Integration and test plans for complex manufacturing systems
Boumen, R.

DOI:
10.6100/IR628120

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 author's 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

Citation for published version (APA):

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?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
INTEGRATION AND TEST PLANS FOR COMPLEX MANUFACTURING SYSTEMS

Roel Boumen
Voorkant: De voorkant van dit proefschrift toont een legpuzzel die gebruikt wordt als analogie voor het integratie- en testplanningsprobleem. In deze analogie wordt het leggen van de verschillende puzzelstukjes, gezien als het integreren van een systeem. Op de achterkant van dit proefschrift wordt deze analogie uitgelegd voor ieder afzonderlijk hoofdstuk. De foto op de voorkant laat een deel van een lithografische machine van ASML zien. Deze systemen zijn gebruikt als voorbeeldstudies voor de gepresenteerde methodieken.

Cover: The cover of this thesis shows a puzzle that is used as analogy of the integration and test planning problem. In this analogy, laying the different puzzle pieces is seen as the integration of a system. On the back of this thesis, we explain for each individual chapter the related analogy. The photograph on the cover shows a part of a lithographic machine of ASML. These systems have been used as case studies for the methods presented.

Cover photo: © Copyright 2004, ASML
Cover design: R. Boumen

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

IPA dissertation series 2007-12

© Copyright 2007, R. Boumen
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission from the copyright owner.

A catalogue record is available from the Eindhoven University of Technology Library

Reproduction: Universiteitsdrukkerij Technische Universiteit Eindhoven

The work described in this thesis has been carried out at ASML Veldhoven and the Eindhoven University of Technology, both in the Netherlands.
INTEGRATION AND TEST PLANS FOR COMPLEX MANUFACTURING SYSTEMS

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op maandag 20 augustus 2007 om 16.00 uur

door

Roel Boumen

geboren te Maasbree
Dit proefschrift is goedgekeurd door de promotor:
prof.dr.ir. J.E. Rooda

Copromotor:
dr.ir. J.M. van de Mortel-Fronczak
This thesis is the final result of my Ph.D. project and is titled ‘Integration and test plans for complex manufacturing systems’. It is a collection of my research papers together with some practical extensions. Some papers have been stripped to avoid duplication. I related the different sections in each chapter with an introduction and conclusion. The case studies are collected in the last chapter to accommodate readers that are merely interested in the applicability of the methods presented. I hope that readers may use this thesis as a guideline, reference, or handbook to solve their own integration and test planning problems.

My Ph.D. project has been performed as part of the Tangram research project. Tangram is a cooperation between ASML, the Embedded Systems Institute, the Eindhoven University of Technology and several other partners, and is partially supported by the Netherlands Ministry of Economic Affairs. Finding a balance between the academic research and the real-life industrial problems was one of the many interesting challenges that this project has offered me.

In the remainder of this preface I would like to thank all the people that contributed to this Ph.D. project. First of all, I would like to thank professor Koos Rooda for his supervision and for keeping me enthusiastic which resulted in this thesis in only three-and-a-half years. I would also like to thank my coach Asia van de Mortel-Fronczak for teaching me everything I know about academic writing and for co-authoring all articles in this thesis. Furthermore, I would like to thank the members of the reading committee, professor Jos Baeten, professor Arjan van Gemund and professor Krishna Pattipati, for their valuable comments. Special thanks to professor Krishna Pattipati for letting me be part of his research team for three months at the University of Connecticut (USA).

Being part of the Tangram project did have the advantage of being part of a great team. Therefore, I would like to thank all Tangram members. Special thanks to Ivo de Jong: together we hopefully contributed to many improvements to the integration and testing at ASML, and to Niels Braspenning who was always willing to listen and discuss the problems we tried to solve. Furthermore, I would like to thank Joris Vermunt and Jeroen Mestrom for their help in implementing the algorithms and co-authoring certain papers. Of course, I would like to thank all other students that were part of ‘LoAr’ for their help in performing case studies and developing prototype tooling.

Without the challenging industrial problems provided by ASML, this thesis would have been a purely academic exercise. Therefore, I would like to thank some members of the ASML staff for their contributions: Luud Engels, Tom Brugman and Tammo van den Berg.
Furthermore, I thank Marco Kunowski, Jan Wegter and many others for their help with the case studies and for actually using our methods. Special thanks to Barend van den Nieuwelaar for making me enthusiastic about performing my Ph.D. project at ASML.

Thanks to all my colleagues at the Systems Engineering Group of the Eindhoven University of Technology, and Mieke Lousberg for her friendly help and care.

I would like to thank my family, relatives and friends for their support. And finally, I thank Lizet who always supported me in my work, even when I spent three months in the USA without her. Lizet, thank you for loving me through all these years.

Venlo, April 2007
The integration and test phases that are part of the development and manufacturing of complex manufacturing systems are costly and time consuming. As time-to-market is becoming increasingly important, it is crucial to keep these phases as short as possible, while maintaining system quality. This is especially true for the time-to-market driven semiconductor industry and for companies providing manufacturing systems to this industry such as ASML, a provider of lithographic systems. The Tangram research project has the goal, to shorten integration and test time by a model-based integration and test approach. The Ph.D. project described in this thesis is part of the Tangram project.

To achieve integration and test time reduction, we developed three methods that each solve one of the following three integration and test problems:

- Construction of an optimal test plan with respect to time, cost and/or quality.
- Construction of an optimal integration plan with respect to time, cost and/or quality.
- Construction of an optimal integration and test plan with respect to time, cost and/or quality.

The test plan optimization method consists of two steps. The first step is the definition of a model of the test problem. This model consists of tests that can be performed with associated cost and duration, possible faults that can reside in the system with associated fault probability and impact (importance), and the relation between the tests and the possible faults, also denoted as the test coverage for each possible fault. The second step consists of calculating the optimal test plan based on this test model given an objective function and possible constraints on time, cost and/or risk, which is a parameter for the quality of the system. By constructing an AND/OR graph of the problem, where AND nodes denote tests and OR nodes denote system states represented by the ambiguous faults, all possible test sequences of this problem are obtained. An algorithm selects the best solution from this AND/OR graph. This solution is a set of test sequences, where the test sequence that is followed depends on the outcome (pass/fail) of the previous tests.

The integration plan optimization method consists of the same two steps as the test plan optimization method. The integration model consists of modules with their development times, interfaces that denote which modules can be integrated with each other, and test phases with their durations. Furthermore, the model consists of the relation between test phases and modules indicating which modules should be integrated before the test phase may start. Also for this problem, an AND/OR graph is constructed. The AND nodes denote integration actions and the OR nodes denote system states represented by the modules that are integrated. An algorithm selects the optimal solution from this AND/OR graph. The optimal solution has the shortest possible integration time. The solution is a tree of integration
actions and test phases indicating, for each module, the sequence of integration actions and test phases.

The integration and test planning method is a combination of the two previously mentioned methods and also consists of two steps. The integration and test model is a combination of the test model and the integration model, with additional relations between modules and possible faults describing in which modules these possible faults are inserted. During the construction of the integration AND/OR graph, a test AND/OR graph is constructed for each integration AND node. This test AND/OR graph represents the test phase that is performed after that integration action. The start and stop moments of these test phases are determined by the test phase positioning strategy. We developed several test phase positioning strategies according to which test phases are started, for example periodically or when a certain risk level is reached.

We applied the methods developed to industrial case studies in ASML to investigate the benefits of these methods. From a case study performed in the manufacturing of lithographic machines, we learned that the duration of a test phase may be reduced by approximately 20% when using the test plan optimization method instead of creating a test plan manually. From a case study performed in the integration phase of a new prototype system, we learned that using the integration planning method may reduce integration time by almost 10% compared to a manually created integration plan. From a case study performed in the integration and test phase of a software system, we learned that the final test phase duration may be reduced by approximately 40% when applying a risk-based test phase positioning strategy instead of the currently used periodic test phase positioning strategy.

We conclude that the methods developed can be used to construct optimal integration and test plans. These optimal integration and test plans are often more efficient than manually created plans, which reduces the time-to-market of a complex system while maintaining the same final system quality. Future research should indicate how to incorporate the methods developed in the complete integration and test process, and how to obtain the information needed to create the integration and test models.
De integratie- en testfases die deel uit maken van de ontwikkeling en fabricage van complexe machines kosten over het algemeen veel geld en duren lang. Omdat de tijd die nodig is om een nieuw systeem op de markt te brengen steeds belangrijker wordt, is het extra belangrijk om deze fases zo kort mogelijk te houden terwijl er niet ingeleverd wordt op de systeemkwaliteit. Zeker voor de door tijdgedomineerde semiconductorindustrie en bedrijven die fabricagesystemen aan deze industrie leveren, zoals bijvoorbeeld ASML (een fabrikant van lithografische systemen) is dit belangrijk. Het Tangram onderzoeksproject heeft juist als doel om dit probleem op te lossen door gebruik te maken van een modelgebaseerde aanpak. Het promotieproject dat in dit proefschrift is beschreven, is onderdeel van dit Tangram project.

Om het beoogde doel te bereiken hebben we drie methodes ontwikkeld die elk een van onderstaande integratie- en testproblemen oplossen:

- Het construeren van een testplan, dat optimaal is met betrekking tot tijd, kosten en/of kwaliteit.
- Het construeren van een integratieplan, dat optimaal is met betrekking tot tijd, kosten en/of kwaliteit.
- Het construeren van een integratie- en testplan, dat optimaal is met betrekking tot tijd, kosten en/of kwaliteit.

De testplan optimalisatiemethode bestaat uit twee stappen. De eerste stap is het definiëren van een testmodel van het betreffende testprobleem. Dit model bestaat uit testen die gedaan kunnen worden met hun bijbehorende testduur en -kosten en mogelijke fouten die in het systeem aanwezig kunnen zijn met hun bijbehorende foutkansen en hun impact (deze geeft aan hoe belangrijk ze zijn). Verder bestaat dit model uit de relatie tussen de testen en mogelijke fouten, oftewel de afdekking van iedere test op iedere fout. De tweede stap is het berekenen van het optimale testplan met dit model, gegeven een doelfunctie en mogelijke restricties op tijd, kosten en/of risico. Risico is een maat voor de kwaliteit van het systeem. Door een zogenaamde AND/OR graaf te maken, bestaande uit AND knopen die een test representeren en OR knopen die een systeemtoestand op basis van uitgesloten fouten representeren, is het mogelijk om alle mogelijke testvolgordes te construeren. Een algoritme kiest vervolgens de optimale oplossing door deze AND/OR graaf te doorzoeken. Een oplossing is een verzameling van testvolgordes waarbij de testvolgorde die wordt doorlopen, bepaald wordt door de uitkomsten (goed/fout) van voorgaande testen.

De integratieplan optimalisatiemethode bestaat uit dezelfde twee stappen als de hiervoor beschreven methode. Het integratiemodel bestaat uit modules met bijbehorende levertijd, koppelingen die aangeven welke modules met elkaar geïntegreerd kunnen worden en testfases met bijbehorende testduur. Verder bestaat het model uit de relatie tussen deze testfases.
en modules. Deze relatie geeft aan welke modules geïntegreerd moeten zijn alvorens een testfase uitgevoerd kan worden. Ook voor dit probleem wordt een AND/OR graaf gemaakt. De AND knopen in deze graaf representeren de integratieacties en de OR knopen de systeem toestand op basis van de al geïntegreerde modules. Een algoritme kiest vervolgens de oplossing die binnen de kortste tijd de verschillende modules integreert tot een compleet systeem. Deze oplossing is een boom van integratieacties en testfases die voor elke module de volgorde van integratieacties en testfases aangeeft.

De integratie- en testplanningsmethode is een combinatie van de twee al reeds genoemde methodes en bestaat ook uit twee stappen. Het integratie- en testmodel is een combinatie van het testmodel en het integratiemodel uitgebreid met de relaties tussen de modules en mogelijke fouten. Deze relatie geeft aan welke mogelijke fouten door welke module gecreëerd worden. Tijdens het construeren van de integratie AND/OR graaf, wordt voor elke AND knoop een test AND/OR graaf gemaakt die de testfase representeert die na deze integratieactie uitgevoerd moet worden. Het start en stop moment van deze testfases wordt bepaald door de testfase positioneringstrategie. Er zijn verder verschillende testfase positioneringstrategies ontwikkeld die bijvoorbeeld elke periode een testfase starten of een testfase starten wanneer een bepaald risiconiveau gehaald is.

De ontwikkelde methodes zijn toegepast op enkele voorbeeldstudies binnen ASML om de voordelen van deze methodes te onderzoeken. Een voorbeeldstudie die tijdens de fabricage van lithografische systemen is gedaan, laat zien dat de duur van een testfase met ongeveer 20% kan worden gereduceerd door het toepassen van de testplan optimalisatiemethode in vergelijking met een handmatig gemaakt plan. Een voorbeeldstudie die tijdens de integratie van een prototype van een nieuw systeem is gedaan, laat zien dat de duur van deze integratiefase met bijna 10% kan worden gereduceerd door het toepassen van de integratieplan optimalisatiemethode in vergelijking met een handmatig gemaakt plan. Een voorbeeldstudie die tijdens de integratie- en testfase van een software systeem is gedaan, laat zien dat de duur van de laatste testfase met ongeveer 40% kan worden gereduceerd door een risicogebaseerde testfase positioneringstrategie te gebruiken in plaats van de nu gebruikte periodieke testfase positioneringstrategie.

We kunnen concluderen dat de ontwikkelde methodes daadwerkelijk integratie- en testplannen efficiënter maken. Hierdoor wordt de tijd benodigd om een nieuw systeem op de markt te brengen korter, terwijl de kwaliteit van deze systemen hetzelfde blijft. Toekomstig onderzoek moet uitwijzen hoe de methodes binnen de huidige integratie- en testprocessen gebruikt moeten worden en welke informatie nodig is om de genoemde integratie- en testmodellen te maken.
The integration and test planning methods developed in this project can be used to optimize real-life industrial integration and test plans. However, these methods may also be used for other problems. We now indicate how the results obtained in this project can be exploited by giving an overview of problems that may be solved using the methods suggested.

During the life-cycle of a multidisciplinary system, many integration and test phases are needed and therefore executed. Integration and test actions are performed during the development phase of a system, the manufacturing of multiple instances of a system and the operational phase of a system. The methods developed in this project can be used in most of these application domains as is demonstrated with several case studies. However, these case studies do not show all applications of the methods. Therefore, we now give an overview of the possible application areas.

The integration and test plan optimization methods can be used in three main application areas:

**Analysis** During analysis of several possible scenarios for integration and test plans, the methods developed can be used to determine the best of these possible scenarios. For example, if more tests can be developed, the methods can be used to determine the additional benefits of these investments. Also, the methods may be used to determine the test time reduction, if an additional prototype of a new system is created.

**Optimization** The primary purpose of the methods developed is to optimize real-life integration and test plans. With several case studies we have shown that this is possible. The results of these case studies show that test time can be reduced by more than 20% and integration time by more than 10% when optimizing a plan using the methods introduced instead of creating a plan manually. Furthermore, the method can be used to keep plans up to date. If a module is delivered later than planned, a new plan can be calculated almost automatically, which reduces planning effort.

**Strategy** The last purpose of the methods developed is to determine strategies that can be used to solve multiple instances of an integration and test problem. For example during manufacturing, multiple systems are integrated and tested. By calculating an integration and test plan for one of these systems, it is possible to determine the strategy that may be used to construct the integration and test plans for all of these systems. Also, during the development of software releases, we have shown that it is possible to determine the test phase positioning strategy that results in the shortest integration and test phases.

The combination of the three main application areas with the application domains shows that these methods can be used in many situations. To make the exploitation of results more concrete, we give a list of example problems.
• During the integration of the first prototype of a new system it is essential to show certain system functionalities as early as possible. By using the integration and test planning method the integration and test plan can be optimized towards this goal and can be kept up to date almost automatically.

• During the manufacturing of lithographic machines, test phases are performed to check system performance and calibrate certain system parameters. The test time of these test phases may be reduced by applying the test planning method.

• During reliability testing of a new system, or a software release, it may be more beneficial to perform efficient subsystem reliability tests in parallel than a system level reliability test. With the test planning method, it is possible to determine the duration of subsystem testing to reach a certain system level reliability.

• During the assembly of systems in the manufacturing process, the most expensive parts should be integrated as late as possible to reduce the interest cost of these parts. With the integration planning method, it is possible to optimize such an integration plan towards the interest cost.

• For the test phases of a new type of a system, prototypes of this system must be manufactured. More prototypes reduce the time-to-market of a system because testing can be done in parallel, but increase the total development cost. With the integration and test plan optimization method it is possible to determine the optimal number of prototypes for a new system.

• During the development of a new system, models can be used to replace components that have not been implemented yet. This way, system level tests can be performed before the complete system is ready, which reduces the time-to-market. Creating these models also costs time and money. With the integration planning method, it is possible to determine which models should be created for which components such that the total integration and test time or cost is minimal.
## CONTENTS

Preface \hspace{1cm} v  
Summary \hspace{1cm} vii  
Samenvatting \hspace{1cm} ix  
Exploitation of Results \hspace{1cm} xi  

1 Introduction  
1.1 ASML \hspace{1cm} 2  
1.2 Tangram project \hspace{1cm} 5  
1.3 Integration and test strategy \hspace{1cm} 8  
1.4 Objectives and approach \hspace{1cm} 13  
1.5 Outline \hspace{1cm} 16  

2 Test Plan Optimization  
2.1 Test sequencing in complex manufacturing systems \hspace{1cm} 17  
2.2 Stopping criteria for test sequencing \hspace{1cm} 41  
2.3 Hierarchical test sequencing \hspace{1cm} 57  
2.4 Extensions \hspace{1cm} 74  
2.5 Conclusions \hspace{1cm} 82  

3 Integration Plan Optimization  
3.1 Integration sequencing in complex manufacturing systems \hspace{1cm} 83  
3.2 Hierarchical integration sequencing \hspace{1cm} 100  
3.3 Extensions \hspace{1cm} 106  
3.4 Conclusions \hspace{1cm} 114  

4 Integration and Test Plan Optimization  
4.1 Integration and test sequencing in complex manufacturing systems \hspace{1cm} 115  
4.2 Optimal test phase positioning \hspace{1cm} 135  
4.3 Hierarchical integration and test sequencing \hspace{1cm} 139  
4.4 Conclusions \hspace{1cm} 140  

5 Case Studies  
5.1 Test planning for lithographic systems \hspace{1cm} 143  
5.2 Integration planning for lithographic systems \hspace{1cm} 149  
5.3 Integration and test planning for software releases \hspace{1cm} 158  

xiii
5.4 Test plans for software releases ........................................... 165
5.5 Conclusions ................................................................. 167

6 Conclusions ................................................................. 169
   6.1 Objectives ............................................................... 169
   6.2 Benefits to industry .................................................... 170
   6.3 Further research and development ................................... 170

Bibliography ................................................................. 173

Curriculum Vitae .......................................................... 179
This thesis is the final result of my Ph.D. project and is titled ‘Integration and test plans for complex manufacturing systems’. This chapter describes the background, objectives and approach of the project, and the outline of this thesis. The original Ph.D. assignment is titled ‘Integration and test strategy’. The assignment is part of the ‘Integration and test strategy’ Line of Attention of the Tangram research project. In essence, the goal of the Ph.D. project was to solve several strategic questions that arise during the integration and test phases of embedded systems and in particular lithographic machines of ASML. The main problem that is solved is the construction of integration and test plans and, moreover, the construction of good or even optimal plans. This thesis describes a methodology that provides a solution to this problem and that is, according to the original assignment, applicable to the integration and test phases of the development and manufacturing of complex embedded systems, like lithographic machines, airplanes, mobile phones and automobiles.

Throughout this thesis, we will use an analogy for an integration and test plan. This analogy is the puzzle as shown on the back of the cover of this thesis. Each puzzle piece (module) is a component of the complete puzzle (system). The puzzle is completed when all puzzle pieces are placed. Furthermore, each puzzle piece provides a certain functionality, which is denoted as one (or more) cogwheel(s). Together, these cogwheels provide the complete system functionality. Completing the puzzle by placing all the puzzle pieces is an analogy for the integration phase of a system where the modules are assembled into the complete system. During integration, tests are applied that show certain functional properties (or requirements) of the system. In the puzzle, certain ‘test’ puzzle pieces are the analogue of these tests. These puzzle pieces show a cogwheel with a needle that indicates in which direction the cogwheels turn and concludes whether this is the correct direction (pass) to provide the overall system functionality, or not (fail). Most chapters in this thesis are represented by puzzle pieces that illustrate the contents of these chapters.

This chapter is constructed as follows. Section 1.1 describes ASML lithographic machines and the market in which ASML operates. Section 1.2 describes the Tangram project. Section 1.3 gives an overview of existing methods and of literature dealing with integration and test strategies; moreover, it describes industrial examples of integration and test strategies. Section 1.4 describes the objectives of this Ph.D. project. Section 1.5 describes the outline of this thesis.
1. INTRODUCTION

1.1 A S M L

This Ph.D. project is performed in cooperation with ASML [ASML, 2006], an equipment supplier to the semiconductor industry. ASML is the world’s leading provider of lithographic systems for the semiconductor industry. These complex machines are critical in the production of integrated circuits or chips. In this thesis, many examples are given that deal with the integration and test phases of these lithographic machines. Furthermore, case studies during these phases were performed to demonstrate the applicability of the methods developed. The first subsection describes these lithographic machines and their main functionalities and requirements. The second subsection describes the semiconductor market and customers of ASML. The specific business drivers of the market drive the development, integration and test phases of ASML lithographic machines. The influence of these business drivers on the development of lithographic machines is described in the last subsection.

1.1.1 Lithographic machines

Lithographic machines are representative examples of complex manufacturing systems. These systems are used in the semiconductor industry and perform the most critical step in the manufacturing process of integrated circuits (IC). Their primary manufacturing process is the exposure of an IC pattern onto a wafer. Typically, the pattern is engraved on a so-called reticle. Light projects the pattern via a demagnification lens onto the wafer. During exposure, the reticle and the wafer make a scanning movement. Exposure must be performed with a very high accuracy. Therefore, reticles as well as wafers must undergo several preprocessing steps before exposure can take place. Preprocessing includes measuring of imperfections of the machine as well as of the wafers and reticles to enable compensation for these imperfections. For more information about semiconductor manufacturing see for example [Shon-Roy et al., 1998].

To actually perform the lithographic processes, several subsystems must be deployed. The main subsystems of a lithographic machine are pointed out in Figure 1.1 which shows the ASML XT:1900i lithographic machine. The reticle handler brings and takes reticles to the reticle stage which holds the reticle during the lithographic process. The wafer handler brings and takes wafers to the dual wafer stage which holds the wafer during the lithographic process. A laser produces the laser beam needed for the lithographic process. An illuminator uniformizes the light produced by the laser and a lens shrinks and images the pattern from the reticle on the wafer. Each of these subsystems consist of several multidisciplinary modules which again consist of several components. The average selling price of these systems was 13.7 million euro in the second quarter of 2006 [ASML, 2006].

The main performance requirements for a lithographic system are: critical dimension, overlay and productivity. The critical dimension denotes the minimal size, in nanometer (nm), of lines that can be produced on an IC. The overlay denotes the minimal difference, in nanometers (nm), between line positions of two lines that are produced by exposing the same wafer two times. The productivity is the maximal throughput in wafers per hour (wph).
Table 1.1: Performance requirements of lithographic machines [ASML, 2006]

<table>
<thead>
<tr>
<th>System</th>
<th>Critical dimension</th>
<th>Overlay</th>
<th>Productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUV Alpha demo</td>
<td>≤ 35 nm</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>XT:1900i</td>
<td>≤ 40 nm</td>
<td>≤ 6 nm</td>
<td>≥ 131 wph</td>
</tr>
<tr>
<td>XT:1700Fi</td>
<td>≤ 45 nm</td>
<td>≤ 7 nm</td>
<td>≥ 122 wph</td>
</tr>
<tr>
<td>XT:1400F</td>
<td>≤ 65 nm</td>
<td>≤ 8 nm</td>
<td>≥ 133 wph</td>
</tr>
</tbody>
</table>

that can be exposed. For the ASML lithographic machine XT1900i shown in Figure 1.1 and other systems types, these performance requirements are shown in Table 1.1.

1.1.2 Business drivers of the semiconductor market

Customers of ASML can be subdivided into three main categories: customers producing memory IC’s, customers producing logic IC’s, and so-called foundry customers. Memory IC’s are, for example, flash memories that can be found in digital cameras, logic IC’s are, for example, CPU’s that can be found in PC’s, and foundry customers produce a wide variety of semiconductors for third parties. The main semiconductor market business driver is time-to-market. The earlier a new CPU or memory unit is brought to the market, the more money can be earned. The time-to-market pressure has grown in the past years. This is illustrated in Figure 1.2, which shows the price drop of Intel CPU’s in the period between 1994 and 1999. As can be seen, the price drops are faster and faster which reduces the time window where a semiconductor manufacturer can reach its return on investments. Furthermore,
1. INTRODUCTION

The semiconductor industry is subject to Moore’s Law [Moore, 1965], which states that the number of transistors on a chip is doubled every 18 months.

The time-to-market pressure dominates the business drivers of equipment manufacturers delivering to this market, such as ASML. Customers need the newest equipment as soon as possible to create the newest products as soon as possible. This results in a rather strange demand for lithographic machines as described in [Prins, 2004]. The sooner the first technology is on the market, the more products will be sold by that equipment manufacturer since the semiconductor manufacturer has adjusted its production process on that specific system. However, there is a gap between the first delivery of a system and the demand for multiple systems that are used for production. This ‘chasm’ gap is used by semiconductor manufacturers to develop their production process and by the equipment manufacturer to mature the system. Therefore, the first system may have good overlay but lacks productivity or reliability. In the chasm gap there is time to improve both such that the system can be used for full production.

1.1.3 System development

The time-to-market driven aspect of lithographic machines dominates the way these systems are developed. ASML uses the standard V-model, shown in Figure 1.3, for the development of lithographic machines. The V-model starts on the left with system design, followed by subsystem design, module design, component design and component implementation. The subsystems and modules are often multidisciplinary complex systems by themselves, while the components are single disciplinary systems for example, software, mechanics, electronics or optics. The design and implementation actions are performed in parallel as much as possible. When components are ready they are tested and integrated in modules, which are again tested and integrated into subsystems and finally into the complete system. During
the final system test phase, the system is tested for its main system requirements such as overlay, critical dimension and productivity.

ASML implements the software of the system in-house. ASML subcontracts the design and implementation of many other modules to other companies such as Philips and Zeiss. The system integration and test is done by ASML. Chuma describes this way of working in more detail in [Chuma and Aoshima, 2003; Chuma, 2006]. These papers are a good start to learn about the lithographic equipment industry and its players.

The V-model does not show the final manufacturing phase of lithographic machines. After the development of the first prototypes of a new system, multiple machines are created during the manufacturing phase. In this phase, parts of the integration and test phases for mechanical, electrical and optical modules are repeated.

1.2 TANGRAM PROJECT

As a result of the high time-to-market pressure in the semiconductor industry, the time between ‘all functionality available’ (i.e., all parts of the machine are integrated) and ‘first shipment date’ (where the system reliability is supposed to be as stated in the system specifications) is very short. This small time gap puts test activities in an interesting bottleneck position, often resulting in shipment of partially untested machines, which is illustrated in Figure 1.4(a). In this figure, test activities start after a part of the development has finished. As soon as development has completely finished, test effort is increased to increase the product quality. However, on a fixed date, the system is shipped after which the rest of the test effort is done at the customer’s site. In this situation, it is hard to guarantee sufficient quality at the moment the customer starts to use the system. The only way to improve this is by means of concurrent engineering: tests are performed in parallel with development actions as much as possible, see Figure 1.4(b). In this figure, testing starts almost together with
1. INTRODUCTION

Development activities

(a) Test activities incomplete at shipment

(b) Concurrent development and testing

Figure 1.4: Concurrent engineering leading to shorter test phases [Brugman and Beenker, 2002]

development. As a result, fewer tests have to be done after development has finished and the system can be shipped earlier. The test effort at the customer’s site is likely to be shorter, since more faults have been discovered before shipment.

In order to develop methods to optimize this concurrent integration and testing, the Tangram project has been started. The project is a close cooperation between the Embedded Systems Institute [ESI, 2006], ASML and several other academic and industrial partners. The project aims at a reduction of cost and lead time in the integration and test phase of complex high-tech products, with special attention to ASML’s wafer scanners. Tangram aims to reduce both the time to shipment and the test time at the customer’s site. The goals and way of working of the project are described in the project plan [Brugman and Beenker, 2002]. A short summary of the Tangram project is given in this section.

The primary project objective is the following: define a test methodology with supporting processes and tools, which provides a right balance between product quality, product reliability, test schedule and test cost in order to optimize the return on investment. This test method should have the following characteristics:

- Focus on software testing and multi-technology integration testing.
- Coupling to a testability viewpoint on system architecture.
- Via an approach of integrating testable components, enable and support an early start of the test and integration activities in the development process.
- Use automation where applicable.
- Allow for diagnostic capabilities for the difficult problems.
• Maintain test quality in a continuously changing environment.

It should be noted that the scope of the Tangram project focusses on the right part of the V-model as shown in Figure 1.3. One could argue that when the design process is improved, less testing would be necessary, nevertheless an integration and test problem remains. Therefore, the Tangram project focusses on improving the integration and test phases to reach its objective.

The Tangram project is subdivided into four Lines of Attention (LoA’s), namely:

**LOA1** Integration and test strategy: Incorporation of testability in both the architecture and the ASML working methods. Development of algorithms and integration and test strategies in order to reduce the integration and test time.

**LOA2** Model-based integration and infrastructure: Development of methods and means for modeling and analysis in order to discover integration problems earlier during development. Definition and development of an infrastructure that can be used to perform integration tests on multiple components with different kinds of interfaces.

**LOA3** Model-based testing: Improvement of the quality of tests while minimizing the impact of the generation of tests, and minimizing the execution of tests while optimizing the quality of tests.

**LOA4** Model-based diagnosis: Development of algorithms that use test results to accurately isolate the cause of system failure and thus to minimize the system down time.

This Ph.D. project is part of LoA1 dealing with integration and test strategy. LoA1 consists of two Ph.D. projects which closely cooperate with each other. The thesis resulting from the other Ph.D. project performed by I.S.M. de Jong can be found in [de Jong, 2007]. The general objectives of Line of Attention 1 are:

• Develop an integral multidisciplinary test strategy such that:
  – Testable system equals sum of testable subsystems.
  – Components can be tested and diagnosed in subsystem environment.
  – Subsystems can be tested early in isolation and in the integrated system and in a hardware-in-the-loop context, the same subsystem can be tested in its environment.

• Allow for and define a growth path from current system architecture to a testable system architecture.

• Tradeoff analysis for testability requirements (e.g., on cost and performance) based on risk analysis.

In Section 1.4, we will explain which objectives are part of this Ph.D. project and are of the other Ph.D. project.
1.3 INTEGRATION AND TEST STRATEGY

This section describes questions that are addressed in an integration and test strategy for complex embedded systems, such as lithographic machines. It also provides an overview of research that has been performed in this field and describes the domains where these problems arise. This is done by giving examples and comparisons of real-life situations and problems in various industries. These examples and situations have been obtained during interviews with various engineers during visits to companies in the Netherlands and abroad.

Integration is, in the most general sense, the process of combining things. In this project, integration is defined as the process of combining several subsystems into one system. Each of these subsystems contributes to the system functionality but only the combination of all subsystems provides the complete system functionality. Testing is the process of showing the correct behavior of a system. A strategy is a longterm plan of actions designed to achieve a particular goal. An integration and test strategy is therefore defined as a long term plan of integration and test actions to achieve that several subsystems are combined into one complete system and to show that the system shows correct behavior.

The main objective of LoAr is to create a methodology that enables to create, maintain and control an integration and test strategy for embedded systems. This methodology should be suited for all domains where integration and test strategies of embedded systems are essential. These domains are the development, manufacturing and operation of systems. In the following subsections, integration and test strategy problems are described and literature overviews for each of these domains are given.

1.3.1 Development integration and test

The most discussed embedded systems domain for integration and test strategies is development. The integration and test phase during development deals with the first instance of a new type of system. We subdivided this subsection in two parts. First, we discuss existing literature on this subject. Subsequently, we discuss several problems in industry and the different approaches taken by companies.

Literature and methods

Each discipline in system development has its own testing and integration techniques, methods and tools that are used in the integration and test phases. We will describe a few of them here.

In software development, a lot of research has been performed on test and integration techniques. According to Beizer [Beizer, 1990], the following integration methods can be distinguished:

- Bottom-up: Start with the integration and test of the lowest parts of the software, e.g., the drivers. Then, combine these parts into one and test the combination, and so on until the complete system is integrated and tested.
1.3. INTEGRATION AND TEST STRATEGY

- Top-down: Start with the integration and test of the highest parts of the software, e.g., the high level control software using stubs (simple module simulation models) to replace the lowest parts. Then, replace the stubs one by one with the module implementations and test the combination, until the complete system is integrated and tested.

- Big-bang: Integrate everything at once and start testing the combination.

These methods are never optimal given a certain integration problem. A combination of them chosen for a specific situation is optimal. For example, if in a top-down approach the development of a stub of module A takes too long compared to the development of module A itself, it is better to integrate module A immediately. For other modules this might not be the case. For object-oriented software systems, Hanh et al. [Hanh et al., 2001] describe a way to develop an optimal integration plan however, this method is only suited for this specific type of software.

Testing methods and techniques are also widely discussed by Beizer [Beizer, 1990] and Graig and Jaskiel [Graig and Jaskiel, 2002]. Some categories of testing methods and techniques can be distinguished:

- Test process
- Test planning
- Test implementation
- Test execution

Testing strategies are mainly considered in the first two categories: the test process and test planning. In a test plan mainly three categories of tests (phases) can be distinguished: unit testing, integration testing and system testing. Unit testing is testing one part of the system in isolation, while integration testing is testing the combination of two parts (that are integrated). System testing is testing the complete system. The normal approach is to perform all these test phases, although this might not be optimal. For example, if the implementation of a particular part is extremely easy, the unit test phase may be skipped. In the literature, some techniques exist that decide what to test and what not to test. One example is risk-based test selection that starts with the execution of tests that have the highest risk. However, the test that excludes the most risk in the system is the system test: this test evidently excludes the most risk if it passes, although the probability that this test passes may be extremely small if the other tests have not been executed.

Another decision that has to be taken in the test process is when to stop testing. There are two decision criteria mentioned in [Graig and Jaskiel, 2002]: 1) stop when resources are exhausted, and 2) stop based on certain metrics such as defect rate. The first criterion is simple but does not guarantee the quality of software; the second criterion only takes failing test cases into account and does not consider passing test cases.

Another discipline is the hardware or mechanical discipline. Integration in this discipline is called assembly. Looking to assembly strategies or plans, the main difficulty is to
find good and feasible sequences. Some research has been done in this area by de Mello [de Mello and Sanderson, 1991b,a] and Boneschanscher [Boneschanscher, 1993] who describe methods to generate optimal and feasible assembly sequences based on mechanical part information. The disadvantage of these methods is that the delivery times of the different parts is not taken into account. Furthermore, the cost of an assembly sequence is based on the assembly costs, often caused by robot movements. During integration, costs are also caused by performing tests.

Not much research has been performed on test strategies for hardware testing. This is because the main difficulty of hardware testing is the (costly) test equipment. Strategic decisions are therefore often based on experience in a certain field.

The last discipline discussed is system integration and test. Not much research has been performed on the strategic decisions that are made during system integration. Normally, these decisions are made using tools like Microsoft Project. At the system level, strategic test problems are often solved by improving the process during the integration and test phases. For example, the test maturity model [Burnstein et al., 1996] can be used to measure the test and integration capabilities of organizations. Furthermore, methods are developed that help in structuring the process, such as the Failure mode effect analysis (FMEA) described in, for example, [Goddard, 2000]. A more general approach is taken in [Levardy et al., 2004; Engel et al., 2004], where a general Verification, Validation and Test approach is described. General disadvantage of these methods is that they focus on improving the process instead of defining techniques that optimize the decisions made during the testing and integration of systems.

**Industrial examples**

In Table 1.2, the time-to-market durations and corresponding durations of the integration and test phases are shown for different types of industries. This table shows the approximate durations of the integration and test phases of the mentioned companies. This overview clearly shows the difference in time-to-market of a new system for the different companies, but also shows that almost always a large part of this time-to-market is spent on integration and testing. The cost involved in these integration and test phases is therefore huge.

Many differences exist between companies and their approaches to the integration and
test strategies. For example, Airbus, manufacturer of civilian airplanes, takes approximately 6 years to integrate and test a new type of airplane. Many tests that are performed are prescribed by aviation organizations before a new airplane is actually allowed to transport people. This quality dominated approach is also used during the development of satellites and spacecrafts or by companies developing systems for the Department of Defense in the U.S.A. In these companies, all possible tests are performed, final system quality is most critical while cost and time are less critical. The difference with this approach and those employed in the semiconductor equipment industry is large. In this industry, it is crucial to get the system to the market as soon as possible, even when the system quality is not perfect.

In the software discipline, another trend is visible. Companies like Microsoft often compromise on quality to reduce costs. The quality is improved through patches after the product is sold. This is of course the cheapest way of testing: let testing be done by your customers. This strategy works since customers want the newest technology as soon as possible and accept the fact that faults can occur.

No matter which integration and test strategy a company uses, one has to decide what drives the integration and test phase and what constrains the phase. For example, a company producing large-scale printers and copiers develops its own software, just like ASML. Although there are no real differences between the business drivers of both companies, their software integration and test strategies are different. While ASML performs every week a validation test of the software release that takes approximately one day, the printer manufacturer performs every four weeks a test phase that lasts a complete week. This strategic choice mainly depends on the availability of prototypes and the properties of the system.

1.3.2 Manufacturing integration and test

After the first system of a type is developed, multiple systems of the same type are manufactured. Also during this manufacturing phase, integration and testing is performed. The focus during manufacturing testing is on implementation faults and not on design faults since these were already tested during the development phase.

Manufacturing integration strategies are closely related to mechanical assembly sequences as described by de Mello [de Mello and Sanderson, 1991b,a], which was already discussed in the previous subsection. These techniques can be used to calculate optimal robot assembly movements for small embedded systems. In the manufacturing test strategies, many methods are described in the literature, each suitable for a different discipline. An example is testing semiconductors as described by [Carmon-Freed, 1996] and [Bahadur et al., 1998]. However, these techniques are only suitable for specific (semiconductor) test problems. In software, usually no manufacturing testing is performed since software can be copied. However, manufacturing tests exist, called calibrations, that test the settings of the software for a specific machine.

The integration and test strategies in an industry heavily depend on the manufacturing volumes of the different companies. In Table 1.3, an overview is given of manufacturing volume estimates and test effort during the manufacturing for different companies. In general,
Table 1.3: Industry manufacturing volumes

<table>
<thead>
<tr>
<th>Industry</th>
<th>Manufacturing volumes</th>
<th>Test effort (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space (NASA)</td>
<td>1-2</td>
<td>50%</td>
</tr>
<tr>
<td>Manufacturing systems (ASML)</td>
<td>10-20</td>
<td>30%</td>
</tr>
<tr>
<td>Airplane (AIRBUS)</td>
<td>100</td>
<td>25%</td>
</tr>
<tr>
<td>Automobile</td>
<td>100,000</td>
<td>10%</td>
</tr>
<tr>
<td>Telephones</td>
<td>1,000,000</td>
<td>1%</td>
</tr>
<tr>
<td>Drug</td>
<td>10,000,000</td>
<td>&lt;1%</td>
</tr>
</tbody>
</table>

we can conclude that the higher the volume, the less testing is done.

We can distinguish two types of manufacturing testing in industry: testing the system and testing the manufacturing process. When testing the system, each system is tested according to some integration and test strategy. For example, ASML tests every system that is manufactured whether it meets certain requirements. This is also done by AIRBUS and NASA. However, a manufacturer of mobile telephones only tests a predefined percentage of its products. This is done to check whether the manufacturing process is working correctly, for example, to check if no manufacturing machine is broken or if no operator performs the wrong actions.

In the system test approach, the focus of integration may also be different for different companies. For AIRBUS, the main difficulty during integration or assembly is the size and shape of the different components. The logistic process dominates the integration strategy. For ASML, the costs and delivery times of components dominate the integration and therefore the test strategy. The interest costs of expensive components are minimized by integrating these components as late as possible.

1.3.3 Operational integration and test

Operation is the final system phase that requires integration and test strategies. The integration strategy questions are often not present during operation since the system is already integrated. Only for spare parts, an integration strategy may be interesting. Testing during operation is often performed to find certain faults if the system is down, or during maintenance.

In the Tangram project, methods are investigated to improve the diagnosis of faults during operation. Pietersma [Pietersma et al., 2005] describe how model-based diagnosis techniques may be used to define which observations should be performed to find a certain cause of a system failure. Other diagnosis techniques are described by Pattipati [Pattipati and Alexandridis, 1990], where a test sequencing method is introduced that determines which test sequence is best to diagnose a certain system. Main assumption of both approaches is that at most one fault is present.

Another difference in maintenance strategies is the capability of repairing faults when a
1.4 OBJECTIVES AND APPROACH

In the previous section, we described certain problems that exist at companies and the decisions that have to be taken by companies regarding integration and test strategies. We also discussed some existing methods from the literature. These methods merely focus on single disciplines or application areas, or focus on the integration and test process. However, the problems and questions of companies are almost always the same: what is the optimal integration and test strategy for a given situation. Such a strategy contains the sequence of actions, the stopping criterion, and quantitative measures such as time cost and/or quality.

In LoA1, we have the goal to develop a method that solves this problem for multidisciplinary embedded systems. There are three main steps that are important for an integration and test strategy. These three steps are shown in the integration and test strategy framework in Figure 1.5. The first step is defining the goal of the integration and test strategy. This goal is different for each domain and for each system. In essence, we denote three main drivers: time, quality and cost. For every company there usually is one objective (minimal cost, minimal time or maximal quality) and several constraints (minimal end quality, maximal cost or time to spend). This choice is denoted as the TQC (time, quality, cost) model.

The second step is defining the strategy. This step is investigated further in this Ph.D. project. In essence, this step consists of making a so-called integration and test plan that contains the actions that should be performed.

The last step is executing and controlling the strategy. This step is investigated in the other Ph.D. project [de Jong, 2007] of Line of Attention 1 of the Tangram project. In this project, the effect of the chosen strategy on different aspects of the system and provided resources is investigated.

These three steps are primarily performed in sequence. However, if during the creation of the plan, the goal cannot be fulfilled, e.g., maximal cost is always exceeded, the goal must be adjusted. Also, if during the execution of the strategy, actions are delayed or resource capacities are limited, a new plan must be determined. It is even possible that a new goal must be determined if resource capacities change, e.g., the shipment date (constraint) is changed.

The objective of this Ph.D. project is to develop methods that can be used to define integration and test plans. This objective is split up into 3 subproblems:

- Develop a method that constructs a test plan.
- Develop a method that constructs an integration plan.
- Develop a method that constructs an integration and test plan.

In the following subsections, these subproblems are discussed in more detail.
1.4.1 Test plan optimization

There are three main research questions that need to be addressed to solve this subproblem:

**Question 1.1** What is the structure of a test plan?

**Question 1.2** Which information is needed to create a test plan?

**Question 1.3** Which method is suited to construct a test plan given the information provided?

Research question 1.1 deals with the actions that are present in a test plan, and the decisions that need to be taken to construct the plan. Research question 1.2 deals with the information of the system under test, test strategy goal and test process that is needed to create a plan. Question 1.3 deals with the method that uses this information to create a plan that is good or even optimal in the given situation.

Since the goal of the project is to solve industrial problems, a fourth research question is added:

**Question 1.4** What are the benefits of creating an optimal test plan in real-life industrial problems?

This question is answered by performing case studies that show the benefits in time or cost reduction during test phases of lithographic systems.

