R^{ms} : M \rightarrow \mathcal{P}(S \times \mathbb{R}^+)$ gives for each module in $M$ the fault states that are introduced when developing this module and the probabilities of each introduced fault state.

This 12-tuple is a combination of the test model and the integration model, except that element $P$ of the test model is replaced by elements $R^{is}$ and $R^{ms}$. With these elements it is possible to calculate for each subassembly consisting of a number of modules and interfaces the probability for each fault state.

The assumptions for this model are the same as for the test and integration models, except that:

- The assumption that each test should be performed exactly once does not hold anymore.
- After each integration and each development action a test phase is performed, testing the fault states that are defined by the test strategy with the tests that are defined by the test strategy.

### 3.2 Illustration

The scanner example introduced in Section 7 is extended with $R^{is}$ and $R^{ms}$ to illustrate the integration and test sequencing method. Elements $M, I, R^{im}, C^i, C^m$ are shown in Table 1, elements $T, R^{tm}, C^T$ in Table 2, elements $T, S, R^{ts}, C^t$ ($P$ is no part of the integration and test model) in Table 4 and the new elements $R^{is}, R^{ms}$ in Table 5. A part of Table 4 has been shown previously as the test model in Table 3.

### 3.3 Objective

A solution to an integration and test sequencing problem is a tree with one root node that represents the completely integrated and tested system (all fault states are isolated and fixed and all modules are assembled), nodes that represent integration actions, nodes that represent test phases and leaf nodes that represent the untested modules. This tree can be represented by a function $G^i : M \rightarrow (\mathcal{P}(I) \cup (\mathcal{P}(S) \times \mathcal{P}(T)))^*$, which gives for each module in $M$ a sequence of integration actions and test phases that integrates this module into the completely integrated and tested system. The test phases are denoted by the fault states $S' \subseteq S$ that are tested and the tests $T' \subseteq T$ that can be used. For each test phase, a function $G^t$ is defined that gives the optimal test sequence for this test phase. Function $G^t : \mathcal{P}(S') \rightarrow (T')^*$, gives for each set $S_S \subseteq S'$ of fault states a test sequence $G^t(S_S)$, with tests from $T'$ that isolates and fixes every fault state in $S_S$. The total time of such a complete test and integration solution is:

Table 4: $T, S, R^{ts}, C^t$ of the integration and test model

| $S$ / $T$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ | $t_{11}$ | $t_{12}$ | $t_{13}$ | $t_{14}$ | $t_{15}$ | $t_{16}$ | $t_{17}$ | $t_{18}$ | $t_{19}$ | $t_{20}$ | $t_{21}$ | $t_{22}$ | $t_{23}$ | $t_{24}$ | $t_{25}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_1$ | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| $s_2$ | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| $s_3$ | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| $s_4$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| $s_5$ | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| $s_6$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| $s_7$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| $s_8$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| $s_9$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| $s_{10}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| $s_{11}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| $s_{12}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| $s_{13}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $s_{14}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $s_{15}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $s_{16}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| $s_{17}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| $s_{18}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| $s_{19}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| $s_{20}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| $s_{21}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| $s_{22}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| $s_{23}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $s_{24}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $C^t$ | 3 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 |

Table 5: Elements $R^{is}$, $R^{ms}$ of the integration and test model

| $S$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $i_1$ | $i_2$ | $i_3$ | $i_4$ | $i_5$ | $i_6$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_1$ | 10% | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_2$ | 10% | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_3$ | 10% | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_4$ | 10% | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_5$ | 10% | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_6$ | 2% | 2% | 0 | 0 | 0 | 0 | 0 | 2% | 0 | 0 | 0 | 0 | 0 |
| $s_7$ | 2% | 2% | 0 | 0 | 0 | 0 | 0 | 2% | 0 | 0 | 0 | 0 | 0 |
| $s_8$ | 0 | 0 | 0 | 5% | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_9$ | 0 | 0 | 0 | 5% | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_{10}$ | 0 | 0 | 0 | 5% | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_{11}$ | 0 | 0 | 2% | 2% | 0 | 0 | 0 | 0 | 0 | 0 | 2% | 0 | 0 |
| $s_{12}$ | 0 | 0 | 2% | 2% | 0 | 0 | 0 | 0 | 0 | 0 | 2% | 0 | 0 |
| $s_{13}$ | 0 | 0 | 0 | 0 | 1% | 1% | 1% | 0 | 0 | 0 | 0 | 1% | 1% |
| $s_{14}$ | 0 | 0 | 0 | 0 | 1% | 1% | 1% | 0 | 0 | 0 | 0 | 1% | 1% |
| $s_{15}$ | 0 | 0 | 0 | 0 | 1% | 1% | 1% | 0 | 0 | 0 | 0 | 1% | 1% |
| $s_{16}$ | 0 | 0 | 0 | 0 | 1% | 1% | 1% | 0 | 0 | 0 | 0 | 1% | 1% |
| $s_{17}$ | 0 | 1% | 0 | 1% | 1% | 1% | 1% | 0 | 0 | 0 | 0 | 1% | 1% |
| $s_{18}$ | 0 | 1% | 0 | 1% | 1% | 1% | 1% | 0 | 0 | 0 | 0 | 1% | 1% |
| $s_{19}$ | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% |
| $s_{20}$ | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% |
| $s_{21}$ | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% |
| $s_{22}$ | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% |
| $s_{23}$ | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% |
| $s_{24}$ | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% | .5% |

$$J^i(G^i) = \max_{m \in M} \Big( C^m(m) + \sum_{i \in G^i_m} C^i(i) + \sum_{(S',T') \in G^{s,t}_m} J^t(S',T') \Big) \quad (3)$$

Where $C^i_m$ is the set of all interfaces in all integration actions of solution $G^i(m)$ and $G^{s,t}_m$ is the set of all test phases of solution $G^i(m)$. The test cost of a test sequence is denoted by:

$$J^t(S',T') = \sum_{S_S \subseteq S'} \sum_{t \in G^t(S_S)} \Big( C^t(t) \prod_{(s,p) \in S_p : s \in S_S} p \prod_{(s',p') \in S_p : s' \in (S' \setminus S_S)} (1 - p') \Big) \quad (4)$$

Where $S_p$ is the set of the fault states and their associated fault probabilities at a certain moment during the integration and test phase and is calculated by equation 15. This fault probability depends on the integrated modules (increase by $R^{sm}$), the created interfaces (increase by $R^{im}$) and the already performed tests (decrease to 0 if tests pass or if the fault state is repaired).