1.4.2 Integration plan optimization

Also for this subproblem, there are three main research questions that need to be addressed:
1.4. OBJECTIVES AND APPROACH

QUESTION 2.1 What is the structure of an integration plan?

QUESTION 2.2 What kind of information is needed to create an integration plan?

QUESTION 2.3 Which method is suited to create an integration plan given the information provided?

Research question 2.1 deals with the actions that are present in an integration plan, and the decisions that need to be taken in the plan. Research question 2.2 deals with the information of the system, integration strategy goal and integration process that is needed to create a plan. Question 2.3 deals with the method that uses this information to create an integration plan that is good or even optimal in the given situation.

Since the goal of the project is to solve industrial problems, a fourth research question is added:

QUESTION 2.4 What are the benefits of using this method in real-life industrial problems?

This question is answered by performing case studies that show the benefits in time or cost reduction during integration phases of lithographic systems.

1.4.3 Integration and test plan optimization

Also for this subproblem, there are three main research questions that need to be addressed:

QUESTION 3.1 What is the structure of an integration and test plan?

QUESTION 3.2 What kind of information is needed to create an integration and test plan?

QUESTION 3.3 Which method is suited to create an integration and test plan given the information provided?

Research question 3.1 deals with the actions that are present in an integration and test plan, and the decisions that need to be taken in the plan. Research question 3.2 deals with the information of the system, integration and test strategy goal, and integration and test process that is needed to create a plan. Question 3.3 deals with the method that uses this information to create an integration and test plan that is good or even optimal in the given situation.

A fourth research question is added:

QUESTION 3.4 What are the benefits of using this method in real-life industrial problems?

This question is answered by performing case studies that show the benefits in time or cost reduction during integration and test phases of lithographic systems.
1.5 OUTLINE

This thesis is structured as follows. Chapter 2 deals with research questions 1.1 through 1.3. This chapter is based on several papers that have been submitted and accepted during the project. Chapter 3 deals with research questions 2.1 through 2.3. This chapter is based on one paper that has been submitted during the project. Chapter 4 deals with research questions 3.1 through 3.3. Also this chapter is based on a paper that has been submitted during the project. Chapter 5 shows several case studies described in several papers that have been published during the project, the benefits of the methods introduced and answers research questions 1.4, 2.4 and 3.4. Chapter 6 gives conclusions about this Ph.D. project.
This chapter introduces a method for optimizing test plans. It is illustrated with a single puzzle piece that shows a test in its most abstract sense: based on the outcome of the turning direction of the cogwheel system determine whether the system works (pass) or not (fail).

In this chapter, three sections introduce the test plan optimization method. Section 2.1 is based on a paper that describes the optimization of the test sequence and selection. Section 2.2 is based on a paper that optimizes the stop moment of testing and introduces several objectives. Section 2.3 is based on a paper that describes how to sequence test phases in a multilevel test plan by making the plan hierarchical. Section 2.4 describes (practical) extensions to the method that were not mentioned in the papers but are sometimes needed to optimize real-life industrial test plans for complex systems. The last section gives conclusions about this chapter.

2.1 TEST SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

This section is based on the paper titled Test sequencing in complex manufacturing systems [Boumen et al., 2006d] that is accepted by IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans in 2006. The paper section dealing with the case studies has been removed in this chapter because a more detailed and elaborate description of the case studies can be found in Section 5.1.

The goal of this paper is to answer research questions 1.1 through 1.3 for simple test planning problems where the goal is to deliver a system with optimal final quality in minimal test time.
Test sequencing in complex manufacturing systems


Abstract

Testing complex manufacturing systems, such as an ASML [ASML, 2006] lithographic machine, takes up to 45% of the total development time of a system. The problem of which tests must be executed in what sequence to ensure in the shortest possible test time that the system works, the test sequencing problem, was already solved by Pattipati [Pattipati et al., 1991; Pattipati and Alexandridis, 1990] for the diagnosis of systems during operation. Test sequencing problems during the development and manufacturing phases of systems, however, require a different approach than test sequencing problems during operation. In this paper, the test problem description and algorithms developed by Pattipati are extended to solve test sequencing problems for the development and manufacturing of manufacturing systems.

2.1.1 Introduction

Testing complex manufacturing systems is expensive and time consuming as described in [Cusumano and Selby, 1995; Engel et al., 2004]. As time-to-market is becoming increasingly important, the issue of which tests must be executed in what sequence to ensure in the shortest possible time that the system works is a problem for many companies. This is especially true for the time-to-market driven semiconductor industry and for companies providing manufacturing systems to this industry such as ASML [ASML, 2006; Prins, 2004], a provider of lithographic systems.

Test selection methods have been widely discussed in the literature. For example risk-based test selection, as described in [Graig and Jaskiel, 2002], is a well known and used selection method. These methods mainly focus on the selection of tests, not on the sequence in which tests must be executed. This is in contrast to Pattipati [Pattipati et al., 1991;
Pattipati and Alexandridis, 1990] who described in several papers a method called sequential diagnosis. This method is used to solve the problem of which tests must be applied in what sequence at minimal expected cost, also known as the test sequencing problem. Additional research by Shakeri et al. [Shakeri et al., 2000] described two different strategies for solving this problem: single-fault and multiple-fault sequential diagnosis. To solve large test sequencing problems, several papers [Shakeri et al., 2000; Raghavan et al., 1999a; Tu and Pattipati, 2003] described heuristic approaches for both strategies.

These aforementioned studies mainly focus on test sequencing problems for diagnosing systems during the operational phase of systems. Test sequencing problems during the development and manufacturing phases of systems, however, require a different approach. These test sequencing problems are essentially different in 2 ways:

- The repair or fix action of a fault influences the system under test and therefore the remaining test sequencing problem. For example, a fix action may introduce faults that were not present before.
- The fault probability of a system is substantially higher than during the operational phase.

Furthermore, previous research never described the possible reduction in test time of solving a test sequencing problem in the development or manufacturing phases of systems by using the sequential diagnosis method instead of creating test sequences manually, which is normally done by experts.

In this paper, we extend the sequential diagnosis method to solve test sequencing problems during the development and manufacturing of machines. These problems are called system test problems. Our extensions are twofold: first we extend the problem description and then we extend the solution algorithms. In addition, we show the potential test time reduction of solving test problems with this method. These results are obtained from a case study performed for a test phase during the manufacturing of lithographic machines.

The structure of the paper is as follows. Section 2.1.2 explains system testing, the test sequencing problem and shows an illustration. Section 2.1.3 starts with the sequential diagnosis problem definition and continues with the extensions made to this definition. Section 2.1.4 discusses how the system test problem can be solved using a single-fault strategy. Section 2.1.5 discusses how the system test problem can be solved using a multiple-fault strategy. Section 2.1.6 shows three measures that reduce computational effort to create a solution. Section 2.1.7 presents the results of a case study that has been performed to show the potential time reduction using the method. Section 2.1.11 gives conclusions about this paper. Section 2.1.8 describes the notations used in Sections 2.1.9 and 2.1.10, which explain the two algorithms.

2.1.2 System testing

Testing, in this paper, is defined as checking whether the system behaves according to its specification, while diagnosis is defined as finding the root cause of a certain system failure.
Test problems are known in the following life cycle phases: the design phase, the implementation phase, the test/qualification phase and in the production phase (according to the life-cycle definition shown in [Engel et al., 2004]). Diagnosis problems are merely known in the use/maintenance phase. During testing, faults are found and must be repaired with a fix action. A fix action changes the system under test and therefore influences the test sequence. Therefore, fix actions must be taken into account during system testing.

In the following subsection, the system test sequencing problem is discussed. In the second subsection, a telephone system is used as illustration for the system test problem.

**System test sequencing problem**

A system test sequencing problem is defined as determining a test and fix action sequence for a system under test, given a set of available tests and a set of possible faults. After executing the tests and fixing detected faults successfully, the system should work according to its specification. The objective of this problem is to minimize the time needed to test the system. Fix actions can have constraints (preconditions) or consequences for the system under test and therefore on the test and fix action sequence:

- A fix action of a fault can only be performed after a certain test.
- A fix action of a fault can introduce new faults.

These constraints or consequences are common for system test sequencing problems, but can also be present in other test sequencing problems.

**Illustration**

A telephone is taken as an example to illustrate a system test sequencing problem. This telephone consists of three modules: the device, the cable and the receiver as shown in Figure 2.1. There are two interfaces between the modules: one between the device and the cable and one between the cable and the receiver.

![Figure 2.1: The telephone](image)

5 possible faults are defined:

1. The device is broken.
2. The cable is broken.
3. The receiver is broken.
4. The connection between the cable and the device is broken.
5. The connection between the cable and the receiver is broken.

The first three faults are specific module faults, while faults 4 and 5 are interface faults. Each fault has a certain probability to be present, in this case study 10% for each fault. Six tests are available to test this system:

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

The cost of each test can be defined in terms of time, manpower requirements or other economic factors. In this paper, cost is defined in time units because the objective is to reduce test time. We assume that test 1 takes 3 time units, tests 2, 3 and 4 each take 1 time unit, and test 5 and 6 each take 2 time units. Each fault has a fix action that repairs that fault. During the test of the device (test 2), information needed for the fix action of the device (fault 1) is obtained. This means that before fixing the device, test 2 must be performed. The fix action of the cable consists of replacing the cable with a new one. After the fix action, the interface faults (fault 4 and 5) may be present again, and must therefore be tested again.

2.1.3 Problem formulation

To solve the system test sequencing problem, two steps are performed. In this section, the system test sequencing problem is formulated in a system test model. In the subsequent two sections, solution algorithms are developed to calculate a test sequence using the system test model.

Test model

According to [Pattipati et al., 1991], a test sequencing problem can be formulated in a quintuple test model $D: (T, S, C, P, R_t)$, where:

- $T$ is a finite set of $k$ tests.
Table 2.1: (System) test model of the telephone

<table>
<thead>
<tr>
<th>S / T</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>P</th>
<th>(Rᵣ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10%</td>
<td>( - )</td>
</tr>
<tr>
<td>s₂</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10%</td>
<td>( s₄, s₅ )</td>
</tr>
<tr>
<td>s₃</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>10%</td>
<td>( - )</td>
</tr>
<tr>
<td>s₄</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10%</td>
<td>( - )</td>
</tr>
<tr>
<td>s₅</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10%</td>
<td>( - )</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- S is a finite set of l fault states.
- C : T → ℝ⁺ gives for each test in T the associated cost of performing that test.
- P : S → [0, 1] gives for each fault state in S the a priori (absolute) probability that the fault state is present.
- Rᵣ : T → ℙ(S) gives for each test in T, a subset of fault states that are covered by that test and is also known as the test signature.

The assumptions for this test model are:

- Tests only have binary outcomes (pass or fail).
- The fault states are independent of each other.
- The fault states each have a unique test signature.
- The tests are 100% reliable, meaning a fault state is certainly present if a test fails.
- The tests are 100% sensitive, meaning a test always fails if a covered fault state is present.

Element Rᵣ can also be represented as a matrix A of dimensions l × k, where Aᵢⱼ = 1 if test tⱼ covers fault state sᵢ, otherwise Aᵢⱼ = 0. In Table 2.1, the test model of the telephone is shown as matrix A and elements T, S, C and P. The elements between brackets are introduced in the next subsection and are not part of this test model.

System test model

It is not possible to define the complete system test sequencing problem in this test model because the test model does not take fix actions into account. Therefore, the test model is extended to the system test model by making the following additional assumptions:

- Each fault state in S has a fix action.
A fix action fixes the fault state completely.

Furthermore, two elements are added to $D$ to define that a fix action may introduce new fault states and that a fix action may only be performed after a certain test. These elements are:

- $R_{ss} : S \rightarrow \mathcal{P}(S)$ gives for each fix action of a fault state in $S$ the fault states that may be introduced by that fix action.

- $R_{st} : S \rightarrow \mathcal{T}$ gives for certain fix actions of fault states in $S$, the test that must be performed before that fix action.

Summarizing, the system test model used to describe the system test sequencing problem is represented by $D = (T, S, C, P, R_{ts}, R_{ss}, R_{st})$. The system test model for the telephone illustration is shown in Table 2.1, the elements between brackets are part of the system test model. Note that the element $R_{st}$ is denoted by $\star$ in the $A$ matrix.

For now we assume that the fault probability of fault states reintroduced is the same as the a priori fault probability. However, the model could be extended such that $R_{ss}$ also denotes the new fault probability of a fault state reintroduced. For now we assume that a test only has binary outcomes. However, some problems during the manufacturing or development of complex machines have tests with more than one fail outcome. For these problems we use the multi-valued tests as described by Tu et al. in [Tu et al., 2003]. This approach is not further discussed in this paper.

Problem objective

A solution $G$ to the system test sequencing problem is a function $G : \mathcal{P}(S) \rightarrow T^*$, which gives for each set $S_U$ of fault states that could be present a fix action and test sequence $G(S_U)$, with tests from $T$ that isolates and fixes every fault state in $S_U$. The cost of such a solution is [Pattipati et al., 1991]:

$$ J(G) = \sum_{S_U \subseteq S} \sum_{S \in G(S_U)} C(t) \prod_{s \in S_U} P(s) \prod_{s' \in (S \setminus S_U)} (1 - P(s')) $$

(2.1)

The objective is to find an optimal solution $G^*$ that has minimal expected test cost $J^*$, from all possible solutions $G$:

$$ J^* = J(G^*) = \min_{G \in G} J(G) $$

(2.2)

2.1.4 Single-fault system testing

One strategy for solving the system test problem is the single-fault strategy which assumes that at most one fault state is present in the system. This strategy can be used for problems with low fault state probabilities. In this section, solution algorithms from sequential diagnosis that use the single-fault strategy are adjusted to solve the system test model.
2. TEST PLAN OPTIMIZATION

Table 2.2: Single-fault system test model of the telephone

<table>
<thead>
<tr>
<th>S / T</th>
<th>t_1</th>
<th>t_2</th>
<th>t_3</th>
<th>t_4</th>
<th>t_5</th>
<th>t_6</th>
<th>P</th>
<th>R_{ss}</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>64.28%</td>
<td>-</td>
</tr>
<tr>
<td>S_1</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td></td>
<td>7.14%</td>
<td></td>
</tr>
<tr>
<td>S_2</td>
<td></td>
<td>I</td>
<td>O</td>
<td>I</td>
<td>O</td>
<td></td>
<td>7.14%</td>
<td>S_0, S_4, S_5</td>
</tr>
<tr>
<td>S_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td>O</td>
<td>7.14%</td>
<td></td>
</tr>
<tr>
<td>S_4</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td>O</td>
<td>O</td>
<td>7.14%</td>
<td></td>
</tr>
<tr>
<td>S_5</td>
<td></td>
<td>I</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>I</td>
<td>7.14%</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>2</td>
<td>2</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

Single-fault system test model

The assumption that at most one fault state exists, requires some changes to the original system test model. First, the possibility that no fault state exists must be modeled explicitly. Therefore an extra fault state $S_0$ is added to $S$, representing that the system is OK. The extended set of fault states is denoted by $\tilde{S}$. Furthermore, the sum of fault state probabilities must be 100%. Therefore, the a priori fault state probabilities $P$ are converted to conditional fault state probabilities $\tilde{P}$ using [Pattipati et al., 1991; Shakeri et al., 2000]:

$$P(S_0) = \left(1 + \sum_{s \in \tilde{S}} \frac{P(s)}{1 - P(s)}\right)^{-1}$$ (2.3)

and

$$P(S_i) = \frac{P(S_i)}{1 - P(S_i)} \left(1 + \sum_{s \in \tilde{S}} \frac{P(s)}{1 - P(s)}\right)^{-1} \text{ for } S_i \in S.$$ (2.4)

Fixing a fault state always introduces the system OK state $S_0$. The single-fault system test model is: $D^s = (T, \tilde{S}, C, \tilde{P}, R_{ss}, R_{st})$. The single-fault system test model of the telephone is shown in Table 2.2.

Solution algorithm

To calculate the optimal solution $G^*$, an AND/OR graph containing all solutions from $G$ is constructed. This graph consists of three types of nodes: AND, OR and leaf nodes, as shown in Figure 2.2(a). An OR node denotes a candidate set of fault states, that is a set of fault states of which at least one is present. The AND nodes represent tests applied to the OR nodes and the leaf nodes represent isolated fault states. Constructing an AND/OR graph is NP-hard [Pattipati and Alexandridis, 1990]. Optimal algorithms are based on Dynamic Programming or AND/OR graph search [Pattipati and Alexandridis, 1990].

The Dynamic Programming technique is a recursive algorithm that constructs a graph from the leaf nodes up by identifying larger subtrees until the complete graph is generated. The Dynamic Programming technique has storage and computational complexity of $O(k^n)$.
The AND/OR graph search algorithm (AO*) constructs a directed graph with a root node and a nonempty set of terminal leaf nodes. The root node represents the given problem to be solved, while the terminal leaf nodes correspond to the subproblems with known solutions. An OR node is solved if at least one of its immediate successor nodes is solved. An AND node is solved when all of its immediate successors are solved. Because the top-down AO* is more efficient than Dynamic Programming [Pattipati and Alexandridis, 1990], the AO* algorithm is used for single-fault test sequencing.

To make the AO* algorithm suitable for system test problems, the AND/OR graph definition is slightly modified. An intermediate leaf node is introduced, as shown in Figure 2.2(b), which represents a fix action of fault states that are present. An OR node is followed by an intermediate node if a fault state is isolated and a fix action can be applied. A test can be performed if it provides additional information or if it is required before a fix of one of the candidate fault states (according to the fix action precedence). A leaf node is defined as a node with an empty candidate set. The intermediate leaf node has at least one successor: a leaf node if (∀s ∈ SF : Rss(s) = ∅), or an OR node containing the subset s of S for which holds (∃s_i ∈ SF : s ∈ Rss_i(s_i)). To reduce calculations, the expected test cost of solved OR nodes are stored during the calculation of the complete AND/OR graph. This extended algorithm, called AO*σ, has been formalized and implemented, as shown in Section 2.1.9.

The test tree shown in Figure 2.3(a) is an optimal test tree for the telephone. The optimal expected test cost J* of this tree is 4.31 which means that on average 4.31 time units are necessary to identify and fix one fault state in the system.
Simulation experiment

The expected test cost $J^*$ of a single-fault tree denotes the average test cost with the assumption that only one fault state is present in a system. In practice, the tree will be used several times until all faults are isolated and fixed. The average test cost is therefore higher than the expected test cost $J^*$. By system test simulation, the average simulated test cost $\tilde{J}$ can be calculated. This is done with a test process simulation model as described in [de Jong et al., 2006]. This model simulates the test, fix and diagnose processes with a discrete-event simulator and is not discussed further in this paper. A simulation experiment is conducted to calculate the average cost for the telephone tree solution in a multiple-fault environment. This experiment consists of simulating a test process that uses the tree from Figure 2.3(a) as test plan. This test process is used for testing 5000 systems that contain faults according to the $a$ priori fault state probabilities. The average test time $\tilde{J}$ of testing the 5000 systems is 6.8 time units. A histogram of the simulation experiment is shown in Figure 2.3(b) showing the variance of the test times.

![Figure 2.3: Single-fault optimal solution and simulation results](image)

2.1.5 Multiple-fault system testing

The application domain of the single-fault strategy is limited to system test sequencing problems with, on average, few fault states present in the system. If more fault states are present, the multiple-fault strategy, that considers all possible combinations of fault states, can be used. Multiple-fault strategies have an exponential complexity of $O(2^l)$ [Shakeri et al., 2000] and therefore require more computational effort.
Solution algorithm

Shakeri [Shakeri et al., 2000] proposes a multiple-fault algorithm based on the single-fault algorithm. This extended single-fault strategy takes multiple faults into account, but the algorithm may indicate that a non-present fault state is present. The cost of the unnecessary repair action is so high for complex manufacturing systems that we consider this algorithm not suitable to solve system test problems.

A second multiple-fault strategy proposed by Shakeri is to consider all possible combinations of fault states as separate fault states and use the single-fault algorithm to solve the problem. However, with this optimal strategy, all present fault states are isolated (and therefore fixed) at the leaf node which is less optimal than fixing in between testing.

A third algorithm proposed by Shakeri is based on Sure near-optimal multiple-fault test strategies. These strategies isolate faults one or more at a time while not making an error when multiple faults are present. The three basic ingredients are: minimal candidate generation, minimal candidate isolation and multiple-fault propagation. From all possible sets of fault states that could have resulted in certain test outcomes, only the sets with the least amount of fault states are considered and investigated. This assumption does not hold for fault state probabilities above 50%, which are rare during diagnosis but can occur during manufacturing or development testing. Therefore, the Sure strategies are not considered suitable for system testing.

For system testing, a new optimal multiple-fault strategy based on multiple-fault propagation is proposed in this paper. This strategy is called the multiple-fault propagation (mfp) strategy. In the mfp strategy, multiple-fault propagation is conducted during the AND/OR graph construction instead of after a single-fault tree calculation. The OR nodes in the AND/OR graph do not denote candidate sets, but the compact set notation [Grunberg et al., 1987] used for multiple-fault propagation. This compact set notation is a short notation for all possible suspected fault state sets (suspected sets). A suspected set is a set of fault states that are suspected to be present, because the set could have caused the previous test outcomes. The compact set notation denotes all possible suspected sets in short denoting test outcomes as candidate sets (at least one fault state is present) and an excluded fault state set, denoting known ‘good’ (not present) fault states. The mfp strategy is implemented in the multiple-fault AO∗σ.

In the AND/OR graph definition used for this algorithm, there are two types of intermediate leaf nodes: fix and diagnose actions. Fix actions are performed as soon as a fault state is isolated. Diagnosis actions are performed when no test gives additional information and no fault states are isolated. These nodes resemble the rectification actions as defined in [Pattipati and Dontamsetty, 1992] because they only have one outcome, but are essentially different from these rectification actions because they are performed only when necessary.

Diagnose actions are needed to terminate the algorithm when fault state sets have the same test signature, and tests cannot isolate the individual fault states. The fix action consequences and constraints are taken into account in the multiple-fault AO∗σ algorithm. The multiple-fault AO∗σ algorithm is formalized in Section 2.1.10.

The multiple-fault tree shown in Figure 2.4(a) is the optimal multiple-fault solution for
Simulation experiment

The simulation experiment with the single-fault tree showed that the optimal cost differs from the average simulated cost, because of the single-fault assumption. For the multiple-fault tree, the optimal expected test cost $J^*$ should be the same as the average simulated test cost $\bar{J}$. A simulation experiment has been performed to show that $J^*$ approaches $\bar{J}$ and to show the cost variations when using a multiple-fault tree in a test process. The performed experiment is the same as the single-fault experiment; only now using the multiple-fault tree from Figure 2.4(a). In Figure 2.4(b), a histogram of the simulation experiment is shown. After 5000 simulation runs, the average test cost $\bar{J}$ of the multiple-fault tree is 5.3, and equals the expected test cost $J^*$. This figure also shows that the variation in test cost is less than testing with the single-fault tree.

2.1.6 Computational reduction measures

To solve large test sequencing problems, computational reduction measures are needed to obtain a solution in reasonable computation time. For the single-fault algorithm several near-optimal search algorithms are known from [Pattipati and Alexandridis, 1990], [Shakeri et al., 2000], [Raghavan et al., 1999a] and [Tu and Pattipati, 2003], for example: the $AO^*$ algorithm, the limited search $AO^*$, and the $AO^*$ algorithm combined with a multi-step Information Gain heuristic. We use the one-step Information Gain (IG) heuristic in the $AO^*$ algorithm, which is sufficient to solve large test sequencing problems in the manufacturing and development of manufacturing machines.

The multiple-fault $AO^*$ algorithm has much more computational cost than the single-fault algorithm. Therefore, three computational reduction measures are proposed in this section that can be used, separately or combined, to solve large problems. Each of them is discussed in the following subsections.

Probability estimator

The computation of pass and fail probabilities of a test is an expensive computation in the multiple-fault solution strategy because the fault probabilities of all possible suspected fault state sets must be calculated from the compact set notation. The probability estimator (PE) estimates the pass and fail probabilities using the compact set notation, without calculating all possible suspected fault state sets. The PE estimates for each fault state covered by a test the probability that this fault state is present. The probability that at least one of the covered fault states is present determines the fail probability of a test. The estimate that a fault state $s$ is present $P(s)$ is based on the compact set notation denoted in $x$, and is calculated with:

$$1 - P'(s) = \left(1 - P(s)\right) \prod_{S_C : S_C \in x \land \exists s \in S_C} \left(1 - \frac{P(s)}{\sum_{s \in S_C} P(s)}\right)$$ (2.5)
Here, $S_C$ is a candidate set of OR node $x$, containing $s$. This estimate is based on the relative fault probability of the fault state in each candidate set. For each candidate set at least one fault state is present, meaning that the sum of the probabilities is 100%. The relative contribution of the investigated fault state in the candidate set is taken as the relative fault probability of that fault state for that candidate fault state set. The product of these relative fault probabilities, including the a priori fault probability, is the new fault probability.

**Information Gain**

The Information Gain (IG) heuristic prunes search alternatives in the AND/OR graph. This measure can be used during the single- and multiple-fault strategies. The Information Gain heuristic determines test $t$ in the candidate test set $T_C$ that maximizes the Information Gain
2. TEST PLAN OPTIMIZATION

Table 2.3: Illustration of divide and conquer method

<table>
<thead>
<tr>
<th>S/T</th>
<th>A₁</th>
<th>A₂</th>
<th>A₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₁</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₂</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₃</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₄</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₅</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t₆</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₇</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₈</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₉</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₁₀</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₁₁</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t₁₂</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

per cost unit [Raghavan et al., 1999a]:

$$\frac{IG(t)}{C(t)} = \max_{t' \in T} \left( \frac{IG(t')}{C(t')} \right)$$  \hspace{1cm} (2.6)

Here, the Information Gain can be calculated with:

$$IG(t) = - \left( p_p(t) \log_2 p_p(t) + p_f(t) \log_2 p_f(t) \right)$$  \hspace{1cm} (2.7)

Above, $p_p(t)$ and $p_f(t)$ are the pass and fail probabilities of test $t$.

Divide and conquer

The divide and conquer (DAC) method divides the system test problem into smaller subproblems, which can be solved in an optimal or near-optimal way. The sequence of subproblems themselves is fixed. The problem can be decomposed according to the system hierarchy, or by investigating the original model: tests that cover the same fault states can be grouped. The group size can be determined using the product of the fault state probabilities. If possible, the total group fault probability $P_{tot}$, defined by equation 2.8, should be at least 50%, otherwise the solution is too far from the optimal one.

$$P_{tot} = 1 - \prod_{s \in S} (1 - P(s))$$  \hspace{1cm} (2.8)

Techniques exist that may help in decomposing an $A$ matrix into smaller submatrices, for example the Mondriaan technique presented in [Vastenhouw and Bisseling, 2005]. In the example model shown in Table 2.3, the test problem is divided into three smaller test problems by grouping tests from the original test problem. The three smaller problems, $A_1, A_2, A_3$, can be solved separately, ignoring the test-fault state relations that are not bold. The three obtained test sequences are then performed sequentially.
Table 2.4: Results computational reduction experiment

<table>
<thead>
<tr>
<th>Measure</th>
<th>Solution cost</th>
<th>Comp. effort</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$J^*$</td>
<td>OR nodes</td>
</tr>
<tr>
<td>-</td>
<td>691</td>
<td>2836</td>
</tr>
<tr>
<td>PE</td>
<td>691</td>
<td>2836</td>
</tr>
<tr>
<td>IG</td>
<td>764</td>
<td>126</td>
</tr>
<tr>
<td>DAC</td>
<td>736</td>
<td>514</td>
</tr>
</tbody>
</table>

Performance

In this subsection, we investigate the performance of the proposed computational measures for the multiple-fault algorithm with the example real-life model shown in Table 2.5. In this example, we extended the method and algorithms presented with multiple outcome tests as introduced in [Tu et al., 2003]. This means that a test may have multiple fail outcomes, depending on the fault state. For example, $t_2$ in this example has outcome 1 if $s_1$ is present, outcome 2 if $s_2$ is present and $s_1$ is not present, and outcome 0 (pass) if none of them are present.

A solution for this model is obtained with each proposed computational measure. The results are shown in Table 2.4. The measures are compared on solution cost ($J^*$) and computational effort, denoted in the amount of OR nodes investigated and the computation time. The computation times are normalized to the computation time without reduction measures. The algorithm is implemented in the $\chi$ language [Hofkamp and Rooda, 2002], that allows for direct implementation of the functions defined. For the DAC method, we divided the problem into two subproblems. The PE measure performs good since it decreases computation time without increasing the expected test cost. The IG measure performs not as good because the measure ignores fix action signatures (element $R_n$) into account. Therefore, this heuristic should not be used for problems with many fix action signatures, such as this example. The performance of the DAC heuristic depends on the choice of decomposition. In this case, the DAC heuristic performs quite good because the decomposition is clear and simple.

2.1.7 Case study

This paper section is intentionally removed from this chapter because a more detailed and elaborate description of the case studies can be found in Section 5.1 or in the original version of this paper.

2.1.8 Notations

In Table 2.6, the definitions and their descriptions used in this paper are shown. The following notations are used:
2. TEST PLAN OPTIMIZATION

Table 2.5: (System) test model for the performance experiment

<table>
<thead>
<tr>
<th>$S / T$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
<th>$t_6$</th>
<th>$t_7$</th>
<th>$t_8$</th>
<th>$t_9$</th>
<th>$t_{10}$</th>
<th>$t_{11}$</th>
<th>$t_{12}$</th>
<th>$t_{13}$</th>
<th>$t_{14}$</th>
<th>$t_{15}$</th>
<th>$P$</th>
<th>$R_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$2^*$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>60 %</td>
</tr>
<tr>
<td>$s_2$</td>
<td>0</td>
<td>$2^*$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100 %</td>
</tr>
<tr>
<td>$s_3$</td>
<td>0</td>
<td>0</td>
<td>$2^*$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>80 %</td>
</tr>
<tr>
<td>$s_4$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$2^*$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>80 %</td>
</tr>
<tr>
<td>$s_5$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$2^*$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>80 %</td>
</tr>
<tr>
<td>$s_6$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$2^*$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>90 %</td>
</tr>
<tr>
<td>$s_7$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$2^*$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10 %</td>
</tr>
<tr>
<td>$s_8$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$1^*$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>60 %</td>
</tr>
<tr>
<td>$s_9$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$2^*$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5 %</td>
</tr>
<tr>
<td>$s_{10}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$2^*$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5 %</td>
</tr>
<tr>
<td>$s_{11}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100 %</td>
</tr>
<tr>
<td>$s_{12}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100 %</td>
</tr>
<tr>
<td>$s_{13}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100 %</td>
</tr>
<tr>
<td>$s_{14}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100 %</td>
</tr>
<tr>
<td>$s_{15}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100 %</td>
</tr>
</tbody>
</table>

| $C$     | 45   | 30   | 20   | 12   | 60   | 60   | 60   | 60   | 60   | 20     | 12   | 60     | 20     | 12   | 60   | 20 | 120  | 90 |

- $\mathcal{P}$ denotes a powerset.
- $H_m(x/(J, t))$ denotes $H_m$ where the value of $x$ is replaced by the tuple $(J, t)$.
- $x \setminus x_f$ denotes $\{s | s \in x \land s \notin x_f\}$.

The single- and multiple-fault OR node representations are defined as:

- $X^s \subseteq \mathcal{P}(S)$, so the OR node consist of one candidate set ($S_C$).
- $X^m \subseteq \mathcal{P}(\mathcal{P}(S)) \times \mathcal{P}(S)$, this notation is adopted from the compact set notation as defined by Grunberg et al. [Grunberg et al., 1987]. The OR node consists of multiple candidate sets ($S_C$) and one excluded set ($S_E$).

The expected test cost of each solved OR node is stored in $H^s$ or $H^m$ to reduce computation time. These are defined as:

- $H^s : X^s \rightarrow (\mathbb{R}^+ \times T) \cup \{\bot\}$ gives for a solved single-fault OR node the minimal test cost and the next test or gives undefined for an unsolved OR node.
- $H^m : X^m \rightarrow (\mathbb{R}^+ \times T) \cup \{\bot\}$ gives for a solved multiple-fault OR node the minimal test cost and the next test or gives undefined for an unsolved OR node.

2.1.9 Single-fault algorithm

This section gives a formal, functional-style [Bird, 1998] description of the single-fault $AO^*_s$ algorithm. A step-by-step description of the algorithm that relates to the functions defined, is shown in Figure 2.5.

The single-fault algorithm consists of two functions $ORs$ and $ANDs$ that are defined for a single-fault system test problem $D^s$. To find the optimal expected test cost $J^*$ for $D^s$, the following expression can be used:
2.1. TEST SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

**Single-fault step-by-step algorithm**

**Input:**
- System Model: $D$;

**Output:**
- The optimal solution graph: $G$;
- The expected cost of the solution graph;

**Step 0:** Initialize a graph $G$ consisting of the root node $x = S$, i.e., initial system ambiguity, mark the node as unsolved.

**Step 1:** Repeat the following steps for the root node $x$ to construct an AND/OR graph until the root node is marked solved. Then exit with $J = F(x)$ as expected test cost and the solution graph $G$ (these steps are performed by function ORs).

**Step 1.0** If $x$ contains one element $s$ and the condition $R_{st}(s)$ is fulfilled, remove element $s$ (fix $s$) from $x$ and insert $R_{ss}(s)$ in $x$

**Step 1.1** If $x$ is empty, mark $x$ solved in $G$ and assign cost 0.0 and exit, otherwise determine the candidate test set $T_C$ and perform for each test $t$ in $T_C$ the following steps (these steps are performed by function ANDs):

**Step 1.1.0** Initialize a subgraph $G'$ consisting of root node $t$

**Step 1.1.1** Determine for $t$ the pass and fail OR nodes $x_p$ and $x_f$, insert them in $G'$ and draw an edge from $t$ to both of them

**Step 1.1.2** If $x_p$ is not solved, mark $x_p$ unsolved and perform steps 1.0 through 1.2 for $x$ replaced by $x_p$; do the same for $x_f$ (this is the recursion by function ORs)

**Step 1.1.3** Determine for $t$ the pass and fail probabilities $p_p$ and $p_f$ and assign the cost $p_p \cdot F(x_p) + p_f \cdot F(x_f)$ to AND node $t$

**Step 1.2** Select the test $t$ and the corresponding subgraph $G'$ that has minimal expected cost. Mark $x$ solved and assign the cost $F(t) + C(t)$ to $x$. Merge graph $G$ with subgraph $G'$, create an edge from node $x$ to the root node of $G'$ and exit. If no subgraph is present ($T_C$ is empty), mark $x$ solved, assign 0.0 as cost (diagnose action is needed) and then exit.

Figure 2.5: Single-fault step-by-step algorithm description
Table 2.6: List of definitions

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>System test problem: $(T, S, C, P, R_{ts}, R_{ss}, R_{st})$.</td>
</tr>
<tr>
<td>$T$</td>
<td>Set of $k$ tests.</td>
</tr>
<tr>
<td>$S$</td>
<td>Set of $l$ fault states.</td>
</tr>
<tr>
<td>$S^s$</td>
<td>$S \cup s_0$.</td>
</tr>
<tr>
<td>$C$</td>
<td>Gives for each test in $T$ the cost of performing that test.</td>
</tr>
<tr>
<td>$P$</td>
<td>Gives for each fault state in $S$ the \textit{a priori} probability that it is present.</td>
</tr>
<tr>
<td>$P_s$</td>
<td>Gives for each fault state in $S$ the conditional probability that it is present.</td>
</tr>
<tr>
<td>$R_{ts}$</td>
<td>Gives the subset of fault states that are covered by a test (test signature).</td>
</tr>
<tr>
<td>$R_{ss}$</td>
<td>Gives for each fix action of a fault state in $S$ the fault states that may be introduced.</td>
</tr>
<tr>
<td>$R_{st}$</td>
<td>Gives for certain fix actions of fault states in $S$, the test that has to be performed.</td>
</tr>
<tr>
<td>$A$</td>
<td>$l \times k$ matrix representation of $R_{ts}$.</td>
</tr>
<tr>
<td>$\delta A$</td>
<td>Density of $A$.</td>
</tr>
<tr>
<td>$P$</td>
<td>Average \textit{a priori} fault state probability of $P$.</td>
</tr>
<tr>
<td>$C_A$</td>
<td>Set of costs for a candidate test set.</td>
</tr>
<tr>
<td>$T_C$</td>
<td>Candidate (may be performed) test set.</td>
</tr>
<tr>
<td>$T_P$</td>
<td>Performed (have been performed) test set.</td>
</tr>
<tr>
<td>$S_U$</td>
<td>Suspected (possible present) fault state set.</td>
</tr>
<tr>
<td>$S_C$</td>
<td>Candidate (at least 1 present) fault state set.</td>
</tr>
<tr>
<td>$S_E$</td>
<td>Excluded (definitely not present) fault state set.</td>
</tr>
<tr>
<td>$t, s, p, c$</td>
<td>A single test, fault state, probability or cost.</td>
</tr>
<tr>
<td>$-F$</td>
<td>Represents a fixed definition.</td>
</tr>
<tr>
<td>$-D$</td>
<td>Represents a diagnosed definition.</td>
</tr>
<tr>
<td>$-P$</td>
<td>Represents a passed definition.</td>
</tr>
<tr>
<td>$G, \mathcal{G}$</td>
<td>A solution and all possible solutions of the test problem.</td>
</tr>
<tr>
<td>$j, J, J^*$</td>
<td>Single path test cost, expected and optimal expected test cost.</td>
</tr>
<tr>
<td>$\tilde{J}, J^0$</td>
<td>Average simulated test cost and original test cost.</td>
</tr>
<tr>
<td>$X^s, X^m$</td>
<td>Single- and multiple-fault OR node domain.</td>
</tr>
<tr>
<td>$x$</td>
<td>Representation of single- or multiple-fault OR node.</td>
</tr>
<tr>
<td>$\mathcal{H}^s, \mathcal{H}^m$</td>
<td>Function that returns the cost of a solved single- or multiple-fault OR node.</td>
</tr>
<tr>
<td>$H$</td>
<td>Set of solved single- or multiple-fault OR nodes.</td>
</tr>
</tbody>
</table>

\[
(J^*, H) = ORs(S, H_{init}, \emptyset)
\]  

(2.9)

Here, $H_{init} : X^s \to \{\perp\}$ is the initial function that gives the cost of a solved OR node. The resulting $H$ can be used to construct the optimal solution $G$. This calculation gives the cost $J^*$ of the optimal solution according to equation 3, because of the following. In principal, with each OR node calculation all tests are considered for the sequence. This results in an AND/OR graph with all possible solutions $\mathcal{G}$ from which the cheapest one is chosen. The best test per OR node is chosen based on the minimal expected test cost per test, starting from the last OR node. This moment of choice does not effect optimality for earlier OR nodes, because choosing a test with more expected test cost will always result in more expected test cost for a previous OR node. Furthermore, at each OR node not all tests
are investigated as AND nodes. Tests that are certain to pass or fail will not be investigated. This does not affect optimality because the result of such a test is exactly the same OR node and therefore this test will always increase the cost of a sequence.

Let function $\text{ORS} : X^e \times \mathcal{H}^e \times \mathcal{P}(T) \to \mathbb{R}^+ \times \mathcal{H}^e$ be a function that calculates the minimal expected test cost $J$ of an OR node and updates $H$, given the OR node $x$, the current $H$ and the performed test set $T_P$. $J$ is:

- 0.0 if $x$ is terminated,
- extracted from $H$ if $x$ has already been solved,
- calculated with a fixed OR node if no further tests are possible,
- calculated otherwise.

where $x$ is terminated if $x$ is empty or only contains $s_0$ and $x$ is solved if $x$ is defined in $H$.

The function is defined as follows:

$$
\text{ORS}(x, H, T_P) =
\begin{cases}
(0.0, 0, H) & \text{if } x \subseteq \{s_0\} \\
(H(x).0, H) & \text{if } x \not\subseteq \{s_0\} \land H(x) \neq \perp \\
(J, H_F) & \text{if } x \not\subseteq \{s_0\} \land H(x) = \perp \\
(J, H_m(x/(J, t_j))) & \text{if } x \not\subseteq \{s_0\} \land H(x) = \perp \\
& \land T_C = \emptyset
\end{cases}
$$

where:

- $(J, H_F) = \text{ORS}(x_F, H, \emptyset)$, are the minimal expected test cost and the function that defines the cost of a solved OR node for $x_F$.

- $T_C = \{t | t \in T \land (x \not\subseteq R_{ts}(t) \land x \cap R_{ts}(t) \neq \emptyset) \\
\lor (t \not\in T_P \land \exists s : s \in x : t \in R_{st}(s))\}$, is the candidate test set. The set contains tests that are not guaranteed to pass or fail, so the candidate fault state set is no subset of the test signature, but the test signature must contain at least one element from the candidate set. Also, the test set contains tests that must be performed before fixing a candidate fault state.

- $m = |T_C|$, is the number of candidate tests.

- $(C_A(t_i), H_i) = \text{ANDs}(x, t_i, H_{i-1}, T_P)$ for $i = 1, \ldots, m$,
  where $H_0 = H$, are the minimal test cost and updated $H$ for each test in $T_C$.

- $J = \min_{t \in T_C}(C(t) + C_A(t))$, is the minimal expected test cost for $x$, and $t_j$ is the test from $T_C$ for which this holds.
2. TEST PLAN OPTIMIZATION

- $x_F = \{s_0\} \cup \{s \in S \land (\exists s_i : s_i \in x : s \in R_{ss}(s_i))\}$, is a fixed OR node. The fixed fault states are removed from the candidate set and the fault states that are introduced by the fixed fault states are added to the candidate set, together with the system OK state $s_0$.

Let function $ANDs : X^i \times T \times H^i \times P(T) \rightarrow \mathbb{R}^+ \times H^i$ be a function that determines the minimal expected test cost $J$ of an AND node and updates $H$, given the OR node $x$, applied test $t$, the current $H$ and the performed test set $T_P$. $J$ is calculated by multiplying the pass probability of $t$ by the minimal expected test cost of the pass OR node, summed with the fail probability of $t$ multiplied by the minimal expected test cost of the fail OR node:

$$ANDs(x, t, H, T_P) = (p_p \cdot c_p + p_f \cdot c_f, H_f)$$

where:

- $x_f = x \cap R_{ts}(t)$, is the fail OR node that consists of the intersection of $x$ with the test signature of $t$.
- $x_p = x \setminus x_f$, is the pass OR node that consists of the remaining fault states.
- $p_p = \left(\sum_{s \in x_p} P(s)\right) \left(\sum_{s \in x} P(s)\right)^{-1}$, is the pass probability, calculated by dividing the sum of the $x_p$ fault probabilities, by the sum of the $x$ fault probabilities.
- $p_f = 1 - p_p$, is the fail probability.
- $(c_p, H_p) = ORs(x_p, H, T_P \cup \{t\})$, are the minimal expected test cost and the updated $H$ of $x_p$.
- $(c_f, H_f) = ORs(x_f, H_p, T_P \cup \{t\})$, are the minimal expected test cost and the updated $H$ of $x_f$.

2.1.10 Multiple-fault algorithm

This section shows the formal, functional description of the multiple-fault $AO^*_\sigma$ algorithm. A step-by-step description of the algorithm is shown in Figure 2.6.

The multiple-fault algorithm consists of two functions $ORm$ and $ANDm$ that are defined for a system test problem $D$. To find the optimal expected test cost $J^*$ for $D$, the following expression can be used:

$$(J^*, H) = ORm((\emptyset, \emptyset), H_{init}, \emptyset)$$

Here, $H_{init} : X^m \rightarrow \{\bot\}$ is the initial function that gives the cost of a solved OR node. The resulting $H$ can be used to construct the optimal solution $G$. This calculation gives the cost $J^*$ of the optimal solution, because of the same reasoning as for the single-fault algorithm.

Let function $ORm : X^m \times H^m \times P(T) \rightarrow \mathbb{R}^+ \times H^m$ be a function that calculates the minimal expected test cost $J$ of an OR node and updates the set of solved OR nodes, given the OR node $x$, the current set of solved OR nodes $H$ and the performed test set $T_P$. $J$ is:
Multiple-fault step-by-step algorithm

Input:  
- System Model: $D^m$;

Output:  
- The optimal solution graph: $G$;  
- The expected cost of the solution graph;

Step 0: Initialize a graph $G$ consisting of the root node $x = (\emptyset, \emptyset)$, i.e., initial system ambiguity, mark the node as unsolved.

Step 1: Repeat the following steps for the root node $x$ to construct an AND/OR graph until the root node is marked solved. Then exit with $J = F(x)$ as expected test cost and the solution graph $G$ (these steps are performed by function $ORm$).

Step 1.0 If $x.0$ contains one or more elements $\{s\}$, do for the elements where the condition $R_{st}(s)$ is fulfilled the following: remove element $s$ (fix $s$) from all elements in $x.0$ and insert $s$ in $x.1$, also remove $R_{ss}(s)$ from $x.1$.

Step 1.1 If $x.1$ is $S$, mark $x$ solved in $G$ and assign cost 0.0 and exit, otherwise determine the candidate test set $T_C$ and perform for each test $t$ in $T_C$ the following steps (these steps are performed by function $ANDm$):

Step 1.1.0 Initialize a subgraph $G'$ consisting of root node $t$.

Step 1.1.1 Determine for $t$ the pass and fail OR nodes $x_p$ and $x_f$, insert them in $G'$ and draw an edge from $t$ to both of them.

Step 1.1.2 If $x_p$ is not solved, mark $x_p$ unsolved and perform steps 1.0 through 1.2 for $x$ replaced by $x_p$; do the same for $x_f$ (this is the recursion by function $ORm$).

Step 1.1.3 Determine for $t$ the pass and fail probabilities $p_p$ and $p_f$, and assign the cost $p_p \cdot F(x_p) + p_f \cdot F(x_f)$ to $t$.

Step 1.2 Select the test $t$ and the corresponding subgraph $G'$ that has minimal expected cost. Mark $x$ solved and assign the cost $F(t)+C(t)$ to $x$. Merge graph $G$ with subgraph $G'$, create an edge from node $x$ to the root node of $G'$ and exit. If no subgraph is present ($T_C$ is empty), create $x_d$ by removing all elements from $x.0$ and insert them in $x.1$ (a diagnose OR node). Insert $x_d$ in $G$ and draw an edge from $x$ to $x_d$.

If $x_d$ is not solved, mark $x_d$ unsolved and perform steps 1.0 through 1.2 to solve $x_d$. Then, mark $x$ solved, assign $F(x_d)$ as cost and exit.

Figure 2.6: Multiple-fault step-by-step algorithm description
- if \( x \) is terminated,
- derived from \( H \) if \( x \) has already been solved,
- calculated with a fixed OR node if fault states are isolated,
- calculated with a diagnosed OR node if no fault states are isolated and further testing has no use,
- calculated otherwise.

where \( x \) is terminated if all fault states are excluded, \( x \) is solved if \( x \) is defined in \( H \) and fault states are isolated if they are within a candidate set of size \( 1 \), and \( T_p \) complies with \( R_d \). The function is defined as follows:

\[
\text{ORm}(x, H, T_p) = \begin{cases} 
(\text{o.o}, H) & \text{if } x.1 = S \\
(H(x), \text{o}, H) & \text{if } x.1 \neq S \land H(x) \neq \bot \\
(J_F, H_F) & \text{if } x.1 \neq S \land H(x) = \bot \\
(J_D, H_D) & \text{if } x.1 \neq S \land H(x) = \bot \land T_C = \emptyset \\
(J, H_m(x/(J, t_j))) & \text{if } x.1 \neq S \land H(x) = \bot \land T_C \neq \emptyset \\
\end{cases}
\]

\( (2.13) \)

Where:
- \((J_F, H_F) = \text{ORm}(x_F, H, \emptyset)\), are the minimal expected test cost and the updated \( H \) of \( x_F \).
- \((J_D, H_D) = \text{ORm}(x_D, H, \emptyset)\), are the minimal expected test cost and the updated \( H \) of \( x_D \).
- \( x_F.\text{o} = \{S_C \setminus S_F | S_C \in x.\text{o}\} \), are the candidate sets of the fixed OR node without \( S_F \).
- \( x_F.1 = (x.1 \cup S_F) \setminus \{s | s \in S \land (\exists s_i : s_i \in S_F : s \in R_d(s_i))\} \), is the excluded set of the fixed OR node. \( S_F \) is added to the excluded set of \( x \) and excluded fault states of \( x \) that may be introduced by \( S_F \) are removed from the excluded set.
- \( x_D.\text{o} = \emptyset \), are the candidate sets of the diagnosed OR node. As all candidate fault states are diagnosed, no more candidate sets exist.
- \( x_D.1 = (x.1 \cup S_D) \setminus \{s | s \in S \land (\exists s_i : s_i \in S_D : s \in R_d(s_i))\} \), is the excluded set of the diagnosed OR node. \( S_D \) is added to the excluded fault states of \( x \) and excluded fault states that may be introduced by \( S_D \) are removed from the excluded set of \( x \).
- \( S_F = \{s | s \in S \land R_d(s) \in (T_p \cup \{\perp\}) \land (\exists S_C : S_C \in x.\text{o} : |S_C| = 1 \land s \in S_C)\} \), are the fixed fault states consisting of fault states that are in a candidate set of size \( 1 \) and where the \( R_d \) relation is in \( T_p \).
2.1. TEST SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

- $S_D = \{s|s \in S \land (\exists S_C : S_C \in x.o : s \in S_C)\}$, are the diagnosed fault states consisting of all fault states in the candidate sets.

- $T_C = \{t|t \in T \land (\forall S_C : S_C \in x.o : S_C \not\subseteq R_{0i}(t) \land S_C \cap R_{0i}(t) \neq \emptyset) \lor (\exists S_C : S_C \in x.o : (t \notin T_p \land \exists s : s \in S_C : s \in R_{st}(s)))\}$, is the candidate test set consisting of the tests of which the test signature is no subset of the excluded fault states of $x$ (a certain pass), and none of the candidate sets of $x$ is a subset of the test signature (a certain fail), together with tests that must be performed before fixing a candidate fault state.

- $m = |T_C|$, is the number of candidate tests.

- $(C_A(t_i), H_i) = ANDm(x, t_i, H_{i-1}, T_P)$ for $i = 1, \ldots, m$, where $H_0 = H$, are the minimal test cost and updated $H$ for each test in $T_C$.

- $J = \min_{t \in T_C} (C(t) + C_A(t))$, is the minimal expected test cost of $x$, and $t_j$ is the test from $T_C$ for which this holds.

Let function $ANDm : X^m \times T \times \mathcal{H}^m \times \mathcal{P}(T) \rightarrow \mathbb{R}^+ \times \mathcal{H}^m$ be a function that determines the minimal expected test cost $J$ of an AND node and updates $H$, given the OR node $x$, applied test $t$, the current $H$ and the performed test set $T_P$ in the same way as in the single-fault algorithm:

$$ANDm(x, t, H, T_P) = (p_p \cdot c_p + p_f \cdot c_f, H_f) \quad (2.14)$$

Where:

- $x_p = \{S_C \setminus R_{0i}(t)|S_C \in x.o\}, x.1 \cup R_{0i}(t)$, is the pass OR node. The test signature of $t$ is removed from the candidate sets of $x$ and added to the excluded set of $x$.

- $x_f = (x.\cup \{R_{0i}(t)|x.1\}, x.1)$, is the fail OR node. The test signature, without the excluded fault states of $x$, is added as candidate set.

- $(c_p, H_p) = ORm(x_p, H, T_P \cup \{t\})$, are the minimal expected test cost and the updated set of solved OR nodes of $x_p$.

- $(c_f, H_f) = ORm(x_f, H_p, T_P \cup \{t\})$, are the minimal expected test cost and the updated set of solved OR nodes of $x_f$.

- $p_p = \frac{\sum_{S_C \in x.o \cap x.1} \prod_{s \in S_C} p(s) \prod_{s \in (S_C \setminus (x.1 \cup x.1))} 1-p(s)}{\sum_{S_C \in x.o \cap x.1} \prod_{s \in S_C} p(s) \prod_{s \in (S_C \setminus (x.1 \cup x.1))} 1-p(s)}$, is the pass probability of $t$ calculated by dividing the sum of the fault probabilities of the suspected sets in $x_p$, by the sum of the fault probabilities of the suspected sets in $x$. The
fault probability of a suspected set is calculated by multiplying the fault probabilities of the suspected fault states by the pass probabilities of the not-suspected, not-excluded fault states.

- \( p_f = 1 - p_p \), is the fail probability of \( t \).

Furthermore, function \( ANDm \) uses function \( \Theta : X^m \rightarrow \mathcal{P}(\mathcal{P}(S)) \), which determines all possible suspected sets \( S_U \) that could have resulted the OR node given the OR node \( x \), and is derived from [Grunberg et al., 1987]. \( S_U \) consists of the subsets of \( S \) of which the intersections with each of the candidate sets in \( x \) are not empty and of which the intersection with the excluded set of \( x \) is empty:

\[
\Theta(x) = \{ S_U | S_U \subseteq S \land S_U \cap x.1 = \emptyset \land(\exists S_C : S_C \in x.0 : S_U \cap S_C = \emptyset) \}
\] (2.15)

### 2.1.11 Conclusion

In this paper, we propose a method that can be used to derive a test sequence which takes a minimal amount of time to test a system, while assuring that the system works. To this end, a system test model is introduced to formulate system test sequencing problems. This model is an extension of the sequential diagnosis model [Patti et al., 1991]. To solve the system test sequencing problem, two algorithms have been constructed.

The first algorithm is the single-fault algorithm \( AO^*_s \), which calculates a time-optimal test tree. The second algorithm is the multiple-fault algorithm \( AO^*_m \), which does not have the single-fault assumption, but has higher computational cost. This algorithm solves problems optimally, using multiple-fault propagation.

By performing a case study in the manufacturing of ASML lithographic machines we have shown that it is possible to model and solve real-life test sequencing problems in manufacturing. We can conclude that the multiple-fault algorithm reduces test time 15% to 30% compared to manually selected test sequences. The single-fault algorithm increases test time and is therefore not suitable to solve manufacturing or development test sequencing problems.

To reduce computational complexity and solve larger test sequencing problems, the algorithm is extended with a probability estimator (PE), the Information Gain (IG) heuristic and a method to divide the test problem into smaller subproblems (DAC). The proposed PE measure reduces computation time drastically while the solution cost is hardly influenced. Therefore we propose to use this measure for every problem. Because the IG heuristic increases the solution cost a lot when the test sequencing problem has many fix action relations, we consider the IG heuristic not suitable for these type of problems. The performance of the DAC measure depends on the decomposition of the problem and can therefore only be used when this decomposition is quite clear and simple.

Apart from reducing test time, a second benefit is that the model provides more insight in the relations between faults and tests. Furthermore, the available test set can be made more
explicit. New tests can be developed that have a better coverage, or that replace multiple other tests.

Integration and test strongly interact with each other. If the realization of a certain module is delayed, tests using this module cannot be performed, while tests concerning other modules can be performed. Also, if the modules are separated, parallel testing is possible, which reduces test time. In our future work, we will investigate how integration and test must be combined in order to determine the optimal test sequence.

2.2 STOPPING CRITERIA FOR TEST SEQUENCING

This section is based on the paper titled *Risk-based stopping criteria for test sequencing* [Boumen et al., 2006c] and is submitted to IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans in 2007. The paper section dealing with the case studies has been removed in this chapter because a more detailed and elaborate description of the case studies can be found in Section 5.1 and in Section 5.4.

The goal of this paper is to answer research questions 1.1 through 1.3 for more advanced test planning problems where the goal of the test plan is to minimize the total sum of end quality and test cost, but also to solve reliability test planning problems.
Risk-based stopping criteria for test sequencing

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

Abstract

Testing complex manufacturing systems, like ASML lithographic machines, can take up to 45% of the total development time. The decision of when to stop testing is often difficult to make because less testing may leave critical faults in the system, while more testing increases time-to-market. In this paper, we solve the problem of deciding when to stop testing by introducing a test sequencing method that incorporates several stopping criteria. These stopping criteria consist of objectives and constraints on the test cost and the remaining risk cost. For a given problem, a suitable stopping criterion can be chosen. For example, with the risk-based stopping criterion, testing stops when the test time exceeds the mitigation of system operation repair time. Furthermore, we show that it is also possible to model reliability problems with this test sequencing method. The method is demonstrated with two case studies at ASML.

2.2.1 Introduction

Testing complex manufacturing systems is expensive both in terms of time and money, as shown in [Cusumano and Selby, 1995; Engel et al., 2004]. As time-to-market for these systems is becoming increasingly important, the decision of when to stop testing becomes more and more important. Less testing may leave undiscovered defects in the system, which increases repair time during system operation, while more testing results in a longer time-to-market. This decision is especially important in the time-to-market driven semiconductor industry and for companies providing manufacturing systems to this industry such as ASML [ASML, 2006; Prins, 2004], a provider of lithographic systems. This is because of the time-to-market pressure of delivering machines to the customer and the high cost associated with solving defects during system operation.

For certain test problems, criteria have been found that help in deciding when to stop testing. These criteria are called test stopping criteria. For example, the SEMI standard [SEMI,
statistically calculates the amount of testing hours needed to show a certain mean time between failures (MTBF) of semiconductor equipment, based on reliability growth models as defined by, for example, Goel and Okumoto [Goel and Okumoto, 1981]. This test stopping criterion is only applicable if there is one test that shows the system reliability and not for a collection of tests that also cover other system requirements, like functional or other performance criteria. Also, if a collection of tests is available that can show the system reliability, for example subsystem tests, it is not clear how many testing hours are required for each test. Mortin et al. [Mortin et al., 1994] suggest to calculate the hours of subsystem testing needed to ensure a certain system reliability assuming that exactly one reliability test is available for each subsystem. However, in practice tests may exist that cover more than one subsystem or certain subsystems may not have their own reliability test.

Brown et al. [Brown et al., 1989] introduce a cost model that determines the optimal number of software test cases based on a probabilistic model that incorporates the cost per test, the error cost, the number of software executions and the estimated number of faults. The main disadvantage of this approach is that no distinction is made between different tests and the fact that these different tests cover different possible faults. This in contrast to Chari and Hevner [Chari and Hevner, 2006] that do incorporate the proportion of a single test case in the total amount of system testing. However, they only select test cases and do not create test sequences, and their approach is limited to software.

Williams and Ambler [Williams and Ambler, 2002] describe a more general test stopping criterion: stop testing if the total cost of testing is minimal, where the total cost of testing is defined as the sum of the test and risk cost. The test cost is the cost made during the test phase: more testing results in more cost. The risk cost is the cost to find and repair problems during system operation, while these problems could have been found during the test phase. Hence, more testing decreases risk cost, because possible faults are found and repaired, as shown in Figure 2.7 taken from Williams and Ambler [Williams and Ambler, 2002]. In this figure, the total cost of testing for a fictitious example is shown. The left side of the figure shows the effect of not testing: the total cost of testing (called overall cost of manufacturing test) is equal to the risk cost (called cost of test escapes). When tests are applied, the test cost (called cost of testing in manufacturing) increases while the risk cost decreases. Therefore, the total cost of testing decreases (after a small increase). However, when more tests are applied the total cost of testing increases again because the risk cost decrease is less than the test cost increase. The right side of the figure shows the effect of testing everything: the total cost of testing is almost equal to the test cost because the risk cost is almost zero. The optimal amount of testing is when the total cost of testing is minimal. The main disadvantage of this test stopping criterion is that it needs a predetermined test sequence. If the test sequence is not known in advance, it is not clear how to determine the stopping moment.

Another test stopping criterion is described by Morali et al. [Morali and Soyer, 2003] who formulate a test stopping criterion for software testing as a sequential decision problem. Their model for the failure rate is based on the same statistical approach as used by the SEMI standards. This approach does not take into account the failure possibilities of tests, like we have done in our previous work, but approximates the effect of fixing faults with the reliability failure rate models.
In our previous work [Boumen et al., 2006d], we have described a method that calculates a time-optimal test sequence excluding all possible faults and thus all risk cost. However, the total cost of testing may be lower when we stop testing earlier and leave a small amount of risk in the system. Therefore, this stopping criteria of excluding all tests may lead to higher total costs. In this paper, we extend the basic test sequencing method presented in [Boumen et al., 2006d] with several stopping criteria to decide the optimal test stopping moment. A stopping criterion consists of an objective function and several possible constraints. This makes it possible to stop testing when for example time has run out, the target quality (risk) is reached or the total test cost is lowest. Furthermore, the impact for a fault state is introduced to calculate the risk per fault state. Also, we demonstrate that reliability problems can be modeled as a test sequencing problem. To do so, we have to use ‘inconclusive’ tests to model reliability tests. Inconclusive tests are not 100% sensitive, meaning that an inconclusive test does not always fail if a covered fault state is present, as opposed to one of the assumptions of the basic system test sequencing problem. We extend the algorithms formulated in [Boumen et al., 2006d] to solve the extended test sequencing problem and introduce heuristics that can be used to solve large (reliability) test sequencing problems.

The structure of the paper is as follows: Section 2.2.2 summarizes the basic test sequencing method and explains the concept of inconclusive tests. Section 2.2.3 explains the risk cost. Section 2.2.4 describes the different stopping criteria that can be defined. Section 2.2.5 introduces an algorithm to solve the defined test sequencing problem. Section 2.2.6 presents the results of two case studies that have been performed at ASML. Section 2.2.8 gives conclusions. Section 2.2.7 shows a table of all used notations and gives a functional description of the algorithm introduced.
2.2. STOPPING CRITERIA FOR TEST SEQUENCING

2.2.2 Test sequencing background

This part of this paper section is intentionally removed from this chapter because a more detailed and elaborate description of the test sequencing method has already been discussed in Section 2.1 and can be found in the original version of this paper.

Inconclusive tests

One of the assumptions of the basic system test sequencing problem defined in [Boumen et al., 2006d] is that a test is 100% sensitive, meaning a test always fails if a covered fault state is present. In real life this may not be the case. An example of a not 100% sensitive test is a ‘reliability’ test that checks the reliability of a system. If such a test passes once, it does not guarantee with 100% that the covered fault states are definitely absent. To model such tests, we extend the model definition with inconclusive tests.

In the literature, some extensions have been made to cope with not 100% sensitive tests. Biasizzo et al. [Biasizzo et al., 1998] describe so-called asymmetrical tests, which are not 100% sensitive. Raghavan et al. [Raghavan et al., 1999b] describe so-called unreliable tests that are not 100% sensitive and not 100% reliable. Sheppard and Simpson have also defined several approaches to describe such tests, see for example [Sheppard and Simpson, 1991]. We propose to call tests that are not 100% sensitive (but are 100% reliable), inconclusive tests, because they do not give a conclusive answer when they pass.

We define the coverage of a test per fault state. If a test passes, the coverage factor of a test, \( p_c \), gives certainty about the absence of a fault state: the uncertainty of the presence of a fault state is decreased by the test coverage. A test that covers a fault state with 100% (a conclusive test) gives 100% certainty about the absence of a fault state (if the test passes) and therefore decreases the so-called uncertainty of a fault state \( p_u \) from 100% to 0%. A test with an 80% coverage of a fault state, gives 80% certainty about the absence of a fault state and therefore decreases \( p_u \) from 100% to 20%. All fault states start with an uncertainty of 100%.

In this paper, we consider non-repeatable and repeatable tests. Non-repeatable inconclusive tests decrease the uncertainty of a fault state just once if they pass. After that one time, the test will always have the same outcome and therefore does not decrease the uncertainty anymore. A repeatable inconclusive test decreases the uncertainty of a fault state each time it is passed. Therefore, it is useful to be applied multiple times. An example of a repeatable test is a reliability test. An example of a non-repeatable test is the measurement of a parameter. A second measurement would not help since the same value would be measured.

To model both repeatable and non-repeatable inconclusive tests, the system model definition is changed. The system model description becomes a sextuple: \((T, S, C, P, R_{ts}, T_r)\), where:

- \( T, S, C, P \) are the same as in the previous test model.
- \( R_{ts} : T \rightarrow \mathcal{P}(S \times \mathbb{R}^+) \) gives for each test in \( T \), a set of covered fault states and their test coverage \( p_c \), which is a number between 0 and 1. This element can again be represented by a matrix \( A \), which denotes the test coverage \( p_c \) for each fault state per test.
Table 2.7: System test model of the telephone with inconclusive tests (and impact)

<table>
<thead>
<tr>
<th>$S$ / $T$</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>$(l)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10 %</td>
</tr>
<tr>
<td>$s_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>10%</td>
<td>(20)</td>
</tr>
<tr>
<td>$s_3$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10%</td>
<td>(20)</td>
</tr>
<tr>
<td>$s_4$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.95</td>
<td>1</td>
<td>10%</td>
</tr>
<tr>
<td>$s_5$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10%</td>
<td>(20)</td>
</tr>
</tbody>
</table>

- $T_r: T \rightarrow \mathbb{B}$ gives for each test in $T$, a boolean that gives true for a repeatable test and false for a non-repeatable test. A non-repeatable test is denoted with a (⋆) for a certain test $t$ (see for example Table 2.7).

In the telephone example we consider $t_5$ as a non-repeatable inconclusive test that has a 95% coverage of fault state $s_4$. The extended system test model of the telephone is shown in Table 2.7.

2.2.3 Risk cost

This section introduces the concept of risk cost for test sequencing. Risk is used for several stopping criteria introduced in the next section.

We define risk in the same way as Williams and Ambler [Williams and Ambler, 2002], who defined risk as the probability that a fault is present multiplied by the impact or repair cost of that fault. Since a fault state is defined as a possible fault with a defined probability, we only have to define the impact per fault state. The impact denotes how important a fault state is. One way of defining this number is to take the time needed to detect and repair the fault state if this fault state is present after the test phase ends. For example, for test sequencing problems during the manufacturing of a lithographic machine, this is the time and cost needed to detect and swap a certain faulty module during operation at the customer. Other definitions are also possible as long as they denote how important a possible fault is.

Formally, the risk cost $j_r$ of a fault state $s$ is denoted by

$$j_r(s) = P'(s)I(s)$$ (2.16)

Here, $P'(s)$ is the actual probability that fault state $s$ is present and $I(s)$ gives the impact of fault state $s$. The initial value of $P'(s)$ is $P(s)$. Depending on the outcomes of tests, this probability decreases or increases (more details can be found in Section 2.2.5). The total risk in the system is the sum of the risks of the individual fault states.

The system test model, as shown in Section 2.2.2, is now extended with impact. The System test model becomes a septuple $(T, S, C, P, R_{ts}, I, T_r)$, where:

- $T, S, C, P, R_{ts}, T_r$ are the same as in the previous test model.
• $I: S \to \mathbb{R}^+$, gives for each fault state in $S$ the impact of that fault state in cost units needed to detect and repair that fault state during operation.

In the telephone example, we assume that the impact of a fault state is 20 time units, which means that if a fault state remains in the telephone and appears during operation, it will take 20 time units to repair it. The extended system test model of the telephone is shown in Table 2.7, where the impact is denoted between brackets.

### 2.2.4 Stopping criteria

We define a stopping criterion as the combination of an objective function and zero or more constraints. There are six possible objective functions that use the following cost functions defined for a solution $G$. Such a solution $G: \mathcal{P}(S) \to T^* \times \mathcal{P}(S)$ gives for each set $S_U$ a test sequence $T_P$ with tests from $T$ and the set of fault states $S_R$ that remain in the system after testing. The test sequence $T_P$ isolates and fixes the fault states in $S_U$ that are not in $S_R$. The expected test cost $J_C$ of a solution $G$ is:

$$J_C^e(G) = \sum_{S_U \subseteq S} P(S_U) \cdot j_C(G(S_U), \mathbf{0})$$

while the maximal test cost of a solution $G$ is:

$$J_C^m(G) = \max_{S_U \subseteq S} j_C(G(S_U), \mathbf{0})$$

$P(S_U)$ denotes the probability that a set of fault states $S_U$ is present and is given by:

$$P(S_U) = \prod_{s \in S_U} P(s) \prod_{s' \in (S \setminus S_U)} (1 - P(s'))$$

By $\mathbf{0}$ we denote the first element of a tuple and by $\mathbf{1}$ the second element. The test cost $j_C$ of a sequence of performed tests $T_P$ is equal to:

$$j_C(T_P) = \sum_{t \in T_P} C(t)$$

The expected risk cost $J_R$ of a solution $G$ is:

$$J_R^e(G) = \sum_{S_U \subseteq S} P(S_U) \cdot j_R(G(S_U), \mathbf{1})$$

The maximal risk cost of a solution $G$ is:

$$J_R^m(G) = \max_{S_U \subseteq S} j_R(G(S_U), \mathbf{1})$$

The risk cost $j_R$ for a certain remaining fault state set $S_R$ is derived from Equation 2.16, and is denoted by:

$$j_R(S_R) = \sum_{s \in S_R} P(s) I(s)$$
Here, \( P'(s) \) is the current probability that fault state \( s \) is present. In Section 2.2.5, it is explained how to calculate this parameter. The expected total test cost \( f'_T \) of a solution \( G \) is:

\[
f'_T(G) = f'_C(G) + f'_R(G)
\] (2.24)

The maximal total test cost \( f'^m_T \) of a solution \( G \) is:

\[
f'^m_T(G) = \max_{S_U \subseteq S} (j_R(G(S_U) \cdot I + j_C(G(S_U) \cdot o)))
\] (2.25)

The total test cost may only be calculated if the risk cost is defined in the same cost units as the test cost since these two costs are added to each other.

The generalized objective is to minimize the cost \( f' \):

\[
f' = \min_{G \in D} J(G)
\] (2.26)

Here, cost \( J(G) \) can be \( f'_C(G), f'^m_C(G), f'_R(G), f'^m_R(G), f'_T(G) \) or \( f'^m_T(G) \) and is chosen beforehand. \( D \) is the set of all possible solutions. Besides the defined objective functions, we define two constraints on the solutions. These constraints define that the test cost or risk cost of a valid solution should not succeed a certain value:

- The maximal test cost constraint: \( f'^m_C(G) \leq n_1 \),

- The maximal risk cost constraint: \( f'^m_R(G) \leq n_2 \),

Here, \( n_1 \) and \( n_2 \) are user-defined.

With a combination of an objective function and zero or more constraints, a stopping criterion can be constructed. Some examples are:

- Determining a test sequence that reduces the risk as soon as possible to 20% of the original risk can be achieved by minimizing the expected test cost \( f'_C(G) \) and by defining a constraint on the maximal risk cost: \( f'^m_R(G) \leq 0.2 \cdot \sum_{s \in S} P(s) I(s) \)

- Determining the test sequence that reduces the risk as low as possible in maximal 1 week of testing can be achieved by minimizing the expected risk cost \( f'_R(G) \) and by defining a constraint on the maximal test cost: \( f'^m_C(G) \leq 1 \) week

- Determining the test sequence that minimizes the total expected test cost can be achieved by minimizing the expected total test cost \( f'_T(G) \) and by defining no constraints.

Several more stopping criteria can be constructed by using the maximal test cost and risk cost. Note that the test cost can be defined in time units or cost units. It is also possible to define the test time as a separate parameter that can be used as objective function or as constraint.
2.2. STOPPING CRITERIA FOR TEST SEQUENCING

2.2.5 Solution algorithm

The algorithm that is used to solve the test sequencing problem is an extension of the original multiple-fault $AO^*_f$ algorithm as defined in [Boumen et al., 2006d]. It performs an AND/OR graph search where an OR node represents the compact set notation of all possible fault state sets that could be present given the previous test outcomes. An AND node represents a test. Furthermore, there are intermediate nodes that represent fix or diagnose actions of fault states. At each OR node all possible AND nodes are investigated. The cost of an AND node depends on the chosen objective function. From all possible AND nodes for a given OR node, the AND node with the least cost is chosen in the solution. The algorithm is equipped with the defined constraints and inconclusive tests.

To implement the stopping criteria, each OR node has an additional leaf node as shown in Figure 2.8, which is the STOP node. In this figure, a square node is an OR node, while a circle node is an AND node and a triangle node is a leaf node. The cost of the STOP node is the risk cost of the OR node $j_R(S_R)$, as defined in (5) where $S_R$ is the set of fault states that have not been excluded. Choosing the successor with minimal cost can only be done by investigating all AND nodes. If one of the AND nodes is a repeatable inconclusive test, it would take infinitely long until all fault states are excluded. Therefore, only the AND nodes of tests that have lower test cost than the risk in the STOP node are investigated. The AND nodes of tests that have higher test cost are always more expensive than the STOP node and are therefore not investigated.

Inconclusive tests are implemented as follows. The original OR node from [Boumen et al., 2006d] is of the form: $X^m \subseteq \mathcal{P}(\mathcal{P}(S)) \times \mathcal{P}(S)$, and consists of multiple candidate sets $S_C$, that each contain at least one fault state present, and one excluded set $S_E$, containing fault states that are certainly not present. The new OR node is of the form: $X^m \subseteq \mathcal{P}(\mathcal{P}(S)) \times \mathcal{P}(S \times \mathbb{R}^+) \times \mathcal{P}(T)$, where the excluded fault state set is extended with the uncertainty $p_u$ for each fault state. If $p_u = 0$, it is 100% certain that the fault state is absent. Furthermore, the test set $T_R$ of non-repeatable tests that have passed is added to the OR node form.

With the running example shown in Figure 2.9, the algorithm is explained in more detail. This figure shows the complete AND/OR graph that is constructed by the algorithm for the example model shown in Table 2.8. In this example, we have chosen to minimize the total
expected test cost with a constraint on the risk cost $J_R^m \leq 0.0$ such that all fault states are removed from the system. The optimal solution is also shown in Figure 2.9 by the non-dashed, bold nodes and edges. The labels of the edges correspond to the cost contribution of the subtree under that node. The percentages on the edges correspond to the pass and fail probabilities of the tests. A square node denotes an OR node which shows the tuple $S_C, S_E, T_R$ for that OR node, or an intermediate node with a diagnosis or fix action of one or more fault states. A circle-shaped node denotes a test. A triangle-shaped node denotes a stop node. The node numbers present in all nodes correspond to the order in which the algorithm investigates the nodes. In Figure 2.10 each step performed by the algorithm to construct the AND/OR graph in Figure 2.9 is described and explained. The step numbers correspond to the node numbers in the AND/OR graph.

The optimal solution to the problem of Table 2.7 without inconclusive tests is shown in Figure 2.11(a). Compared with Figure 2.4(a), an extra stop node is introduced after $t_2$ failed and $s_1$ is fixed. This stop node represents a final situation where $s_4$ has not been excluded and causes 2 hours of risk in the system. To exclude $s_4$, $t_5$ should have been performed which also costs 2 hours. Therefore, it does not matter if $t_5$ is performed or not: both solutions in Figure 2.4(a) and in Figure 2.11(a) are considered optimal solutions for this problem. In the figure, multiple stop nodes are visible because it depends on the outcome of the tests whether or not stopping is allowed and profitable. The optimal solution to the problem of Table 2.7 with inconclusive tests is shown in Figure 2.11(b). This solution shows an extra stop node after $t_5$ has passed. Test $t_5$ partly (95%) covers $s_4$. When $t_5$ passes some risk remains in the system. Excluding that risk costs more than the remaining risk cost.

**Computational reduction measures**

Inconclusive tests increase the computational effort of the test sequencing algorithm drastically. To deal with large test sequencing problems two heuristic measures are proposed.

During reliability testing, we are only interested in the amount of testing that is needed to show a certain reliability. Therefore, the first heuristics only calculates the pass trace of the AND/OR graph. This, so-called, pass-tracing (PT) heuristics does not investigate fail OR nodes.

The second heuristics selects the best next test(s) based on the risk decrease per cost unit. The risk decrease per cost unit $\delta$ for a test $t_i$ at a certain OR node $x$, is defined by:

<table>
<thead>
<tr>
<th>$S$</th>
<th>$T$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$P$</th>
<th>$I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$0.5$</td>
<td>$1$</td>
<td>$25%$</td>
<td>$10$</td>
<td></td>
</tr>
<tr>
<td>$s_2$</td>
<td>$0$</td>
<td>$1$</td>
<td>$25%$</td>
<td>$10$</td>
<td></td>
</tr>
<tr>
<td>$C$</td>
<td>$1$</td>
<td>$3$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.2. STOPPING CRITERIA FOR TEST SEQUENCING

Figure 2.9: Complete constructed AND/OR graph for the example in Table 2.8

\[ \delta(t_j, x) = \frac{j_R(x_p) - j_R(x)}{C(t_j)} \]  

(2.27)

Here, \( x_p \) denotes the OR node if test \( t_j \) passes. This Risk Decrease (RD) heuristic is mainly used for reliability test sequencing problems and resembles risk-based test selection as described by, for example, Harrold and Rothermel [Harrold et al., 2001; Rothermel and Harrold, 1996].

2.2.6 Case studies

This paper section is intentionally removed from this chapter because a more detailed and elaborate description of the case studies can be found in Section 5.1 and in Section 5.4 or in the original version of this paper.
Step-by-step explanation of constructing the AND/OR graph of Figure 2.9

[1] For this unsolved OR node; create a leaf node [2] and for $t_1, t_2$ an AND node [3],[16]; return $J_1 = \min(J_2, J_3, J_9) = J_{19} = 3.75$:

[2] Return the cost of stopping in this leaf node $J_2 = j_R(\{s_1, s_2\}) = 5.0$.

[3] For test $t_1$ create for each test outcome an OR node [4],[12]; return $J_3 = 1.0 + 0.656 \cdot J_7 + 0.344 \cdot J_9 = 3.0$.

[4] For this unsolved pass OR node, create leaf node [5] and an AND node [6] for $t_2$, $t_1$ may not be performed again; return $J_4 = \min(J_3, J_6) = J_6 = 3.0$:

[5] Return the cost of stopping in this leaf node $J_5 = j_R(\{s_1, s_2\}) = 3.75$.

[6] For test $t_2$ create for each test outcome an OR node [7],[9]; return $J_6 = 3.0 + 0.5625 \cdot J_{11} + 0.4375 \cdot J_{17} = 3.75$.

[7] This pass OR node has not been solved, so determine a leaf node [8]; return $J_7 = J_8 = 0.0$:

[8] Return the cost of stopping in this leaf node $J_8 = j_R(\{\}) = 0.0$.

[9] For this unsolved fail OR node, where $T_C = \emptyset$ and $s_c \neq \emptyset$, create an intermediate diagnosis node [10]; return $J_9 = J_{10} = 0.0$:

[10] Determine the new OR node [11] after the diagnosis of $s_1, s_2$; return $J_{10} = J_{11} = 0.0$:

[11] For the unsolved diagnosed OR node create a leaf node [8]; since this node has already been solved, it does not need to be investigated; return $J_{11} = J_8 = 0.0$.

[12] For this unsolved fail OR node, where $s_c = \{s_1\}$; create an intermediate fix node [13]; return $J_{12} = J_{13} = 2.5$.

[13] Determine the OR node after fixing $s_1$ [14]; return $J_{13} = J_{14} = 2.5$:

[14] For this unsolved fixed OR node, create a leaf node [15] and no AND nodes since $T_C(t_2) > j_R$ and $t_1$ has no contribution because $p_u(s_1) = 0.0$; return $J_{14} = J_{15} = 2.5$:

[15] Return the cost of stopping in this leaf node $J_{15} = j_R(\{s_2\}) = 2.5$.

[16] For test $t_2$, create for each test outcome (pass or fail) an OR node [11],[17]; since node [11] has already been solved, it does not need to be investigated; return $J_{16} = 3.0 + 0.5625 \cdot J_{11} + 0.4375 \cdot J_{17} = 3.75$.

[17] For this unsolved fail OR node, create a leaf node [18] and only for $t_1$ an AND node [19] because $t_2$ will always fail; return $J_{17} = \min(J_{18}, J_{19}) = J_{19} = 1.71$:

[18] Return the cost of stopping in this leaf node $J_{18} = j_R(\{s_1, s_2\}) = 11.4$.

[19] For test $t_1$, create for each test outcome (pass or fail) an OR node [9],[12]; since both nodes have been solved, they do not need to be investigated, return $J_{19} = 1.0 + 0.714 \cdot J_9 + 0.286 \cdot J_{12} = 1.71$.

Figure 2.10: Step-by-step algorithm example explanation
2.2. STOPPING CRITERIA FOR TEST SEQUENCING

Figure 2.11: Optimal solutions for telephone examples

(a) Optimal test sequence for Table 2.7 without inconclusive test, \( J' = 5.24 \) (\( J_T = 5.04 \) and \( J_R = 0.20 \))

(b) Optimal test sequence for Table 2.7 with inconclusive test, \( J' = 5.30 \) (\( J_T = 5.03 \) and \( J_R = 0.27 \))
2. TEST PLAN OPTIMIZATION

2.2.7  Functional algorithm description

This section gives a formal, functional-style [Bird, 1998] description of the $AO^*_\sigma$ algorithm with inconclusive test and risk-based optimization.

In Table 2.9, the notations and their descriptions used in this paper are shown.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>Test model: $(T, S, C, P, R_{ts}, I, T_r)$.</td>
</tr>
<tr>
<td>$T$</td>
<td>Set of $k$ tests.</td>
</tr>
<tr>
<td>$S$</td>
<td>Set of $l$ fault states.</td>
</tr>
<tr>
<td>$C$</td>
<td>Gives for each test in $T$ the cost of performing that test.</td>
</tr>
<tr>
<td>$P$</td>
<td>Gives for each fault state in $S$ the \textit{a priori} probability that it is present.</td>
</tr>
<tr>
<td>$R_{ts}$</td>
<td>Gives the subset of fault states that are covered by a test and their coverage (test signature).</td>
</tr>
<tr>
<td>$I$</td>
<td>Gives for each fault state in $S$ the impact of that fault state.</td>
</tr>
<tr>
<td>$T_r$</td>
<td>Gives for each test in $T$ a boolean indicating whether or not the test is repeatable or not.</td>
</tr>
<tr>
<td>$A_{ij}$</td>
<td>$l \times k$ matrix representation of $R_{ts}$.</td>
</tr>
<tr>
<td>$G, G'$</td>
<td>A solution and all possible solutions of the test problem.</td>
</tr>
<tr>
<td>$J, J'$</td>
<td>Total cost and optimal total cost.</td>
</tr>
<tr>
<td>$J_C, J_R, J_T$</td>
<td>Expected test cost, risk cost and total cost.</td>
</tr>
<tr>
<td>$J_{C}, J_{R}, J_{T}$</td>
<td>Maximal test cost, risk cost and total cost.</td>
</tr>
<tr>
<td>$T_p$</td>
<td>Performed (have been performed) test set.</td>
</tr>
<tr>
<td>$S_R$</td>
<td>Remaining fault state set.</td>
</tr>
<tr>
<td>$J_{C}, J_{R}, J_{T}$</td>
<td>Maximal test cost, risk cost and total cost of a solution $G$.</td>
</tr>
<tr>
<td>$T_C$</td>
<td>Candidate (may be performed) test set.</td>
</tr>
<tr>
<td>$x$</td>
<td>A system state in an OR node.</td>
</tr>
<tr>
<td>$t, s, p, c$</td>
<td>A single test, fault state, probability or cost.</td>
</tr>
<tr>
<td>$X^m$</td>
<td>OR nodes domain.</td>
</tr>
<tr>
<td>$H^m$</td>
<td>Function that returns the cost of a solved OR node.</td>
</tr>
<tr>
<td>$H$</td>
<td>Set of solved OR nodes.</td>
</tr>
<tr>
<td>$S_E$</td>
<td>Excluded (definitely not present) fault state set.</td>
</tr>
<tr>
<td>$n_1, n_2$</td>
<td>User-defined variables for the constraints on $J_{C}$ and $J_{R}$.</td>
</tr>
</tbody>
</table>

The set of all possible OR nodes is defined as: $X^m = \mathcal{P}(\mathcal{P}(S)) \times \mathcal{P}(S \times \mathbb{R}^+) \times \mathcal{P}(T)$. This notation is adopted from the compact set notation as defined by Grunberg et al. [Grunberg et al., 1987].

$H^m$ is redefined as a set of function: $X^m \rightarrow (\mathbb{R}^+ \times \{T\} \cup \{\top\}) \cup \{\bot\}$ and gives for a solved multiple-fault OR node the minimal test cost and the next test, or the remaining risk cost and a stop ($\top$) node, or gives undefined ($\bot$) for an unsolved OR node.

The multiple-fault algorithm consists of two functions, $ORm$ and $ANDm$, that are defined for a system test problem $D$. Both functions are explained below.

To find the optimal expected test cost $J'$ for $D$, the following expression is used:

$$ (J', H) = ORm((\emptyset, S_{init}, \emptyset), H_{init}, \emptyset) $$  \hspace{1cm} (2.28)
where:

- \( H_{\text{init}} : X^m \rightarrow \{ \bot \} \) is the initial function that gives the cost of solved OR nodes.
- \( S_{E_{\text{init}}} = \{(s, 1.0) \mid s \in S \} \), is the excluded set of fault states which for all fault states in \( S \) the uncertainty \( p_u \). The initial value of \( p_u = 1.0 \) denotes that the fault state has not been excluded, as explained before.

The resulting \( H \in \mathcal{H}^m \) can be used to construct the optimal solution \( G \) and returns the cost \( J' \) of the optimal solution.

Let function \( ORm : X^m \times \mathcal{H}^m \times \mathcal{P}(T) \rightarrow \mathbb{R}^+ \times \mathcal{H}^m \) be a function that calculates the minimal expected test cost \( J' \) of an OR node and updates the set of solved OR nodes, given the OR node \( x \), the current set of solved OR nodes \( H \) and the performed test set \( T_p \). \( J' \) is:

- 0.0 if \( x \) is terminated,
- derived from \( H \) if \( x \) has already been solved,
- calculated with a fixed OR node if fault states are isolated,
- calculated with a diagnosed OR node if no fault states are isolated and further testing is of no use,
- the current risk cost if this cost is less than the cost of each test from \( T_C \) and the constraint is met,
- determined by investigating the possible AND nodes.