The objective is to find an optimal solution $G^{i\prime}$ that has minimal expected test cost $J^{i\prime}$, from all possible solutions $\mathcal{G}^i$:

$$J^{i\prime} = J^i(G^{i\prime}) = \min_{G^i \in \mathcal{G}^i} J^i(G^i) \quad (5)$$

# 4 Test strategies

A test strategy defines per test phase which fault states are tested. If no fault states are tested, no test phase is started. The simplest strategy is to test all fault states present in a subassembly that can be tested with the possible tests as soon as possible. This 'Test fault states as soon as possible' strategy is often used in quality-driven projects and industries because this strategy keeps the risk of fault states low during integration. However, this strategy may take a lot of test effort, and therefore time, because fault states may be introduced more than once and are therefore also tested more than once. In Figure 2, the strategies described in this section are illustrated. This figure illustrates the system fault probability in time for each strategy for a fictive example. The system fault probability can be seen as a measure of the system quality. The lower the system fault probability, the higher the system quality. The system fault probability is defined as:

$$\mathrm{P}_s(S_p) = 1 - \prod_{p:(s,p) \in S_p} 1 - p \quad (6)$$

The integration of a module in the system increases the fault probability of certain fault states (according to relations $R^{ms}$ and $R^{is}$). As a result, the system fault probability increases. Testing decreases the fault probability of certain fault states because either fault states pass or faults are found and fixed. As a result, the system fault probability decreases. The 'Test fault states as soon as possible' strategy is illustrated by the first graph. The graph shows that after each integration action (increase of risk) a test phase is performed that reduces the system fault probability to zero.

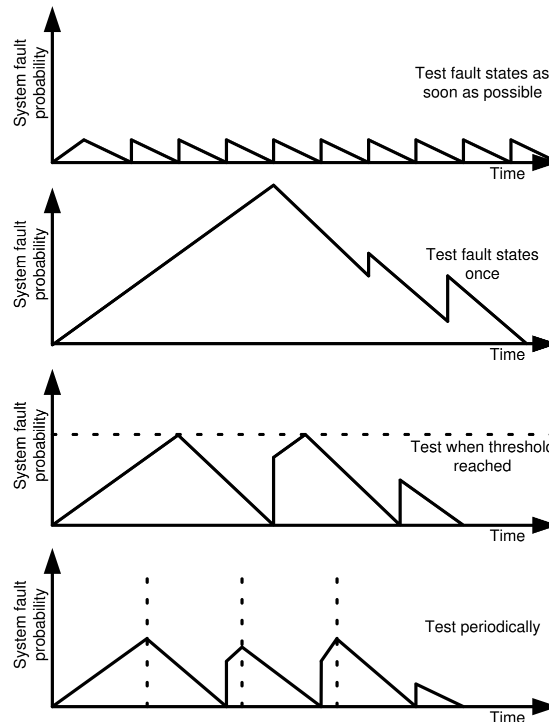



Figure 2: Overview of test strategies (integration causes an increase while testing causes a decrease of the system fault probability)

The second strategy 'Test fault states once' is more efficient in the sense that it only tests the fault states that cannot be introduced anymore when integrating the remaining modules with the current subassembly. A drawback of this strategy is that the system fault probability during the integration phase is higher than during the 'Test fault states as soon as possible' strategy. This strategy is therefore suited for integration problems that in general have low fault probabilities. The strategy is illustrated by the second graph of Figure 2. This graph first shows a large increase of the system fault probability, and then several large test phases that reduce the system fault probability.

The third strategy is the 'Test when threshold reached' strategy which tries to control the quality of the system while reducing the total test effort by looking at the system fault probability. If the system fault probability is higher than a certain user-defined threshold all possible fault states that can be tested are tested, otherwise no testing is done. This strategy can be profitable for time-driven projects or industries that accept some risk during system integration. This strategy is illustrated by the third graph of Figure 2. The graph shows that testing starts when the system fault probability has reached a certain threshold and ends when all fault states have a probability of zero and thus the system fault probability reaches zero.

The fourth strategy is a 'Test periodically' strategy which is often used to probe the quality under development on a periodic basis. According to this strategy, test phases are started once every period. This strategy is illustrated by the fourth graph of Figure 2. The graph shows that a test phase starts after a certain time interval.

In the following subsections it is explained how the tested fault states $S'$ and used tests $T'$ are

determined for each test strategy.

## 4.1 Test fault states as soon as possible

The 'Test fault as soon as possible' strategy (strategy A) tests all fault states that can be tested. That is, the fault states from $S_p$ which can be tested by at least one test that may be performed. A test may be performed if at least one essential subassembly of this test is a subset of the current subassembly $M'$. The fault states ($S'_A$) that are tested during a test phase, given the current set of fault states $S_p$ and the current subassembly $M'$, are therefore determined by:

$$\begin{aligned} S'_A(S_p, M') = & \\ \{s|(s,p) \in S_p \wedge (\exists (t, M'') & : t \in T \wedge M'' \subseteq M' \\ & : s \in R^{ts}(t) \wedge M'' \in R^{tm}(t)) \\ \} & \end{aligned} \tag{7}$$

Furthermore, $P'_A$ denotes the fault probabilities of the tested fault states $S'_A$ and is defined as:

$$P'_A = (s,p) \in S_p \text{ for } s \in S'_A \tag{8}$$

This element is needed in the remainder of the paper. The tests that can be used during a test phase of a subassembly consisting of modules $M' \subseteq M$ are:

$$\begin{aligned} T'_A = \quad & \{t|t \in T \\ & : (\exists M'' : M'' \in R_{tm}(t) : M'' \subseteq M') \\ & \wedge R^{ts}(t) \cap S'_A \;/\!= \emptyset \\ & \} \end{aligned} \tag{9}$$

## 4.2 Test fault states once

The 'Test fault states once' strategy (strategy O) tests certain fault states when they cannot be introduced anymore by modules that have not been integrated yet. That is, all modules and interfaces that can introduce a certain fault state should be integrated in the subsystem before that fault state is tested. A test may be performed if at least one essential subassembly of this test is a subset of the current subassembly $M'$. A small example of this strategy is the following. Suppose two modules $m_1$ and $m_2$ each may introduce fault state $s_1$ with a probability of 10% and can be integrated with each other. Furthermore, test $t_1$ may be executed when only $m_1$ is integrated and covers fault state $s_1$, while test $t_2$ may be performed when only $m_2$ is integrated and also covers fault state $s_1$. The previous strategy ('Test fault states as soon as possible') would create two parallel test phases each testing the same fault state $s_1$ before the two modules are integrated with each other. This strategy would create one test phase after the two modules have been integrated with each other. This test phase would test fault state $s_1$ only once using $t_1$ or $t_2$.