\( x \) is terminated if all fault states are excluded, \( x \) is solved if \( x \) is defined in \( H \) and fault states are isolated if they are within a candidate set of size 1, and \( T_p \) complies with \( R_{st} \). The function is defined as follows:

\[
ORm(x, H, T_p) =
\begin{cases}
(O.0, H) & \text{if } \{s \mid (s, r) \in x.I \land r = O.0\} = S \\
(H(x), O, H) & \text{if } \{s \mid (s, r) \in x.I \land r = O.0\} \neq S \\
& \wedge H(x) \neq \bot \\
(J_F, H_F) & \text{if } \{s \mid (s, r) \in x.I \land r = O.0\} \neq S \\
& \wedge H(x) = \bot \\
(J_D, H_D) & \text{if } \{s \mid (s, r) \in x.I \land r = O.0\} \neq S \\
& \wedge H(x) = \bot \land T_C = \emptyset \\
(J_R, H(x/(J_R, T))) & \text{if } \{s \mid (s, r) \in x.I \land r = O.0\} \neq S \\
& \wedge H(x) = \bot \land T_C \neq \emptyset \\
& \wedge (\forall t: t \in T_C : C(t) > j_R) \\
& \wedge j_R \leq n_2 \\
(J_m, H_m(x/(J, t))) & \text{if } \{s \mid (s, r) \in x.I \land r = O.0\} \neq S \\
& \wedge H(x) = \bot \land T_C \neq \emptyset \\
& (\exists t: t \in T_C : C(t) \leq j_R) \\
& \vee j_R > n_2
\end{cases}
\quad (2.29)
\]
Where:

- $J_F$ is the cost of the fixed OR node and $H_F$ is the updated $H$.
- $J_D$ is the cost of the diagnosed OR node and $H_D$ is the updated $H$.
- $J_R$ is the cost of the current risk in the system if the objective function is minimizing $J_{\text{eR}}, J_{\text{mR}}, J_{\text{eT}}$ or $J_{\text{mT}}$, else $J_R$ is 0.0.
- $T_C$ is the candidate test set consisting of the tests for which the test signature is not a subset of the excluded fault states of $x$ (a certain pass), and none of the candidate sets of $x$ is a subset of the test signature (a certain fail), together with tests that must be performed before fixing a candidate fault state. Moreover, non-repeatable tests in $x$ may not be performed again. In addition, all tests must meet the test cost constraint such that for every $t \in T_C: \sum_{t_i \in T_P} C(t_i) + C(t) \leq n_r$, where $T_P$ is the already performed test sequence.
- $J_T$ is the minimal cost of $x$, and $t_i$ is the test from $T_C$ for which this holds.
- $J = \min(J_T, J_R)$ if $J_R \leq n_2$ else $J = J_T$. Furthermore, $t_j$ is $t_i$ if $J_T$ is the minimum or $t_j$ becomes $\top$ if $J_R$ is the minimum.

Let $\text{AND}_m : X^m \times T \times H^m \times \mathcal{P}(T) \rightarrow \mathbb{R}^+ \times H^m$ be a function that determines the minimal expected test cost $J$ of an AND node and updates $H$, given the OR node $x$, an applied test $t$, the current $H$ and the performed test set $T_P$. This function is already explained in [Boumen et al., 2006d] and therefore not explained in detail.

2.2.8 Conclusions

In this paper, we introduced several stopping criteria for test sequencing. These stopping criteria use the risk in the system to decide when to stop testing. By defining the impact for each fault state, this risk can be calculated. Testing reduces risk while increasing test cost. A stopping criterion can be defined by choosing an objective function and zero or more constraints.

We expanded the test model with tests which are not 100% sensitive, called inconclusive tests, such that reliability or indicative tests can be modeled. Two inconclusive tests are defined: 1) repeatable and 2) non-repeatable tests.

The $AO_t^r$ algorithm introduced in [Boumen et al., 2006d] has been adjusted to solve the test sequencing problem presented with the stopping criteria introduced. Two heuristics are developed that can be used to solve large problems with inconclusive tests.

By performing a case study we have shown that it is possible to calculate the optimal test sequence for a software test phase. This test sequence reduces the risk as much as possible in a given time frame. The optimal sequence reduces the risk with almost 35% in 8 hours testing, while a manually created sequence only reduced the risk with 22% in the same time.
frame. The obtained results of course depend on how accurately the model parameters are estimated. We also showed with a model independent comparison between the manual way of working and the optimal way of working that the last way of working gives better results.

By modeling reliability tests on the subsystem level, the optimal amount of subsystem testing can be calculated. As shown with the second case study, subsystem reliability testing is often faster than system reliability testing. Other techniques (such as SEMI [SEMI, 2000]) are not capable of modeling subsystem reliability tests.

Furthermore, we have shown that the sequence of subsystem reliability tests is important for the decline of the system uncertainty. In general, the total system risk decreases more when for each subsystem a little risk is decreased than when all risk in just one subsystem is decreased.

2.3 HIERARCHICAL TEST SEQUENCING

This section is based on the paper titled *Hierarchical test sequencing for systems* [Boumen et al., 2006e]. This paper has been based on joined work with Professor K.R. Pattipati and S. Ruan from the University of Connecticut, Storrs CT, USA. This work has been performed during a visit of myself to Professor Pattipati from August 2005 until October 2005.

The goal of this paper is to answer research questions 1.1 through 1.3 for test planning problems where the information of the problem is hierarchically described.
Hierarchical test sequencing for systems

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

Abstract

Testing complex manufacturing systems, like the ASML TWINSCAN lithographic machine, is expensive and time consuming. In previous work, a test sequencing method to calculate time-optimal test sequences has been developed. Because today complex systems are composed of several subsystems, which are again composed of several modules, the need exists to model test sequencing problems hierarchically. Such a hierarchical test sequencing problem consists of a high level model describing a test sequencing problem on system level, and one or more low level models describing test sequencing problems on subsystem or module level. The tests on system level correspond to the solutions of the low level problems. This paper describes the hierarchical test sequencing model and proposes two algorithms with heuristics to calculate an optimal test sequence. Benefits of modeling a problem hierarchically are: less computational effort and less modeling effort, both because not all relations are needed. This is shown by a small example. The industrial relevance of this method is illustrated with a case study for the testing phase in the manufacturing of a lithographic machine.

2.3.1 Introduction

As time-to-market for complex manufacturing systems is becoming increasingly important, companies such as ASML [ASML, 2006] develop their systems concurrently, in parallel, to reduce development time. Often, such a system has a hierarchical structure: a number of components forms a subsystem and a number of subsystems forms the complete system. To structure the development, the knowledge about a system is spread over different engineers. This knowledge spread is one of the reasons why integrating and testing these complex systems is difficult and takes a lot of time as is shown in [Cusumano and Selby, 1995; Engel et al., 2004].

This work has been carried out as part of the TANGRAM project under the responsibility of the Embedded Systems Institute Eindhoven, the Netherlands, and in cooperation with several academic and industrial partners. This project is partially supported by the Netherlands Ministry of Economic Affairs under grant TSIT2026.

R. Boumen, I.S.M. de Jong, J.M. van de Mortel-Fronczak and J.E. Rooda are with the Systems Engineering Group, Department of Mechanical Engineering, Eindhoven University of Technology, 5600 MB, Eindhoven, The Netherlands (e-mail: {r.boumen, i.s.m.d.jong, j.m.v.d.mortel, j.e.rooda}@tue.nl). S. Ruan and K.R. Pattipati are with the University of Connecticut, Storrs CT, USA (e-mail: {sruan, krishna}@engr.uconn.edu).
During a test phase in the development or manufacturing of a complex system, decisions have to be made about which components or subsystems must be tested and in what sequence. This problem is almost similar to the problem solved in our previous work [Boumen et al., 2006d,c] where we solved the test sequencing problem. The difference between test sequencing in a system consisting of multiple components and the original test sequencing is that one test model is not sufficient to describe the problem. Because of the knowledge spread, the need exists for defining test models on different levels in a systems hierarchy. For example, on subsystem level an engineer has to decide which component test phases need to be performed and in what sequence this should happen. A component test phase by itself may be the result of a test sequencing problem on a lower level. In this paper, it is shown how to model and solve a test sequencing problem consisting of individual test sequencing subproblems. This problem is called the hierarchical test sequencing problem.

Besides the need for decomposition, because of knowledge spread, there are two more benefits of decomposing a test sequencing problem towards a hierarchical test model. First, modeling effort decreases because not all test dependencies need to be described. Second, because there are less dependencies computation effort decreases. A consequence of the hierarchical approach is that the solution might not be optimal compared to solving the problem with just one (large) test model. These effects are also investigated in this paper.

The structure of the paper is as follows. Section 2.3.2 summarizes our previous work and explains the basic test sequencing method. Section 2.3.3 describes the new hierarchical test model and explains how test phases can be modeled. Section 2.3.4 describes several algorithms to solve the hierarchical test sequencing problem and heuristics to reduce computational effort. Section 2.3.5 shows the effects of a hierarchical model on computational and modeling effort. Section 2.3.6 presents the results of a case study that has been performed at ASML, a provider of lithographic systems. Section 2.3.7 gives conclusions.

### 2.3.2 Test sequencing background

This paper section is intentionally removed from this chapter because a more detailed and elaborate description of the test sequencing method has already been discussed in Section 2.1 or can be found in the original version of this paper.

### 2.3.3 Hierarchical test sequencing

The basic idea of hierarchical test sequencing is that the solution, which is a test sequence, of a low level test sequencing problem can be described as a test in a high level model. This high level test corresponds to the low level solution while a high level fault state corresponds to the set of low level fault states that is covered by tests in the low level model. 'High level' means for example the system level and 'low level' the subsystem level. If a high level model contains a low level model it has a parent-child relation: the high level model is the parent of the low level child model. This modeling is hierarchical in the sense that a child of a parent model, can have its own child. The parent-child relationship is constructed as a tree,
as shown in Figure 2.12. If a test in a high level model corresponds to a low level model, this test is called a group test, otherwise a single test. If a fault state in a high level model corresponds to a low level model, this fault state is called a group fault state, otherwise a single fault state. Each child model corresponds to exactly one group test and one group fault state in the parent model.

The main assumption of hierarchical test sequencing is that a solution to a test sequencing problem, the test sequence, has the same properties as a test, which are: a) a set of covered fault states, b) an outcome and c) a cost of performing. A low level test sequence covers at least the high level group fault state. Also, a test sequence may cover other single or group fault states in the parent model which are not considered in the child model. Furthermore, a group test representing the child test sequence may have the following three outcomes in the parent model:

- **Pass**: the test sequence has been executed successfully and no fault states have been identified.

- **Repair**: the test sequence has been executed successfully and at least one child fault state has been identified and repaired.

- **Fail**: the sequence has been executed with an exception, i.e., a child fault state has been identified and a repair action has been applied but the fault is still present. This occurs when a fault state in the parent model is present that was not considered in the child
test sequencing problem. The failing test sequence is considered as a failing group test in the parent model.

The cost of performing a test sequence depends on the path taken, and therefore on the outcome of the test sequence. If the test sequence passes, its cost is equal to the cost of the pass path in the tree (for example $t_6$ and $t_5$ in Figure 2.4(a)). If the test sequence gives the repair or the fail outcome, the cost is the expected test cost of all paths except the pass path in the solution tree.

The other assumptions for the hierarchical test sequencing problem are:

- There is exactly one test sequencing model on the highest level 0, which is the root of a tree of test sequencing models as shown in Figure 2.12.
- A group fault state covered by a group test that correspond to the same child model can only be repaired by performing the group test, i.e. it is not enough that a test identifies a fault state in the group fault state, but the actual child fault state must be identified using the child tests. To ensure this, the $Rt$ relation must be used (see for example Table 2.11). This relation is denoted with a $\ast$ in the $D$ matrix.
- A parent test only has one child test sequencing problem. A child test sequencing problem is designed to identify one set of child fault states, which is represented as one parent fault state. Not all tests in a high level model correspond to a child test sequencing model; they may also be normal tests.

Hierarchical test sequencing problem definition

In this subsection, the hierarchical test sequencing problem as described above is formalized. The hierarchical test sequencing problem is a model tree, with a root node $H^0$, as can be seen in Figure 2.12. Each node of this tree denotes a single test sequencing model, i.e., $H^k = \{ l^k, S^k, T^k, D^k, C^k, P^k, Rs^k, Rt^k \}$ for $k = 0, ..., K$, which includes:

- $l^k$, is the layer index which indicates the depth of model $H^k$ in the model tree.
- $S^k = \{ s_1^k, s_2^k, ..., s_m^k \}$ is a finite set of $m$ fault states. Each fault state is either a single fault state or a group fault state.
- $T^k = \{ t_1^k, t_2^k, ..., t_n^k \}$ is a finite set of $n$ available tests. Each test is either a single test or a group test.
- A dependency matrix $D^k = \{ d_{ij}^k \}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$, where $d_{ij}^k = 1$ if test $t_j^k$ can indicate whether fault state $s_i^k$ is present, $d_{ij}^k = 0$ if test $t_j^k$ can not indicate whether fault state $s_i^k$ is present and $d_{ij}^k = H^u$ if test $t_j^k$ is a group test that tests group fault state $s_i^k$ by executing the solution of model $H^u$. Then $S^u$ is the detailed set of fault states representing the group fault state $s_i^k$ in model $H^k$, while $T^u$ is the detailed set of tests representing the group test $t_j^k$ in $H^k$. $l^u = l^k + 1$ because $H^u$ is a child model of $H^k$. $H^0$ is defined as the root model, so $l^0 = 0$. 
• For each single test, \( t^k \), there is a cost \( c^k_j \in C^k \); the cost for a group test is unknown and must be calculated with the solution of the child model \( H^u \).

• For each single fault state, \( s^k_i \), there is the a priori probability of failure appearance, \( p^u_i \in P^k \). The probability of the group fault state \( s^k_i \) is equal to the probability that at least one of the fault states in \( S^u \) is present, which is defined by:

\[
p^k_i = 1 - \prod_{s^u_j \in S^u} (1 - p^u_j)
\] (2.30)

• For some fault states, \( s^k_i \), there is a set of fault states \( R^k_s \in R^k \) that is introduced by the fix action of that fault state.

• For some fault states, \( s^k_i \), there is a test \( R^k_t \in R^k \) that needs to be performed before fixing this fault state.

• A fix action of a fault state is 100% reliable.

• For each group test \( t^k \), only one group fault state \( s^k_i \) corresponds to a lower level model \( H^u \).

If test \( t^k \) is a single test, the outcomes are either fail or pass. If \( t^k \) is a group test, representing the lower level model \( H^u \), it has three possible outcomes: pass, fail or repair. These outcomes are defined as follows.

\[
\begin{align*}
\text{Pass} & \quad \text{if } (\forall s^u_i \in S^u : a^u_i = 0) \\
& \quad \quad \land (\forall s^k_i \in \{ s^k_i \in S^k : d^k_{ij} = 1 \} : a^k_i = 0) \\
\text{Repair} & \quad \text{if } (\exists s^u_i \in S^u : a^u_i = 1) \\
& \quad \quad \land (\forall s^k_i \in \{ s^k_i \in S^k : d^k_{ij} = 1 \} : a^k_i = 0) \\
\text{Fail} & \quad \text{if } (\exists s^k_i \in \{ s^k_i \in S^k : d^k_{ij} = 1 \} : a^k_i = 1)
\end{align*}
\] (2.31)

Here, the status of fault state \( s^k_i \) is denoted as \( a^k_i \); \( a^k_i = 0 \) if \( s^k_i \) is not present and \( a^k_i = 1 \) otherwise. The status of a fault state is unknown during the construction of a test sequence and is only used here to explain the concept of the different outcomes of a group test.

**Objective**

A solution \( G^k \) to one test sequencing problem \( H^k \) is a function \( G^k : \mathcal{P}(S^k) \rightarrow (T^k)^* \), which gives for each set of fault states \( S_U \) that could be present a test sequence \( G^k(S_U) \), with tests from \( T^k \) that isolates and fixes every fault state in \( S_U \). The cost of such a solution is:

\[
J(G^k) = \sum_{S_U \subseteq S^k} \sum_{s^k_i \in G^k(S_U)} \left( c^k_j \prod_{s^k_i \in S_U} p^k_i \prod_{s^k_i \in (S^k \setminus S_U)} (1 - p^k_i) \right)
\] (2.32)

Here, \( c^k_j \) for a single test is known and \( c^k_j \) for a group test representing model \( H^u \) must be calculated. The objective is to find an optimal solution \( G^{k*} \) that has the minimal expected test cost \( J^{k*} \) for a model \( H^k \), from all possible solutions \( G^k \):
2.3. HIERARCHICAL TEST SEQUENCING

The solution of the complete hierarchical test sequencing problem consisting of models \( H^0, H^1, ..., H^K \) is \( G^* = (G^0, G^1, ..., G^K) \). The total expected test cost of this solution is \( J^* = J(G^*) \).

Illustration

With a telephone exchange example we illustrate how to model a hierarchical test sequencing problem. The telephone exchange example as shown in Figure 2.13 consists of two telephones, a switch and two cables. The telephones are the same as the telephone shown in the illustration of Section 2.3.2.

We model this problem in two ways: 1) with one non-hierarchical model \( H \) and 2) with three hierarchical models \( H^0, H^1, H^2 \), to make a comparison between the two methods in Section 2.3.5. In Table 2.10, the non-hierarchical model of this problem is shown. In Table 2.11, the high level model \( H^0 \) of the hierarchical model is shown. This high level model considers each telephone as one group fault state that can be tested by a group test. Furthermore, it has two child models \( H^1 \) and \( H^2 \) describing each one telephone. Both \( H^1 \) and \( H^2 \) are the same as the model shown in Table 2.1.

In the hierarchical model, \( t_2^1 \) has a pass outcome if none of the fault states in \( H^1 \) are present (which means that \( s_1^0 \) is not present) and \( s_2^2 \) of \( H^0 \) is not present. \( t_2^2 \) has a repair outcome if at least one of the fault states in \( H^2 \) is present (which means that fault state \( s_1^0 \) is present in model \( H^0 \)) and fault state \( s_2^2 \) is definitely not present. \( t_2^3 \) fails if fault state \( s_2^2 \) is present and it is unknown whether fault state \( s_1^0 \) is present.

2.3.4 Solution algorithm

In this section, solution algorithms are introduced that calculate optimal solutions for the hierarchical test sequencing problem as defined in the previous section. The solution algorithms are all based on an AND/OR graph search as explained in [Boumen et al., 2006d; Shakeri et al., 2000]. The basic test sequencing algorithm that performs this AND/OR
search is shown in Figure 2.14 and is also used by the hierarchical solution algorithms. Before these hierarchical solution algorithms are discussed, some essential equations used in the basic test sequencing algorithm are introduced.

The ambiguity OR node, explained in [Shakeri et al., 2000; Boumen et al., 2006d], is denoted with the compact set notation: \[ X = (L; F_1, ..., F_L; G) \] to denote the multiple-fault ambiguity group at each OR node. The \( F_l \) for \( l = 1, ..., L \) and \( G \) are subsets of \( S \), where \( G \) denotes the set of fault states which are definitely not present, and \( F_l \) for \( l = 1, ..., L \) are the sets that are known to contain at least one present fault state each.

When a group test \( t_{kj} \) relating to group fault state \( s_k \) is applied on an ambiguity OR node \( X \), there are three possible result sets, i.e., \( X_p \) for the pass result, \( X_f \) for the fail result and \( X_r \) for the repair result:

- \[ X_p = (L; F_1 \setminus D(t_{kj}^k), ..., F_L \setminus D(t_{kj}^k); G \cup D(t_{kj}^k)) \]

- \[ X_f = (L + 1; F_1, ..., F_L, D(t_{kj}^k) \setminus \{s_k\}; G) \]

- \[ X_r = (L + 1; F_1 \setminus D(t_{kj}^k), ..., F_L \setminus D(t_{kj}^k), \{s_k\}; G \cup D(t_{kj}^k) \setminus \{s_k\}) \]

Here, \( D(t_{kj}^k) \) denotes the complete test signature of group test \( t_{kj}^k \). Given a test \( t_{kj}^k \) applied on \( X \), the pass probability of that test, \( P_p(X, t_{kj}^k) \), the repair probability of that test, \( P_r(X, t_{kj}^k) \) and the fail probability of that test \( P_f(X, t_{kj}^k) \) are calculated as follows:

---

**Table 2.10: Non-hierarchical model \( H^o \) of the telephone exchange illustration**

| \( S/T \) | \( t_1 \) | \( t_{21} \) | \( t_{22} \) | \( t_{23} \) | \( t_{24} \) | \( t_{25} \) | \( t_{31} \) | \( t_{32} \) | \( t_{33} \) | \( t_{34} \) | \( t_{35} \) | \( t_{4} \) | \( t_{5} \) | \( P \) | \( R_{33} \) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| \( s_{11} \) | 1 | I | I | I | O | O | O | O | O | O | I | O | 10 % | - |
| \( s_{12} \) | I | I | O | I | I | I | O | O | O | O | O | O | I | O | 10 % | - |
| \( s_{13} \) | 1 | I | O | O | O | O | O | O | O | O | O | I | O | 10 % | - |
| \( s_{14} \) | I | I | O | O | O | I | O | O | O | O | O | O | I | O | 10 % | - |
| \( s_{15} \) | I | I | O | O | O | O | I | O | O | O | O | O | I | O | 10 % | - |
| \( s_{2} \) | I | I | O | O | O | I | O | O | O | O | O | O | I | O | 10 % | - |
| \( s_{31} \) | 1 | O | O | O | O | O | O | I | I | O | O | I | O | I | 10 % | - |
| \( s_{32} \) | I | O | O | O | O | O | O | I | O | I | I | I | O | I | 10 % | - |
| \( s_{33} \) | I | O | O | O | O | O | O | I | O | I | I | O | I | O | 10 % | - |
| \( s_{34} \) | 1 | O | O | O | O | O | O | O | I | I | O | O | I | O | I | 10 % | - |
| \( s_{35} \) | I | O | O | O | O | O | O | O | I | O | O | O | I | O | I | 10 % | - |
| \( s_{4} \) | I | O | O | O | O | O | O | I | O | O | I | O | I | O | I | 10 % | - |
| \( s_{5} \) | 1 | O | O | O | O | O | O | O | O | O | O | O | 1 | 1 | 20 % | - |
| \( s_{6} \) | I | O | O | O | O | O | O | O | O | O | O | O | O | I | O | 10 % | - |
| \( s_{7} \) | I | O | O | O | O | O | O | O | O | O | O | O | O | O | O | 10 % | - |
| \( C \) | 10 | 3 | 1 | 1 | 1 | 2 | 2 | 3 | 1 | 1 | 1 | 2 | 2 | 5 | 5 | - | - |
### Basic step-by-step algorithm

**Input:**
- Model: $H = (S, T, D, C, P, Rs, Rt)$

**Output:**
- The optimal solution graph: $G$
- The expected cost of the solution graph $J(G)$

**Step 0:** Initialize a graph $G$ consisting of the root node $X = (0, \emptyset, \emptyset)$, i.e., initial system ambiguity, mark the node as unsolved.

**Step 1:** Repeat the following steps for the root node $X$ to construct an AND/OR graph until the root node is marked solved. Then exit with $J = F(G)$ as expected test cost and the solution graph $G$.

1. **Step 1.0** If $X.0 \neq \emptyset$, do for the elements where the condition $R_{st}(s)$ is fulfilled the following: remove element $s$ (fix $s$) from all elements in $X.0$ and insert $s$ in $X.1$, also remove $R_{ss}(s)$ from $X.1$

2. **Step 1.1** If $X.1$ is $S$, mark $X$ solved in $G$ and assign cost $0.0$ and exit, otherwise determine the candidate test set $T_C$ and perform for each test $t$ in $T_C$ the following steps:

   - **Step 1.1.0** Initialize a subgraph $G'$ consisting of root AND node $t$
   - **Step 1.1.1** Determine for $t$ the pass and fail OR nodes $X_p$ and $X_f$, insert them in $G'$ and draw an edge from $t$ to both of them.
   - **Step 1.1.2** If $X_p$ is not solved, mark $X_p$ unsolved and perform steps 1.0 through 1.2 for $X$ replaced by $X_p$; do the same for $X_f$ (this is the recursion function ORs).
   - **Step 1.1.3** Determine for $t$ the pass and fail probabilities $p_p$ and $p_f$, and assign the cost $p_p \cdot F(X_p) + p_f \cdot F(X_f)$ to $t$.

3. **Step 1.2** Select the test $t$ and the corresponding subgraph $G'$ that has minimal expected cost. Mark $x$ solved and assign the cost $F(t) + C(t)$ to $x$. Merge graph $G$ with subgraph $G'$, create an edge from node $x$ to the root node of $G'$ and exit. If no subgraph is present ($T_C$ is empty), create $X_d$ by removing all elements from $X.0$ and inserting them in $X.1$ (a diagnose OR node as introduced in [Boumen et al., 2006d]). Insert $X_d$ in $G$ and draw an edge from $X$ to $X_d$. If $X_d$ is not solved ($X_d.1 \neq S$), mark $X_d$ unsolved and perform steps 1.0 through 1.2 to solve $X_d$. Then, mark $X$ solved, assign $F(X_d)$ as cost and exit.

Figure 2.14: Basic test sequencing algorithm step-by-step description
Table 2.11: Hierarchical model $H^k$ of the telephone exchange illustration

<table>
<thead>
<tr>
<th>$S$</th>
<th>$T$</th>
<th>$t_1^k$</th>
<th>$t_2^k$</th>
<th>$t_3^k$</th>
<th>$t_4^k$</th>
<th>$t_5^k$</th>
<th>$P$</th>
<th>$R_{ss}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1^0$</td>
<td>$H_1^k$</td>
<td>$t_1^k$</td>
<td>$t_2^k$</td>
<td>$t_3^k$</td>
<td>$t_4^k$</td>
<td>$t_5^k$</td>
<td>$P$</td>
<td>$R_{ss}$</td>
</tr>
<tr>
<td>$S_2^0$</td>
<td>$H_2^k$</td>
<td>$t_1^k$</td>
<td>$t_2^k$</td>
<td>$t_3^k$</td>
<td>$t_4^k$</td>
<td>$t_5^k$</td>
<td>$P$</td>
<td>$R_{ss}$</td>
</tr>
<tr>
<td>$S_3^0$</td>
<td>$H_3^k$</td>
<td>$t_1^k$</td>
<td>$t_2^k$</td>
<td>$t_3^k$</td>
<td>$t_4^k$</td>
<td>$t_5^k$</td>
<td>$P$</td>
<td>$R_{ss}$</td>
</tr>
<tr>
<td>$S_4^0$</td>
<td>$H_4^k$</td>
<td>$t_1^k$</td>
<td>$t_2^k$</td>
<td>$t_3^k$</td>
<td>$t_4^k$</td>
<td>$t_5^k$</td>
<td>$P$</td>
<td>$R_{ss}$</td>
</tr>
<tr>
<td>$S_5^0$</td>
<td>$H_5^k$</td>
<td>$t_1^k$</td>
<td>$t_2^k$</td>
<td>$t_3^k$</td>
<td>$t_4^k$</td>
<td>$t_5^k$</td>
<td>$P$</td>
<td>$R_{ss}$</td>
</tr>
<tr>
<td>$S_6^0$</td>
<td>$H_6^k$</td>
<td>$t_1^k$</td>
<td>$t_2^k$</td>
<td>$t_3^k$</td>
<td>$t_4^k$</td>
<td>$t_5^k$</td>
<td>$P$</td>
<td>$R_{ss}$</td>
</tr>
<tr>
<td>$S_7^0$</td>
<td>$H_7^k$</td>
<td>$t_1^k$</td>
<td>$t_2^k$</td>
<td>$t_3^k$</td>
<td>$t_4^k$</td>
<td>$t_5^k$</td>
<td>$P$</td>
<td>$R_{ss}$</td>
</tr>
<tr>
<td>$C$</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$P_p(X, t_j^k) = \frac{\sum_{S_k^0 \in \Theta(X)} \prod_{S_k^0 \in S_k^0} P(S_k^0) \prod_{S_k^0 \in S_k^0} (1 - P(S_k^0))}{\sum_{S_k^0 \in \Theta(X)} \prod_{S_k^0 \in S_k^0} P(S_k^0) \prod_{S_k^0 \in S_k^0} (1 - P(S_k^0))}$$  \hspace{1cm} (2.34)

$$P_r(X, t_j^k) = \frac{\sum_{S_k^0 \in \Theta(X)} \prod_{S_k^0 \in S_k^0} P(S_k^0) \prod_{S_k^0 \in S_k^0} (1 - P(S_k^0))}{\sum_{S_k^0 \in \Theta(X)} \prod_{S_k^0 \in S_k^0} P(S_k^0) \prod_{S_k^0 \in S_k^0} (1 - P(S_k^0))}$$  \hspace{1cm} (2.35)

$$P_f(X, t_j^k) = 1 - P_p(X, t_j^k) - P_r(X, t_j^k)$$  \hspace{1cm} (2.36)

Here, $\Theta(X)$ gives all possible fault state sets in model $H^k$ that could have resulted in the ambiguity node $X$:

$$\Theta(X) = \{S : S \subseteq S^k \wedge S \cap X.2 = \emptyset \wedge \exists f : f \in X.1 : S \cap f = \emptyset\}$$  \hspace{1cm} (2.37)

Here, $X.1$ denotes the second element of tuple $X$ and $X.2$ denotes the third element of tuple $X$. For a single test, $X_r$ does not exist and therefore $P_r(X, t_j^k) = 0$.

The cost of a group test $t_j^k$ depends on the outcome of the solution of $H^u$ associated with $t_j^k$. Three costs ($C_p$, $C_f$, $C_r$) are defined for each outcome of $t_j^k$:

$$C_p(X, t_j^k) = \sum_{t_i \in G^u(\emptyset)} c_i^u$$  \hspace{1cm} (2.38)

$$C_f(X, t_j^k) = C_f(X, t_j^k) = \sum_{(S_n \subseteq S^k) \wedge S_n \neq \emptyset} \sum_{u \in G^u(S_n)} \left( c_i^u \prod_{i \in S_n} P_{ss}^{i} \prod_{i \in (S^k \setminus S_n)} (1 - P_{ss}^{i}) \right)$$  \hspace{1cm} (2.39)
Then, the expected cost \( c^j_k \) of a group test \( t^j_k \) given an ambiguity node \( X \) is defined as:

\[
c^j_k(X) = c(X, t^j_k) = C_p(X, t^j_k)P_p(X, t^j_k) + C_f(X, t^j_k)P_f(X, t^j_k) + C_r(X, t^j_k)P_r(X, t^j_k)
\]  

In papers [Boumen et al., 2006d; Shakeri et al., 2000], the Information Gain heuristic is proposed to reduce computational effort. If the Information Gain heuristic is used for the hierarchical test sequencing problem, a group test \( t^j_k \) is selected in an ambiguity state \( X \), if it maximizes the Information Gain per unit cost of all possible (group and single) tests \( T^k \):

\[
\frac{IG(X, t^j_k)}{c^j_k(X)} = \max_{t^i_k \in T^k} \frac{IG(X, t^i_k)}{c^i_k(X)} \tag{2.41}
\]

For single tests \( c^i_k \) is already known in \( C^k \). Furthermore, \( IG(X, t^j_k) \) is the Information Gain given by:

\[
IG(X, t^j_k) = - \left( P_p(X, t^j_k) \log(P_p(X, t^j_k)) \right) + P_f(X, t^j_k) \log(P_f(X, t^j_k)) + P_r(X, t^j_k) \log(P_r(X, t^j_k)) \tag{2.42}
\]

We propose two strategies to solve the hierarchical test sequencing problem: bottom-up and top-down. The bottom-up strategy first solves all test sequencing problems on the lowest level, then all test sequencing problems on a higher level, ending with the highest level test sequencing problem. The top-down strategy starts with the highest level problem and calculates, when needed, a lower level test sequencing problem. For the bottom-up strategy, the assumption is made that the cost of a child test sequencing problem is not influenced by the parent test sequencing problem. For the top-down strategy this assumption is not made. In the following subsections it is explained that the computational effort of the bottom-up strategy is less than the computational effort of the top-down strategy. But the cost of a solution obtained by the bottom-up strategy may be higher than the cost of a solution obtained by the top-down strategy. It depends on the size of the problem which strategy should be used. Both strategies are embedded in algorithms which are discussed in the following subsections.

**Bottom-up solution algorithm**

The bottom-up algorithm calculates a solution for every test sequencing problem just once, starting at the lowest level, ending at the highest level. At each level the bottom-up algorithm uses the basic algorithm as described in Figure 2.14 to calculate a single test sequencing problem. A step-by-step description of the bottom-up algorithm is shown in Figure 2.15. This algorithm may use the Information Gain heuristic as described in [Boumen et al., 2006d] to calculate a solution for a single test sequencing problem.
**Bottom-up algorithm**

**Input:**
- Tree of $K$ system Models: $H^k = (l^k, S^k, T^k, D^k, C^k, P^k, R^k, R^t^k)$ for ($k = 0, ..., K$);

**Output:**
- A set of $K$ solution trees: $G_* = (G^{0*}, G^{1*}, ..., G^{K*})$;
- The expected cost of the root solution tree ($G^{0*}$);

**Step 1:** Select a leaf node, i.e., $H^u = (l^u, S^u, T^u, D^u, C^u, P^u, R^u, R^t^u)$, of the complete modeling tree and perform the multiple-fault procedure as shown in Figure 2.14 until the solution $G^u$ and the expected cost $J^u(G^u)$ are obtained;

**Step 2:** If $H^u$ is a child model perform steps 2.1 and 2.2, else $H^o$ is solved so continue with step 4:

**Step 2.1:** Based on the constructed test tree $G^u$ and optimal expected cost $J^u$, calculate the three cost values $C_p, C_f, C_r$ using equations 2.38 and 2.39 and the total fault state probability $p^u$ of all fault states in $S^u$;

**Step 2.2:** identify the $H^u$’s parent model $H^k$, and its position $d^k_{i,j}$ in $D^k$, set $c^k_{i,j} = (C_p, C_f, C_r)$, and $p^k_{i,j} = p^u$;

**Step 3:** Remove $H^u$ from the modeling tree and repeat steps 1, 2 and 3;

**Step 4:** Output the set of solution trees $G_* = (G^{0*}, G^{1*}, ..., G^{K*})$ and the expected cost $J^{0*}$ of solution $G^{0*}$.

**Top-down solution algorithm**

Because of the assumption that a child test sequencing is not influenced by the parent test sequencing problem, the bottom-up strategy will not give the correct cost of a solution. The reason for that is the following. The probability that a fault state is present may increase or decrease because tests fail or pass. If the probability of a group fault state changes, the probabilities of the corresponding child fault states also change. This change in probability influences the solution of the child sequencing problem.

The top-down strategy takes into account that a child sequencing problem can be influenced by the parent sequencing problem. The top-down solution algorithm starts with the highest level model and calculates the solution of a child sequencing problem when needed. Each time a child test sequencing solution needs to be calculated, the new child fault state probabilities are determined based on the information available in the parent ambiguity state. This is done in 2 steps:

- If the group fault state is certainly present, a candidate set is introduced in the child
ambiguity node containing all child fault states. This is done because at least one of them is present.

- If the probability that the group fault state is present is different than the \textit{a priori} probability (but not 100\%), the probabilities of the child fault states are changed. Each child fault state probability is multiplied with the change of the group fault state.

That is, given the group fault state \( s^i_k \) and the parent ambiguity node \( X^k \), the root ambiguity node \( X^u \) for the child model \( H^u \) is determined by:

\[
X^u = \begin{cases} 
(1, \{ S \}, \{ \} ) & \text{if } \{ s^i_k \} \in X^k, i \\
(0, \{ \}, \{ \} ) & \text{otherwise}
\end{cases}
\] (2.43)

Furthermore, \( p'_u^i = p_u^i \times c \), where \( c = p_j^k / p_j^k \) for \( i \in 1, \ldots, n \) and \( j \) is the group fault state. \( p_j^k \) is the current fault state probability defined by:

\[
p_j^k = \frac{\sum_{S_U \in \Theta(X^k) \cup s^k_j \in S_U} \prod_{i \in S_U} p_i^k \prod_{i \in (S_k \cup X^k)} 1 - p_i^k}{\sum_{S_U \in \Theta(X^k)} \prod_{i \in S_U} p_i^k \prod_{i \in (S_k \cup X^k)} 1 - p_i^k}
\] (2.44)

Here, \( X^k \) is the parent ambiguity node. A step-by-step description of the top-down algorithm is shown in Figure 2.16.

\textit{Top-down solution algorithm with revision}\n
The top-down algorithm has to calculate for each step all low level trees and is therefore computationally expensive, while the bottom-up approach may have more costs than the top-down approach. Therefore, we introduce two algorithms that combine the top-down approach with a bottom-up Information Gain approach.

If the Information Gain is used during the top-down algorithm, the cost of each group test must still be calculated by solving the child sequencing problem in each OR node. To prevent this, first the cost of every test sequencing problem is calculated on the bottom-up way, assuming that the cost is not influenced by the parent test sequencing problem. These costs are used to determine the best test according to the Information Gain heuristic. For the best test, we calculate the actual cost, with the top-down approach. Then, we revise the Information Gain of this test and see whether this test has still the highest Information Gain. If so, we continue, if not, the next best test is taken and the cost of this test is revised. In the worst case, all tests are further investigated and we obtain the computational effort of the top-down algorithm without Information Gain heuristic. There are two approaches to revise the cost of a test: either once or constant. The constant revision has the advantage that it detects sooner if another test has a better Information Gain but has the disadvantage that it must calculate for each level in the hierarchy the complete sequence every revision action. Revising once, has therefore less computational effort when the test is still the best test after revision.
**Top-down algorithm**

**Input:**
- Tree of K system Models: \( H^k = \{ l^k, S^k, T^k, D^k, C^k, P^k, R_{s}^k, R_{t}^k \} \) for \( k = 0, \ldots, K \);

**Output:**
- A set of K solution trees: \( G^* = (G^{0*}, G^{1*}, \ldots, G^{K*}) \);
- The expected cost of the root solution tree \( G^{0*} \);

**Step 1:** Select root model \( H^0 \).

**Step 2:** Perform the multiple-fault procedure 2.14 as normal for \( H^0 \); but for every group test perform the following steps

**Step 2.1** For a group test \( t^k_j \), calculate for the associated group fault state the change in fault probability \( c \);

**Step 2.2** Select the corresponding model \( H^u \) and create root node \( X^u \) according to Equation 2.43 and change \( p^u_i \) according to Equation 2.44;

**Step 2.3** Perform step 2 for model \( H^u \) and root node \( X^u \)

**Step 2.4** Based on the constructed test tree \( G^u \) and optimal expected cost \( J^u \), calculate the three cost values \( C_p, C_f, C_r \) using equations 2.38 and 2.39 and the total fault state probability \( p^u_i \) of all fault states in \( S^u \);

**Step 2.5** set \( c^k_j = (C_p, C_f, C_r) \), and \( p^0_i = p^u_i \);

**Step 3:** Output the set of solution trees \( G^* = (G^{0*}, G^{1*}, \ldots, G^{K*}) \) and the expected cost \( J^{0*} \) of solution \( G^{0*} \).

Figure 2.16: Top down algorithm

### 2.3.5 Results

In this section, we perform two experiments to investigate the effect of hierarchical modeling versus normal modeling and to compare the performance of the top-down and the bottom-up algorithms. Both experiments are explained in the following subsections.

**Hierarchical modeling experiment**

In this experiment, we investigate the effects of modeling a problem hierarchically versus non-hierarchically. The solutions of the non-hierarchical and hierarchical telephone exchange model are calculated using the basic and top-down solution algorithms. The expected test cost of the non-hierarchical solution of the telephone exchange model from Table 2.10 is 20.23 cost units. The expected test cost of the hierarchical solution of the telephone exchange
model in Table 2.11 is 21.90 cost units. The solution of model \( H^0 \) is shown in Figure 2.17. Consider that for each \( T_2 \) and \( T_3 \) in the solution tree of model \( H^0 \) in reality the solution tree of \( H^1 \) or \( H^2 \) is performed which is shown in Figure 2.4(a).

Beside the hierarchical model shown in Table 2.11, three different hierarchical models are created of the same telephone example. Each hierarchical test model describes the same problem with a certain amount of models on a certain amount of levels, see the first three columns of Table 2.12 for more details. We compare the hierarchical modeling approaches versus the non-hierarchical modeling approach. We compare the approaches on 3 criteria: 1) modeling effort, 2) solution cost, and 3) computational effort. Modeling effort is denoted as the number of test-fault state relations that have to be filled (number of tests times the number of fault states). The solution cost is the expected test cost of the solution. The computational effort is denoted in the number of OR nodes that have to be investigated to obtain a solution (without heuristics). For each hierarchical model the solution is obtained with the top-down algorithm. The results of this comparison are shown in Table 2.12. In the
2. TEST PLAN OPTIMIZATION

Table 2.12: Results of the hierarchical modeling experiment

<table>
<thead>
<tr>
<th>Properties</th>
<th>Mod. effort</th>
<th>Sol. Cost</th>
<th>Comp. effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>(H )</td>
<td>(I^k )</td>
<td>(K )</td>
<td>(</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>225</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>160</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>95</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
<td>93</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>5</td>
<td>83</td>
</tr>
</tbody>
</table>

Table 2.13: Results of the algorithm performance experiment

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Sol. cost</th>
<th>Comp. effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom-up</td>
<td>21.89</td>
<td>1133</td>
</tr>
<tr>
<td>Bottom-up + IH</td>
<td>25.24</td>
<td>205</td>
</tr>
<tr>
<td>Top-down</td>
<td>21.90</td>
<td>6605</td>
</tr>
<tr>
<td>Top-down + rev + IH</td>
<td>25.24</td>
<td>499</td>
</tr>
</tbody>
</table>

table, the model properties denoted in the number of levels and number of test models, the modeling effort, the solution cost and computational effort are denoted.

The experiment shows that modeling a test sequencing problem with more models on more levels reduces modeling effort and computational effort while the solution cost increase. However, the choice of decomposition also influences the solution and computational effort. Example 2 in Table 2.12 has a high computational effort, while example 4 is the best decomposition in terms of solution cost and computational effort.

Algorithm performance experiment

In this experiment, we investigate the performance of the proposed algorithms. Each algorithm is used to solve the hierarchical problem as shown in Table 2.11. We compare the algorithms based on computational effort and solution cost. The results are shown in Table 2.13. The algorithms used are: bottom-up, bottom-up using the Information Gain, top-down and top-down with revision together with the Information Gain. In this experiment the bottom-up algorithm gives the same solution with less computational effort. This is mainly because it is a relatively small model, where the parent model does not influence the child models.

2.3.6 Case study

This section describes a hierarchical test sequencing problem during the manufacturing of a lithographic machine. During the manufacturing of a lithographic machine, two test phases
are performed: test phase 1 at ASML and test phase 2 at the customer. Both test phases are almost identical. The only difference between the two test phases are the fault state probabilities which are lower in the second test phase since the system has already been tested before. Each test phase consists of multiple job-steps executed in a sequence. Each job-step by itself consists of multiple tests executed in a sequence. During these tests, faults introduced during manufacturing, assembly and transport must be discovered and system parameters need to be calibrated. In a previous project [Boumen et al., 2006d], the basic test sequencing method has been used to compute optimal test sequences for several job-steps. These models are now used again in a complete model of the problem. The total problem is modeled in one high level test model and several low level models. In the highest-level model, each job-step is modeled as a test. For 11 out of 22 job-steps, a low level model is available that describes the individual tests in the job-step. The remaining 11 job-steps are considered as normal tests on the highest level.

In this case study, we calculate test sequences for the two test phases using the top-down algorithm with the Information Gain heuristic. The cost of the solution for test phase 1 is 140.14 hours versus the 148 hours it normally costs. This is a test time reduction of 5%. The cost of the solution for test phase 2 is 112.23 hours versus the 132 hours it normally costs. This is a test time reduction of 15%. These results are also shown in Table 2.14, together with the model properties of each problem.

It is not possible to make a one complete model of this problem on one level, because:

- The knowledge to construct such a model is spread among different people. In the hierarchical case, each model is made by a different person.

- Not all relations are known. The relations between a test in one job step and a fault state in another job step is simply not known. Only on a higher level the relations are known and can be captured in a high level model.

2.3.7 Conclusions

A hierarchical test sequencing method is proposed in this paper that enables the modeler to create test sequencing problems on different levels within, for example, the systems hierarchy. The combination of these individual test sequencing problems and their relations can be captured in a hierarchical test model. A solution can be obtained with the bottom-up or top-down algorithms suggested. The bottom-up algorithm can best be used when the
individual models have little influence on each other, because it has less computational effort. The top-down algorithm can best be used if the individual models influence each other, that is a group test in a high level model has many relations to fault states in that high level model. In these cases, the top-down algorithm obtains a better solution but requires more computational effort.

With a small example the effect of modeling a problem hierarchically is investigated. This investigation showed that dividing a problem results in less computational effort but higher solution cost. It is also shown that the choice of decomposition influences the solution. We did not investigate which decomposition is best for which problems, but we can give some suggestions. The total fail probability of a child model should be around 50% to achieve the highest Information Gain on a higher level (see [Raghavan et al., 1999a] for more information about Information Gain). Also, tests that cover the same fault states must be grouped together. Techniques exists that may help in decomposing a large model. For example, the Mondriaan partitioning algorithm presented in [Vastenhouw and Bisseling, 2005] may help in ordering the $D$ matrix into groups.