The fault states that are tested during a test phase, given the current set of fault states $S_p$ and the current subassembly $M'$, are therefore determined by:

$$
\begin{array}{lll}
S'_O(S_p, M') = & & \\
\{ & s|(s,p) \in S_p & \\
: & s \notin (\cup m' : m' \in M \setminus M' : R^{ms}(m')) & \\
\wedge & s \notin (\cup i' : i' \in I \wedge R^{im}(i') \in (M \setminus M') : R^{is}(i')) & \\
\wedge & \exists(t, M'') \quad : t \in T \wedge M'' \subseteq M' & \\
 & \qquad\qquad\quad : s \in R^{ts}(t) \wedge M'' \in R^{tm}(t) & \\
\} & &
\end{array} \tag{10}
$$

Furthermore $P'_O$ can be calculated using equation 8 only now using $S'_O$ instead of $S'_A$. $T'_O$ can be calculated using equation 9 only now using $S'_O$ instead of $S'_A$. These elements are needed in the remainder of the paper.

### 4.3 Test when threshold reached

The 'Test when threshold reached' strategy (strategy T) tests all the fault states that are present when the total system fault probability reaches a certain user-define value $a$. This check is performed after each integration action. The tests that can be used during the test phase should test the fault states that are chosen and should be possible to execute.

The fault states that are tested during a test phase, given the current set of fault states $S_p$ and the current subassembly $M'$, are therefore determined by:

$$
S'_T(S_p, M') = \begin{cases} S'_A(S_p, M') & \text{if } \mathrm{P}_s(S_p) > b \\ \emptyset & \text{else} \end{cases} \tag{11}
$$

Where, $\mathrm{P}_s(S_p)$ is the current system fault probability and is determined using equation 6. Furthermore $P'_T$ is calculated using equation 8 only now using $S'_T$ instead of $S'_A$. $T'_T$ can be calculated using equation 9 only now using $S'_T$ instead of $S'_A$. These elements are needed in the remainder of the paper.

### 4.4 Test periodically

The 'Test periodically' strategy (strategy P) tests the fault states that are present when the start of the last test phase is at least one user-defined period $a$ ago. This check is performed after each integration action.

The fault states that are tested during a test phase, given the current set of fault states $S_p$, the current subassembly $M'$, a user-defined period $b$ and the time passed since the last test phase started $d$, are therefore determined by:

$$
S'_P(S_p, M') = \begin{cases} S'_A(S_p, M') & \text{if } d \geq b \\ \emptyset & \text{else} \end{cases} \tag{12}
$$

Where $P'_P$ can be calculated using equation 8 only now using $S'_P$ instead of $S'_A$. $T'_P$ can be calculated using equation 9 only now using $S'_P$ instead of $S'_A$. These elements are needed in the remainder of the paper.

Besides these mentioned strategies, many other strategies can be thought of, for example a combination of periodic and threshold. This and several other strategies will be investigated in our future work.

# 5 Solving algorithm

In this section, we propose a solving algorithm for the integration and test sequencing problem. This algorithm is based on the 'assembly by disassembly' approach used by de Mello *et al.* [3] and suggested by Delchambre *et al.* [9] for the integration sequencing part and the 'sequential diagnosis approach' suggested by Pattipati *et al.* [10] for the test sequencing part. The algorithm used by the 'assembly by disassembly' approach has been extended towards an integration sequencing algorithm in our previous work [2], while the algorithm used by the sequential diagnosis approach has been extended in [6, 7] to a test sequencing algorithm.

Both the test sequencing and the integration sequencing algorithms are AND/OR graph searches. The integration sequencing algorithm starts with the completely integrated system and constructs an AND/OR graph that denotes all possible sequences to disassemble the system into single modules. An OR node denotes the system state $x^i \in X^i$, where $X^i = \mathcal{P}(M)$, which consists of the set of integrated modules. An AND node denotes a possible disassembly action (breaking a set of interfaces) on a certain system state and results in two new OR nodes which denote the two subassemblies that remain after the disassembly action.

An example of an integration AND/OR graph is shown in Figure 3. Each square node in the graph denotes an OR node, while each hexagonal node denotes an AND node, the edges denote the search direction. This AND/OR graph is constructed for a very simple integration model consisting of 3 modules $(m_1, m_2, m_3)$, which are all connected to each other with 3 interfaces ($i_1$ connects $m_1$ and $m_2$, $i_2$ connects $m_1$ and $m_3$, $i_3$ connects $m_2$ and $m_3$) and 4 tests that need to be applied: 3 tests that each need one of the modules ($t_1$ requires $m_1$, $t_2$ requires $m_2$, $t_3$ requires $m_3$) and one system test $t_4$ that requires all modules. The AND/OR graph in Figure 3 shows all possibilities in which the system can be disassembled, and therefore contains all solutions to the integration sequencing problem. In this example, there are three possible solutions ($G_1^i, G_2^i, G_3^i$) that can be distinguished at the root OR node where there are 3 AND nodes to choose from. For each solution $G^i$ the cost $J^i(G^i)$ can be calculated. Then, the cheapest solution is chosen. The chosen solution is then used in the reversed order: starting with the single modules and ending with the integrated system.

The test sequencing algorithm also constructs an AND/OR graph, but now the OR nodes denote the test system state $x^t$ with domain: $X^t = \mathcal{P}(\mathcal{P}(S))$, which denotes candidate sets of fault states, that is all possible sets of fault states that could be present. The AND nodes represent tests applied to the OR nodes and the leaf node represents the system state where all fault states are fixed or shown not present. An example of an AND/OR graph is shown in Figure 4. Each round node represents an AND node while each square node represents an OR node or a leaf node. This AND/OR graph is constructed for a very simple model consisting of two tests that each cover one fault state: $t_1$ covers $s_1$, $t_2$ covers $s_2$. The AND/OR graph shows all possible test sequences: in this example either the sequence $t_1$, $t_2$ or the sequence $t_2$, $t_1$. For each solution $G^t$ the cost $J^t(G^t)$ can be calculated. Then, the cheapest solution is chosen.