Finally, with a case study we also demonstrated that the proposed technique can be used for industrial problems. The optimal test sequences each cost 5% and 15% less than the current sequences. Without the hierarchical test sequencing method we would not be able to model this problem because of the knowledge spread and lack of knowledge for some relations.

2.4 EXTENSIONS

This section describes several extensions to the test plan optimization method as described in the previous three sections. The content of this section has not been published in a paper. Each extension is explained in a separate subsection.

2.4.1 Tests with multiple outcomes

The first extension to the test plan optimization method is that tests may have multiple fail outcomes. An example is a test that checks the productivity of a lithographic machine. If the productivity is less than 125 wafer per hour (wph) (fault state $s_1$) the test gives a not ok (outcome A) as outcome. If the productivity is at least 125 wph the test gives pass as outcome. However, if the system fails at startup because of, for example, a software fault (fault state $s_2$), the test gives fail (outcome B) as outcome. The model of this small example is shown in Table 2.15.

This model deviates from the previous models since this test has multiple fail outcomes (A and B) and a pass outcome. If the productivity test would have had only one fail outcome, no distinction can be made between $s_1$ and $s_2$ when the test fails. However, since this test has multiple fail outcomes (A or B), the test already indicates whether $s_1$ or $s_2$ is present when the test gives the fail outcome. Normally, a test covers a fault state with a certain coverage between 0 and 1. Now, a test covers a fault state with one or more outcomes and for each
outcome a certain coverage is defined between 0 and 1.

Furthermore, we have to define the sequence of outcomes. If both fault states \( s_1 \) and \( s_2 \) are present, the test will not give both outcomes but only one of the two. In the previous example, the test gives outcome B since the system fails at startup and the throughput cannot be measured at all. Therefore, the sequence of outcomes in this example is \([B,A]\). This is also shown in Figure 2.18 depicting the AND/OR graph of this small example. This figure also shows the resulting system states after outcomes A or B.

The extended test model is a nontuple: 
\[
D = (T, S, C, P, R_{ss}, R_{st}, I, T_r),
\]
where:

- \( T, S, C, P, R_{ss}, R_{st}, I, T_r \) are the same as in the test model defined in [Boumen et al., 2006c].
- \( R_{ts} : T \rightarrow \mathcal{P}(S \times \mathcal{P}(\mathbb{R}^+ \times \mathbb{N})) \) gives for each test in \( T \), a set of covered fault states and for each fault state, the set of outcomes (naturals) that the test may have when that fault state is present, and the coverage of that test outcome on that fault state, which is a number between 0 and 1. The order of outcomes is determined by the natural number assigned to the outcome, where the lowest number is the outcome that occurs first (e.g., outcome B has number 1 and outcome A has number 2 in the example).

The algorithm needs to be changed to calculate the AND/OR graph solution as shown in Figure 2.18. Instead of creating one AND node for a test, one AND node is constructed for each fail outcome of a test according to the numbered list of fail outcomes. This is illustrated in Figure 2.19 that shows a detail of the AND/OR graph construction. In this figure, a test with two fail outcomes is split into two tests that are performed after each other. The first test has the same cost as the original test, while the second test has no test cost. The first test has outcome B or pass, while the second test has outcome A or pass. The system states are updated according to the normal rules as defined in [Boumen et al., 2006c].

### 2.4.2 Objectives and constraints

The second extension deals with the objective of the test sequencing problem. The objective of the test sequencing problem is defined in [Boumen et al., 2006d] as follows: minimize the expected test cost, and in [Boumen et al., 2006c] as follows: minimize the sum of expected test cost and expected remaining risk. In [Boumen et al., 2006c], several other possible objectives and constraints are defined. However, by using cost and time even more objectives and constraints can be defined. To incorporate these objectives, the test model is extended.
such that the cost of a test becomes a tuple containing the duration in time units and the cost in cost units. The same is done for the impact and fix cost of a fault state. Therefore, the model becomes a dectuple: $D = (T, S, C, P, R_{ss}, R_{st}, I, T_{r}, F)$, where:

- $T, S, P, R_{ss}, R_{st}, T_{r}$ are the same as in the test model defined in [Boumen et al., 2006c].
- $C : T \rightarrow \mathbb{R}^+ \times \mathbb{R}^+$ gives for each test in $T$ the associated duration in time units and cost in cost units of performing that test.
- $I : S \rightarrow \mathbb{R}^+ \times \mathbb{R}^+$ gives for each fault state in $S$ the impact of that fault state in time units and in cost units needed to detect and repair that fault state during operation.
- $F : S \rightarrow \mathbb{R}^+ \times \mathbb{R}^+$ gives for each fault state in $S$ the associated duration in time units and cost in cost units of repairing that fault state.

With this extended model, it is possible to create numerous objectives and constraints consisting of several variables. A solution $G : \mathcal{P}(S) \rightarrow T^* \times \mathcal{P}(S)$ to the test sequencing
problem gives for each fault state set \( S_U \), a test sequence \( T_P \) with tests from \( T \) and the set of fault states \( S_R \) that remain in the system after testing. The test sequence \( T_P \) isolates and fixes the fault states in \( S_U \) that are not in \( S_R \). For such a solution, the possible variables of an objective function are:

- Expected test cost:
  \[
  EC = \sum_{S_U \subseteq S} \sum_{t \in G(S_U) \cdot o} \left( C(t) \cdot o \prod_{s \in S_U} P(s) \prod_{s' \in (S \setminus S_U)} \left( 1 - P(s') \right) \right) \quad (2.45)
  \]

- Expected test time:
  \[
  ET = \sum_{S_U \subseteq S} \sum_{t \in G(S_U) \cdot i} \left( C(t) \cdot i \prod_{s \in S_U} P(s) \prod_{s' \in (S \setminus S_U)} \left( 1 - P(s') \right) \right) \quad (2.46)
  \]

- Expected remaining risk in cost units:
  \[
  ERC = \sum_{S_U \subseteq S} \sum_{s \in G(S_U) \cdot i} P'(s) I(s) \cdot o \prod_{s' \in S_U} P(s') \prod_{s'' \in (S \setminus S_U)} \left( 1 - P(s'') \right) \quad (2.47)
  \]

- Expected remaining risk in time units:
  \[
  ERT = \sum_{S_U \subseteq S} \sum_{s \in G(S_U) \cdot i} P'(s) I(s) \cdot i \prod_{s' \in S_U} P(s') \prod_{s'' \in (S \setminus S_U)} \left( 1 - P(s'') \right) \quad (2.48)
  \]

- Maximal test cost:
  \[
  MC = \max_{S_U \subseteq S} \sum_{t \in G(S_U) \cdot o} C(t) \cdot o \quad (2.49)
  \]

- Maximal test time:
  \[
  MT = \max_{S_U \subseteq S} \sum_{t \in G(S_U) \cdot i} C(t) \cdot i \quad (2.50)
  \]
2. TEST PLAN OPTIMIZATION

• Maximal remaining risk in cost units:

\[ MRC = \max_{\mathcal{S}_U \subseteq \mathcal{S}} \sum_{s \in G(S_U)} P'(s) I(s)_0 \] (2.51)

• Maximal remaining risk in time units:

\[ MRT = \max_{\mathcal{S}_U \subseteq \mathcal{S}} \sum_{s \in G(S_U)} P'(s) I(s)_1 \] (2.52)

Here, \( .0 \) denotes the first element of a tuple and \( .1 \) the second element. The general objective function of the optimization problem now becomes:

\[ J(G) = c_1 \cdot EC + c_2 \cdot ET + c_3 \cdot ERC + c_4 \cdot ERT + c_5 \cdot MC + c_6 \cdot MT + c_7 \cdot MRC + c_8 \cdot MRT \] (2.53)

Here, \( c_1 \ldots c_8 \) are user-defined coefficients denoting the importance of the different variables. Besides a user-defined objective function, constraints on some of the objective variables may be needed. For example, to create a test phase that reduces as much risk as possible in one week of testing. This requires a maximal test time constraint of one week and an objective function containing the expected remaining risk. Constraints can be defined on \( MC, MT, MRC, MRT \).

With the objective function depending on the user-defined coefficients and the constraints it is possible to define numerous combinations. We only give a few examples:

• A test plan that reduces as much risk as possible in one week can be created using an objective function with \( c_4 = 1 \) (and \( c_1, \ldots, c_3, c_5, \ldots, c_8 = 0 \)) and the constraint: \( MT \leq 1 \) week.

• A test plan that reduces the risk until 20% of the original risk with minimal cost can be created using an objective function with \( c_i = 1 \) (and \( c_2, \ldots, c_8 = 0 \)) and the constraint: \( MRC \leq (0.2 \cdot \sum_{s \in S} P(s) I(s)) \).

• A test plan of maximally $100.000 and maximally 1 week that reduces the risk cost to maximally $20.000 in the shortest amount of time can be created using the objective function with \( c_5 = 1 \) (and \( c_1, \ldots, c_4, c_7, c_8 = 0 \)) and three constraints: \( MT \leq 1 \) week, \( MC \leq $100.000 \) and \( MRC \leq $20.000 \). Note that this objective and these constraints may not yield a valid solution since a test plan that is able to comply to all constraints does not necessarily exist.

2.4.3 Fix strategies

Another extension to the test plan optimization methods is the use of different fix and diagnosis strategies. In [Boumen et al., 2006d], all fix actions are performed sequentially and
immediately when possible, while diagnosis actions are performed sequentially when no more tests are available. In [Boumen et al., 2006c], a strategy is introduced as a computational reduction measure that only calculates the pass possibilities of a test and not the fail possibilities, such that fix and diagnosis actions are never performed. In this subsection, two other strategies are introduced: parallel fixing and fixing as late as possible (ALAP). An overview of all four possible fix and diagnosis strategies is given below:

**SEQUENTIAL** Fix and diagnosis actions are performed sequentially after tests. Fix actions are performed immediately (when a fault state is present), diagnosis actions when no more tests are possible. This strategy is used in situations where fixes can be applied immediately and should be performed before testing can continue, for example during hardware testing.

**PARALLEL** Fix and diagnosis actions are performed in parallel with tests. Fix actions are started immediately (when a fault state is present), diagnosis actions when no more tests are possible. If a test is performed that covers a fault state that at that moment is fixed or diagnosed, the test has to wait until the fix or diagnosis action has finished. This strategy is used in situations where fixes must first be developed and may take long, while testing can continue.

**ALAP** Fix and diagnosis actions are performed sequentially as late as possible. Both fix and diagnosis actions are performed when no more tests can be performed. This strategy can be used when it is not possible to apply fixes during testing. For example, when new hardware has a long delivery time or when a test sequence is performed automatically without operators (at night). Testing continues until no more tests are possible and a list of faults needs to be solved before a new test sequence can be started.

**NEVER** Fix and diagnosis actions are never performed. This strategy is used in situations where only the pass trace of a test sequence is interesting, for example to determine which tests are needed to reach a certain risk level.

With a small example, the different strategies are illustrated. The test model of this example is shown in Table 2.16 and contains four tests. The test and fix cost and impact in this model are expressed in time units. In Figures 2.20(a) through 2.20(d) the solutions for the different fix strategies are shown. The sequential solution is the same as in [Boumen et al., 2006d]. In the parallel solution, the square nodes denote the start of a fix action. Subsequent tests are performed in parallel with a fix unless the test covers the fixed fault state. Then, the test must wait for the fix. In this particular solution, no test has to wait for a fix. The as late as possible (ALAP) solution, denotes all faults that are certainly present at the end of testing. This solution may look like a single fault solution (see [Boumen et al., 2006d]) but is essentially different since the multiple-fault approach is conducted. In the never fix solution, only the pass trace is shown that excludes the fault states as soon as possible.
Figure 2.20: Solutions for different fix strategies
2.4. Extensions

### Table 2.16: Fixing strategy example

<table>
<thead>
<tr>
<th>$S / T$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$P$</th>
<th>$R_{ss}$</th>
<th>$I_1$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>10%</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$s_2$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>10%</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$s_3$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10%</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$C_f$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 2.4.4 Computational reduction measures

In [Boumen et al., 2006d,c], several computational reduction measures are introduced. Both the Information Gain (IG) and Risk Decrease (RD) heuristics can be used to reduce the number of investigated tests at an OR node. These heuristics can be used in two modes: absolute or relative. When either the IG or RD heuristic is used in absolute mode, a predefined number $n$ best tests is selected every OR node. When they are used in relative mode, a predefined percentage $p$ of the best tests is selected every OR node. Given a set of tests $T$ that is analyzed at an OR node, the investigated tests $T'$ are:

- $T' = \{ t \in T : \{ t' \in T : IG(t') \geq IG(t) \} \leq n \}$ for the absolute mode (equal for the RD heuristic)
- $T' = \{ t \in T : IG(t) \geq p \cdot \max_{t' \in T} IG(t') \}$ for the relative mode (equal for the RD heuristic)

The advantage of the relative mode is that when the IG or RD values are close to each other, more tests are investigated while if the maximal value is much larger than the other values only a few (or even one) tests are investigated. This is also the disadvantage of the relative mode, since it is never known in advance how many tests are investigated and thus how many calculations need to be performed.

#### 2.4.5 Other test model applications

The test plan method as introduced in this chapter uses the test model. Beside this method, other methods developed in the Tangram project use this test model. In the following overview, these other methods are briefly discussed.

**Next best test case** Given a test model consisting of tests, fault states and the test coverage of fault states, it is possible to calculate which new test would make the model more testable. In [de Jong, 2007], this method is explained in more detail and applied to some examples and a case study. This method is convenient when, for example, there is budget to develop a new test, and a decision must be made what the test should cover.

**Test process simulation** Given a test model and a test process configuration it is possible to simulate the test process. With the test process simulation it is possible
to analyze the risk in the system and to determine the variation in the end time. In the process model, variability on test execution, diagnosis and fix actions can be taken into account. In [de Jong et al., 2006], the test process simulation is described in more detail and applied to examples and case studies.

**Test Model Partitioning** To parallelize test phases, the test model partitioning method can be used to split a test model (matrix) into two separate model matrices that can be executed in two separate test phases, either sequentially or in parallel. The test model partitioning method is based on existing matrix partitioning methods and is described in more detail in [de Jong, 2007]. This method can also be used to create hierarchical test models.

### 2.5 Conclusions

This chapter has answered research questions 1.1, 1.2 and 1.3 as defined in the first chapter:

**Answer to Question 1.1:** The structure of a test plan is defined as a tree of test sequences combined with fix and diagnosis actions. The sequence that is followed depends on the outcome of the executed tests. Furthermore, each sequence has a certain stop moment. Since we consider one system (one resource), all tests are executed sequentially, fix and diagnosis actions can sometimes be executed in parallel to test actions. A plan has expected and maximal test cost, test time and remaining risk. A plan is judged on these parameters.

**Answer to Question 1.2:** The information needed to construct a test plan is the test model consisting of tests and their properties, fault states and their properties and the relation between these elements. This model defines the test problem. Furthermore, an objective function consisting of several parameters must be provided that is used to determine the optimal plan. Also, constraints can be defined for certain parameters.

**Answer to Question 1.3:** The method creates an optimal test plan, given the test model and objectives and constraints, is constructing an AND/OR graph containing all possible solutions. In this graph, an AND node denotes a test and an OR node denotes a system state consisting of the possible fault states that can be present in the system. The best solution is chosen using an AND/OR graph search algorithm that is adjusted for this specific problem.

With the test plan optimization method we are able to optimize test plans for different test domains, such as manufacturing test phases, reliability test phases and performance test phases. Furthermore, this method reduces the effort of creating and maintaining test plans because optimization is performed automatically. The test model gives insight in the test problem and can be used as a knowledge container to train new test engineers. Furthermore, the method can be used to construct ‘what-if’ scenarios that can show the benefits of certain new investments. In Chapter 5, several case studies show how this method can be used in real life, and show the benefits of using this method.
This chapter introduces a method for optimizing integration plans; or in what sequence should we bring subsystems together to create the complete system as soon as possible. We abstract from the test phases and concentrate on integrating modules with each other. This chapter is therefore illustrated with two puzzle pieces that can be put together or integrated. By combining the two puzzle pieces, functionality is added and more tests can be applied. By creating independent puzzle subsystems, we can work in parallel and create the puzzle faster.

Integration sequencing problems arise in system development but also in software releasing and in manufacturing. Section 3.1 is based on a paper that describes the integration planning method. Section 3.2 deals with the hierarchical aspect of this problem. Section 3.3 describes several extensions not described in the paper. The last section gives conclusions about this chapter.

3.1 INTEGRATION SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

This section is based on the paper titled Integration sequencing in complex manufacturing systems [Boumen et al., 2006a] and is submitted to IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans in 2006. The paper section dealing with the case studies has been removed in this chapter because a more detailed and elaborate description of the case studies can be found in Section 5.2.
Integration sequencing in complex manufacturing systems

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

Abstract

The integration and test phase of complex manufacturing machines, like an ASML [ASML, 2006] lithographic machine, are expensive and time consuming. The tests that can be performed at a certain point in the integration phase depend on the modules that are integrated. Therefore, the test sequence depends on the integration sequence. Thus, by optimizing the integration sequence of these modules, more tests can be done in parallel and valuable integration and test time can be reduced. In this paper, we introduce a mathematical model to describe an integration sequencing problem and we propose an algorithm to solve this problem optimally. Furthermore, we propose two heuristics to solve large industrial problems in limited computation time.

3.1.1 Introduction

In today's industry, time-to-market of systems is more and more important. Therefore, the development of systems is done concurrently. During the integration phase of a system, the different subsystems are assembled 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 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 plan is currently made by hand and is usually not optimal for time. Creating an optimal integration plan could decrease integration and test time and planning effort.

In the literature, integration plans for complex systems are seldom discussed. In general, the basic integration test phases identified are unit testing, integration testing, system testing and acceptance testing as for example explained in [Graig and Jaskiel, 2002]. However,
this integration and test plan does not define the sequence of integration: which modules are integrated when? It only states that tests should be done before and after integration. More research is done in the software discipline on integration strategies. The three basic strategies are: "big bang", where all modules are integrated at once, "top-down" where stubs are used to simulate the behavior of low level modules while testing the high level modules first, and "bottom-up" where first all lower level modules are tested and then the high level module is integrated [Beizer, 1990]. The disadvantage of these three strategies is that none of them is optimal given a certain integration problem: a combination of them is optimal.

For object-oriented software, Hanh et al. [Hanh et al., 2001] developed a technique that tries to either minimize the number of stubs used or the total test resource allocation (test effort). This is done using a test-dependency-graph model describing the modules and their test dependencies. This approach tries to optimize an integration strategy towards a combination of "big-bang", "bottom-up" and "top-down" strategy. The disadvantage of this approach is that it has only been used for object-oriented programming and that it does not take into account that certain modules are not physically present at the start time of the integration phase. Furthermore, it only considers a single system under test, while multiple subassemblies could be used as systems under test to test in parallel.

In the mechanical assembly disciplines, methods and algorithms do exist that take late deliveries and parallel assemblies into account. For example, De Mello et al. [de Mello and Sanderson, 1991b,a] describe how to represent and optimize mechanical assembly sequences such that robot assembly actions are minimized. Also, Boneschanscher [Boneschanscher, 1993] describes how to use these techniques to optimize assembly plans. However, these methods have never been applied to integration and test plans. The difference between assembly and integration is that the cost of an assembly plan is dependent on the robot movements, while the cost of an integration plan is dependent on the tests that are performed.

In this paper, we extend the mechanical assembly sequencing method and algorithm towards an integration sequencing method. This is done by combining the assembly sequencing representation with the test-dependency-graph model of Hanh and by extending the AND/OR graph algorithms as described by De Mello et al. We incorporate for certain tests that stubs or simulation models can replace modules that are not yet available. This way, we can optimize an integration plan with respect to time while taking into account that certain models can replace modules as is proposed by Braspenning et al. [Braspenning et al., 2006].

The paper is structured as follows. Section 3.1.2 explains system integration sequencing and shows an example. Section 3.1.3 shows the formal definition of an integration sequencing problem and the objective for solving this problem. Section 3.1.4 shows the algorithm used to solve the integration sequencing problem. In Section 3.1.6 a case study is conducted to show the benefits of applying this method on the development phase of a lithographic machine. Section 3.1.9 provides the conclusions. Section 3.1.7 shows the notations used in Section 3.1.8, which shows a formal and a step-by-step description of the algorithm.
3. INTEGRATION PLAN OPTIMIZATION

3.1.2 System integration

System integration deals with assembling modules into a working system. To ensure that the system works, tests have to be performed that test the functionality and performance of the system. In the first subsection, we describe several system integration problems. In the second subsection, we introduce an illustration of an integration problem which is used throughout the paper.

Integration problem

Integration problems can be found in many different development phases of systems that all have their specific properties. The three main integration problems during the development and manufacturing of ASML lithographic machines are:

- Software integration
- First-of-a-kind machine integration
- Manufacturing integration

During software integration, multiple copies of the software are available and the assembly of two software modules into one system does not take any (significant) time. Often, stubs can be used to perform certain tests early, when modules are not yet available.

For the first-of-a-kind machine integration, the development cost of each module is extremely high and only one or maybe two modules of (almost) the same type are available. Furthermore, the assembly of two modules into a single system takes a significant amount of time.

For manufacturing integration, a large number of machines is created which means that multiple modules of the same type are available. Also, assembling modules takes time.

In general, an integration phase consists of assembling a number of modules, that are available at a certain time, into one system and performing a number of tests, each of which takes some time to perform and requires a certain set of modules to be assembled. To reduce the overall integration time, development, assembly and test actions should be performed in parallel whenever possible. The integration sequencing problem is defined as choosing an assembly sequence of which the duration of the critical path is minimal which means that the development, assembly and test actions are performed in parallel as much as possible. The critical path is defined as the longest duration path in the integration sequence.

Illustration

To illustrate an integration sequencing problem we use a simplified ASML wafer scanner. A scanner performs the lithographic step in the manufacturing of a semiconductor or IC. Two items are cycling through a scanner: a wafer that is the basis of the IC and contains a photo resistant, and a reticle that contains a part of the negative of an IC [Shon-Roy et al., 1998]. The simplified scanner consists of 7 modules: a reticle handler \( m_1 \) that brings
and takes reticles to the reticle stage \( m_2 \) which holds the reticle during the lithographic process, a wafer handler \( m_3 \) that brings and takes wafers to the wafer stage \( m_4 \) which holds the wafer during the lithographic process, a laser \( m_5 \) that produces the light needed for the lithographic process, an illuminator \( m_5 \) that makes the light produced by the laser uniform and a lens \( m_7 \) that shrinks and images the pattern from the reticle on the wafer. Reticles and wafers are not considered part of the machine in this example. In this machine 3 essential types of interfaces exist: the light interfaces \( i_1, i_2, i_4, i_5 \) which connect the laser to the illuminator to the reticle stage to the lens to the wafer stage, the reticle flow interface \( i_3 \) which connects the reticle handler to the reticle stage and the wafer flow interface \( i_6 \) which connects the wafer handler to the wafer stage.

This illustration of an integration sequencing problem deals with the integration phase of a first type of a scanner. Besides the mentioned modules and interfaces, for this scanner, a model of a lens \( m_8 \) is available which can be used for certain tests. This model also has light interfaces \( i_7 \) and \( i_8 \) with the reticle handler and the wafer handler. The development time for each of the modules is known. For example, the development time of the lens model is 5 time units, while the development time of the actual lens is 25 time units. Also, the integration of two modules into a subsystem takes time. Furthermore, 25 tests have to be performed that each require certain modules to be integrated and cost a certain time to perform. The objective is to find an integration sequence that integrates each module into a complete system such that the overall integration and test time is minimized.

### 3.1.3 Problem formulation

In this section, we formalize the integration sequencing problem. We define a model in the first subsection, show an illustrative model in the second subsection and define the objective
Integration model

The formal integration model consists of two parts: 1) the assembly part describing the system based on models from De Mello et al. [De Mello and Sanderson, 1991b,a], and 2) the test part describing tests and the required modules based on ideas from Hanh et al. [Hanh et al., 2001]. The complete integration sequencing model is an octuple $D$: ($M$, $I$, $T$, $C_m$, $C'_i$, $C_t$, $R_{im}$, $R_{tm}$), where:

- $M$ is a finite set of $k$ modules.
- $I$ is a finite set of $l$ interfaces.
- $T$ is a finite set of $m$ tests.
- $C_m : M \rightarrow \mathbb{R}^+$ gives for each module in $M$ the associated cost in time units of developing that module.
- $C'_i : I \rightarrow \mathbb{R}^+$ gives for each interface in $I$ the associated cost in time units of constructing that interface.
- $C_t : T \rightarrow \mathbb{R}^+$ gives for each test in $T$ the associated cost in time units of performing that test.
- $R_{im} : I \rightarrow M \times M$ gives for each interface in $I$ the modules the interface is constructed in between.
- $R_{tm} : T \rightarrow \mathcal{P}(\mathcal{P}(M))$ gives for each test in $T$ a set of essential assemblies; where an essential assembly is a set of 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 $M$.
- For every test in $T$, there exists a module that is present in all essential assemblies of this test.
- Each test is performed exactly once at the moment that one of the essential assemblies of this test is integrated.

Furthermore, we define that an assembly consists of one or more modules which are integrated with each other and is therefore represented by an element of $\mathcal{P}(M)$ (except $\emptyset$). An integration action is defined as instantiating all interfaces between exactly two assemblies and is therefore represented by an element of $\mathcal{P}(I)$ (except $\emptyset$). A test phase consists of the set of tests that can be performed on a subassembly, but could not be performed before the last integration action. A test phase is therefore represented by an element of $\mathcal{P}(T)$ (except $\emptyset$).
3.1. INTEGRATION SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

Figure 3.2: Elements $M$, $I$, $C^m$, $C^i$ and $R^{im}$ of the integration model

Illustration

For the scanner illustration presented in Section 3.1.2, an integration model is created. Elements $M$, $I$, $C^m$, $C^i$ and $R^{im}$ are represented in Figure 3.2 (the integration part). In this figure, a square node is a module and an edge is an interface between two modules. Elements $T$, $C^i$ and $R^{im}$ are shown in Table 3.1 (the test part).

Objective

A solution to an integration sequencing problem is a sequence of test phases and integration actions for each individual module. This solution can be represented by a function $G : M \rightarrow (P(T) \cup P(I))^*$. ($B^*$ denotes a set of sequences over $B$), that gives for a subassembly consisting of a single element of $M$, a sequence of integration actions (sets of interfaces $P(I)$) and test phases (sets of tests $P(T)$) that integrates this single module subassembly into the completely integrated and tested system. The total time of such a solution is:

$$J(G) = \max_{m \in M} \left( C^m(m) + \sum_{i \in G(m) \upharpoonright T} C^i(t) + \sum_{i \in G(m) \upharpoonright I} C^i(i) \right)$$ (3.1)

Here, $G(m) \upharpoonright I$ takes all elements from $G(m)$ that are a subset of $I$ and uniforms them into one set. The objective is to find an optimal solution $G^*$ that has minimal expected test
Table 3.1: Elements $T$, $C$ and $R^{im}$ of the integration model

<table>
<thead>
<tr>
<th>$T$</th>
<th>$R^{im}$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1 \cdots t_6$</td>
<td>${{m_1}}$</td>
<td>1</td>
</tr>
<tr>
<td>$t_7 \cdots t_8$</td>
<td>${{m_1, m_2}}$</td>
<td>2</td>
</tr>
<tr>
<td>$t_9 \cdots t_{10}$</td>
<td>${{m_4}}$</td>
<td>1</td>
</tr>
<tr>
<td>$t_{11} \cdots t_{13}$</td>
<td>${{m_2, m_4}}$</td>
<td>2</td>
</tr>
<tr>
<td>$t_{14} \cdots t_{17}$</td>
<td>${{m_5, m_6, m_7}, {m_5, m_6, m_8}}$</td>
<td>3</td>
</tr>
<tr>
<td>$t_{18} \cdots t_{20}$</td>
<td>${{m_2, m_4, m_5, m_6, m_7}}$</td>
<td>3</td>
</tr>
<tr>
<td>$t_{21} \cdots t_{25}$</td>
<td>${{m_1, m_2, m_3, m_4, m_5, m_6, m_7}}$</td>
<td>5</td>
</tr>
</tbody>
</table>

Cost $J^*$, from all possible solutions $\mathcal{G}$:

$$J^* = \min_{G \in \mathcal{G}} J(G)$$  \hspace{1cm} (3.2)

3.1.4 Solution algorithm

In this section, we propose an algorithm to solve the integration sequencing problem. This algorithm is based on the “assembly by disassembly” approach suggested by Delchambre et al. [Delchambre et al., 1989]. The approach takes the assembled, or in this case integrated, system as starting point and evaluates all possible sequences in which the complete system can be disassembled. De Mello et al. [de Mello and Sanderson, 1991b,a], suggests to implement this approach in an algorithm that performs a search through an AND/OR graph. In this AND/OR graph, an OR node represents a system state and an AND node represents an assembly action.

To solve the integration sequencing problem we also propose an algorithm based on an AND/OR graph search. However, different definitions of the AND and OR nodes are used as opposed to De Mello. An OR node represents the system state $x$ which is the set of integrated modules, and is defined as an element of $\mathcal{P}(M)$. In a system state, tests are applied that are only possible in that state and not in following states, according to the assumptions made in Section 3.1.3. The initial system state is $x_{initial} = M$. An AND node defines the integration action of two OR nodes into one OR node and is denoted by the set of interfaces that are broken, which is an element of $\mathcal{P}(I)$.

An example of an AND/OR graph is shown in Figure 3.3. 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 ($t_4$ requires $m_1$, $t_5$ requires $m_2$, $t_6$ requires $m_3$) and one system test $t_4$ that requires all modules. The graph in Figure 3.3 shows all possibilities in which the system can be disassembled, and therefore contains all solutions
3.1. INTEGRATION SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

Figure 3.3: AND/OR graph illustration

to the integration sequencing problem. In this example, there are three possible solutions 
\((G_1, G_2, G_3)\) that can be distinguished at the root OR node where there are 3 AND nodes to 
choose from. For each solution \(G\) the cost \(J(G)\) can be calculated, then the cheapest solution 
is chosen.

The algorithm that constructs AND/OR graphs is a depth-first algorithm. The basics 
of the AND/OR search algorithm are explained in this section. The complete algorithm 
is described in Section 3.1.8. The search starts with the initial OR node that denotes the 
complete integrated and tested system: \(x_{init}\). The cost of this particular OR node is called 
\(J_s(x_{init})\) and is determined as follows.

First, the possible set of disassembly actions is determined. The set of all possible inte-
gration actions \(A_x\) are all cut-sets that separate the integrated system with system state \(x\) into 
exactly two unique subsystems \((x_1, x_2)\). For a given system state \(x\), this can be determined 
as follows:

\[
A_x(x) = \{a : a \subseteq I_x : \\
\exists(x_1, x_2) \subseteq x : \\
x_1 \cap x_2 = \emptyset \land x_1 \cup x_2 = x \\
x_1 \neq \emptyset \land x_2 \neq \emptyset \land \forall(m', m'') \in x_1 : connected(m', m'', I_x \setminus a) \\
\land \forall(m', m'') \in x_2 : connected(m', m'', I_x \setminus a) \\
\land \exists m' \in x_1, m'' \in x_2 : connected(m', m'', I_x \setminus a) \}
\]

(3.3)

Here, \(I_x(x)\) denotes all interfaces between the modules in system state \(x\) and function
**connected** checks whether two modules are connected, i.e. there exists a path of interfaces between two modules. Cut-set algorithms exist in the literature that determine all possible cut-sets of a system in linear time per cut-set. We use the algorithm as described by Tsukiyama [Tsukiyama et al., 1980]. For the simple system of which the AND/OR graph is shown in Figure 3.3, the cut-sets are: \( \{i_1, i_2\}, \{i_2, i_3\}, \{i_1, i_3\} \). For the system illustrated in Figure 3.2, the possible cut-sets are: \( \{i_1\}, \{i_2\}, \{i_3\}, \{i_4, i_7\}, \{i_8, i_5\}, \{i_6\}, \{i_4, i_5\}, \{i_7, i_8\} \).

Then, for all cut-sets \( a \in A_x(x) \) given system state \( x \), an AND node is constructed. This AND node represents the disassembly of a system state into two system states \( x_1 \) and \( x_2 \) (determined in equation 3.3), by breaking the interfaces in \( a \). For each of the two system states, \( x_1 \) and \( x_2 \), the tests that can still be performed, \( T_{x_1}(x_1) \) and \( T_{x_2}(x_2) \), can be calculated using:

\[
T_x(x) = \{ t : t \in T : (\exists M' \in R^m(t) : M' \subseteq x) \}
\]

(3.4)

Then, the required tests \( T_r(x) \) that have to be performed in system state \( x \) can be calculated with:

\[
T_r(x) = (T_x(x) \setminus (T_x(x_1) \cup T_x(x_2)))
\]

(3.5)

The total cost of an AND node \( J_a(x, a) \) for a system state \( x \) and a disassembly action \( a \in A_x(x) \) which disassembles the system into two subsystems \( x_1, x_2 \), is defined as the maximal integration and test cost of each formed system state \( x_1 \) and \( x_2 \), and the cost of the disassembly of the system state \( x \) into the two system states, or:

\[
J_a(x, a) = \sum_{i \in a} C_i(i) + \max(J_{x_1}(x_1), J_{x_2}(x_2)) \]

(3.6)

Finally, the cost of the OR node is the minimal cost of each AND node that is constructed and the cost of performing the required tests \( T_r(x) \), or the development cost of a module if one module remains and the cost of the tests \( T_x(x) \) that can still be performed in system state \( x \):

\[
J_x(x) = \begin{cases} 
C_x^m(m) + \sum_{t \in T_x(x)} C(t) & \text{if } x = \{m\} \\
\min_{a \in A_x(x)} (J_a(x, a) + \sum_{t \in T_r(x)} C(t)) & \text{otherwise}
\end{cases}
\]

(3.7)

Here, \(|x|\) denotes the size (number of elements) of \( x \). If the root node is solved, which means that \( J_x(x_{\text{init}}) \) 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.

**Illustration**

With the algorithm presented the solution for the problem described in Section 3.1.3 has been calculated. The solution for this problem is shown in Figure 3.4(a). This figure shows for each module (square node), the integration steps (hexagonal node) which consist of creating interfaces, and the test steps (circular node) that should be done. Note that this tree is the reverse of the disassembly tree. The edges denote the precedence relations between the actions, and therefore show the sequence of integration action and test phases for each
module. The longest paths in this tree are the paths of modules $m_1$, $m_2$, and $m_6$ which are all 70 time units. Therefore the cost of this optimal solution is also 70. In this situation, the actual lens ($m_7$) is integrated late because it has a long development time, and the model of the lens ($m_8$) can be used to perform tests $t_{14}$ through $t_{17}$.

In the situation shown in Figure 3.4(b) it is assumed that the model of the lens ($m_8$) is not available during the integration phase of this new scanner. The cost of this solution is 73 time units. The longest path in the tree is now the path of the lens ($m_7$), which is integrated earlier to perform tests $t_{14}$ through $t_{17}$. For this illustration, we can conclude that the time-to-market of the scanner is 3 time units less when the model of the lens ($m_8$) is developed and used. Therefore, one has to consider whether this time reduction is worth the effort of building the model of the lens.

Figure 3.4: Solutions of the scanner illustration
3. INTEGRATION PLAN OPTIMIZATION

3.1.5 Computational reduction measures

The computational effort for searching an AND/OR graph is NP-hard:

\[
\text{Number of OR nodes to be investigated} = 2^{\left| M \right|} - 1
\]  
(3.8)

according to De Mello in [de Mello and Sanderson, 1991b] for a strongly connected module graph with the check that only feasible cut-sets remain. This means that problems of 20 modules need over 1 million OR node investigations. Since industrial problems often have more than 20 modules, we need some computational reduction measures to be able to solve them in a reasonable computation time.

Two measures (heuristics) are proposed to reduce this computational effort. Both heuristics try to reduce the computational effort by reducing the number of possible cut-sets (AND nodes) per OR node. This is done by only investigating the most promising cut-sets.

The first heuristic is the ‘Early Time’ (ET) heuristic. The idea is that the cost for test and integration should be spent as early as possible in the integration sequence to obtain a sequence that is as parallel as possible. Thus, integration actions (cut-sets) are selected that have the lowest test and integration cost, such that more cost is spent in the lowest parts of the AND/OR graph. The time spent on integration and test in an AND node \( a \in A_x(x) \) is:

\[
ET(a, x) = \sum_{i \in a} C^i(i) + \sum_{t \in T_r(x)} C^t(t)
\]  
(3.9)

Here, \( T_r(x) \) denotes the tests that are applied on that subsystem, as calculated by equation 3.5. The ET heuristic selects the \( n \) cut-sets that have the lowest ET to be investigated further, where \( n \) is a user-defined natural number.

The second heuristic is the ‘Parallel Time’ (PT) heuristic. The idea is that the more actions (test and integration) are done in parallel, the more optimal the total integration plan is. This is done by selecting cut-sets that disassemble a system state \( x \) into two system states \( x_1 \) and \( x_2 \) that have (almost) the same total duration. Therefore, for each cut-set from \( a \in A_x(x) \), the total duration of the two formed subsystems \( J_{tot}(x_1) \) and \( J_{tot}(x_2) \) is calculated using the following equation, for \( x' = x_1 \vee x' = x_2 \):

\[
J_{tot}(x') = \sum_{m \in x'} C^m(m) + \sum_{i \in I_x(x_1)} C^i(i) + \sum_{t \in T_x(x')} C^t(t)
\]  
(3.10)

Here, \( I_x(x') \) denotes all interfaces that are present in system state \( x' \). Furthermore, \( T_x \) can be calculated using Equation 3.4. The total duration for a system state is the time that it takes to develop all modules, create all interfaces and perform all tests sequentially, so the sum of development actions, integration actions and tests. Then, the Information Gain approach is used to determine to what extent the two total times are equal:

\[
PT(a, x) = IG(J_{tot}(x_1), J_{tot}(x_2))
\]  
(3.11)
Table 3.2: Experimental results using various heuristics

<table>
<thead>
<tr>
<th>Experiment</th>
<th>( J^* )</th>
<th>OR nodes</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>70</td>
<td>531</td>
<td>100%</td>
</tr>
<tr>
<td>ET(( n = 1 ))</td>
<td>86</td>
<td>15</td>
<td>3.7%</td>
</tr>
<tr>
<td>ET(( n = 2 ))</td>
<td>76</td>
<td>89</td>
<td>14.9%</td>
</tr>
<tr>
<td>ET(( n = 3 ))</td>
<td>71</td>
<td>177</td>
<td>30.2%</td>
</tr>
<tr>
<td>ET(( n = 5 ))</td>
<td>71</td>
<td>345</td>
<td>59.8%</td>
</tr>
<tr>
<td>PT(( n = 1 ))</td>
<td>74</td>
<td>15</td>
<td>2.7%</td>
</tr>
<tr>
<td>PT(( n = 2 ))</td>
<td>70</td>
<td>65</td>
<td>8.7%</td>
</tr>
<tr>
<td>PT(( n = 3 ))</td>
<td>70</td>
<td>113</td>
<td>17.4%</td>
</tr>
<tr>
<td>PT(( n = 5 ))</td>
<td>70</td>
<td>235</td>
<td>43.0%</td>
</tr>
</tbody>
</table>

Here, the IG is defined by [Johnson, 1960]:

\[
IG(x, y) = - \left( \frac{x}{x+y} \cdot \log_2 \left( \frac{x}{x+y} \right) + \frac{y}{x+y} \cdot \log_2 \left( \frac{y}{x+y} \right) \right)
\]  

(3.12)

The PT heuristic selects the \( n \) best cut-sets that have the highest PT to be investigated further, where \( n \) is a user-defined number.

With the following experiments, the influence of the proposed heuristics is investigated. For each experiment, different heuristics are used to calculate a solution for the Scanner illustration in Figure 3.2. As shown in Figure 3.4(a), the cost of the optimal solution is 70 time units. In Table 3.2, the cost and computational effort, measured in terms of the number of OR node calculations and time (relative to the time needed to calculate the optimal solution), are shown for each of the experiments. For the ET heuristic it is shown that for \( n = 1 \) the solution is far from optimal, while for \( n = 5 \) the solution approaches the optimal one. For the PT heuristic it is shown that for \( n = 1 \) the solution is better than the ET(\( n = 1 \)) heuristic, and that for \( n = 2 \) the solution is already optimal. For this illustration, it is best to use the PT heuristic.

### 3.1.6 Case study

This paper section is intentionally removed from this chapter because a more detailed and elaborate description of the case studies can be found in Section 5.2 or in the original version of this paper.

### 3.1.7 Definitions and notations

In Table 3.3, the definitions and their descriptions used in this paper are shown.
3. INTEGRATION PLAN OPTIMIZATION

Table 3.3: List of definitions

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>Integration model: $(M, I, T, C^m, C^i, C^t, R_{im}, R_{tm})$</td>
</tr>
<tr>
<td>$M$</td>
<td>set of $k$ modules.</td>
</tr>
<tr>
<td>$I$</td>
<td>set of $l$ interfaces.</td>
</tr>
<tr>
<td>$T$</td>
<td>set of $m$ tests.</td>
</tr>
<tr>
<td>$C^m$</td>
<td>the cost in time units of developing a module $m$.</td>
</tr>
<tr>
<td>$C^i$</td>
<td>the cost in time units of constructing an interface $i$.</td>
</tr>
<tr>
<td>$C^t$</td>
<td>the cost in time units of performing a test $t$.</td>
</tr>
<tr>
<td>$R_{im}$</td>
<td>the modules where interface $i$ is constructed between.</td>
</tr>
<tr>
<td>$R_{tm}$</td>
<td>the essential assemblies of test $t$.</td>
</tr>
<tr>
<td>$m, i, t$</td>
<td>a single module, interface or test.</td>
</tr>
<tr>
<td>$J, J^*$</td>
<td>Cost and optimal cost of an integration sequencing problem.</td>
</tr>
<tr>
<td>$G, G^*$</td>
<td>All solutions, a solution and the optimal solution of the integration sequencing problem.</td>
</tr>
<tr>
<td>$X, x, x_{init}$</td>
<td>Domain of system states, a single system state and the initial system state.</td>
</tr>
<tr>
<td>$A_x, a$</td>
<td>All possible cut-sets of a system state $x$, and a single cut-set.</td>
</tr>
<tr>
<td>$J_x, J_{ar}, J_{tot}$</td>
<td>Cost of an OR node denoted with system state $x$ and an AND node denoted with cut-set $a$, and total cost-to-go for a system state $x$.</td>
</tr>
<tr>
<td>$I_x, T_x, T_r$</td>
<td>All interfaces that are constructed in a system state $x$, all tests that can be performed and all tests that are performed in a system state $x$.</td>
</tr>
<tr>
<td>$ET, PT$</td>
<td>’Early time’ and ’Parallel time’ heuristics.</td>
</tr>
<tr>
<td>$n$</td>
<td>User-defined variable.</td>
</tr>
<tr>
<td>IG</td>
<td>Information Gain.</td>
</tr>
<tr>
<td>$H, H_{init}$</td>
<td>Domain, instantiation, and initial instantiation of the function that returns the cost of solved OR nodes.</td>
</tr>
</tbody>
</table>

3.1.8 Algorithm

This section describes the algorithm that performs the AND/OR graph search. The AND/OR graph search is a recursive search over AND and OR nodes. Each OR node (except for the leaf nodes) has one or more following AND nodes, depending on the number of possible cut-sets that are possible in that OR node. Each of those AND nodes has exactly two resulting OR nodes for each of the created subsystems, and so on. This section gives a formal, functional-style [Bird, 1998] description of the algorithm. A step-by-step description of the algorithm that relates to the functions defined, is shown in Figure 3.5.

To prevent double calculations of the same OR nodes we introduce $H : X \rightarrow (\mathbb{R}^+ \times \mathcal{P}(I)) \cup \{\perp\}$ which gives the optimal cost and the chosen cut-set for a solved OR node, or gives undefined for an unsolved OR node. The function $H$ is an instantiation of $\mathcal{H}$.

To find the optimal integration cost $J^*$ for an integration model $D = (M, I, T, C^m, C^i, C^t, R_{im}, R_{tm})$, the following expression can be used:

$$ (J^*, H) = \text{OR}(x_{init}, A_x(x_{init}), H_{init}) $$

(3.13)

where $H_{init} : \mathcal{P}(M) \rightarrow \{\perp\}$ is the initial function that gives the cost of a solved OR node. $A_x(x_{init})$ are all cut-sets of the initial system state $x_{init}$, calculated using the algorithm
Step-by-step algorithm

Input: – System model $D$;

Output: – The optimal solution tree $G$;
             – The cost of the solution;

Step 0: Initialize a graph $G$ consisting of the root node $x_{\text{init}} = M$, i.e., all modules are integrated, mark the node as unsolved.

Step 1: Repeat the following steps for the system state $x = x_{\text{init}}$ to construct an AND/OR graph until the root node is marked solved. Then exit with $J = J_x(x_{\text{init}})$ as expected test cost and the solution graph $G$(these steps are performed by function OR).

Step 1.0 If $x$ contains one element $m$ mark $x$ solved in $G$ and assign the cost of the tests that must still be performed $T_r(x)$ and the development cost $C^m(m)$ to $x$, then stop. Else, continue with step 1.1

Step 1.1 Determine the possible cut-sets $A_x(x)$ and perform for each cut-set $a$ in $A_x(x)$ the following steps (performed by function AND)

Step 1.1.0 Initialize a subgraph $G'$ consisting of root node $a$
Step 1.1.1 Determine for $a$ the new subsystem OR nodes $x_1$ and $x_2$, insert them in $G'$ and draw an edge from $a$ to both of them
Step 1.1.2 If $x_1$ is not solved, mark $x_1$ unsolved and perform steps 1.0 through 1.2 for $x$ replaced by $x_1$; do the same for $x_2$ (this is the recursion by function OR)
Step 1.1.3 Determine for $a$ the cost of creating the interfaces in $a$, the cost of tests that have to be performed $T_r$, and the maximal cost for $x_1$ or $x_2$, and assign these cost to AND node $a$

Step 1.2 Select the cut-set $a$ and the corresponding subgraph $G'$ that has minimal cost. Mark $x$ solved and assign the cost $J_a(a, x)$ to $x$. Merge graph $G$ with subgraph $G'$, create an edge from node $x$ to the root node of $G'$ and exit.

Figure 3.5: Step-by-step algorithm description
presented in [Tsukiyama et al., 1980]. Here \(x_{\text{init}} = M\), which denotes the initial state which 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 subassemblies. This prevents calculating the cut-sets for every system state which reduced computation time.

The resulting \(H\) can be used to construct the optimal solution \(G\). This calculation gives the cost \(J^*\) of the optimal solution according to equation 3.2. For each 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 subsystems, 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 \(\text{OR}: X \times P(P(I)) \times H \to R^* \times H\) calculates the cost of a system state \(x\) given the possible disassembly actions and the initial \(H\). The cost of such an OR node is denoted by Equation 3.7. The OR function is defined as follows. If the state has already been solved (i.e. if \(H(x) \neq \perp\)), the solution of that solved state is taken from \(H\) and returned. Otherwise, several options are possible. If \(x\) consists of one module, the node is a leaf node and the resulting cost is the module development cost and the remaining test cost. If \(x\) consists of multiple modules, one or more cut-sets are available which are evaluated and compared to each other with the AND function:

\[
\text{OR}(x, A_x, H) = \begin{cases} 
(H(x), H) & \text{if } H(x) \neq \perp \\
(J', H(x/(J', \{\}))) & \text{if } H(x) = \perp \land |x| = 1 \\
(J'', H''(x/(J'', a''))) & \text{if } H(x) = \perp \land |x| > 1 
\end{cases}
\]

Here:

- \(J' = \sum_{m \in x} C^m(m) + \sum_{t \in T_x(x)} C'(t)\)

- \((J_a(a_i), H_i) = \text{AND}(x, \text{rmv}(A_x, a_i), a_i, H_{i-1}) + \sum_{t \in T_x(x)} C'(t)\) 
  for \(i = 1, \ldots, |A_x|\) (where \(H_0 = H\), are the minimal test cost and updated \(H\) for each cut-set in \(A_x\).

- \(J'' = J(a'') = \min_{a'' \in A_x} (J_a(a))\), is the minimal cost of \(x\), and \(a''\) is the cut-set from \(A_x\) for which this holds.

- Function \(\text{rmv}\) removes all interfaces in the cut-set \(a_i\) from all cut-sets in \(A_x\) while ensuring that the resulting cut-sets still split the new system state into exactly two sub-systems, and thus results in the new set of cut-sets ensuring that this set still satisfies Equation 3.3.
The function \( \text{AND} : X \times \mathcal{P}(I) \times \mathcal{P}(\mathcal{P}(I)) \times \mathcal{H} \rightarrow \mathbb{R}^* \times \mathcal{H} \) takes a system state \( x \) 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.

\[
\text{AND}(M, a, A_x, H) = \\
\left( \sum_{i \in a} C(i) + \max(J', J''), H'' \right)
\]

(3.15)

Where:

- \( x_1 \) and \( x_2 \) are defined as the new system states resulting from applying cut-set \( a \) on system state \( x \).
- \( (J', H') = \text{OR}(x_1, A_x, H) \), calculates the cost of system state \( x_1 \) and the updated \( H' \).
- \( (J'', H'') = \text{OR}(x_2, A_x, H') \), calculates the cost of system state \( x_2 \) and the updated \( H'' \).

### 3.1.9 Conclusions

In this paper, we proposed a method to calculate an optimal integration sequence. The optimal integration sequence is the sequence with the shortest duration that develops, integrates and tests modules into a system, and therefore tries to perform the development, integration and test actions as much as possible in parallel. The method consists of defining a model which is able to describe an integration sequencing problem mathematically and an AND/OR algorithm which is able to determine the optimal solution.

The model consists of the modules or stubs/models to be integrated in the system and their development times, the interfaces that denote the possible assemblies of modules and their creation times, the tests that should be performed and the time it takes to perform them, for each test the modules that need to be present before the test can be performed and for each interface the modules it is created in between.

We introduced two heuristics for the algorithm in order to obtain solutions for large problems in reasonable time limits.

We performed a case study in the development of a first-of-a-kind lithographic machine to optimize the integration plan. With this case study, we learned that feasible and good solutions can be obtained using the model.

The benefits of this method are two-fold: 1) optimal integration plans can be calculated which are usually better than manually created integration plans, and 2) replanning the integration plan when for example a module is delivered late takes less effort because only the model needs to be adjusted.

In this method we assumed that all tests should be applied exactly once. However, in previous work [Boumen et al., 2006d,c] we have shown that not all tests must be applied, or that tests must be applied multiple times. In our future work, we will combine both the test sequencing method presented in previous work with the integration sequencing method such that optimal integration sequences with optimal test phases can be obtained.
3.2 HIERARCHICAL INTEGRATION SEQUENCING

The content of this section has not been published in a paper. In today’s industry, embedded systems are becoming more and more complex. This complexity growth can be shown by the growing number of components that are formed and the growing number of hierarchical layers in a system. In ASML systems, four hierarchical levels can be distinguished:

- **Components** at the lowest level, a lithographic machine consists of 1000+ components.
- **Building Blocks** several components form a mono-disciplinary building block. There are 100+ building blocks in the system.
- **Functional Clusters** several building blocks form a multidisciplinary functional cluster. There are 30+ functional clusters.
- **System** all functional clusters together form the final system.

This growing number of components often makes it impossible to calculate an optimal integration sequence in a reasonable computation time. In the previous section, we therefore introduced several heuristics that reduce this computation time. However, these heuristics become inefficient when used for very large systems (more than 100 modules). Another solution is to use a hierarchical integration sequencing method that uses for example the system’s hierarchy to find a good solution with reasonable computational effort. A hierarchical approach also reduces the modeling effort since there are multiple levels on which one can reason. Such an approach also makes it possible to combine several low level models created by different engineers into one system integration model.

This section introduces a hierarchical integration sequencing method. In Subsection 3.2.1, the hierarchical integration model is introduced. In Subsection 3.2.2, the solution algorithm is presented. In Subsection 3.2.3, several methods are discussed that help in creating a hierarchical integration model.

### 3.2.1 Integration model

The basic idea of a hierarchical integration model is that a subassembly of modules has the same properties as a single module. These properties are:

- **Development time and cost** The development time of a subassembly is the duration of the critical integration path of that subassembly including test phases and development times of the individual modules. The development cost of a subassembly is the sum of the development costs of the individual modules.

- **Interfaces with other modules** This property is satisfied by allowing interfaces between subassemblies and individual modules in the hierarchical model.

- **Test module relations** This property is satisfied by allowing subassemblies to be part of the essential modules of a test.
From now on, a subassembly that is modeled hierarchically is called a cluster. For each cluster, a single integration model needs to be defined. The resulting set of models can be represented as a tree as shown in Figure 3.6. In this figure, nodes represent modules and clusters, and edges represent interfaces. This graph shows three levels of hierarchy with one or more models on each level. The graph also shows the relation between a cluster on one level and a model on a lower level. For example, cluster $l_2^0$ on level 0 relates to the subassembly on level 1 containing modules $m_1^2$, $m_2^2$.

The formal hierarchical integration model becomes $D = \{D_1, \ldots, D_K\}$, a set of $k$ single integration models. The single integration model $D_k$ is a dectuple $D_k = (L_k, M_k, I_k, T_k, C_{m}^k, C_{i}^k, C_{t}^k, R_{im}^k, R_{tm}^k, R_{ld}^k)$, where:

- $M_k, I_k, T_k, C_{m}^k, C_{i}^k, C_{t}^k$ are defined in the original model.
- $L_k$ is a finite set of $g$ clusters.
- $R_{im}^k : I_k \rightarrow (L_k \cup M_k) \times (L_k \cup M_k)$ gives for each interface in $I_k$ the modules or clusters the interface is constructed between.
- $R_{tm}^k : T_k \rightarrow \mathcal{P}(\mathcal{P}(M_k \cup L_k))$ gives for each test in $T_k$ its essential assemblies; where an essential assembly describes the modules or clusters that should be integrated with each other before the test can be performed.
• $R_{k}^{ld} : L_k \rightarrow D$ gives for each cluster in $L_k$, the associated low level single integration model $D_i$.

The assumptions for this single integration model are:

• All modules in $M_k$ and all clusters in $L_k$ must be connected to each other, so there exists a path of interfaces that connects every module in $M_k$ or cluster in $L_k$ to every other module in $M_k$ or cluster in $L_k$. This ensures that there only exists one system.

• For every test in $T_k$, there exists at least one module or cluster that is present in all essential assemblies of this test. This ensures that each test is only performed once.

• Each test in $T_k$ is performed exactly once at the moment that one of the essential assemblies of this test is integrated.

• There is exactly one single integration model at the highest level of the hierarchical integration model (one root node of the hierarchical integration model tree).

• A single integration model can only be related to exactly one cluster that is present in a higher level single integration model, except the highest model.

To illustrate a hierarchical model, we introduce the following example which has also been used in Chapter 2 to illustrate hierarchical test sequencing. The example system is a telephone exchange system illustrated in Figure 3.7. This system consists at the highest level of five subsystems: Telephone 1($l_1^1$), Telephone 2($l_2^1$), Cable 1($m_1^1$), Cable 2($m_3^1$) and the Switch($m_2^1$), which are connected using four interfaces ($i_1^2, i_2^2, i_3^2, i_4^2$). Each telephone consists of three modules at a lower level: the horn ($m_1^2, m_2^2, m_3^2$), the cable and the device which are connected using two interfaces ($i_1^2, i_2^2$).

![Figure 3.7: Telephone exchange example](image)

A hierarchical model is created that consists of three single integration models: $D_1$ is the model at the highest level, $D_2$ is the model of Telephone 1 and $D_3$ is the model of Telephone 2. The modules $M_i$, clusters $L_i$ and interfaces $I_i$ of the highest level model $D_i$ are shown in Figure 3.8(a). The modules $M_2$ and interfaces $I_2$ of the model $D_2$ are shown in Figure 3.8(b). Model $D_3$ is the same as model $D_2$ and is therefore not shown here. In Table 3.4(a), the relation $R_{k}^{ld}$ is shown that relates the clusters within model $D_i$ to the other models $D_2$ and $D_3$. In Table 3.4(b), the remaining relations and elements of model $D_i$ are shown. In
3.2. HIERARCHICAL INTEGRATION SEQUENCING

Table 3.4(c), the remaining relations and elements of model \( D_2 \) are shown. Model \( D_3 \) is the same as model \( D_2 \) and therefore not shown here. Note that in this example, each module has exactly one essential assembly.

Table 3.4: Elements of models \( D_1 \) and \( D_2 \)

<table>
<thead>
<tr>
<th>( L / D )</th>
<th>( D_1 )</th>
<th>( D_2 )</th>
<th>( D_3 )</th>
<th>( T_1 )</th>
<th>( R_{1m}^m )</th>
<th>( R_{1m}^m )</th>
<th>( C_1^m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 )</td>
<td>I</td>
<td>O</td>
<td>1</td>
<td>{{l_1, m_1^1, m_1^2, m_1^3, l_1^2}}</td>
<td>10</td>
<td>{{m_2^1, m_2^2, m_2^3}}</td>
<td>3</td>
</tr>
<tr>
<td>( l_2 )</td>
<td>O</td>
<td>I</td>
<td>1</td>
<td>{{m_2^1}}</td>
<td>1</td>
<td>{{m_2^1}}</td>
<td>1</td>
</tr>
</tbody>
</table>

3.2.2 Solution algorithm

The solution algorithm that calculates the optimal hierarchical integration sequence for a hierarchical model \( D \) is almost the same as the one level algorithm. The hierarchical solution algorithm also constructs an AND/OR graph. An OR node represents the system state \( (x) \) which is the set of integrated modules or clusters and is defined as an element of \( \mathcal{P}(M \cup L) \), where \( M \) is the total set of modules and \( L \) the total set of clusters in \( D \). In a system state, tests are applied that are only possible in that state and not in following states. The initial system state of the highest model \( D_1 \) is \( x_{i\text{ini}}^1 = M_1 \cup L_1 \). The algorithm starts with the highest level model \( D_1 \). An AND node defines the integration action of two OR nodes into one OR node and is denoted by the set of interfaces that are broken, which is an element of \( \mathcal{P}(I) \), where \( I \) is the set of all interfaces of \( D \).

The algorithm that constructs AND/OR graphs is a depth-first algorithm. The search
starts with the initial OR node of the highest level model. The cost of this particular OR node is called $J_x(x_{i_{init}})$ and is determined as follows.

First, the possible set of disassembly actions is determined for the highest level model $D_l$. For a given system state $x$ and a particular model $D_k$, this can be determined as follows:

$$A_x(x, D_k) = \{a : a \subseteq I^x_k(x) : \exists(x_1, x_2) \subseteq x :$$

$$x_1 \cap x_2 = \emptyset \land x_1 \cup x_2 = x$$

$$x_1 \neq \emptyset \land x_2 \neq \emptyset \land \forall(m', m'') \in x_1 : \text{connected}(m', m'', I^x_k(x) \setminus a)$$

$$\land \forall(m', m'') \in x_2 : \text{connected}(m', m'', I^x_k(x) \setminus a)$$

$$\land \exists m' \in x_1, m'' \in x_2 : \text{connected}(m', m'', I^x_k(x) \setminus a)$$

$$\}$$

Here, $I^x_k(x)$ denotes all interfaces between the modules and clusters in system state $x$ and for model $D_k$. Function connected checks whether two modules or clusters are connected, i.e. there exists a path of interfaces between two modules or clusters. We use the same cut-set algorithm as in the single integration model algorithm.

For all cut-sets $a \in A_x(x)$ an AND node is constructed given system state $x$ and model $D_k$. The tests that can still be performed, $T^x_k(x_1, D_k)$ and $T^x_k(x_2, D_k)$, after breaking the interfaces in the cut set can be calculated using:

$$T^x_k(x, D_k) = \{t : t \in T^x_k : (\exists M' \in R^m_k(t) : M' \subseteq x)\}$$

(3.17)

The required tests $T^x_k(x, D_k)$ that have to be performed in system state $x$ can be calculated using:

$$T^x_k(x, D_k) = T^x_k(x, D_k) \setminus (T^x_k(x_1, D_k) \cup T^x_k(x_2, D_k))$$

(3.18)

The total cost of an AND node $J_a(x, a)$ for a system state $x$ and a disassembly action $a \in A_x(x)$ which disassembles the system into two subsystems $x_1, x_2$, is defined as the maximal integration and test cost of each formed system state $x_1$ and $x_2$, plus the cost of the disassembly of the system state $x$ into the two system states, or:

$$J_a(x, a, D_k) = \sum_{i \in a} C_i(i) + \max(J_x(x_1), J_x(x_2))$$

(3.19)

Finally, the cost of the OR node is the minimal cost of each AND node that is constructed and the cost of performing the required tests $T^x_k(x)$, or the development cost of a module if one module remains and the cost of the tests $T^x_k(x)$ that can still be performed in system state $x:

$$J_x(x, D_k) =$$

$$\min_{a \in A_x(x_{i_{init}}, D)} \left( J_a(x, a, D_i) + \sum_{t \in T^m_k(x_{i_{init}}, D)} C_i(t) \right) + \sum_{t \in T^m_k(x, D_k)} C_i(t) \quad \text{if } x = \{l\}$$

$$C^m_k(m) + \sum_{t \in T^m_k(x, D_k)} C_i(t) \quad \text{if } x = \{m\}$$

(3.20)

$$\min_{a \in A_x(x_{i_{init}}, D)} \left( J_a(x, a, D_k) + \sum_{t \in T^m_k(x, D_k)} C_i(t) \right) \quad \text{otherwise}$$
Figure 3.9: Telephone exchange example optimal hierarchical solution

where $D_l$ is the model corresponding to cluster $l$ in model $D_k$ and $X_l^{\text{init}} = M_l \cup L_l$ denotes the initial node of model $D_l$.

For the illustration introduced in the previous section, the optimal solution is shown in Figure 3.9. The rectangles around certain parts of the graph denote the solutions of clusters $D_2$ and $D_3$. The longest path time of this solution is 30 time units. It must be noted that the hierarchical model may result in a less optimal solution compared to a complete single model. This is because the clusters are constructed completely before they are integrated in the system. This approach reduces the computational effort but may increase the integration time.
3.2.3 Cluster creation

The main difficulty of the hierarchical integration sequencing approach presented is the creation of the clusters. We discuss a few approaches that can be followed. The hierarchy of a model can be retrieved from:

**THE SYSTEM HIERARCHY** As shown in the small example, it is possible to create a system level model and to create for each subsystem a new model and so on. This is only possible if there is a predefined system hierarchy and if this hierarchy is also followed during integration.

**THE DELIVERY TIMES OF MODULES** Cluster the modules that arrive approximately at the same time during the integration process. For example, during the integration of software updates, the system hierarchy is not followed and updates are grouped according to the delivery times, see for example the case study in [Boumen et al., 2007a].

**THE ENTANGLEMENT OF MODULES** The number of shared interfaces may indicate which modules belong to each other. Group the modules that have many interfaces in common.

The first two approaches are quite easy to apply, the last one may be more difficult to apply. For this last approach, tools are available that may help with clustering modules. An example of such a tool is GraphViewer [van Ham, 2004] which is based on small-world graphs as described by van Ham in [van Ham and van Wijk, 2004]. This visualization package groups the nodes (modules) based on the shared edges (interfaces). We applied this tool to create a hierarchical model of the complete ASML wafer scanner software system which consists of 221 modules and 9428 interfaces. The resulting graph is shown in Figure 3.10(a) with interfaces and in Figure 3.10(b) without interfaces. One clearly can distinguish the modules that have many interfaces in common and are therefore positioned close to each other. In the second graph, the (grey-scale) colors denote the current subdivision into three subsystems which is questionable for some modules.

3.3 EXTENSIONS

This section describes several extensions to the integration plan optimization method as described in the previous two sections. The content of this section has not been published in a paper. Each extension is explained in the following subsections.

3.3.1 Copyable modules

In the basic integration model, each module can be used for one test phase at a time. However in some cases, modules exist that can be used in parallel when they have been developed because they can be copied. An example of such a module is a software module. It is quite easy to copy a software module to another computer and perform two test phases in parallel.
using the same module. To cope with such modules, we adjust the integration model and integration algorithm as follows.

The new (single) integration model is a nontuple $D = (M, I, T, C^m, C^i, C^t, M^c, R^{im}, R^{im})$, where:

- $M, I, T, C^m, C^i, C^t, R^{im}, R^{im}$ are the same as in the basic integration model shown in [Boumen et al., 2006a]

- $M^c : M \rightarrow \mathbb{B}$, indicates for each module in $M$, whether this module is copyable or not by a boolean.

The assumptions for this model are the same as for the basic model extended with the following assumption: for every test in $T$, there exists at least one non-copyable module that is present in all essential assemblies of this test to ensure that each test is performed only once.

With a small example we demonstrate this extended integration sequencing model. Table 3.5 shows model elements $M, T, C^m, C^i$ and $R^{im}$ of a system consisting of 3 modules
Table 3.5: Model elements $M, T, C^m, C^l, R^{lm}$ and $M^c$

<table>
<thead>
<tr>
<th>$T / M$</th>
<th>$m_1^*$</th>
<th>$m_2$</th>
<th>$m_3$</th>
<th>$C^l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>I</td>
</tr>
<tr>
<td>$t_2$</td>
<td>I</td>
<td>O</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>$t_3$</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>$C^m$</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.11: AND/OR graph with copyable module $m_1$.

$(m_1, m_2, m_3)$ all connected with three interfaces $(i_1, i_2, i_3)$. Module $m_1$ may be copied which is indicated with a $^*$. 

To cope with copyable modules in the algorithm, we simply extend the valid set of cut-sets with cut-sets that also enable copyable modules. This is illustrated with the example AND/OR graph shown in Figure 3.11. This figure shows the first part of the AND/OR graph of the example introduced. Besides the three normal cut-sets, two additional cut-sets (grey nodes) are created. For these additional cut-sets, $m_1$ is copied and is therefore present in both the resulting system states.

In the basic algorithm, the set of cut-sets $A_x$ is determined by the function $A_x$. In the new algorithm, a new function $Copy$ is added that constructs the additional cut-sets given the original cut-sets and the system state $x$. The subassemblies $x_1$ and $x_2$ are formed by disassembling $x$ according to a disassembly action $a \in A_x$. This new function is defined as:

$$
Copy(x, A_x) = \bigcup_{a \in A_x} \{ (a, M') : M' \in \{ M'' : M'' \subseteq \{ m \in M : M^c(m) \} \} \\
\wedge x \neq x_1 \cup M' \wedge x \neq x_2 \cup M' \\
\wedge \forall (m', m'') \in x_1 \cup M' : connected(m', m'', I_\setminus a) \\
\wedge \forall (m', m'') \in x_2 \cup M' : connected(m', m'', I_\setminus a) \}
$$

(3.21)

The result of this function is a set of cut-sets with the corresponding copied modules. For the initial node of the AND/OR graph shown in Figure 3.11, the resulting set is:
3.3. EXTENSIONS

The optimal solution for the example is shown in Figure 3.12. In this solution, \( m_1 \) is copied to make it possible that \( t_1 \) and \( t_2 \) can be performed in parallel. The cost of the critical path of this example is 3 time units. If module \( m_1 \) would not be copied, the cost would be 4 time units.

In [Braspenning et al., 2007], a case study is described where the integration method presented with copyable modules is used to investigate whether it is beneficial to use executable software component models to replace real components.

3.3.2 Objectives and constraints

The objective of the integration sequencing problem as defined in [Boumen et al., 2006a] is to minimize the duration of the critical path of an integration sequence. However, there are more objectives possible. To incorporate these objectives the integration model is extended as follows. The cost of the development of a module now becomes a tuple containing the duration in time units and the cost in cost units. Also, the costs of tests, and costs of interfaces are defined in this way. Furthermore, we introduce the concept of interest on monetary values of components. During the integration and test phase, components represent a certain value. If these components are expensive, the interest on that value is important for the cost of the integration sequence: one would like to integrate the most expensive components as late as possible to reduce the interest cost. Therefore, we introduce for each module a value in cost units that is used to calculate the interest cost of a certain sequence. Furthermore, \( r_i \) is a user-defined variable that denotes the interest rate per time unit.

The new integration model is a nontuple \( D = (M, I, T, C^m, C^i, C^f, M^c, R^{im}, R^{im}) \), where:

- \( M, I, T, M^c, R^{im}, R^{im} \) are the same as in the basic integration model shown in [Boumen et al., 2006a].
3. INTEGRATION PLAN OPTIMIZATION

- \( C^m : M \rightarrow \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \) gives for each module in \( M \) the associated duration in time units and cost in cost units of developing that module, and the value of that module in cost units.

- \( C^i : I \rightarrow \mathbb{R}^+ \times \mathbb{R}^+ \) gives for each interface in \( I \) the associated duration in time units and cost in cost units of creating that interface.

- \( C^t : T \rightarrow \mathbb{R}^+ \times \mathbb{R}^+ \) gives for each test in \( T \) the associated duration in time units and cost in cost units of performing that test.

With this extended model, it is possible to create numerous objectives and constraints consisting of several parameters. A solution \( G : M \rightarrow (P(T) \cup P(I))^* \) (\(^* \) denotes a set of sequences) to this problem gives for a subassembly consisting of a single element of \( M \), a sequence of integration actions (sets of interfaces \( P(I) \)) and test phases (sets of tests \( P(T) \)). This sequence of actions integrates this single module subassembly into the completely integrated and tested system. The possible variables of an objective function are:

- The total integration time, which is the sum of all durations and is denoted as (since this parameter is independent of the solution and thus constant, it is less interesting):

\[
TT = \sum_{m \in M} C^m(m).o + \sum_{t \in T} C^t(t).o + \sum_{i \in I} C^i(i).o \tag{3.22}
\]

- The total integration cost, which is the sum of all costs including the interest cost and is denoted as:

\[
TC = \sum_{m \in M} C^m(m).1 + \sum_{t \in T} C^t(t).1 + \sum_{i \in I} C^i(i).1
+ \sum_{m \in M} i_r \cdot C^m(m).2 \cdot (C^m(m).o + \sum_{t \in G(m) \mid T} C^t(t).o + \sum_{i \in G(m) \mid I} C^i(i).o) \tag{3.23}
\]

Here, \( G(m) \mid I \) takes all elements from \( G(m) \) that are a subset of \( I \) and uniforms them into one set.

- The critical integration path time, which is the duration of the longest path of an integration plan and is denoted as:

\[
CT = \max_{m \in M} \left( C^m(m).o + \sum_{t \in G(m) \mid T} C^t(t).o + \sum_{i \in G(m) \mid I} C^i(i).o \right) \tag{3.24}
\]

- The critical integration path cost, which is the cost of the most expensive path of an integration plan including the interest cost and is denoted as:

\[
CC = \max_{m \in M} \left( C^m(m).1 + \sum_{t \in G(m) \mid T} C^t(t).1 + \sum_{i \in G(m) \mid I} C^i(i).1
+ i_r \cdot C^m(m).2 \cdot (C^m(m).o + \sum_{t \in G(m) \mid T} C^t(t).o + \sum_{i \in G(m) \mid I} C^i(i).o) \right) \tag{3.25}
\]
Here, \( \cdot \) denotes the first element of a tuple and \( \cdot \) the second element, and so on. The general objective function for the optimization problem now becomes:

\[
J(G) = c_1 \cdot TT + c_2 \cdot TC + c_3 \cdot CT + c_4 \cdot CC
\]

Here, \( c_1 \ldots c_4 \) are user-defined coefficients that can be chosen, just as user-defined interest rate \( r_i \). Besides a user-defined objective function, constraints on some of the objective variables are needed to model real-life cases. For example, to calculate an integration sequence that has a maximal total cost of $100,000 we need to define a constraint. Constraints can be defined on \( TT, TC, CT, CC \).

It is possible to define numerous combinations of objectives and constraints. Since there are many combinations possible, we only give a few examples:

- An integration plan that integrates and tests a system as soon as possible with a maximal total test cost of $100,000 and with an interest of 10% per year can be created using the objective function with \( c_1 = 1 \) and \( r_i = 10\% \) per year (and \( c_2, c_3, c_4 = 0 \)) and the constraint: \( TC \leq 100,000 \).

- An integration plan that integrates and tests a system with minimal total test cost in three weeks can be created using the objective function with \( c_2 = 1 \), \( r_i = 10\% \) per year (and \( c_1, c_3, c_4 = 0 \)) and the constraint: \( CT \leq 3 \) weeks.

### 3.3.3 Computational reduction measures

In [Boumen et al., 2006a], several computational reduction measures are introduced. Both the Early time (ET) and Parallel time (PT) heuristics can be used to reduce the number of investigated cut-sets at an OR node. Also, the hierarchical models can be used to reduce the computational effort. Besides the heuristics that focus on time, a third heuristic is introduced that focuses on the minimization of interest cost. The ‘Minimal Interest’ heuristic (MI), only investigates the cut-sets that have the lowest possible interest cost. For a cut-set \( a \) that disassembles a state \( x \) into two subsystems \( x_1 \) and \( x_2 \), the maximal interest cost is when every module in either \( x_1 \) or \( x_2 \) is used to perform every test and is used during each integration. The maximal interest cost is determined by:

\[
MI(a, x) = r_i \cdot \left( \sum_{m \in x_1} C^m(m).2 \cdot \left( \sum_{i \in I_x(x_1)} C^i(i).o + \sum_{t \in T_x(x_1)} C^t(t).o \right) \right. \nonumber
\]

\[
+ \sum_{m \in x_2} C^m(m).2 \cdot \left( \sum_{i \in I_x(x_2)} C^i(i).o + \sum_{t \in T_x(x_2)} C^t(t).o \right) \right)
\]

Here, \( r_i \) denotes the interest rate per time unit. Note that the interest over the development of each module is not taken into account because these costs are always present and are therefore constant for all cut-sets.

All heuristics can be used in two modes: absolute or relative. When either the ET, PT or MI heuristic is used in the absolute mode, a predefined number \( n \) of the bests cut-sets is selected for every OR node. When they are used in the relative mode, a predefined percentage
\( p \) of the best cut-sets is selected for every OR node. For a set of possible cut-sets \( A_x \) that is analyzed at an OR node \( x \), the investigated cut-sets \( A'_x \) are:

- \( A'_x(x) = \{ a \in A_x(x) : |\{ a' \in A_x(x) : ET(a', x) \leq ET(a, x) \}| \leq n \} \), for the absolute mode of the ET heuristic.

- \( A'_x(x) = \{ a \in A_x(x) : |\{ a' \in A_x(x) : PT(a', x) \geq PT(a, x) \}| \leq n \} \), for the absolute mode of the PT heuristic.

- \( A'_x(x) = \{ a \in A_x(x) : |\{ a' \in A_x(x) : MI(a', x) \leq MI(a, x) \}| \leq n \} \), for the absolute mode of the MI heuristic.

- \( A'_x(x) = \{ a \in A_x(x) : ET(a, x) \leq \frac{1}{p} \cdot \min_{a' \in A(x)} ET(a', x) \} \), for the relative mode of the ET heuristic.

- \( A'_x(x) = \{ a \in A_x(x) : PT(a, x) \geq p \cdot \max_{a' \in A(x)} PT(a', x) \} \), for the relative mode of the PT heuristic.

- \( A'_x(x) = \{ a \in A_x(x) : MI(a, x) \leq \frac{1}{p} \cdot \min_{a' \in A(x)} MI(a', x) \} \), for the relative mode of the MI heuristic.

The advantage of the relative mode is that when the ET, PT or MI values are close to each other, more cut-sets are investigated. While if they are not, a few (or even only the best) cut-sets are investigated. The disadvantage of the relative mode is that it is never known in advance how many cut-sets are investigated and thus how many calculations need to be performed.

### 3.3.4 Integration replanning

After the integration plan has been calculated, the plan is executed. During this execution, certain disturbances may occur. Some examples are:

- Development phases, integration phases or test phases may take longer or shorter than planned.

- Modules or tests are added or removed from the plan.

If these disturbances occur, the integration model needs to be updated and a new optimal plan needs to be calculated. However, certain development, test and integration phases are already finished or being executed at that moment. To cope with such a situation, we added the replan functionality. By indicating which actions are already executed or currently being executed, it is possible to calculate the updated plan for the remainder of the phases. This is done as follows. The model \( D_0 = (M_0, I_0, T_0, C^m_0, C^i_0, R^{im}_0, R^{tm}_0) \) is the initial model which is used to create the initial integration sequence. The new
model $D_i = (M_i, I_i, T_i, C_i^m, C_i^r, C_i^t, R_i^m, R_i^r)$ is the model where the disturbances are already incorporated. This new model is then updated according to the set of $l$ subassemblies $\{(M_{0}^l, I_{0}^l, T_{0}^l, C_l), \ldots, (M_{0}^l, I_{0}^l, T_{0}^l, C_l)\}$ that are already formed and tested by executing a part of the integration plan of the initial model $D_o$. Here, $C_k$ is the time it takes to develop modules $M_{0}^k$, to integrate these modules by creating $I_{0}^k$ and to execute the tests $T_{0}^k$ according to the initial sequence, for $1 \leq k \leq l$. The already integrated modules, created interfaces and executed tests are now removed from the new model. For every subassembly, an additional module is added to the new model. This additional module, represents the subassembly and takes all properties of the modules and interfaces present in this subassembly. The updated new model then is $D'_i = (M'_i, I'_i, T'_i, C^m_i, C^r_i, C^t_i, R^m_i, R^r_i)$, where:

- $M'_i = \{ m \in M_i : m \notin \cup (M_{0}^l, \ldots, M_{0}^l) \} \cup \{ m' : 1 \leq k \leq l \land m' \notin M_i \}$, which are all modules that have not been integrated yet and exactly one module for each already formed subsystem.

- $I'_i = \{ i \in I_i : i \notin \cup_{1 \leq k \leq l} I_{0}^k \}$, which are all interfaces that have not been created yet in the old plan.

- $T'_i = \{ t \in T_i : t \notin \cup_{1 \leq k \leq l} T_{0}^k \}$, which are all tests that have not been performed yet in the old plan.

- $C^m_i(m) = C^m_i(m)$ for all $m \in M_i \cap M'_i$ and $C^m_i(m') = C_i$ for $m' \in \{ m', \ldots, m' \}$, which are the development costs of all modules that have not been integrated yet in the old plan and the total duration of the already formed subsystems in the old plan.

- $C^r_i(i) = C_i(i)$ for all $i \in I'_i$, which are the creation costs for all interfaces that have not been created yet.

- $C^t_i(t) = C_i(t)$ for all $t \in T'_i$, which are the execution costs for all tests that have not been performed yet.

- $R^m_i(i) = \{ m' : 1 \leq k \leq l : (\exists m \in R^m_i(i) : m \in M_{0}^k) \} \cup \{ m : m \in R^m_i(i) : m \in (M_i \cap M'_i) \}$ for all $i \in I'_i$, which are the modules that have not been integrated yet in the already executed part of the plan and the additional modules for each subsystem developed.

- $R^m_i(t) = \{ m' : 1 \leq k \leq l : (\exists m \in R^m_i(t) : m \in M_{0}^k) \} \cup \{ m : m \in R^m_i(t) : m \in (M_i \cap M'_i) \}$ for all $t \in T'_i$, which are the modules that have not been integrated yet in the already executed part of the plan and the additional modules for each subsystem developed.

Examples of replan actions are described in the case studies of [Boumen et al., 2006a], [Boumen et al., 2007b] and in Chapter 5.
3.4 CONCLUSIONS

This chapter has answered research questions 2.1, 2.2 and 2.3 as defined in the first chapter:

**ANSWER TO QUESTION 2.1:** A structure of an integration plan is defined as a tree where the nodes represent integration actions and test phases, the leaf nodes represent the development actions of modules and the root node represents the completely integrated and tested system. Each module is integrated with other modules and test phases are performed until the final system is completely integrated. The different test and integration actions are executed in parallel on the subsystems. Such a tree has a critical path that determines the time-to-market of the plan. Furthermore, a plan has associated with it a total cost and duration. A plan is judged on these parameters.

**ANSWER TO QUESTION 2.2:** The information needed to construct an integration plan is the integration model, consisting of modules, interfaces, test phases and their properties, and the relation between these elements. This model defines the integration problem. Furthermore, an objective function, consisting of several parameters, should be defined, and these are used to determine the optimal plan. Also, constraints can be defined for certain parameters.

**ANSWER TO QUESTION 2.3:** The method that creates an optimal integration plan, given the integration model and objectives and constraints, involves constructing an AND/OR graph containing all possible solutions. In this graph, an AND node denotes an integration action and an OR node denotes a system state consisting of the integrated modules in the system. The AND/OR graph is constructed using the assembly by disassembly approach, this means starting with the completely integrated system and disassembling this system into single modules. The best solution is chosen using an AND/OR graph search algorithm that is adjusted for this specific problem.

With the integration plan method, we are able to optimize integration plans for different integration domains, such as prototype integration, manufacturing integration and software integration. Furthermore, this method reduces the effort in creating and maintaining integration plans because the optimization is performed automatically. The integration model gives insight in the integration problem and can be used as a knowledge container to train new engineers. Furthermore, the method can be used to judge ‘what-if’ scenarios that are constructed to show the benefits of certain new investments, or to show the benefits of using models of components to perform test phases earlier. In Chapter 5, several case studies show how this method can be used in real life and show the benefits of using this method.
This chapter introduces a method for optimizing integration and test plans. We combine the two methods that were described in the previous chapters into one method. Therefore, this chapter is illustrated with multiple puzzle pieces of which one is a test puzzle piece. Combining puzzle pieces represents the integration of modules into one system and the test puzzle piece shows a test in its most abstract sense: by looking at the outcome of the test (turning direction) we can determine whether the (sub-)system that was created works (pass) or not (fail). By creating larger puzzles (systems), functionality is added and more tests may be applied. By creating independent puzzle subsystems, we can work in parallel and create the puzzle faster.

In this chapter, we solve the problem of integration and test sequencing or in what sequence should we bring subsystems together and in what sequence should tests be executed such that the total time-to-market of the complete integration and test plan is minimal. These problems arise in system development but also in software releases and in manufacturing. Section 4.1 is based on a paper that deals with these problems. Section 4.2 continues with an optimal method to determine when test phases should start and stop. Section 4.3 deals with the hierarchical aspect of this problem. The last section gives conclusions about this chapter.

4.1 INTEGRATION AND TEST SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

This section is based on the paper titled Integration and test sequencing in complex manufacturing systems [Boumen et al., 2007a] and is submitted to IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans in 2007. The paper section dealing with the case studies has been removed in this chapter because a more detailed and elaborate description of the case studies can be found in Section 5.3.
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,

Abstract

The integration and test phase of complex manufacturing machines, like an ASML [ASML, 2006] 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.

4.1.1 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 sub-systems 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 takes a lot of effort and often results in a plan that is not optimal 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 [Boumen et al., 2006a], we developed a method that optimizes an integration sequence. This method is based on de Mello et al. [de Mello and Sanderson, 1991b,a] where a method is introduced for optimizing mechanical assembly sequences and on Hanh et al. [Hanh et al., 2001] where optimal integration strategies for object-oriented

This work has been carried out as part of the TANGRAM project under the responsibility of the Embedded Systems Institute Eindhoven, the Netherlands, and in cooperation with several academic and industrial partners. This project is partially supported by the Netherlands Ministry of Economic Affairs under grant TSIT2026.

R. Boumen, I.S.M. de Jong, J.M.G. Mestrom, J.M. van de Mortel-Fronczak and J.E. Rooda are with the Systems Engineering Group, Department of Mechanical Engineering, Eindhoven University of Technology, 5600 MB, Eindhoven, The Netherlands (e-mail: r.boumen@tue.nl; i.s.m.d.jong@tue.nl; j.w.h.vermunt@student.tue.nl; j.m.v.d.mortel@tue.nl; j.e.rooda@tue.nl).
systems are determined. In [Boumen et al., 2006a], 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. In addition, certain tests are never performed because other tests cover the same requirements. Therefore, the integration method as proposed in [Boumen et al., 2006a] 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 phase in 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 [Boumen et al., 2006b,c]. This method is based on sequential diagnosis methods as described by Pattipati et al. [Pattipati et al., 1991]. The combined method is able to create the optimal complete integration and test sequence with regard to time or cost.

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

The paper is constructed as follows. Section 4.1.2 explains in short the separate integration sequencing and test sequencing methods. Section 4.1.3 explains the combined integration and test sequencing method. Section 4.1.4 describes the strategies that can be used to determine the starting moment of test phases. Section 4.1.5 shows the algorithm used to solve integration and test sequencing problems. In Section 4.1.6, a case study in the development of a lithographic machine software system is performed to show the benefits of applying this method. Finally, Section 4.1.8 contains the conclusions.

4.1.2 Background

This paper section is intentionally removed from this chapter because more detailed and elaborate descriptions of the test sequencing and integration sequencing methods have already been discussed in Sections 2.1 and 3.1 or can be found in the original version of this paper.

4.1.3 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 sequences 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 is calculated using the test sequencing algorithm. Therefore, the objective function of the integration sequencing algorithm is changed.

**Model**

The complete integration and test sequencing problem called $D$ can be formulated in terms of the 12-tuple $(M, I, T, S, C^m, C^i, C^t, R_{im}, R_{tm}, R_{is}, R_{ms})$, 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.
4.1. INTEGRATION AND TEST SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

Table 4.1: $M, I, C^m, C^i$ 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        | 0        | 1        |
| $i_5$   | 0        | 0        | 0        | 1        | 0        | 0        | 1        | 1        |
| $i_6$   | 0        | 0        | 1        | 1        | 0        | 0        | 1        | 1        |
| $C^m$   | 10       | 15       | 10       | 10       | 15       | 20       | 25       |

Table 4.2: $T$, $C^i$ and $R^{im}$ of the integration model

<table>
<thead>
<tr>
<th>$T$</th>
<th>$R^{im}$</th>
<th>$C^i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$ through $t_6$</td>
<td>${m_1}$</td>
<td>1</td>
</tr>
<tr>
<td>$t_7$ through $t_8$</td>
<td>${m_1, m_2}$</td>
<td>2</td>
</tr>
<tr>
<td>$t_9$ through $t_{13}$</td>
<td>${m_4}$</td>
<td>1</td>
</tr>
<tr>
<td>$t_{14}$ through $t_{17}$</td>
<td>${m_1, m_4}$</td>
<td>2</td>
</tr>
<tr>
<td>$t_{18}$ through $t_{20}$</td>
<td>${m_3, m_5, m_6, m_7}$</td>
<td>3</td>
</tr>
<tr>
<td>$t_{21}$ through $t_{25}$</td>
<td>${m_1, m_2, m_3, m_4, m_5, m_6, m_7}$</td>
<td>5</td>
</tr>
</tbody>
</table>

Illustration

The scanner example introduced in Section 3.1.2 is extended with $R^i$ and $R^{ms}$ to illustrate the integration and test sequencing method. Elements $M, I, R^{im}, C^i, C^m$ are shown in Table 4.1, elements $T, R^{im}, C^f$ in Table 4.2, elements $T, S, R^i, C^f$ ($P$ is no part of the integration and test model) in Table 4.3 and the new elements $R^i, R^{ms}$ in Table 4.4. A part of Table 4.3 has been shown previously as the test model in Table 2.1.