The main difference between the two AND/OR graphs is the order of execution. The integration solution is executed in the opposite order in which the AND/OR graph is constructed. This means that we start with the separate modules and end with the completely integrated

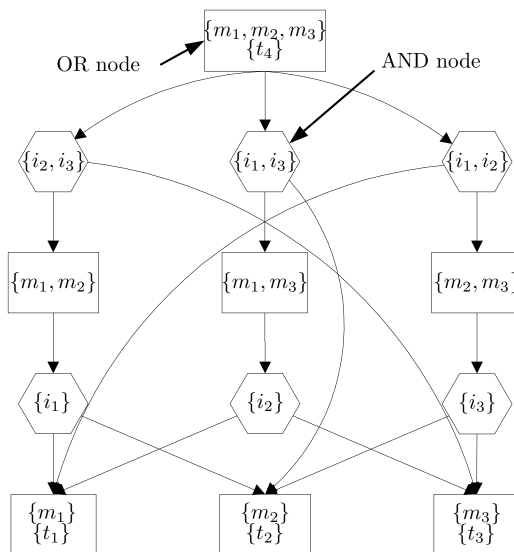



Figure 3: Integration AND/OR graph illustration

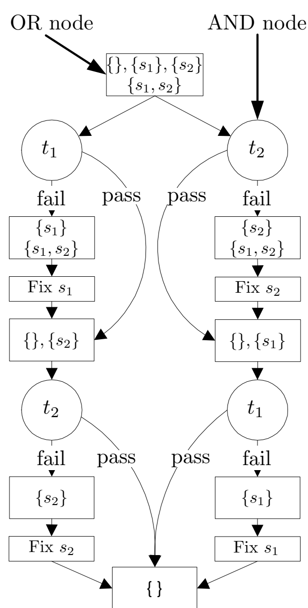



Figure 4: Test AND/OR graph illustration

system. A solution of the test sequencing problem is executed in the same order in which the AND/OR graph is constructed: starting with the root node, ending with the state in which all fault states are found and repaired or not present. This difference makes it impossible to combine the two AND/OR graphs into one and develop an algorithm that constructs a combined AND/OR graph. Therefore, we propose an algorithm that is a combination of the integration and test sequencing algorithms. It constructs one integration AND/OR graph and several test AND/OR graphs. During the integration AND/OR graph search a test AND/OR graph is constructed for each test phase. The contents of these test AND/OR graphs depends on the system state, the chosen test strategy and the available tests.

The test sequencing algorithm as presented in [7] can be used without any changes. The integration AND/OR search algorithm needs to be changed such that it calculates the fault states that should be tested based on the test strategy and the test sequencing algorithm is used to calculate the test cost of a test phase. The search starts with the initial root integration OR node that denotes the completely integrated and tested system: $x^i_{init}$. The cost of this particular OR node is called $J^i_x(x^i_{init})$ and is determined as follows.

The first step is to determine the possible set of disassembly actions. The set of all possible integration actions $A_x$ consists of all cut-sets that separate the integrated system with system state $x^i$ into exactly two unique sub-systems ($x^i_1$ and $x^i_2$). For a given system state $x^i$, this can be determined as follows:

$$
\begin{aligned}
&A_x(x^i) = \\
&\{a | a \subseteq I_x(x^i) : \\
&\quad \exists x^i_1, x^i_2 \\
&\quad\quad : x^i_1 \cap x^i_2 = \emptyset \wedge x^i_1 \cup x^i_2 = x^i \wedge x^i_1 \neq \emptyset \wedge x^i_2 \neq \emptyset \\
&\quad\quad \wedge \forall m', m'' \in x^i_1 \wedge m', m'' \in x^i_2 : conn(m', m'', I_x(x^i) \setminus a) \\
&\quad\quad \wedge \nexists m' \in x^i_1, m'' \in x^i_2 : conn(m', m'', I_x(x^i) \setminus a) \\
&\}
\end{aligned}
\tag{13}
$$

where $I_x(x^i)$ denotes all interfaces between the modules in system state $x^i$ and function *conn* checks whether two modules are connected, i.e. where there exists a path of interfaces between two modules. Cut-set algorithms exist that determine all possible cut-sets of a system in lineair time per cut-set. We use the algorithm as described by Tsukiyama [11]. For the simple system of which the AND/OR graph is shown in Figure 3, the cut-sets are: $\{i_1, i_2\}, \{i_2, i_3\}, \{i_1, i_3\}$ for the initial OR node. For the scanner illustration in Section 2, the cut-sets are: $\{i_1\}, \{i_2\}, \{i_3\}, \{i_4\}, \{i_5\}, \{i_6\}$ for the initial OR node.

The second step is to construct an AND node for every cut-set $a \in A_x(x^i)$ given the system state $x^i$. This AND node represents the disassembly of a system state into two system states $x^i_1$ and $x^i_2$ (determined in equation 13), by breaking the interfaces in $a$.

The total cost of an AND node $J^i_a(x^i, a)$ for a system state $x^i$ and a disassembly action $a \in A_x(x^i)$ which disassembles the system into two subsystems $x^i_1, x^i_2$, is defined as the maximal integration and test cost of each formed system state $x^i_1$ and $x^i_2$, plus the cost of disassembling the system state $x^i$ into the two system states plus the cost of the associated test phase, or:

$$
\begin{aligned}
&J^i_a(x^i, a) = \\
&\quad \max(J^i_x(x^i_1), J^i_x(x^i_2)) + \sum_{i \in a} C^i(i) + J_t(x^i, S_p(x^i, a))
\end{aligned}
\tag{14}
$$

Where $S_p(x^i, a)$ is a tuple of fault states and their belonging fault probabilities and is calcu-

lated by:

$$\begin{array}{lll} S_p(x^i, a) = & & \\ \quad Unify & ( & \{(s,p) \in S_p(x^i_1, a_1) | s \notin S'_1\} \\ & \cup & \{(s,p) \in S_p(x^i_2, a_2) | s \notin S'_2\} \\ & \cup & \{\cup R^{is}(i) | i \in a\} \\ & ) & \end{array} \tag{15}$$

Where $a_1, a_2$ are the cut-sets that had minimal cost at respectively system states $x_1$ and $x_2$. $S'_1, S'_2$ are the sets of fault states that are tested during the test phases associated to the AND nodes of $a_1$ and $a_2$. Furthermore, function $Unify$ makes all fault states in $S_p$ unique by combining fault states that are multiple times present in $S_p$ into one fault state with corresponding fault probability. This function is defined by:

$$\begin{array}{l} Unify(S_p) = \\ \{(s, 1 - \prod_{(s,p') \in S_p} (1 - p')) | (s,p) \in S_p\} \end{array} \tag{16}$$