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 $S_U \subseteq S'$ of fault states a test sequence $G^t(S_U)$, with tests from $T'$ that iso-
lates and fixes every fault state in $S_U$. The total time of such a complete test and integration solution is:

$$J^i(G^i) = \max_{m \in M} \left( C^m_i(m) + \sum_{t \in G^i_t(m)} C^i(t) + \sum_{(S', T') \in G^i_{st}^m} J^i(S', T') \right)$$  (4.1)

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

$$J^i(S', T') = \sum_{S_U \subseteq S'} \sum_{t \in G^i_t(S_U)} \left( C^i(t) \prod_{(s, p) \in S_p \cap S_U} p \prod_{(s', p') \in S_{p'} \cap (S' \setminus S_U)} (1 - p') \right)$$  (4.2)

Here, $S_p$ 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 4.13. This fault
4.1. INTEGRATION AND TEST SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

Table 4.4: Elements $R^i$, $R^{ims}$ of the integration and test model

<table>
<thead>
<tr>
<th>$S$</th>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$m_3$</th>
<th>$m_4$</th>
<th>$m_5$</th>
<th>$m_6$</th>
<th>$m_7$</th>
<th>$i_1$</th>
<th>$i_2$</th>
<th>$i_3$</th>
<th>$i_4$</th>
<th>$i_5$</th>
<th>$i_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>10%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_2$</td>
<td>10%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_3$</td>
<td>10%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_4$</td>
<td>10%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_5$</td>
<td>10%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_6$</td>
<td>2%</td>
<td>2%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_7$</td>
<td>2%</td>
<td>2%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_8$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_9$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_{10}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_{11}$</td>
<td>0</td>
<td>2%</td>
<td>2%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_{12}$</td>
<td>0</td>
<td>2%</td>
<td>2%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$s_{13}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>$s_{14}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>$s_{15}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>$s_{16}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>$s_{17}$</td>
<td>0</td>
<td>1%</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>$s_{18}$</td>
<td>0</td>
<td>1%</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>$s_{19}$</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
</tr>
<tr>
<td>$s_{20}$</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
</tr>
<tr>
<td>$s_{21}$</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
</tr>
<tr>
<td>$s_{22}$</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
</tr>
<tr>
<td>$s_{23}$</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
</tr>
<tr>
<td>$s_{24}$</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
<td>5%</td>
</tr>
</tbody>
</table>

The probability depends on the integrated modules (increase by $R^{ims}$), the created interfaces (increase by $R^{ims}$) 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 $G^i$ that has minimal expected test cost $J^i$, from all possible solutions $G^i$:

$$J^i = f^i(G^i) = \min_{G^i \in \mathcal{G}} f^i(G^i)$$

(4.3)

4.1.4 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 4.1, 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:

\[ P_s(S_p) = 1 - \prod_{p:(s,p) \in S_p} (1 - p) \]  

(4.4)

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.

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 4.1. 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 4.1. 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 4.1. 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.

**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:

$$S'_A(S_p, M') = \{ s(t, p) \in S_p \land (\exists t, M'') : t \in T \land M'' \subseteq M' \land s \in R^{ts}(t) \land M'' \in R^{tm}(t) \}$$  (4.5)

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

$$P'_A = (s, p) \in S_p \text{ for } s \in S'_A$$  (4.6)

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:

$$T'_A = \{ t \mid t \in T : (\exists M'' : M'' \in R^{im}(t) : M'' \subseteq M') \land R^{ts}(t) \cap S'_A \neq \emptyset \}$$  (4.7)
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:

\[
S'_O(S_p, M') = \{ s | (s, p) \in S_p \land s \not\in \bigcup m' : m' \in M \setminus M' : R^{\text{int}}(m') \land s \not\in \bigcup i' : i' \in I \land R^{\text{int}}(i') \in (M \setminus M') : R^{\text{int}}(i') \land \exists (t, M'') : t \in T \land M'' \subseteq M' \land s \in R^{\text{int}}(t) \land M'' \in R^{\text{int}}(t) \}\]

(4.8)

Furthermore \( P'_O \) can be calculated using equation 4.6 only now using \( S'_O \) instead of \( S'_A \). \( T'_O \) can be calculated using equation 4.7 only now using \( S'_O \) instead of \( S'_A \). These elements are needed in the remainder of the paper.

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-defined 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:

\[
S'_T(S_p, M') = \begin{cases} 
S'_A(S_p, M') & \text{if } P_s(S_p) > b \\
\emptyset & \text{else} 
\end{cases}
\]

(4.9)

Here, \( P_s(S_p) \) is the current system fault probability and is determined using equation 4.4. Furthermore \( P'_T \) is calculated using equation 4.6 only now using \( S'_T \) instead of \( S'_A \). \( T'_T \) can be
calculated using equation 4.7 only now using \( S'_T \) instead of \( S'_A \). These elements are needed in the remainder of the paper.

**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:

\[
S'_p(S_p, M') = \begin{cases} 
S'_A(S_p, M') & \text{if } d \geq b \\
\emptyset & \text{else}
\end{cases} 
\]  

(4.10)

Here, \( P'_p \) can be calculated using equation 4.6 only now using \( S'_p \) instead of \( S'_A \). \( T'_p \) can be calculated using equation 4.7 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.

**4.1.5 Solution algorithm**

In this section, we propose a solution algorithm for the integration and test sequencing problem. This algorithm is based on the ‘assembly by disassembly’ approach used by de Mello *et al.* [de Mello and Sanderson, 1991b] and suggested by Delchambre *et al.* [Delchambre *et al.*, 1989] for the integration sequencing part and the ‘sequential diagnosis approach’ suggested by Pattipati *et al.* [Pattipati and Alexandridis, 1990] 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 [Boumen *et al.*, 2006a], while the algorithm used by the sequential diagnosis approach has been extended in [Boumen *et al.*, 2006d,c] 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 4.2. 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
4. INTEGRATION AND TEST PLAN OPTIMIZATION

Figure 4.2: Integration AND/OR graph illustration

integration model consisting of 3 modules \((m_1, m_2, m_3)\), which are all connected to each other with 3 interfaces \((i_1 \text{ connects } m_1 \text{ and } m_2, i_2 \text{ connects } m_1 \text{ and } m_3, i_3 \text{ connects } m_2 \text{ and } m_3)\) and 4 tests that need to be applied: 3 tests that each need one of the modules \((t_1 \text{ requires } m_1, t_2 \text{ requires } m_2, t_3 \text{ requires } m_3)\) and one system test \(t_4\) that requires all modules. The AND/OR graph in Figure 4.2 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 \((G_1, G_2, G_3)\) that can be distinguished at the root OR node where there are 3 AND nodes to choose from. For each solution \(G\) the cost \(J(G)\) can be calculated. Then, the cheapest solution is chosen. The chosen solution is then used in the reverse 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 \(x^t\) which is an element of \(X^t = \mathcal{P}(\mathcal{P}(S))\), 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 4.3. 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 \(G\) the cost \(J(G)\) 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 con-
structured. This means that we start with the separate modules and end with the completely integrated 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.

Figure 4.3: Test AND/OR graph illustration

The test sequencing algorithm as presented in [Boumen et al., 2006c] 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 cost of a test phase. The search starts with the initial root integration OR node that denotes the completely integrated and tested system: $x_{init}'$. The cost of this particular OR node is called $f_i^j(x_{init}')$ 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 ($x_1^i$ and $x_2^i$). For a given system state $x^i$,
this can be determined as follows:

\[
A_s(x^i) = \{a | a \subseteq I_s(x^i) : \exists x^i_1, x^i_2 \\
\text{s.t. } x^i_1 \cap x^i_2 = \emptyset \land x^i_1 \cup x^i_2 = x^i \land x^i_1 \neq \emptyset \land x^i_2 \neq \emptyset \\
\land \forall m', m'' \in x^i_1 \land m', m'' \in x^i_1 : \text{conn}(m', m'', I_s(x^i) \setminus a) \\
\land \forall m' \in x^i_1, m'' \in x^i_2 : \text{conn}(m', m'', I_s(x^i) \setminus a) \}
\]  

(4.11)

where \(I_s(x^i)\) denotes all interfaces between the modules in system state \(x^i\) and function \(\text{conn}\) 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 linear time per cut-set. We use the algorithm as described by Tsukiyama [Tsukiyama et al., 1980]. For the simple system of which the AND/OR graph is shown in Figure 4.2, the cut-sets are: \({i_1, i_2}\}, \{i_1, i_3\}, \{i_4, i_5\}\ for the initial OR node. For the scanner illustration in Section 4.1.2, the cut-sets are: \({i_1}, \{i_2\}, \{i_4\}, \{i_5\}\) for the initial OR node.

The second step is to construct an AND node for every cut-set \(a \in A_s(x^i)\) 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 4.11), by breaking the interfaces in \(a\).

The total cost of an AND node \(f^i_s(x^i, a)\) for a system state \(x^i\) and a disassembly action \(a \in A_s(x^i)\) 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:

\[
f^i_s(x^i, a) = \max(f^i_s(x^i_1), f^i_s(x^i_2)) + \sum_{i \in a} C^i(i) + J_s(x^i, S_p(x^i, a))
\]  

(4.12)

Here, \(S_p(x^i, a)\) is a tuple of fault states and their corresponding fault probabilities and is calculated by:

\[
S_p(x^i, a) = \text{Unify} (\{\{s, p\} \in S_p(x^i_1, a_1) | s \notin S^i\} \cup \{\{s, p\} \in S_p(x^i_2, a_2) | s \notin S^i\} \cup \{\cup R^i(i) | i \in a\})
\]  

(4.13)

Here, \(a_1, a_2\) are the cut-sets that had minimal cost at respectively system states \(x_1\) and \(x_2\). \(S^i_1, S^i_2\) 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 \(\text{Unify}\) 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:

\[
\text{Unify}(S_p) = \{(s, \prod_{(s, p') \in S_p} (1 - p')) | (s, p) \in S_p\}
\]  

(4.14)
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:

\[
J^i(x^i) = \begin{cases} 
C^m(m) + J_t(x^i, R^{ms}(m)) & \text{if } x = \{m\} \\
\min_{a \in A_i(x^i)} (J^i_a(x^i, a)) & \text{otherwise} 
\end{cases}
\] (4.15)

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:

\[
J_t(x^i, S_p) = \begin{cases} 
J^i_o(P(S'_A), T'_A, P'_A) & \text{if } w = 'A' \\
J^i_o(P(S'_O), T'_O, P'_O) & \text{if } w = 'O' \\
J^i_o(P(S'_T), T'_T, P'_T) & \text{if } w = 'T' \\
J^i_o(P(S'_P), T'_P, P'_P) & \text{if } w = 'P'
\end{cases}
\] (4.16)

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

\[
J^i_o(x^i, T', P') = \begin{cases} 
0 & \text{if } x^i = \emptyset \\
J^i_o(x^d, T', P') & \text{if } x^i \neq \emptyset \land T'' = \emptyset \\
\min_{t \in T''} (J^i_a(t, x^i_{fi}) + C^i(t)) & \text{else}
\end{cases}
\] (4.17)

Where:

- \(x^i_{fi}\) is the fixed system state. This is the system state without the fault states that are definitely present.
- \(x^i_d\) 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, \(J^i_a(t, x^i)\) denotes the cost of an AND node which is determined by the cost of the two succeeding OR nodes (pass or fail) and the probabilities that these OR nodes are reached, that is:
\[ J_i(t, x^t) = p_p \cdot J_i(x_{p}^t, t', P') + p_f \cdot J_i(x_{f}^t, t', P') \quad (4.18) \]

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 Section 4.1.7. If the root node is solved, which means that \( J_i(x_{init}) \) 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.

**Illustration**

For the illustration introduced in Section 4.1.2, we calculated the optimal solution for the four introduced strategies. The results of this experiment are shown in Table 4.5. 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 4.4(a), while test phases \( T_{m1}, T_{re} \) and \( T_{waf} \) are shown in Figures 4.4(b), 4.4(c) and 4.4(d). Test phase \( T_{m1} \) is the same as shown in Figure 2.4(a). 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 4.5 as a Microsoft Project Gantt chart. In this chart the durations and sequence of actions are shown.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>( J^r )</th>
<th>( \sum J^r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>36.68</td>
<td>20.97</td>
</tr>
<tr>
<td>O</td>
<td>36.68</td>
<td>20.97</td>
</tr>
<tr>
<td>T (45%)</td>
<td>38.79</td>
<td>19.38</td>
</tr>
<tr>
<td>P (40 hours)</td>
<td>39.23</td>
<td>18.82</td>
</tr>
</tbody>
</table>
4.1 INTEGRATION AND TEST SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

4.1.6 Case study

This paper section is intentionally removed from this chapter because a more detailed and elaborate description of the case studies can be found in Section 5.3 or in the original version of this paper.

4.1.7 Algorithm

This section gives a formal, functional-style [Bird, 1998] 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 \((M, I, T, S, C^m, C^i, C^t, R_{im}, R_{it}, R_{is}, R_{ms})\). To this end, the following expression can be used:

\[
(J^i, H^i) = \text{OR}(x_{\text{init}}^i, A_x(x_{\text{init}}^i), H_{\text{init}}^i)
\]  

where \( H_{\text{init}}^i : \mathcal{P}(M) \times \mathcal{P}(S \times \mathbb{R}^+) \rightarrow \{\perp\} \) 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. \( A_x(x_{\text{init}}^i) \) are all cut-sets of the initial system state \( x_{\text{init}}^i \), calculated using the algorithm presented in [Tsukiyama et al., 1980]. Furthermore, \( x_{\text{init}}^i = M \) 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 4.3. 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 $\text{OR}_i : X^i \times P(P(I)) \times H^i \rightarrow \mathbb{R}^+ \times P(S \times \mathbb{R}^+) \times H^i$ calculates the cost of a system state $x'$ given the possible disassembly actions and the initial $H^i$. The cost of such an integration OR node is defined by Equation 4.15. The $\text{OR}_i$ function is defined as follows. If the state has already been solved (i.e. $H^i(x') \neq \bot$), the solution of that solved state is taken from $H^i$ and returned. Otherwise, several options are possible. If $x'$ consists of one module, the node is a leaf node and the resulting cost is the module development cost and the remaining test cost. If $x'$ 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:
4.1. INTEGRATION AND TEST SEQUENCING IN COMPLEX MANUFACTURING SYSTEMS

\[
OR_i(x^i, A_x, H^i) = \begin{cases} 
(H^i(x^i), H^i) & \text{if } H^i(x^i) \neq \bot \\
(J', S'_p, H^i(x^i/(J', \{\}))) & \text{if } H^i(x^i) = \bot \land x^i = \{m'\} \\
(J'', S''_p, H''(x^i/(J'', a''))) & \text{if } H^i(x^i) = \bot \land x^i \neq \{m'\} 
\end{cases}
\]

(4.20)

where:

- \( J' = C^m(m') + J_i(x^i, R^ms(m'))..0 \), where .0 denotes the first element of a tuple, and .1 the second.
- \( S'_p = \{(s, p) \in R^ms(m') | s \notin J_i(x^i, R^ms(m'))..1\} \)
- \( J'' = J_a(a'') = \min_{a \in A_x} J_a(a) \), is the minimal cost of \( x^i \), and \( a'' \) is the cut-set from \( A_x \) for which this holds.
- \( \{J_a(a), H^i\} = \text{AND}_i(x^i, rmv(A_x, a), a, H^i)\.0 + J_i(x^i, S_p(x^i, a))\.0 \)
  
  for \( j = 1, \ldots, |A_x| \) (where \( H^i = H^i \)), are the minimal test cost and updated \( H^i \) for each cut-set in \( A_x \). Furthermore, \( H'' = H''|_{A_x} \). In this equation \( S_p \) can be determined as follows: \( S_p(x^i, a) = \text{AND}_i(x^i, rmv(A_x, a), a, H^i)\.1 \)
- \( S''_p = \{(s, p) | (s, p) \in \text{AND}_i(x^i, rmv(A_x, a''), a'', H'')\.1 \)
  \& \( s \notin J_i(x^i, S_p(x^i, a''))..1\} \), 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 4.11.

The function \( \text{AND}_i : X^i \times \mathcal{P}(I) \times \mathcal{P}^{\mathcal{P}(I)} \times \mathcal{H} \rightarrow \mathbb{R}^* \times \mathcal{P}(S \times \mathbb{R}^*) \times \mathcal{H} \) 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.

\[
\text{AND}_i(M, a, A_x, H^i) = (\sum_{i \in a} C(i) + \max(J', J''), S_p, H'') 
\]

(4.21)

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 \( H'' \).
4. INTEGRATION AND TEST PLAN OPTIMIZATION

- \((J^w, S_p^w, H^w)\) = \(OR_i(x^i_2, A_x, H^w)\), calculates the cost of system state \(x^i_2\) and the updated \(H^w\).

- \(S_p = S_p^1 \cup S_p^2 \cup \sum_{i \in a} R^i(i)\) 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 \(J_i\) that is used in the integration sequencing algorithm.

Let function \(J_i : X^i \times P(S \times \mathbb{R}_+) \times P(I) \rightarrow \mathbb{R}_+ \times P(S)\) 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 \(a\):

\[
J_i(x^i, S_p, a) = \begin{cases} 
(OR(S^i_A, T^i_A, P^i_A, x^i_{init}, H^i_{init}, T^i_{init}), a, S^i_p) & \text{if } w = 'A' \\
(OR(S^i_O, T^i_O, P^i_O, x^i_{init}, H^i_{init}, T^i_{init}), a, S^i_O) & \text{if } w = 'O' \\
(OR(S^i_T, T^i_T, P^i_T, x^i_{init}, H^i_{init}, T^i_{init}), a, S^i_T) & \text{if } w = 'T' \\
(OR(S^i_p, T^i_p, P^i_p, x^i_{init}, H^i_{init}, T^i_{init}), a, S^i_p) & \text{if } w = 'P'
\end{cases}
\] (4.22)

Where:

- Function OR\(_i\) returns the cost of a test phase. This is done by constructing a test AND/OR graph using two functions OR\(_i\) and AND\(_i\) that are both described formally in [Boumen et al., 2006d].

- \(S^i_A, S^i_O, S^i_T, S^i_p, T^i_A, T^i_O, T^i_T, T^i_p, P^i_A, P^i_O, P^i_T\) and \(P_p\) are defined in section 4.1.4.

- \(H^i_{init} : X^i \rightarrow \left\{ \perp \right\}\) is the initial function that gives the cost of a solved OR node.

- \(x^i_{init} = (\varnothing, \varnothing)\) denotes the initial test state.

- \(T^i_{init} = \varnothing\) denotes the initial performed test set.

4.1.8 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 in 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.

4.2 OPTIMAL TEST PHASE POSITIONING

In the previous section, we used test positioning strategies to determine when a test phase starts and when it stops. We introduced several strategies that can be used for this purpose. However, none of these strategies is truly optimal given a certain situation. Therefore, in this section we show a method that may be used to determine the optimal test positioning strategy given a predefined integration sequence. This method calculates the optimal division of test phases given an integration sequence by determining for each integration action the best test phase such that the total time-to-market for the integration and test phase is minimal. In the first subsection, the method is explained and illustrated with a small example. In the second subsection, we discuss how this method may be used in practice. The content of this section has not been published in a paper.

4.2.1 Method

The method that is proposed consists of defining a model and an integration sequence and then calculating the optimal solution. The model that is used is the same as the integration and test model defined in the previous section \( (M, I, T, S, C^m, C^i, C^t, R_{im}, R_{tm}, R_{ts}, R_{ms}) \). Furthermore, the input consists of a predefined integration sequence, represented by a function \( G_i: \mathcal{P}(M) \rightarrow \mathcal{P}(I) \times (\mathcal{P}(M) \times \mathcal{P}(M)) \), that gives for certain subassemblies consisting of a set of modules from \( M \), the integration action (a set of interfaces \( \mathcal{P}(I) \)) and the two subassemblies that formed this subassembly. Note that this integration sequence is predefined and therefore only has one integration action for every subassembly.

Illustration

As an illustration, we use the telephone example as introduced in Chapter 2. The telephone consists of three modules that are connected by two interfaces; interface \( i_1 \) connecting the device \( m_1 \) and cable \( m_2 \), and interface \( i_2 \) connecting the cable \( m_2 \) and horn \( m_3 \). As shown in Table 4.6(a), this telephone has five fault states and six tests that can be performed. All tests can be performed with all possible subassemblies. The modules and interfaces introduce fault states with a certain probability as shown in Table 4.6(b). The development times and interface creation times are also shown in this table. The last part of the input is the integration sequence as illustrated in Figure 4.6. In this example, the device and cable (\( m_1 \) and \( m_2 \))
are integrated first, then the horn \((m_3)\) is integrated. The cost of this integration sequence is 6 time units. This cost is determined by the longest path in the integration sequence consisting of the development of \(m_3\) (5 time units) and the interface creation time of \(i_2\) (1 time unit).

**Optimization**

The test positioning problem is now defined as follows. Find for each predefined integration action a test sequence such that the time-to-market of the complete integration and test plan is minimal.

To calculate the optimal solution, an AND/OR graph is constructed that represents all possible solutions to this problem. An AND node in this AND/OR graph represents a test phase and a disassembly action. Such an AND node results in one or two OR nodes. From such an OR node, multiple AND nodes are possible and the best AND node must be chosen for each OR node. An OR node represents the system state \(x\) which is denoted by a tuple of excluded fault states and the integrated modules, and is defined as an element of \(\mathcal{P}(M) \times \mathcal{P}(S)\).
A test phase is defined as a set of fault states that are excluded during that test phase, and is an element of $\mathcal{P}(S)$. The initial OR node is the completely integrated and tested system. By applying AND nodes (disassembly actions) the system is disassembled to single modules that have not been tested.

An example of such an AND/OR graph is shown in Figure 4.7 which shows a part of the AND/OR graph of the example introduced and the optimal solution to this example. The numbers on the edges denote the cost of the underlying graph, which is the cost of the longest path in that graph. Here, a sketch of an algorithm that constructs such an AND/OR graph and finds the optimal solution is presented:

1. Start with the initial OR node: $x = (M, S)$, namely the state in which all modules are integrated and all fault states are excluded (the final desired state). If the subassembly consists of more than two modules, and must therefore be disassembled, continue with step two. Else, the OR node is a leaf node so proceed to step 4.

2. Determine the fault states that absolutely need to be excluded in the next test phase. That is, the fault states set $S_e(x)$ that either cannot be tested anymore since certain tests cannot be performed anymore after the disassembly action, or with fault states that were introduced by that integration action. This faulty states set is defined as:

$$S_e(x) = \left\{ s : s \in x.1 \ni \exists p : (s, p) \in \bigcup_{i \in G_i(x.0).0} R_i(i) \vee s \not\in S_{\text{st}}(G_i(x.0).1.0) \cup S_{\text{st}}(G_i(x.0).1.1) \right\}$$  \hspace{1cm} (4.23)

where $S_{\text{st}}$ determines the fault states that can be tested with the tests that are possible given the set of modules that are integrated.

3. Determine the possible test phases $S_T(x)$, where a test phase is defined as an element of $\mathcal{P}(S)$, for the system state $x$ as follows:

$$S_T(x) = \{ S' : S' \subseteq x.1 \land S_e(x) \subseteq S' \}$$  \hspace{1cm} (4.24)

that is, all fault state sets that are a subset of the excluded fault state set $x.1$, including the fault state that has to be excluded $S_e(x)$. Create for all these possible test phases an AND node that excludes fault states $S'$ and performs the following steps:

(a) Calculate the cost $J_T$ of the test phase excluding fault states $S'$ given the test set $T'(x.0)$ that is possible given the modules integrated (see section 4.1.4). This calculation can be done with the test sequencing algorithm. Determine the cost of the disassembly action $J_I$, that is:

$$J_I = \sum_{i \in G_i(x.0).0} C^i(i)$$  \hspace{1cm} (4.25)
(b) Determine the two resulting OR nodes, \( x_1 \) and \( x_2 \), that are formed after the disassembly action and test phase as follows for \( x_1 \):

\[
x_1 = (G_i(x.o).i.o, \{ s : s \in x.I \setminus S' \land s \in S_i(G_i(x.o).i.o) \})
\]

\[
x_2 = (G_i(x.o).i.1, \{ s : s \in x.I \setminus S' \land s \in S_i(G_i(x.o).i.1) \})
\]

where \( S_i(G_i(x.o).i.o) \) is the set of fault states that are introduced by the modules in the system state \( x_1 \) or by the creation of the interfaces between these modules. Calculate for each of these OR nodes the cost \( J_1 \) and \( J_2 \) by starting the algorithm again with step 1, only now with \( x_1 \) or \( x_2 \) as initial OR node.

(c) return the cost of the AND node (test phase and disassembly action), which is:

\[
J = J_T + J_1 + \max(J_1, J_2)
\]

4. If the subassembly needs to be disassembled select the AND node that has the minimal cost \( J \) and return the cost of this AND node as the cost of the optimal solution. If not, determine the cost \( J_T \) of testing the final fault states \( x.I \) and return the cost

\[
J = J_T + \sum_{m \in x.o} C^m(m)
\]

The optimal solution for the telephone example as shown in Figure 4.7, takes 9 time units to perform and starts with a test phase that only tests fault state \( s_5 \). Then, fault state \( s_3 \) is tested in parallel with fault states \( s_1, s_2, s_4 \) on the two different subassemblies that were formed. If all fault states would be tested during the first test phase, the total cost of the tree would be 11.25 and thus higher than of the optimal solution. Also, if fault states \( s_1 \) and \( s_2 \) would be tested in parallel, the cost would be higher than testing them together with fault state \( s_4 \). The test phase testing fault states \( s_1, s_2, s_4 \) is much more efficient than testing first fault state \( s_4 \) and after that testing \( s_1 \) and \( s_2 \) in parallel. This test phase is more efficient because more tests are available to use.

4.2.2 Conclusions

With the proposed method, it is possible to determine the optimal test positioning strategy for a given integration sequence. However, the construction of an AND/OR graph can be very time consuming because there are many test phases possible given a set of fault states. Even in the small example illustrated in Figure 4.7, there are 16 possible test phases at the initial OR node. This makes this method very hard to use for practical problems. We have therefore not applied this method to any case study (yet). More research is required to create algorithms that more efficiently determine optimal solutions to this problem, or to create heuristics that determine good but near-optimal solutions to this problem.

A second drawback of this approach is the fact that it is not possible to use inconclusive tests because this would make the solution space infinite and thus unsolvable. The reason for this is that there are infinitely many test phases possible since it is also possible to reduce a part of the probability of a fault state during a test phase.
4.3 HIERARCHICAL INTEGRATION AND TEST SEQUENCING

In Chapters 2 and 3, we explained that in some situations it is necessary to define and solve a test or integration planning problem hierarchically. For the combination of integration and test planning this may also be needed. This section briefly explains a method to solve an integration and test problem hierarchically using the already developed hierarchical test planning and hierarchical integration planning methods. The content of this section has not been published in a paper.

To do so, we need a formal definition of a hierarchical integration and test model. This
The hierarchical model is a combination of the hierarchical test model and the hierarchical integration model and is defined as a quintuple: \((D^i, D^t, R^{ms}, R^{is}, R^{tm})\) where:

- \(D^i = \{D^i_1, \ldots, D^i_K\}\) is a set of \(K\) single integration models \(D^i_k\). Each single integration model \(D^i_k\) is a septime: \(D^i_k = (L^i_k, M^i_k, I^i_k, C^m_k, C^i_k, R^{tm}_k, R^{ld}_k)\) for \(k = 1, \ldots, K\), where the elements are defined in Chapter 3.

- \(D^t = \{D^t_1, \ldots, D^t_J\}\) is a set of \(J\) single test models \(D^t_j\). Each single test model \(D^t_j\) is a quintuple: \(D^t_j = (l^t_j, S^t_j, T^t_j, R^{st}_j, C^t_j)\) for \(j = 1, \ldots, J\), where the elements are defined in Chapter 2.

- \(R^{tm} : T_j \rightarrow \mathcal{P}(\mathcal{P}(M_k) \times \mathcal{P}(L_k))\) for \(j = 1, \ldots, J\) and \(k = 1, \ldots, K\), gives for each single test in \(T_j\) (no group tests) its essential assemblies; where an essential assembly describes the modules or clusters that must be integrated with each other before the test can be performed.

- \(R^{is} : I_k \rightarrow \mathcal{P}(S^t_j \times \mathbb{R}^+)\) for \(j = 1, \ldots, J\) and \(k = 1, \ldots, K\), gives for each interface in \(I_k\) the single fault states (no group fault states) that are introduced when creating this interface and the probabilities of each fault state introduced.

- \(R^{ms} : M_k \cup L_k \rightarrow \mathcal{P}(S^t_j \times \mathbb{R}^+)\) for \(j = 1, \ldots, J\) and \(k = 1, \ldots, K\), gives for each module and cluster in \(M_k, L_k\) the single fault states (no group fault states) that are introduced when developing this module and the probabilities of each fault state introduced.

We can use the same hierarchical algorithms presented in Chapters 2 and 3 to calculate the hierarchical solution to this problem. The only additional assumption we have to make is that a group test may only be performed when all its children tests may be performed (according to the \(R^{tm}\) relations).

Note that the test model hierarchy and the integration model hierarchy may be different because they are independent of each other. They are independent because the relations between these models do not have any hierarchy: the \(R^{tm}, R^{is}\) and \(R^{ms}\) relations are defined on single tests and single fault states.

Because this hierarchical method does not differ from the already presented methods, we do not give an example or present a case study.

### 4.4 Conclusions

This chapter has answered research questions 3.1, 3.2 and 3.3 as defined in the first chapter:

**Answer to Question 3.1:** The structure of an integration and test plan is defined as a tree of integration actions and test phases, where the leaf nodes of that tree are the development actions of modules and the root node is the completely integrated and tested system. Each module is integrated with other modules, and test phases are performed until the final system is completely integrated. Each test phase is defined as a tree of test sequences combined with fix and diagnosis actions. The test sequence
that is followed depends on the outcome of the executed tests. Furthermore, each test sequence has a certain stop moment. The total integration and test plan has associated maximal and total costs and durations. A plan is judged on these parameters.

**Answer to Question 3.2:** The information needed to construct an integration and test plan is the integration and test model, consisting of modules, interfaces, tests, fault states, the properties of these elements and the relation between these elements. This model describes the integration and test problem. Furthermore, an objective function, consisting of several parameters, must be provided that is used to determine the optimal plan. Constraints can be defined for certain parameters. Finally, a test phase positioning strategy is provided that decides when test phases are started and when they should stop.

**Answer to Question 3.3:** The method that creates an optimal integration and test plan based on a test phase positioning strategy, given the integration and test model and objectives and constraints is constructing several AND/OR graphs containing all possible solutions. To determine the integration sequence, an AND/OR graph is constructed where an AND node denotes an integration action and an OR node denotes a system state consisting of the integrated modules in the system. The AND/OR graph is constructed using the assembly by disassembly approach, this means starting with the completely integrated system and disassembling this system into single modules. After each integration action, a test phase is constructed if this is allowed by the test phase positioning strategy. This is done by constructing an AND/OR graph where an AND node denotes a test and an OR node denotes a system state consisting of the possible fault states that can be present in the system. The best solution for this test phase is determined using an AND/OR graph search algorithm that is created for this specific problem. Also, for the integration sequencing problem the best solution is determined using an AND/OR graph search algorithm that is created for this specific problem.

With the integration and test plan method, we are able to optimize integration and test plans for different domains, such as prototype integration and test, manufacturing integration and test, and software integration and test domains. Furthermore, this method reduces the effort of creating and maintaining integration and test plans because the optimization can be done automatically. The integration and test model gives insight in the integration and test problem and can be used as a knowledge container to train new engineers. Furthermore, the method can be used to judge ‘what-if’ scenarios that are constructed to show the benefits of certain new investments. In Chapter 5, several case studies show how this method can be used in real life and show the benefits of using this method.
In this chapter, we describe several case studies that were performed with the methods introduced in the previous chapters at ASML [ASML, 2006]. These case studies were performed to show the benefits of using this method and to show how the methods can be used in practise for real-life problems.

Section 5.1 is based on a paper that describes how to reduce the test time for system manufacturing test phases, using the test plan optimization method. Section 5.2 is based on a paper that describes how to reduce the integration time for system development integration phases, using the integration plan optimization method. Section 5.3 is based on a paper that describes how to reduce the integration and test time for software integration and test phases, using the integration and test plan optimization method. In the last section, conclusions are drawn about the performed case studies.

5.1 **TEST PLANNING FOR LITHOGRAPHIC SYSTEMS**

This section is based on the paper titled *Test time reduction by optimal test sequencing* [Boumen et al., 2006b] that is accepted and presented at the International Council of Systems Engineering (INCOSE) 2006 Symposium and has appeared in the proceedings of this Symposium. The paper section dealing with the method has been removed in this thesis because a more detailed and elaborate description of the method can be found in Chapter 2.
Test time reduction by optimal test sequencing

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

Abstract

Testing complex manufacturing systems, like ASML lithographic machines [ASML, 2006], can take up to 45% of the total development time of a system. This test time can be reduced by choosing wisely which test cases must be performed in which sequence, without making investments in test cases or the system. With the test sequencing method, developed by [Boumen et al., 2006d,c], it is possible to make these decisions such that a time, cost and/or quality optimal test sequence can be constructed. This paper shows with two case studies that test time can be reduced up to 20% without loss of quality with this method. The first case study is related to the test phase during the manufacturing of a lithographic machine. The second case study is related to the reliability testing of a first of a kind lithographic machine.

5.1.1 Introduction

Testing complex manufacturing systems is expensive both in terms of time and money, as shown by [Cusumano and Selby, 1995] and [Engel et al., 2004]. To reduce time-to-market of a new system or to reduce lead time during the manufacturing of these systems, it is crucial to reduce the test time. Reducing test time can be done by: 1) making testing faster, for example by automation of test cases, 2) making testing easier, for example by changing the system and 3) doing testing smarter, for example by choosing wisely which test cases to perform and in what sequence. Much research has been done on the first two aspects, less research is being done on the third aspect. In this paper we show that test time can be reduced by optimizing test sequences which allows testing to be performed more efficiently. The decision of which tests are performed and which are not is important. Not performing a test case may leave crucial faults in the system, while performing a test case can lead to an increase of lead time or time-to-market. Deciding what to test is especially important in the time-to-market driven semiconductor industry and for companies providing manufacturing systems to this industry such as ASML, a provider of lithographic systems. This is caused by the time-to-market pressure of delivering such machines to customers and the high costs associated
with solving defects during system operation. For the optimization of test sequences three parameters are of interest: the test time, the test cost and the quality of the system after testing. In [Boumen et al., 2006d,c], a test sequencing method has been developed that optimizes the selection and sequencing of tests. The main focus in these papers is on the mathematical models and algorithms. In this paper, we explain how this method can be used to reduce test time. In addition, we describe two case studies in which test time is reduced by using this method. The first case study is related to the manufacturing test phase of a lithographic machine. The second case study is related to the reliability test phase of a new lithographic system. The structure of the paper is as follows. Section 5.1.2 shortly explains the test sequencing method. Section 5.1.3 describes the two case studies. The last section gives the conclusions of our work.

5.1.2 Test sequencing method

This paper section is intentionally removed from this chapter because a more detailed and elaborate description of the test sequencing method has already been discussed in Section 2.1 and can be found in the original version of this paper.

5.1.3 Case studies

In this section, we describe two case studies where we reduced test time with the test sequencing method presented. The first case study is related to a test phase during the manufacturing of a lithographic machine. The second case study is related to a reliability test phase during the development of a new type of lithographic machine.

Manufacturing case study

During the manufacturing of a lithographic machine, two large test phases are performed. Test phase 1 is performed at ASML. After successful accomplishment of this phase, the system is disassembled, transported to the customer, and there assembled again. After the system is installed at the customer’s site, test phase 2 is performed. The two test phases are almost identical, and both include multiple job-steps. Each job-step consists of multiple tests executed in a sequence. Currently, the sequence does not depend on the outcome of a test: if a test fails, the problem is diagnosed until the cause of the failure is found and repaired. In these job-steps, faults introduced during manufacturing, assembly and transport must be discovered and system parameters need to be calibrated. The case study is performed for three job-steps in both test phases. Test models have been made by ASML engineers and the test sequences have been optimized towards test time, while testing stops when the risk is 0 hours (stop criterion 1). The parameters of the three models related to three job-steps are shown in Table 5.1. Since the test problems in test phase 1 and 2 are identical almost the same models can be used: only the probabilities differ between the two test phases, as also shown in Table 5.1.
Table 5.1: Model properties of the manufacturing case study

<table>
<thead>
<tr>
<th>Job-step</th>
<th>Number of tests</th>
<th>Number of fault states</th>
<th>Average fault probability test phase 1</th>
<th>Average fault probability test phase 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>39</td>
<td>73</td>
<td>30%</td>
<td>16%</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>15</td>
<td>86%</td>
<td>71%</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>60</td>
<td>46%</td>
<td>No data</td>
</tr>
</tbody>
</table>

Table 5.2: Results of the manufacturing case study

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12.2</td>
<td>10.2</td>
<td>17%</td>
<td>12.2</td>
<td>8.3</td>
<td>32%</td>
</tr>
<tr>
<td>2</td>
<td>13.6</td>
<td>13.1</td>
<td>4%</td>
<td>13.6</td>
<td>11.5</td>
<td>16%</td>
</tr>
<tr>
<td>3</td>
<td>33.0</td>
<td>27.0</td>
<td>18%</td>
<td>No data</td>
<td>No data</td>
<td>No data</td>
</tr>
</tbody>
</table>

In general, the fault probabilities are lower in the second test phase, because the system was already performing according to its specifications after the first test phase. The faults found during the second test phase are merely transportation and assembly faults. No data is available for job-step 3 in test phase 2. The results of this case study are shown in Table 5.2, presenting the currently average test time (determined from historical data), the new average test time and the reduction in test time. Due to the currently used fixed test sequence, the test times in both test phases are equal. However, when using the method presented we notice that more test time can be reduced during the second test phase because fault probabilities are in general lower. Due to these lower fault probabilities, testing can start at a higher system level which, in most cases, reduces test time.

Reliability case study

For a new type of a lithographic machine, a test phase needs to be performed to show that the first of a kind machine has a reliability of at least 50 hours MTBF. For the most critical modules (the modules that have been changed) in the system, separate tests have been developed that test a particular module faster than a total system test (normal production) would do. For example, during normal production a robot accomplishes 1 cycle per minute; in a specific test, the robot is tested such that it accomplishes 10 cycles per minute, without testing the rest of the system. In the current situation, these specific module tests are not used to show the total system reliability. Only the system test is used, because current methods, for example SEMI standards (SEMI 2005), only consider this type of test when testing the system reliability. The model of this problem is shown in Table 5.3. There are five critical modules, and a fault state (s1-s5) is associated with each module. An additional fault state (s6) is associated with the rest of the system. The normal production is the system test.
Table 5.3: Model of reliability case study

<table>
<thead>
<tr>
<th>S / T</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Test 4</th>
<th>Test 5</th>
<th>Test 6</th>
<th>P</th>
<th>I(hour)</th>
<th>I($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module 1</td>
<td>1/50</td>
<td>6/50</td>
<td>3/50</td>
<td>2/50</td>
<td>2/50</td>
<td>0</td>
<td>90%</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td>Module 2</td>
<td>1/50</td>
<td>0</td>
<td>10/50</td>
<td>0</td>
<td>4/50</td>
<td>0</td>
<td>80%</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td>Module 3</td>
<td>1/50</td>
<td>0</td>
<td>3/50</td>
<td>0</td>
<td>2/50</td>
<td>0</td>
<td>80%</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td>Module 4</td>
<td>1/50</td>
<td>0</td>
<td>0</td>
<td>2/50</td>
<td>4/50</td>
<td>4/50</td>
<td>80%</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td>Module 5</td>
<td>1/50</td>
<td>0</td>
<td>0</td>
<td>1/50</td>
<td>4/50</td>
<td>10/50</td>
<td>90%</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td>System</td>
<td>1/50</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>50%</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td>C(hour)</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C($)</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4: Results of the reliability case study

<table>
<thead>
<tr>
<th>Test</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>26</td>
</tr>
<tr>
<td>Test 2</td>
<td>0</td>
</tr>
<tr>
<td>Test 3</td>
<td>7</td>
</tr>
<tr>
<td>Test 4</td>
<td>0</td>
</tr>
<tr>
<td>Test 5</td>
<td>37</td>
</tr>
<tr>
<td>Test 7</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>70</td>
</tr>
</tbody>
</table>

(t1) that covers each fault state with a probability of 1/50. The other five tests (t2-t6) provide higher coverage for certain modules, because these tests make more cycles per hour for these modules than normal production would make.

The solution is optimized towards test time, while testing stops when the total cost of testing is minimal (stop criterion 3). The optimal solution has a maximal test time of 70 hours, which is shown in Table 5.4. This means that 70 hours of testing is needed to show the 50 hours MTBF. During this test phase, none of these tests may fail. Otherwise, the MTBF requirement is not fulfilled.

In Figure 5.1, the risk in the system denoted in time units is shown for two situations. The first situation (Test 1 only) shows the risk decrease when only Test 1 is used. This situation is the current situation at ASML. Situation 2 (optimal sequence) shows the risk decrease of the optimal test phase, calculated with the test selection method. The optimal test stop moment for situation 1 is 111 hours of testing, and for situation 2 it is 70 hours of testing. From this figure we can conclude that it is profitable to use the critical module tests to test the system reliability instead of using only Test 1. Also, the remaining risk in the system is only 119 hours when using the critical module tests, while 160 hours when using only Test 1.
Today, reducing time-to-market or lead time for complex systems is extremely important. Test and integration time is almost always on the critical development path. Therefore, reducing this time will definitely reduce time-to-market or lead time. There are several approaches to reduce test time: testing can be made faster by automating test cases; testing can be made easier, for example, by changing the system; or testing can be done smarter, for example, by applying the method described in this paper and in [Boumen et al., 2006d,c]. The benefit of our method is avoidance of additional investments: we still use the same tests on the same system. This method incorporates the possibility to define different optimization criteria for different industries and systems. A test sequence can be optimized towards time or costs. Testing stops when the maximal time or cost is reached, or the system has reached a certain quality, denoted with risk, or when the total cost of testing is the lowest. Because the impact of a fault is different for each system, a different solution will be optimal. For example, the impact of a fault in an airplane is much higher and will therefore result in more testing than for a manufacturing machine. Test time in the testing of complex manufacturing systems can be reduced by using the methods presented. This is shown with two case studies in this paper that apply to the development and manufacturing phases of a lithographic machine. From these case studies we learn that this method can reduce up to 20% in test time, compared with test sequences made by hand by ASML engineers. The test model as presented here is a intuitive and easy method to write down a test sequencing problem. Furthermore, this model can be used for many other analyses, for instance, for test strategy simulation as described in [de Jong et al., 2006], to decide which test needs to be developed next, or for a
5.2. INTEGRATION PLANNING FOR LITHOGRAPHIC SYSTEMS

This section is based on the paper titled *Optimal integration and test planning applied to lithographic systems* [Boumen et al., 2007b] that is accepted and presented at the International Council of Systems Engineering (INCOSE) 2007 Symposium and has appeared in the proceedings of this Symposium. The paper section dealing with the method has been removed in this thesis because a more detailed and elaborate description of the method can be found in Chapter 3.

static analysis of the test problem.
Optimal integration and test planning applied to lithographic systems

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

Abstract

In the current integration and test phase of the development of a complex system, (the "right side" of the V-model) planning is becoming more and more difficult because of: 1) variability in delivery times of components, 2) failing tests and subsequent repairs, 3) resource changes and the use of component models, and 4) the growing system complexity and growing number of components and tests. Manually created integration and test plans are often not optimal regarding time-to-market. Furthermore, creating and maintaining these plans takes a lot of effort. In this paper, we introduce a method that allows to automatically create optimal integration and test plans. This method can be used by intelligent enterprises to shorten the time-to-market of a system and to reduce the effort needed to create and maintain integration and test plans. We illustrate this method with two cases studies related to the development of ASML lithographic machines [ASML, 2006].