The last step is to determine the cost of the OR node. The cost of the OR node is the development cost of a module plus the cost of the required test phase if one module remains, or the minimal cost of each AND node that is constructed:

$$J^i_x(x^i) = \begin{cases} C^m(m) + J_t(x^i, R^{ms}(m)) & \text{if } x = \{m\} \\ \min_{a \in A_x(x^i)} \left(J^i_a(x^i, a)\right) & \text{else} \end{cases} \tag{17}$$

The cost of a test phase $J_t(x^i, S_p)$, where $S_p$ is either $S_p(x^i, a)$ for an AND node or $R^{ms}(m)$ for a leaf node, depends on the chosen strategy $w$, where $w$ is either $A$ for strategy A, $O$ for strategy O, $T$ for strategy T, or $P$ for strategy P. Each strategy has its associated set of fault states ($S'_w$), the fault probabilities of these fault states ($P'_w$), and the set of tests ($T'_w$) that can be used (as explained in the previous section). The cost of a test phase is then defined as:

$$\begin{array}{l} J_t(x^i, S_p) = \\ \begin{cases} J^t_o(\mathcal{P}(S'_A), T'_A, P'_A) & \text{if } w = \text{‘A’} \\ J^t_o(\mathcal{P}(S'_O), T'_O, P'_O) & \text{if } w = \text{‘O’} \\ J^t_o(\mathcal{P}(S'_T), T'_T, P'_T) & \text{if } w = \text{‘T’} \\ J^t_o(\mathcal{P}(S'_P), T'_P, P'_P) & \text{if } w = \text{‘P’} \end{cases} \end{array} \tag{18}$$

The cost of a test phase is the cost of the initial test OR node, with the initial system state $x^t_{init} = \mathcal{P}(S'_w)$. A system state $x^t$ indicates all possible combinations of fault states that could be present. The cost of an OR node given the system state $x^t$, the set of tests $T'$ that can be performed, the fault probabilities of all individual fault states $P'$, and their properties and relations $C^t$, $R^{ts}$, $R^{ss}$, $R^{st}$, is determined by the following function:

$$\begin{array}{l} J^t_o(x^t, T', P') = \\ \begin{cases} 0.0 & \text{if } x^t = \emptyset \\ J^t_o(x^t_d, T', P') & \text{if } x^t \;/= \emptyset \wedge T'' = \emptyset \\ \min_{t \in T''}(J^t_a(t, x^t_t) + C^t(t)) & \text{else} \end{cases} \end{array} \tag{19}$$

Where:

- $x_t^t$ is the fixed system state. This is the system state without the fault states that are definitely present.

- $x_d^t$ is the diagnosed system state. This is the system state without the fault states that are diagnosed. These are the fault states that could be present because they are covered by tests that failed.

- $T'' \subseteq T'$ is the set of tests that are useful to apply. These are the tests that may either give a pass outcome or a fail outcome.

Furthermore, $J_a^t(t, x^t)$ denotes the cost of an AND node which is determined by the cost of the two succeeding OR nodes (pass an fail) and the probabilities that these OR nodes are reached, that is:

$$J_a^t(t, x^t) = p_p \cdot J_o^t(x_p^t, T', P') + p_f \cdot J_o^t(x_f^t, t', P') \quad (20)$$

Where:

- $p_p$ is the pass and $p_f$ the fail probability of test $t$.

- $x_p^t$ is the pass and $x_f^t$ is the fail OR node after applying test $t$ on system state $x^t$.

The complete functional description of the algorithm is shown in Appendix 7. If the root node is solved, which means that $J_x^i(x_{init}^i)$ is known, the complete solution is known and can be constructed. Then, the integration tree is the reverse sequence of the disassembly tree, i.e. starting with the separate modules and ending with the integrated system. For each test phase, the corresponding test sequence can also be constructed.

## 5.1 Illustration

For the illustration introduced in Section 2, we calculated the optimal solution for the four introduced strategies. The results of this experiment are shown in Table 6. The total integration and test sequence is optimized towards minimal duration of the integration and test sequence, but also the total test time, which is the sum of all test times, is important since this number reflects the total costs that are made for testing.

For this problem, both the 'Test fault states as soon as possible' and the 'Test fault states once' strategies give the best solution in terms of minimal duration. However, the total test time for both strategies is the highest, whereas the total test time for the periodic strategy is the lowest. Therefore, a choice must be made regarding what is more important: the total test time or the minimal duration .

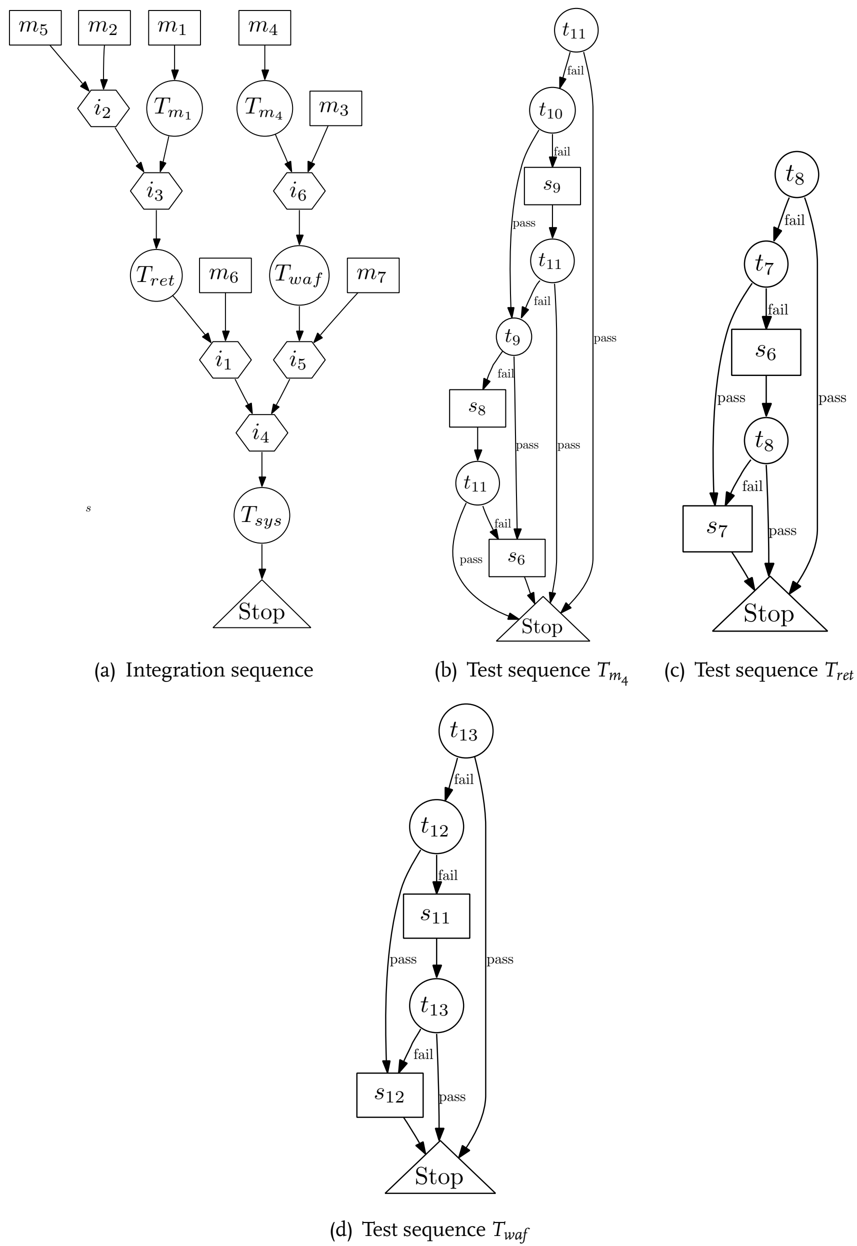
For illustration, we show the solutions of the 'as soon as possible' strategy. The integration sequence of this solution is shown in Figure 5(a), while test phases $T_{m_4}$, $T_{ret}$ and $T_{waf}$ are shown in Figures 5(b), 5(c) and 5(d). Test phase $T_{m_1}$ is the same as shown in Figure 1(b). Test phase $T_{sys}$ is not shown in this paper, since it is too large. Furthermore, the total integration and test sequence is shown in Figure 6 as a Microsoft Project Gantt chart. In this chart the durations and sequence of actions are shown.

Table 6: Results illustration

| Strategy | $J^{i*}$ | $\sum J^t$ |
|---|---|---|
| A | 36.68 | 20.97 |
| O | 36.68 | 20.97 |
| T (45%) | 38.79 | 19.38 |
| P (40 hours) | 39.23 | 18.82 |

(100%)

(100%)




(a) Integration sequence

(b) Test sequence $T_{m_4}$

(c) Test sequence $T_{ret}$

(d) Test sequence $T_{waf}$

Figure 5: Integration and test sequences for the scanner illustration

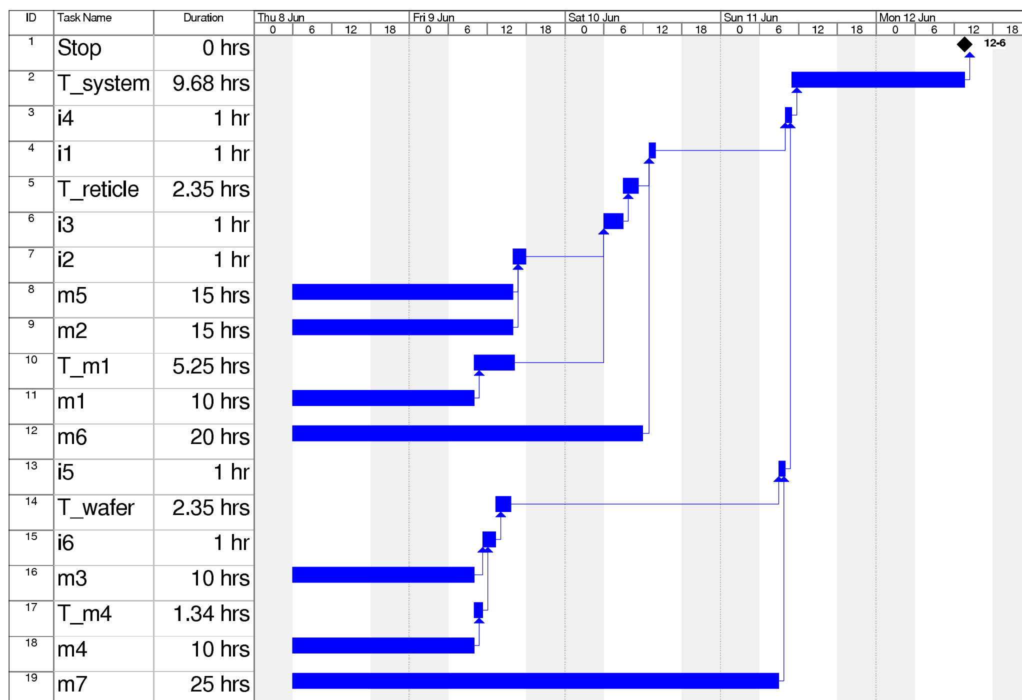



Figure 6: Integration and test sequence represented as a Microsoft Project Gantt chart

# 6 Case study

The presented method can be used to optimize integration and test sequences for several problems. We performed a case study during the development of a new software release of an ASML lithographic machine. During the development of such a software release, components are updated in parallel and integrated in the so-called qualified baseline (QBL) that consists of a complete software release. This QBL is tested every week for one day. Furthermore, when all changes of a specific software release are developed, the QBL is tested completely and then released. After this final test phase, the software release is installed on lithographic systems at customers. ASML currently uses a periodic strategy to define when to start testing. In this case study, we investigated whether this strategy is optimal for time. To do so, we modeled one specific software release. The properties of the integration and test model are shown in Table 7(a).

The model is constructed as follows. Every developed change to the software is modeled as one module. Also, the original software release is modeled as one module. Furthermore, every (sub)-requirement to the system is modeled as a fault state. A test set is available to test the different requirements. Then, we estimate for every developed change of the software (module), the probability that a certain (sub)-requirement is not met anymore. Using this model, we can investigate which test strategy provides the integration and test plan with minimal duration. In Table 7(b), the solutions are shown for the best strategies that are investigated. The total integration and test sequence is optimized towards minimal duration (time-to-market), but also the total test time is important since this number reflects the total costs that are made for testing.

We can conclude based on these experiments, that the 'Test when threshold reached' strategy using a system fault probability threshold of 10% results in the lowest time-to-market.

Table 7: Case study

(a) Case study properties

| property | value |
|---|---|
| $\lvert M \rvert$ | 260 |
| $\lvert I \rvert$ | 259 |
| $\lvert T \rvert$ | 169 |
| $\lvert S \rvert$ | 55 |
| $\overline{C^t}$ | 0.25 hours |

(b) Case study results

| Strategy | $J^i$ | $\sum J^t$ |
|---|---|---|
| P (40 hours) | 2617 | 2769 |
| A | 2618 | 2792 |
| O | 2626 | 2593 |
| T (10%) | 2566 | 3800 |
| T (90%-1%) | 2585 | 2719 |

However, the total test cost shows a large increase when using this strategy. Therefore, the threshold is made time-dependant. This strategy starts with 90% and ends with 1% system fault probability and shows a decrease of both time-to-market and total test time.