5.2.1 Introduction

In today’s industry, time-to-market of systems is becoming more and more important. The integration and test phase of a complex system typically takes more than 45% of the total development time [Engel et al., 2004]. Reducing this time shortens the time-to-market of a new system.

In the integration and test phase of system development, components are concurrently developed and integrated into a subsystem. Subsequently, the subsystems are integrated into a system. In between these integration actions, tests are applied to check system requirements. An integration plan describes the sequence of integration actions and tests that need to be performed. For new ASML lithographic systems, integration and test plans are currently created manually.
An inefficient integration and test plan may cause faults to be found late in the integration and test process, because tests are performed late in the process. The rework caused by these faults increases the integration and test phase duration. Furthermore, not keeping a plan up to date causes an inefficient way of working that increases the duration of the complete phase. A good integration and test plan performs tests as early and in parallel as much as possible such that faults are found early in the process. Furthermore, when a plan is kept up to date, it is easier to make the correct decisions during the often chaotic integration and test phase. An optimal integration and test plan in general does not increase the system quality but increases the efficiency of working such that cost and/or time are minimized. Creating good or even optimal integration and test plans is becoming more and more difficult because of:

- The growing number of components in today’s systems. This results in numerous possible integration and test plans.

- The parallelism in the plan. Subsystems or modules should be tested in parallel as much as possible. Also, component models can be used to perform certain tests before actual components are delivered (see model-based integration by [Braspenning et al., 2006]).

Maintaining an integration and test plan is also becoming more and more difficult because of:

- The variability in delivery times of components. If a component arrives later than planned, the plan should be updated.

- The variability in test phase duration. Failing tests initiate a repair and diagnosis action and may increase the test phase duration.

- The varying number of components. During integration, it is possible that more components are needed than originally planned, such as software patches that were not included in the original system design.

Due to these difficulties, a method is needed that allows for automatic creation of integration and test plans that are optimal for the time-to-market of a system. This method should also minimize the effort needed to keep a plan up to date. In this paper, we introduce such a method. This method is called the integration and test planning method and consists of the following steps. First, a model is created of the integration and test problem that describes the problem mathematically. Second, an algorithm is used to automatically calculate the optimal integration and test plan. Finally, the plan is executed. A new plan can be calculated automatically after updating the model if a plan update is needed during the execution of the original plan. The paper is structured as follows. Section 5.2.2 describes the integration and test phases of lithographic machines. Section 5.2.3 describes the proposed integration and test planning method. Section 5.2.4 describes two case studies where this method is applied to the integration and test phases of lithographic machines. The last section gives conclusions.
5.2.2 Integration and test of lithographic systems

To explain the integration and test phases of a lithographic machine, we first describe shortly how such a system works. A lithographic machine (or wafer scanner) performs the lithographic step in the manufacturing of a semiconductor or IC. Two items are cycling through a scanner: a wafer that is the basis of the IC and contains a photo resistant coating, and a reticle that contains (a part of) the negative of the image that needs to be placed on the wafer using laser light. The system consists of 7 basic subsystems: a reticle handler that brings and takes reticles to the reticle stage which holds the reticle during the lithographic process, a wafer handler that brings and takes wafers to the wafer stage which holds the wafer during the lithographic process, a laser that produces the light needed for the lithographic process, an illuminator that uniformizes the light produced by the laser and a lens that shrinks and images the pattern from the reticle onto the wafer. Figure 5.2 shows a picture of an ASML lithographic machine with its main subsystems.

To reduce time-to-market of a new type of lithographic system, often multiple prototypes are created to perform tests in parallel. Normally, each of these prototypes is used for a specific goal; for example the first prototype is used to test all functional requirements, while the second prototype is used to test all performance requirements. At this moment, for each of these prototypes an integration and test plan is manually created by an integration engineer in Microsoft Project. The integration and test phase of these systems is characterized by a large time-to-market pressure and a huge number of multidisciplinary components (mechanical, electrical, optical, software) that are developed in parallel and should be integrated. During such an integration phase, first an old type lithographic system is manufactured. This system is then upgraded to the new type of system by replacing certain modules with
the new modules, upgrading the software and performing tests to check the system requirements. This approach reduces the risk on possible faults because a complete working machine is taken as starting point. Often, models or 'dummy' components are used during the integration phases to perform tests earlier in the integration phases, before the actual modules are delivered. During the execution of an integration and test plan, the plan often needs to be updated. If a module arrives later than planned, the duration of the module development is updated in the plan. Microsoft (MS) Project then automatically delays all tasks that depend on this development task. However, the sequence of tasks is not changed by MS project, which results in not optimal plans. Therefore, the sequence of tasks needs to be updated manually which increases the effort to update a plan.

5.2.3 Integration and test planning method

This paper section is intentionally removed from this chapter because a more detailed and elaborate description of the integration sequencing method has already been discussed in Section 3.1 and can be found in the original version of this paper.

5.2.4 Case studies

This section shows the results of two case studies that were performed during the integration and test phase of the development of two new ASML systems. The first case study shows the optimization of the integration and test plan of a new lithographic prototype and shows a plan update that was performed when the deliveries of certain modules were delayed. The second case study shows the optimization of the integration and test plan of two prototypes of a completely new type of system where some tests must be performed on one specific prototype and other tests may be performed on either the first or the second prototype.

Case study 1

This new lithographic machine is constructed based on an old type of system. Only the upgrade of certain modules is considered and not the integration of the old type of system. Therefore, the old system is modeled as one assembly (M1) that is completely present at the start of the project. There are 16 other modules (M2 through M17) that are integrated in the old system to upgrade this system to the new system. Modules M10, M11 and M12 are different laser system types. Some tests require one of these modules to be integrated before they can be performed while other tests require one specific laser to be integrated. The complete model of this system is shown in Figure 5.3 and Table 5.5. All modules are connected to the old system (M1), while the three lasers (M10, M11, M12) are also connected to M9.

The integration plan for this model is shown in Figure 5.4(a). The total duration of the plan is 1469 hours. The critical path is the path that module M2 follows and is shown in dark grey.
At a certain point in time during the execution of this plan the delivery times of some modules have been changed. In Table 5.6, the new development durations of these modules are shown. Furthermore, module M15 is removed in the new plan. After a simple update of the model, a new plan has been calculated automatically. This new plan, shown in Figure 5.4(b), shows the new critical path of module M14 in dark grey. The vertical dark grey line in the figure shows the time at which the plan is updated.

**Case study 2**

In this case study, two prototypes that have been developed in parallel are used to test some specific requirements of a new type of system. These two prototypes have been built from scratch, so no old system type is upgraded. Important detail of the problem is that 80% of the 66 tests are required to be performed on a specific system while 20% of the tests can be performed on either the first or the second prototype. Therefore, the two prototypes cannot be considered separately but have to be considered as one system. This means that both prototypes are defined in one model to create the optimal combined integration and test plan. Afterwards, the individual integration plans for both prototypes can be retrieved from...
### 5.2. Integration Planning for Lithographic Systems

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Develop M1</td>
<td>0 hrs</td>
</tr>
<tr>
<td>2</td>
<td>Test T0</td>
<td>96 hrs</td>
</tr>
<tr>
<td>3</td>
<td>Develop M10</td>
<td>0 hrs</td>
</tr>
<tr>
<td>4</td>
<td>Integrate I1-10</td>
<td>25 hrs</td>
</tr>
<tr>
<td>5</td>
<td>Develop M2</td>
<td>184 hrs</td>
</tr>
<tr>
<td>6</td>
<td>Integrate I1-2</td>
<td>27 hrs</td>
</tr>
<tr>
<td>7</td>
<td>Test T1</td>
<td>165 hrs</td>
</tr>
<tr>
<td>8</td>
<td>Develop M3</td>
<td>208 hrs</td>
</tr>
<tr>
<td>9</td>
<td>Integrate I1-3</td>
<td>16 hrs</td>
</tr>
<tr>
<td>10</td>
<td>Replan point</td>
<td>0 hrs</td>
</tr>
<tr>
<td>11</td>
<td>Test T2, T4</td>
<td>352 hrs</td>
</tr>
<tr>
<td>12</td>
<td>Develop M13</td>
<td>520 hrs</td>
</tr>
<tr>
<td>13</td>
<td>Integrate I1-13</td>
<td>2 hrs</td>
</tr>
<tr>
<td>14</td>
<td>Test T6</td>
<td>10 hrs</td>
</tr>
<tr>
<td>15</td>
<td>Develop M5</td>
<td>456 hrs</td>
</tr>
<tr>
<td>16</td>
<td>Integrate I1-5</td>
<td>4 hrs</td>
</tr>
<tr>
<td>17</td>
<td>Develop M17</td>
<td>632 hrs</td>
</tr>
<tr>
<td>18</td>
<td>Integrate I1-17</td>
<td>2 hrs</td>
</tr>
<tr>
<td>19</td>
<td>Test T10</td>
<td>6.5 hrs</td>
</tr>
<tr>
<td>20</td>
<td>Develop M8</td>
<td>1048 hrs</td>
</tr>
<tr>
<td>21</td>
<td>Integrate I1-8</td>
<td>2 hrs</td>
</tr>
<tr>
<td>22</td>
<td>Develop M7</td>
<td>1288 hrs</td>
</tr>
<tr>
<td>23</td>
<td>Integrate I1-7</td>
<td>0 hrs</td>
</tr>
<tr>
<td>24</td>
<td>Test T16</td>
<td>120 hrs</td>
</tr>
<tr>
<td>25</td>
<td>Develop M14</td>
<td>1416 hrs</td>
</tr>
<tr>
<td>26</td>
<td>Develop M6</td>
<td>1048 hrs</td>
</tr>
<tr>
<td>27</td>
<td>Integrate I1-6</td>
<td>0 hrs</td>
</tr>
<tr>
<td>28</td>
<td>Integrate I1-14</td>
<td>2 hrs</td>
</tr>
<tr>
<td>29</td>
<td>Test T7</td>
<td>10 hrs</td>
</tr>
<tr>
<td>30</td>
<td>Develop M4</td>
<td>344 hrs</td>
</tr>
<tr>
<td>31</td>
<td>Integrate I1-4</td>
<td>2 hrs</td>
</tr>
<tr>
<td>32</td>
<td>Test T3</td>
<td>5 hrs</td>
</tr>
<tr>
<td>33</td>
<td>Develop M16</td>
<td>608 hrs</td>
</tr>
<tr>
<td>34</td>
<td>Integrate I1-16</td>
<td>7 hrs</td>
</tr>
<tr>
<td>35</td>
<td>Test T11</td>
<td>12 hrs</td>
</tr>
<tr>
<td>36</td>
<td>Develop M11</td>
<td>1216 hrs</td>
</tr>
<tr>
<td>37</td>
<td>Develop M9</td>
<td>672 hrs</td>
</tr>
<tr>
<td>38</td>
<td>Develop M12</td>
<td>664 hrs</td>
</tr>
<tr>
<td>39</td>
<td>Integrate I9-11</td>
<td>0 hrs</td>
</tr>
<tr>
<td>40</td>
<td>Integrate I9-12</td>
<td>0 hrs</td>
</tr>
<tr>
<td>41</td>
<td>Integrate I1-9, I1-12, I1-11, I9-12</td>
<td>39 hrs</td>
</tr>
<tr>
<td>42</td>
<td>Test T5, T9, T12, T13, T14, T15</td>
<td>515 hrs</td>
</tr>
<tr>
<td>43</td>
<td>Finished</td>
<td>0 hrs</td>
</tr>
</tbody>
</table>

(a) Case study 1 solution represented as an MS project Gantt chart

(b) Case study 1 replan solution represented as an MS project Gantt chart

Figure 5.4: Case study 1 solutions
Table 5.6: Changed development times for Case study 1

<table>
<thead>
<tr>
<th>M</th>
<th>Old development duration</th>
<th>New development duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>M7</td>
<td>904</td>
<td>1288</td>
</tr>
<tr>
<td>M8</td>
<td>688</td>
<td>1048</td>
</tr>
<tr>
<td>M11</td>
<td>688</td>
<td>1216</td>
</tr>
<tr>
<td>M12</td>
<td>888</td>
<td>664</td>
</tr>
<tr>
<td>M14</td>
<td>536</td>
<td>1416</td>
</tr>
<tr>
<td>M15</td>
<td>552</td>
<td>Removed</td>
</tr>
</tbody>
</table>

Table 5.7: Case study 2 model properties

<table>
<thead>
<tr>
<th>Element</th>
<th>Number</th>
<th>Min and max times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules</td>
<td>94</td>
<td>0 to 880 hour</td>
</tr>
<tr>
<td>Interfaces</td>
<td>113</td>
<td>0 to 40 hour</td>
</tr>
<tr>
<td>Tests</td>
<td>66</td>
<td>4 to 80 hour</td>
</tr>
</tbody>
</table>

the combined plan. The properties of the combined model are shown in Table 5.7.

The solution to this problem is shown in Figure 5.5. In this combined plan, the two individual sequences for each prototype can be distinguished. Both prototypes are on the critical path which is denoted in dark grey. The total duration of this plan is 1346 hours. The plan that was created manually by an engineer without using this method takes 1536 hours to perform. This is mainly due to the fact that tests are scheduled less efficiently over the two prototypes compared to the optimal plan. The optimal plan is therefore more than 10% shorter than the plan created manually. Note that we compare two initial plans with each other and not the actual executed plans. These initial plans do not contain the disturbances that may occur during the integration and test phase (although new plans can be created automatically as shown in the previous case study).

5.2.5 Conclusions

In this paper, we have introduced a method that allows to create optimal integration and test plans for the integration and test phase during the development of a complex system. This method consists of: 1) defining a model of the problem, 2) creating a plan and 3) executing the plan. Two case studies in the development of new ASML lithographic systems showed the benefits of the method, which are:

- The integration and test plans created with the proposed method are optimal and may therefore be shorter than manually created plans. The second case study shows that the optimal plan is more than 10% shorter than a manually generated plan.
- Planning and re-planning effort can be reduced. The first case study shows that it is
Figure 5.5: Case study 2 solution represented in an MS project Gantt chart
easy to re-plan when certain modules arrive later than planned. The only step that has to be performed is updating the model with the new times. The plan can then be updated automatically. Unfortunately, we cannot give any hard numbers on the real effort reduction because the method has not been used on a large scale yet.

Another benefit of this method is the model. The model can be used as a knowledge container and denotes how the integration and test problem is defined in a simple and precise way. This makes it easy to review the model with peer engineers. The planning method does not influence the quality of the system because the selection of tests is not taken into account. This is different in [Boumen et al., 2007c] where we used the method presented in combination with a test selection method to determine the optimal integration and test plan.

In this paper, we have shown that the integration and test planning method can be used to optimize an integration and test plan for the development of a new system. However, this method is used to solve other problems, such as the optimization of integration and test plans for (evolutionary) software releases (see [Boumen et al., 2007c]) and the optimization of integration and test plans for the manufacturing of multiple systems. In [Boumen et al., 2007c], the method has been used to determine which component models should be developed to perform tests earlier when the realizations of components are not yet ready. The method presented can also be used to optimize integration and test plans of complex systems other than lithographic systems. In the case studies we did not use specific properties of lithographic systems. Although we did not perform real studies with other types of systems, the method may also be suitable for systems that have integration and test phases where large numbers of components developed in parallel should be integrated and where time-to-market is crucial.

5.3 INTEGRATION AND TEST PLANNING FOR SOFTWARE RELEASES

This section is based on the paper titled Optimal integration and test plans for software releases of lithographic systems [Boumen et al., 2007c] that is accepted for the 5th Annual Conference on Systems Engineering Research (CSER) 2007 and has appeared in the proceedings of this Conference. The paper section dealing with the method has been removed in this thesis because a more detailed and elaborate description of the method can be found in Chapter 4.
Optimal integration and test plans for software releases of lithographic systems

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

Abstract

This paper describes a method to determine the optimal integration and test plan for embedded systems software releases. The method consists of four steps: 1) describe the integration and test problem in an integration and test model which is introduced in this paper, 2) determine possible test positioning strategies, 3) calculate integration and test plans for each test positioning strategy and 4) analyze the resulting plans on time-to-market and total test time and choose the optimal one. A test positioning strategy determines when a test phase starts and stops. We introduce several possible test phase strategies that can be used in the method. Using this method, for two specific ASML lithographic machine software releases [ASML, 2006], we determined the optimal integration and test plans and, hence, the best test positioning strategy for ASML software testing.

5.3.1 Introduction

Integration and testing of embedded systems software, like the software of an ASML lithographic system, is costly and time consuming. Therefore, testing is performed as early as possible and parallel to the development process. This is done by performing several test phases during the development process. At the end of the development process, a system test phase is performed. After this test phase, the software is released and can be installed on the embedded systems. To reduce time-to-market of a software release, it is important to choose the best integration and test plan. Such a plan describes the integration sequence of the different modules developed in parallel, the tests that should be performed in the test phases, and when test phases should be performed. The integration and test plan depends on the test positioning strategy that determines when a test phase starts and stops. For example, during the development of an ASML software release an integration test phase is...
performed every week and takes approximately one day. The question is whether this test positioning strategy is the best strategy aimed at releasing software as soon as possible. Many other strategies are possible.

In this paper, we introduce a method that can be used to calculate the optimal integration and test plan for a software integration and test problem based on a test positioning strategy. With this method it is possible to investigate which of the test phase strategies introduced results in an optimal integration and test plan. Optimal integration and test plans minimize time-to-market for a specific ASML software release while not increasing the total test effort compared to the current way of working. The method uses the integration and test sequencing method described in [Boumen et al., 2006d,a] which has been successfully applied to optimize integration and test plans of lithographic systems prototypes [Boumen et al., 2006b, 2007b].

The paper is structured as follows. Section 5.3.2 describes the general software development approach that is used in ASML. Section 5.3.3 introduces the integration and test planning method and the possible test phase strategies. Section 5.3.4 describes the optimization of ASML software release integration and test plans using the integration and test planning method. The final section gives conclusions.

5.3.2 Software integration and test

Before explaining the integration and test planning method, we first discuss the development approach of software in ASML. ASML uses an incremental concurrent software development approach. Incremental means that the development starts with an old software release that is updated to a new software release. Up to a few hundred software engineers make copies of a (part of) the software and concurrently update each part according to the new requirements. If the development of such an update is finished, it is integrated back in the release. Since this is an ongoing process, the release will slowly migrate to a complete new release. During this migration process the current release (with all updates) is called the baseline (BL). When all functionality is updated, the new release is ready and can be released to the customer. This development approach is illustrated in Figure 5.6, where each action is shown as a rectangle and the precedence relations are denoted by directed edges.

Software integration

The integration of software is the assembly of a software update with the BL. In this update, one or more components may be changed. During the development of this update, the software may be changed such that certain system requirements are not met anymore. To check whether all requirements are met, tests need to be performed on the BL.

Software test

In the ASML software development approach three test phases can be distinguished: 1) an update test phase, 2) an integration test phase, and 3) a system test phase. The update test
phase is always performed on the software update by the software developers. This test phase is considered part of the development of the update and not discussed further in this paper. The integration test phase is performed on the BL. In the current way of working, test engineers perform this test phase every week. The duration of this test phase is one day. The system test is performed when all functionality is available and all updates are integrated. The duration of this test phase depends on the desired final quality of the software release. The choice of when test phases should start and stop is called the test positioning strategy. The current test positioning strategy of one 8 hour integration test per week and one system test after every update is integrated is a choice that might not result in the shortest time-to-market. If for example, many updates are integrated, longer integration test phases may be needed, while if nothing is integrated, no integration test phase is needed. Using the method described in the following section, the test positioning strategy can be determined that results in the optimal integration and test plan.

Illustration

To illustrate the software integration and test problem we introduce a small example that is also used in the following sections. Suppose we have a software baseline version 1.0 and by developing 4 updates in parallel, this baseline is updated to version 1.1. The software system has 8 requirements that can be tested by performing 8 tests that cover a certain set of these requirements. Each of the 4 updates that are made might cause that these requirements are not met. The goal is to determine the optimal integration and test plan. An integration and test plan consists of the integration sequence, the sequence of the integration test phases and the system test phase that should to be performed. In addition, it contains the tests that need to be performed in each of these test phases.

5.3.3 Integration and test planning method

This paper section is intentionally removed from this chapter because a more detailed and elaborate description of the integration and test sequencing method has already been discussed in Chapter 4 or can be found in the original version of this paper.
5.3.4 Case studies

In the previous section, we introduced a method to determine the optimal integration and test plan for an integration and test problem. We also showed with a simple illustration how this method can be used for a software release problem. In this section, we determine an optimal integration and test plan for two ASML software releases. This is done in two case studies described in the following subsections. The assumptions that were made during these case studies are:

- the modules are independent of each other,
- the quality of the BL during development is not important,
- only the test times are taken into account.

Case study 1

This case study deals with a large software release. In Table 5.8, the properties of the model are shown. The software updates and their corresponding delivery times are known in advance. For each of these software updates, a test engineer estimates the probability that the update introduces a fault state (requirement failure). Also, test engineers have estimated the coverage of each test on each requirement for the complete available test set. The test durations are also estimated. After the model is created, several test positioning strategies are considered, including the standard ASML test positioning strategy that starts an integration test phase every week for one day. For each of these strategies, an optimal integration and test plan is calculated. The optimal strategy should reduce the time-to-market as much as possible, while not increasing the total test time.

In Table 5.9, the results of different test phase strategies are shown, that is the time-to-market and total test time of the resulting solutions. In Figure 5.7, the risk profiles of the most interesting solutions are shown. We can conclude from this case study that the test positioning strategy with start criterion RB(2,2) results in the least time-to-market (almost 53% reduction in the system test duration). However, this strategy also results in a large total test time. Strategy RB(25,2) (a risk-based line that starts at 25 and ends at 2) also reduces time-to-market and reduces the total test time a little compared to the current ASML situation.

<table>
<thead>
<tr>
<th>Property</th>
<th>Case study 1</th>
<th>Case study 2</th>
</tr>
</thead>
<tbody>
<tr>
<td># Modules</td>
<td>260</td>
<td>122</td>
</tr>
<tr>
<td># Interfaces</td>
<td>259</td>
<td>121</td>
</tr>
<tr>
<td># Fault states</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td># Tests</td>
<td>169</td>
<td>169</td>
</tr>
</tbody>
</table>
### Table 5.9: Results case study 1

<table>
<thead>
<tr>
<th>Test positioning strategy</th>
<th>Start criterion</th>
<th>Stop criterion</th>
<th>Time-to-market (hour)</th>
<th>System test duration (hour)</th>
<th>Δ system test time</th>
<th>Total test time (hour)</th>
<th>Δ total test time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE(1 week) DU(8 hours)</td>
<td></td>
<td></td>
<td>2617</td>
<td>97</td>
<td>-</td>
<td>305</td>
<td>-</td>
</tr>
<tr>
<td>FA</td>
<td>DU(8 hours)</td>
<td></td>
<td>2618</td>
<td>98</td>
<td>1%</td>
<td>328</td>
<td>7%</td>
</tr>
<tr>
<td>FO</td>
<td>RL(0.1,0.1)</td>
<td></td>
<td>2626</td>
<td>106</td>
<td>9%</td>
<td>129</td>
<td>-57%</td>
</tr>
<tr>
<td>RB(2,2) RL(0.1,0.1)</td>
<td></td>
<td></td>
<td>2566</td>
<td>46</td>
<td>-53%</td>
<td>1336</td>
<td>336%</td>
</tr>
<tr>
<td>RB(25,2) RL(20,0.2)</td>
<td></td>
<td></td>
<td>2571</td>
<td>51</td>
<td>-47%</td>
<td>284</td>
<td>-6%</td>
</tr>
</tbody>
</table>

Figure 5.7: Case study 1 risk results
Table 5.10: Results case study 2

<table>
<thead>
<tr>
<th>Test positioning strategy</th>
<th>Time-to-market (hour)</th>
<th>System test duration (hour)</th>
<th>Δ system test time</th>
<th>Total test time (hour)</th>
<th>Δ total test time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE(1 week) DU(8 hours)</td>
<td>704</td>
<td>80</td>
<td>-</td>
<td>148</td>
<td>-</td>
</tr>
<tr>
<td>FA</td>
<td>700</td>
<td>76</td>
<td>-5%</td>
<td>173</td>
<td>17%</td>
</tr>
<tr>
<td>FO</td>
<td>697</td>
<td>73</td>
<td>-9%</td>
<td>106</td>
<td>-28%</td>
</tr>
<tr>
<td>RB(1, 1) RL(0.2, 0.2)</td>
<td>669</td>
<td>45</td>
<td>-44%</td>
<td>550</td>
<td>272%</td>
</tr>
<tr>
<td>RB(20, 2) RL(15, 0.2)</td>
<td>684</td>
<td>60</td>
<td>-25%</td>
<td>135</td>
<td>-9%</td>
</tr>
<tr>
<td>RB(25, 2) RL(22 - 0.2)</td>
<td>674</td>
<td>50</td>
<td>-38%</td>
<td>153</td>
<td>3%</td>
</tr>
</tbody>
</table>

Case study 2

The second case study is the optimization of the integration and test plan of a smaller software release. The properties of this model are also shown in Table 5.8. For different test phase strategies, the optimal integration and test plan is calculated. In Table 5.10, the time-to-market and total test time of the different solutions are shown. The test positioning strategy with start criterion RB(1,1) has the best time-to-market, but this test positioning strategy also increases the total test time drastically. The strategy with start criterion RB(20,2) both reduces the time-to-market and total test time, but the RB(25,2) strategy reduces the time-to-market even more for a little increase in the total test time. Depending on the specific importance of total test time, either of them can be chosen.

5.3.5 Conclusions

In this paper, we introduced a method to determine an optimal test positioning strategy for a software release integration and test problem. The method consists of creating a model of the problem, determining a set of possible test phase strategies, and calculating for each strategy an optimal integration and test plan. A test positioning strategy determines when a test phase should start and stop. The different integration and test plans can be compared on total test time and time-to-market which is the duration of the critical path in the integration and test plan.

We used this method to determine an optimal integration and test plan for ASML software releases that are used to control lithographic systems. This is done for two different software releases. Both case studies showed that the current test positioning strategy used by ASML is theoretically not optimal. This test positioning strategy determines that every week a test phase is started with a fixed duration of one day. The optimal integration and test plan was obtained by using a risk-based test positioning strategy that starts testing when the risk reaches a certain risk level. This risk threshold may decrease in time to reduce the total test
5.4 TEST PLANS FOR SOFTWARE RELEASES

This section contains a case study that has been described in the paper titled *Risk-based stopping criteria for test sequencing* [Boumen et al., 2006c] and is submitted to IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans in 2007. Section 2.2 is based on the same paper but does not include the case study presented in this section. This work has been done as part of the Tangram project.

*Software weekly validation test phase*

This case study deals with the optimization of a test phase in the development of a software release that is used to control a lithographic machine. The total size of the complete release is over 15 millions lines of code. In a period of approximately 6 months, up to 300 software engineers change certain parts of the code to develop the new software version with new requirements. Every week, changed parts of the code are integrated in the so-called baseline software release. When all changes are integrated and tested, this baseline can be released to the customers. Every week, the so-called weekly validation test is performed on the current baseline. The duration of this test phase is one day. Previously, this test phase consisted of performing approximately the same test set.

In this case study, we use the presented test sequencing method to create an optimal test sequence on a weekly basis. The goal of this case study was to find more issues during this weekly test phase without increasing test time, instead of finding all issues during the final test phase before the software is released to the customer.

The test model was constructed as follows by ASML test engineers. The set of tests consists of 442 test cases of the approximate total of 1000 test cases available. These 442 test cases were considered most important. The average duration of one test case is approximately 45 minutes. The normal weekly validation test consists of 9 tests out of these 442 test cases. The set of fault states in the model are the 209 known (sub)-requirements of the software system. The importance or impact of these fault states is estimated by the test engineers based on their expert knowledge and is a number from the set: \(0.1, 0.3, 0.5, 0.9\). This number does not represent the actual repair cost but represents the relative importance of fault states. The test engineers estimate the coverage of each test on each requirement based on
available test documentation and previously performed test cases. The coverage number is chosen from the set: \((0.1, 0.3, 0.5, 0.9, 1.0)\). The parameters of the model do not change (frequently) during the total development process of a software release. This is different for the probability that a requirement is not met. This probability changes every week, depending on the software changes integrated in the software release. Therefore, the probability that a requirement is not met is estimated for every software change. The total fault probability \(p_i\) for a requirement in week \(i\), given a set of \(n\) probabilities for every software change in that week \(P_i = \{p_1^i, \ldots, p_n^i\}\) and given the probability of that requirement after the test phase of the previous week \(p_{i-1}\), then is:

\[
p_i = 1 - \left( (1 - p_{i-1}) \cdot \prod_{p' \in P_i} (1 - p') \right) \tag{5.1}
\]

The probability for every software change is a number from the set \((0.01, 0.1, 0.3, 0.5, 0.9)\) and is based on the following criteria:

- The probability is based on the number of changed components of a software change divided by the total number of components that make sure that this requirement is met. For example, if during a certain software change 2 out of 5 of these components are changed, the fault probability is approximately \(\frac{2}{5}\).

- The severity of the changes is taken into account. By performing an FMEA analysis prior to the development process, the risk areas are identified. These risk areas obtain higher probabilities.

- Requirements that are often not met are identified and get a higher fault probability.

This way of working is of course not perfect and may be improved by using other techniques such as estimations on the number of changed lines of code [Brown et al., 1989] or the number of software executions. These approaches have not been investigated in this paper.

Now that we are able to define a model, we can calculate the optimal test sequence for a weekly validation test phase. The objective of such a test sequencing problem is to exclude as much risk as possible in a time frame of maximal 8 hours. Because diagnosing and fixing faults is not part of the test phase and is done at a different time, we are only interested in the pass trace and therefore use the PT-heuristic. The objective is then to minimize \(J^m\) with a constraint of \(J^m_C \leq 8\) hours. As an illustration, we calculated the optimal test sequence for a randomly chosen week.

The resulting test sequence consists of 16 tests that reduce almost 35% of the initial risk in 8 hours. The old test sequence that was normally used consisted of 9 tests that reduced 22% of the initial risk cost in 8 hours. In Figure 5.8(a), the excluded risk in time is shown for both test sequences. As can be seen in the graph, the optimal test sequence performs better than the manually created sequence.

The comparison between the optimal and manual test sequence in the previous experiment shows that in theory the optimal test sequence performs better because more risk is
5.5. CONCLUSIONS

This chapter has answered research questions 1.4, 2.4 and 3.4 as defined in the first chapter:

**Answer to Question 1.4:** The benefits for using the test plan optimization method are:

- A test sequence can be optimized towards an objective of test time, cost and/or remaining risk (quality). For several case studies we showed that test time can be reduced by approximately 20% with the same final system quality (remaining risk).
- The optimal stop moment can be determined based on several stopping criterion.
- The test model can be used as a knowledge container, as a means of communication and as training method.

Figure 5.8: Comparison between the manual and optimal way of working

Excluded risk included in the same period of time. However, this conclusion of course depends on the estimated model parameters, especially the probability and impact estimations. In Figure 5.8(b), we show a comparison, that is model parameter independent, between the old way of working in the year 2005 and the new way of working using the test sequencing method in 2006. The graphs in this figure show the cumulative number of issues found in the weekly validation tests for a certain number of weeks in a particular year. With the new way of working, 95 issues were found in 26 weeks, while with the old way of working only 52 issues were found in 26 weeks. If we assume that the same number of issues were created by developers in both periods, we can conclude that the new way of working gives better results since more issues were discovered. This assumption does not hold if during one period more issues were inserted than during the other period. However, at this moment no more data is available to improve this comparison.

5.5 CONCLUSIONS
**ANSWER TO QUESTION 2.4:** The benefits for using the integration plan optimization method are:

- An integration sequence can be optimized towards an objective of integration time and/or interest cost. For a case study we showed that integration time can be reduced by approximately 10%.
- Using this integration planning method reduces the effort to create and update the integration plan. Updating a plan because of a delayed delivery of a module can now be done automatically instead of manually.
- The integration model can be used as a knowledge container and as a means of communication.

**ANSWER TO QUESTION 3.4:** The benefits for using the integration and test plan optimization method are:

- An integration and test plan can be optimized towards an objective of time, cost and/or remaining risk. For a case study, we showed that the duration of the last test phase can be reduced by approximately 40% by changing the test positioning strategy.
- With the integration and test planning method one can determine the best test positioning strategy for a given integration and test phase.
- The integration and test model can be used as a knowledge container and as a means of communication.

Furthermore, we have shown by performing different case studies in different domains that the methods introduced can be used in many application areas: the development of a new system, the manufacturing of several instances of a complex system, hardware and software integration phases. The results of these case studies convinced ASML that the methods actually solve some of their integration and test problems. Therefore at this moment, the prototype tool that contains implementations of the proposed methods is being used by ASML engineers on a daily basis.
In this chapter, conclusions are drawn about the methods presented and suggestions for future research are given. Section 6.1 discusses the conclusions that can be drawn about the objectives that are identified in Chapter 1. Section 6.2 discusses the benefits of the methods introduced for companies developing complex embedded systems, ASML in particular. Section 6.3 discusses suggestions for future research and suggestions for the further development of the methods presented.

6.1 OBJECTIVES

In the first chapter, we identified three main research problems that are solved, namely:

- Develop a method that constructs a test plan.
- Develop a method that constructs an integration plan.
- Develop a method that constructs an integration and test plan.

We conclude that a method has been developed that creates an optimal test plan. This method is based on sequential diagnosis methods from the literature and is adjusted to solve test planning problems for test phases of complex systems. We extended the method with risk-based stopping criteria and introduced an approach to solve hierarchical test planning problems. Furthermore, practical extensions such as computational reduction measures are introduced to solve real-life industrial problems.

We conclude that a method has been developed that creates an optimal integration plan. This method is based on assembly sequencing methods from the literature and is adjusted to solve integration planning problems for system integration phases. We introduced an approach to solve hierarchical problems. Furthermore, practical extensions such as computational reduction measures have been introduced to solve real-life industrial problems.

Finally, we conclude that a method has been developed that creates integration and test plans. This method is a combination of the integration planning and the test planning method. We introduced several strategies that may be used to combine integration and test plans. We also introduced an approach to solve hierarchical problems. Furthermore, practical extensions such as computational reduction measures have been introduced to solve real-life industrial problems.
6.2 BENEFITS TO INDUSTRY

In Chapter 1, we described several industrial problems that are related to integration and test strategies. With the integration and test planning methods developed we can solve many of these problems. An elaborate overview of the exploitation of the methods presented is given in the Exploitation of Results in the beginning of this thesis.

We conclude that the methods presented have the following benefits:

- Integration and test time and cost can be reduced while maintaining the system quality by creating optimal integration and test plans. With a case study for several manufacturing test phases we showed that the test time can be reduced by approximately 20% when using the test planning method. Furthermore, we showed with a case study for the integration phase of a new type of complex system that the integration duration can be reduced by almost 10% when using the integration planning method. Also, we showed that a different test positioning strategy for the integration and test phase of a new software release reduces the final test phase by approximately 40%.

- Less manual effort needs to be spent on creating integration and test plans and maintaining these plans because the methods are automated. We showed with a case study that it is possible to automatically re-plan an integration plan when certain components arrived later than planned. We also showed that instead of performing the same test sequence for every system that is manufactured, it is better to create individual test sequences for every manufactured system, since the quality of these systems may vary. Creating these individual plans can now be automated instead of creating them manually.

- The models developed provide one information source that may be used for training new people, communication and for creating ‘what-if’ scenarios. The models used in the methods have been used as training material for new ASML production engineers. Furthermore, the models provide the integration and test problem in one overview, which has been demonstrated in many presentations and discussions at ASML.

6.3 FURTHER RESEARCH AND DEVELOPMENT

Suggestions for further work are split into two parts: the suggestions for further research and the suggestions for further development. The suggestions for further research deal with open research questions. The suggestions for further development deal with extending the methods introduced such that they can be used by industry in a larger scale.

For further research we recommend to:

- Investigate in more depth whether these methods can be used to make the decision of which component models should be created for the model-based integration approach as suggested by Brasperning et al. in [Brasperning et al., 2007]. In this thesis, we gave
a short introduction to this subject in Section 5.3, but more research and a case study are needed on this topic.

- Develop an optimal integration and test planning method that is able to calculate real-life large problems. In Chapter 4, several test phase positioning strategies are suggested to create integration and test plans. We explained that it is possible to calculate the optimal test phase positioning strategy. However, the approach suggested is computationally very complex and no real-life problems can be solved at this moment.

- Investigate how the methods presented can be used to design manufacturing test phases using the models of the development test phases. By extending the methods presented it may be possible to investigate which requirements or performance criteria should be tested during development and which ones during manufacturing.

- Investigate the real test time reduction and effort reduction obtained by using the methods presented. In this thesis, we mostly showed theoretical results. To show more quantitative results, experiments can be conducted to measure the real reductions.

- Apply the methods presented to other areas than embedded systems that deal with integration and testing, such as drug testing. It is possible that the method must be extended to cope with other application areas.

For further development on the methods presented we recommend to:

- Extend the method with more approaches to collect data and information in companies to create the models suggested. In the case studies, we described some approaches to create these models. However, for every company or industry these approaches may be different and should be tailored.

- Integrate the methods with existing test and integration applications such as version control systems and automated test environments. This makes it possible to fully automate the process of, for example, software integration and testing.

- Develop a complete process in which the methods suggested are incorporated. Such a process should describe the way of working, and determine when and how to create and use the models and methods suggested during the development of a new system.

- Extend the prototype tooling developed with easy to use modeling interfaces and fast implementations of the algorithms suggested. This makes the introduction of the methods in companies easier and therefore faster.


Roel Boumen was born on the 18th of March, 1980 in Maasbree, The Netherlands. In 1998 he finished the VWO at the Blariacum College in Venlo. From 1998-2004 he studied Mechanical engineering at the Eindhoven University of Technology. In 2004, he received his M.Sc. degree in Mechanical Engineering from the Systems Engineering Group of the Eindhoven University of Technology, the Netherlands. During his work as a master student he worked in the field of supervisory machine control of lithographic machines in cooperation with ASML, a provider of lithographic systems for the semiconductor industry. Since 2004 he is a Ph.D. student at the Eindhoven University of Technology. His research concerns test strategy in the TANGRAM project. The TANGRAM project is a Dutch research project under the responsibility of the Embedded Systems Institute and in cooperation with ASML and several other academic and industrial partners.
M.C. van Wezel. Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. Formal Specification and Analysis of Industrial Systems. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. Techniques for Understanding Legacy Software Systems. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03


N. van Vugt. Models of Molecular Computing. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. On-line Scheduling and Bin Packing. Faculty of Mathematics and Natural Sciences, UL. 2002-09


M.B. van der Zwaag. Models and Logics for Process Algebra. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. Probabilistic Extensions of Semantical Models. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12


J.I. van Hemert. Applying Evolutionary Computation to Constraint Satisfaction and Data Mining. Faculty of Mathematics and Natural Sciences, UL. 2002-14


Y.S. Usenko. Linearization in $\mu$CRL. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. Random Redundant Storage for Video on Demand. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. To Reuse or To Be Reused: Techniques for component composition and
construction. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02


S.M. Bohte. Spiking Neural Networks. Faculty of Mathematics and Natural Sciences, UL. 2003-04


S.V. Nedea. Analysis and Simulations of Catalytic Reactions. Faculty of Mathematics and Computer Science, TU/e. 2003-06


H.P. Benz. Casual Multimedia Process Annotation – CoMPAs. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08


M.H. ter Beek. Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. The λ Abroad – A Functional Approach to Software Components. Faculty of Mathematics and Computer Science, UU. 2003-11


G.I. Jojgov. Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. Theory of Molecular Computing – Splicing and Membrane systems. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. Models of Tree Translation. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. Data Synchronization and Browsing for Home Environments. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05


L. Cruz-Filipe. Constructive Real Analysis: a Type-Theoretical Formalization and Applications. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications. Faculty of Technology Management, TU/e. 2004-08

N. Goga. Control and Selection Techniques for the Automated Testing of Reactive Systems. Faculty of Mathematics and Computer Science, TU/e. 2004-09


J. Pang. Formal Verification of Distributed Systems. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. Evolutionary Agent-Based Economics. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. Indoor Ultrasonic Position Estimation Using a Single Base Station. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. On Distributed Verification and Verified Distribution. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17


P.J.L. Cuijpers. Hybrid Process Algebra. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. Supervisory Machine Control by Predictive-Reactive Scheduling. Faculty of Mechanical Engineering, TU/e. 2004-21


R. Ruimerman. Modeling and Remodeling in Bone Tissue. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. Experiments in Rights Control - Expression and Enforcement. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. Design and Verification of Lock-free Parallel Algorithms. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. Specification and Analysis of Internet Applications. Faculty of Mathematics and Computer Science, TU/e. 2005-05


I. Kurtev. Adaptability of Model Transformations. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. Computational Aspects of Treewidth - Lower Bounds and Network Reliability. Faculty of Science, UU. 2005-09
O. Tveretina. Decision Procedures for Equality Logic with Uninterpreted Functions. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. Evolution of Finite Populations in Dynamic Environments. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. Data Mining using Genetic Programming: Classification and Symbolic Regression. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. Top Quality Type Error Messages. Faculty of Science, UU. 2005-13

G.F. Frehse. Compositional Verification of Hybrid Systems using Simulation Relations. Faculty of Science, Mathematics and Computer Science, RU. 2005-14


T. Gelsema. Effective Models for the Structure of pi-Calculus Processes with Replication. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. Composing Constraint Solvers. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18


M. Valero Espada. Modal Abstraction and Replication of Processes with Data. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. Stepping through Haskell. Faculty of Science, UU. 2005-21


E. Dolstra. The Purely Functional Software Deployment Model. Faculty of Science, UU. 2006-01


P.R.A. Verbaan. The Computational Complexity of Evolving Systems. Faculty of Science, UU. 2006-03


M. Kyas. Verifying OCL Specifications of UML Models: Tool Support and Compositionality. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. Model Checking Timed Automata - Techniques and Applications. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. Böhm-Like Trees for Rewriting. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. On JML: topics in tool-assisted verification of JML programs. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. Towards Hybrid Molecular Simulations. Faculty of Biomedical Engineering, TU/e. 2006-09
S.G.R. Nijssen. Mining Structured Data. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. Separation and Adaptation of Concerns in a Shared Data Space. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. Reconciling Nondeterministic and Probabilistic Choices. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. Verification techniques for Extensions of Equality Logic. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. Constructive formal methods and protocol standardization. Faculty of Mathematics and Computer Science, TU/e. 2006-14


M.E. Warnier. Language Based Security for Java and JML. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. At Home In Service Discovery. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. Expressivity of Timed Automata Models. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. Formalising Interface Specifications. Faculty of Mathematics and Computer Science, TU/e. 2006-19


J.V. Guillen Scholten. Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. Flexible Heterogeneous Software Systems. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. A run-time reconfigurable Network-on-Chip for streaming DSP applications. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems. Faculty of Mathematics and Computing Sciences, RUG. 2007-03


M.W.A. Streppel. Multifunctional Geometric Data Structures. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. Silent Steps in Transition Systems and Markov Chains. Faculty of Mathematics and Computer Science, TU/e. 2007-08
R. Brinkman. *Searching in encrypted data.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use.* Faculty of Science, Mathematics and Computer Science, RU. 2007-10


R. Boumen. *Integration and Test plans for Complex Manufacturing Systems.* Faculty of Mechanical Engineering, TU/e. 2007-12