# 7 Conclusions

In this paper, we introduced a method to create integration and test strategies. The method is based on an existing integration sequencing algorithm and an existing test sequencing algorithm that are combined into one algorithm. The input of this algorithm is an integration and test model describing the modules to be integrated, the interfaces between the modules, the possible tests and the possible fault states. Furthermore, the model describes certain properties of these elements such as development and execution times and the relation between these elements. Besides this model, the method requires a strategy that defines when a test phase is started and which fault states are tested when. In this paper, we introduced four possible strategies: 'Test fault states as soon as possible', 'Test fault states once', 'Test when threshold reached' and 'Test periodically'. With the method, it is possible to calculate the optimal integration and test sequence for a given strategy. In this setting, optimal relates to the time-to-market of a system, which is determined by the critical path in the integration sequence. The case study within a lithographic software release shows that it is possible to solve real life problems with this method. By comparing the optimal test and integration sequences for different strategies we were able to determine the best strategy for ASML software releases.

# Acknowledgment

The authors would like to thank Mathijs Dohmen from the Eindhoven University of Technology and Jan Wegter from ASML for their help in performing the case study.

# Algorithm

This appendix gives a formal based on recursion, functional style [12] description of the algorithm that is used to solve the integration and test sequencing problem. This algorithm calculates the optimal cost $J^{i*}$ of the optimal integration and test sequence for an integration model $(M, I, T, S, C^m, C^i, C^t, R^{im}, R^{tm}, R^{ts}, R^{is}, R^{ms})$. To this end, the following expression can

be used:

$$(J^{i\prime}, H^i) = OR_i(x^i_{init}, A_x(x^i_{init}), H^i_{init}) \tag{21}$$

where $H^i_{init} : \mathcal{P}(M) \times \mathcal{P}(S \times \mathbb{R}^+) \to \{\perp\}$ is the initial function that gives the cost of a solved OR node. An OR node is denoted by the integrated modules and the present fault states with corresponding fault state probabilities. $A_x(x^i_{init})$ are all cut-sets of the initial system state $x^i_{init}$, calculated using the algorithm presented in [11]. Furthermore, $x^i_{init} = M$ thus the initial state is the complete integrated system. We only calculate the cut-sets for the initial system state. The cut-sets that are needed for the other system states (that are formed by disassembling the initial state) can be obtained by taking the initial set of cut-sets and then remove the cut-sets that do not split that system state into exactly two sub-assemblies. This prevents calculating the cut-sets for every system state which reduces computation time.

The resulting $H^i$ can be used to construct the optimal integration sequence $G^i$. This calculation gives the cost $J^{i*}$ of the optimal solution according to equation 5. For each integration OR node, all cut-sets are considered for the integration sequence except for the cut-sets that do not split the system into exactly two sub-systems, which is not allowed. The best cut-set per OR node is chosen based on the minimal integration and test cost per cut-set, starting from the last OR node. If more cut-sets have the same minimal cost, one of them is chosen.

The function $OR_i : X^i \times \mathcal{P}(\mathcal{P}(I)) \times \mathcal{H}^i \to \mathbb{R}^+ \times \mathcal{P}(S \times \mathbb{R}^+) \times \mathcal{H}^i$ calculates the cost of a system state $x^i$ given the possible disassembly actions and the initial $H^i$. The cost of such an integration OR node is defined by Equation 17. The $OR_i$ function is defined as follows. If the state has already been solved (i.e. if $H^i(x^i) \neq \perp$), the solution of that solved state is taken from $H^i$ and returned. Otherwise, several options are possible. If $x^i$ consists of one module, the node is a leaf node and the resulting cost are the module development cost and the remaining test cost. If $x^i$ consists of multiple modules, one or more cut-sets are available which are evaluated and compared with each other with the integration $AND_i$ function. The functional description of this function is:

$$OR_i(x^i, A_x, H^i) = \begin{cases} (H^i(x^i), H^i) & \text{if } H^i(x^i) \neq \perp \\ (J', S'_p, H^i(x^i/(J^{i'}, \{\}))) & \text{if } H^i(x^i) = \perp \wedge x^i = \{m'\} \\ (J'', S''_p, H^{i''}(x^i/(J^{i''}, a''))) & \text{if } H^i(x^i) = \perp \wedge x^i \neq \{m'\} \end{cases} \tag{22}$$

where:

- $J' = C^m(m') + J_t(x^i, R^{ms}(m')).0$, where .0 denotes the first element of a tuple, and .1 the second.
- $S'_p = \{(s, p) \in R^{ms}(m') | s \notin J_t(x^i, R^{ms}(m')).1\}$
- $J'' = J_a(a'') = \min_{a \in A_x} J_a(a)$, is the minimal cost of $x^i$, and $a''$ is the cut-set from $A_x$ for which this holds.
- $(J_a(a_j), H^i_j) =$
  $AND_i(x^i, rmv(A_x, a_j), a_j, H^i_{j-1}).0 + J_t(x^i, S_p(x^i, a_j)).0$
  for $j = 1, \ldots, |A_x|$ (where $H^i_0 = H^i$), are the minimal test cost and updated $H^i$ for each cut-set in $A_x$. Furthermore, $H^{i''} = H^i_{|A_x|}$. In this equation $S_p$ can be determined as follows: $S_p(x^i, a) = AND_i(x^i, rmv(A_x, a), a, H^i).1$

- $S''_p =$
$\{(s,p)|(s,p) \in AND_i(x^i, rmv(A_x, a''), a'', H^i).1$
$\wedge s \notin J_t(x^i, S_p(x^i, a'')).1\}$, is the resulting set of fault states that are still present with their fault probabilities.

- Function $rmv$ removes all interfaces in the cut-set $a$ from all cut-sets in $A_x$, while ensuring that the resulting cut-sets, still split the new system state into exactly two subsystems, and thus results in the new set of cut-sets ensuring that still satisfies Equation 13.

The function $AND_i : X^i \times \mathcal{P}(I) \times \mathcal{P}(\mathcal{P}(I)) \times \mathcal{H} \rightarrow \mathbb{R}^+ \times \mathcal{P}(S \times \mathbb{R}^+) \times \mathcal{H}$ takes a system state $x^i$ as input and applies the disassembly action $a$ to that system. It returns the cost made by that disassembly action and the path cost of the resulting subsystems. Also, the remaining fault state set $S_p$ is returned.

$$AND_i(M, a, A_x, H^i) = \\ (\sum_{i \in a} C^i(i) + max(J', J''), S_p, H^{i''}) \tag{23}$$

Where:

- $x^i_1$ and $x^i_2$ are defined as the new system states resulting from applying cut-set $a$ on system state $x^i$.
- $(J', S'_p, H^{i'}) = OR_i(x^i_1, A_x, H^i)$, calculates the cost of system state $x^i_1$ and the updated $H^{i'}$.
- $(J'', S''_p, H^{i''}) = OR_i(x^i_2, A_x, H^{i'})$, calculates the cost of system state $x^i_2$ and the updated $H^{i''}$.
- $S_p = S'_p \cup S''_p \cup \sum_{i \in a} R^{is}(i)$ is the updated fault state set which is a combination of the two fault states sets from the two assemblies and the fault state set introduced by the creation of the interfaces.

The above $AND_i$ and $OR_i$ functions are for constructing the integration sequence, the following functions are for constructing the test sequences. We start with the function $J_t$ that is used in the integration sequencing algorithm.

Let function $J_t : X^i \times \mathcal{P}(S \times \mathbb{R}^+) \times \mathcal{P}(I) \rightarrow \mathbb{R}^+ \times \mathcal{P}(S)$ be a function that calculates the test cost of a test phase and the removed fault states during that test phase for a test strategy $w$ and given the system state $x^i$, the current present fault states $S_P$ and the integration action $a$:

$$J_t(x^i, S_p, a) = \\ \begin{cases} (OR_t(S'_A, T'_A, P'_A, x^t_{init}, H^t_{init}, T^p_{init}).0, S'_A) & \text{if } w = \text{'A'} \\ (OR_t(S'_O, T'_O, P'_O, x^t_{init}, H^t_{init}, T^p_{init}).0, S'_O) & \text{if } w = \text{'O'} \\ (OR_t(S'_T, T'_T, P'_T, x^t_{init}, H^t_{init}, T^p_{init}).0, S'_T) & \text{if } w = \text{'T'} \\ (OR_t(S'_P, T'_P, P'_P, x^t_{init}, H^t_{init}, T^p_{init}).0, S'_P) & \text{if } w = \text{'P'} \end{cases} \tag{24}$$

Where:

- Function $OR_t$ returns the cost of a test phase. This is done by constructing a test AND/OR graph using two functions $OR_t$ and $AND_t$ that are both described formally in [6].
- $S'_A$, $S'_O$, $S'_T$, $S'_P$, $T'_A$, $T'_O$, $T'_T$, $T'_P$, $P'_A$, $P'_O$, $P'_T$ and $P'_P$ are defined in section 4.
- $H^t_{init} : X^t \to \{\perp\}$ is the initial function that gives the cost of a solved OR node.
- $x^t_{init} = (\emptyset, \emptyset)$ denotes the initial test state.
- $T^p_{init} = \emptyset$ denotes the initial performed test set.

# Bibliography

[1] ASML, 2006, http://www.asml.com. [Online]. Available: http://www.asml.com

[2] R. Boumen, I. S. M. de Jong, J. M. van de Mortel-Fronczak, and J. E. Rooda, “Integration sequencing in complex manufacturing systems,” Eindhoven University of Technology, Den Dolech 2 5600 MB Eindhoven the Netherlands, SE-report ISSN: 1872-1567 2006-02, october 2006. [Online]. Available: seweb.se.wtb.tue.nl/sereports

[3] L. S. H. de Mello and A. C. Sanderson, “Representations of mechanical assembly sequences,” *IEEE transactions on Robotics and Automation*, vol. 7, no. 2, pp. 211–227, April 1991.

[4] ——, “A correct and complete algorithm for the generation of mechanical assembly sequences,” *IEEE transactions on Robotics and Automation*, vol. 7, no. 2, pp. 228–240, April 1991.

[5] V. L. Hanh, K. Akif, Y. L. Traon, and J.-M. Jézéquel, “Selecting an efficient oo integration testing strategy: An experimental comparison of actual strategies,” *Proceedings of ECOOP 2001*, pp. 381–401, 2001.

[6] R. Boumen, I. S. M. de Jong, J. W. H. Vermunt, J. M. van de Mortel-Fronczak, and J. E. Rooda, “Test sequencing in complex manufacturing systems,” *Accepted for IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 2006.

[7] ——, “A risk-based stopping criterion for test sequencing,” Eindhoven University of Technology, Internal Report SE 420460, January 2006, submitted to IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans.

[8] K. R. Pattipati, S. Deb, R. W. Dontamsetty, and A. Maitra, “Start: System testability analysis and research tool,” *IEEE Aerosp. Electron. Syst. Mag.*, vol. 6, pp. 13–20, January 1991.

[9] A. W. A. Delchambre and P. Gaspart, “Knowledge based assembly by disassembly planning,” *Proceedings of the International Conference on Expert Systems in Engineering Applications*, October 1989.

[10] K. R. Pattipati and M. G. Alexandridis, “Application of heuristic search and information theory to sequential diagnosis,” *IEEE Trans. Syst. Man, Cybern.*, vol. 20, pp. 872–887, July/August 1990.

[11] S. Tsukiyama, I. Shirakawa, and H. Ozaki, “An algorithm to enumerate all cutsets of a graph in linear time per cutset,” *Journal of the Association for Computing Machinery*, vol. 27, no. 4, pp. 619–632, October 1980.

[12] R. Bird, *Introduction to Functional Programming using Haskell*, 2nd ed. Prentice Hall Press, 1998.

[13] D. B. Grunberg, J. L. Weiss, and J. C. Deckert, “Generation of optimal and suboptimal strategies for multiple fault isolation, Tech. rep. TM-248,” 1987.

# Biography

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 concerns test strategy within the TANGRAM project.

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 PhD student at the Eindhoven University of Technology.

J.M.G. Mestrom received his M.Sc. degree in Mechanical Engineering from the Eindhoven University of Technology, the Netherlands, in 2006. During his work as a master student he worked in the field of integration and testing of complex manufacturing systems. His research within the TANGRAM project concerned strategies and algorithms for integration and test sequencing.

J.M. van de Mortel-Fronczak graduated in computer science at the AGH University of Science and Technology of Cracow, Poland, in 1982. In 1993, she received the Ph.D. degree in computer science from the Eindhoven University of Technology, the Netherlands. Since 1997 she works as an 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 control of manufacturing lines and in supervisory control of manufacturing machines.