Integration and test strategies for complex manufacturing machines

Integration and test strategies for complex manufacturing machines

Ivo de Jong

Cover: This thesis is the result of ten years experience in integration and testing at ASML and four years of research at the Eindhoven University of Technology and ASML. The cover represents the cooperation that is required to finish a thesis, like this thesis, as well as the cooperation that is required to integrate and test an ASML wafer scanner.

Voorkant: Dit proefschrift is het resultaat van tien jaar integratie en test ervaring opgedaan bij ASML en vier jaar onderzoek naar integratie en test strategieën, uitgevoerd aan de Technische Universiteit Eindhoven en bij ASML. The voorkant van dit proefschrift geeft de samenwerking weer welke nodig is om een proefschrift, zoals hier voor u ligt, af te ronden, alsook de samenwerking welke nodig is om een ASML wafer scanner te integreren en te testen.

Cover photo: ©ASML. Cover design: ©I.S.M. de Jong.

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 2008-08

ISBN 978-90-386-1219-5. 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 and the Eindhoven University of Technology as part of the TANGRAM research project for integration and testing. The TANGRAM research project was performed under responsibility of the Embedded Systems Institute (ESI) and has been partially sponsored by the Netherlands Ministry of Economic Affairs under grant TSIT2026.

Integration and test strategies for complex manufacturing machines

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 donderdag 27 maart 2008 om 16.00 uur

door

Ivo Samuël Maria de Jong

geboren te Rotterdam

Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. J.E. Rooda

Copromotor:
dr.ir. J.M. van de Mortel-Fronczak

PREFACE

These are the last words that I write in this two hundred page thesis. It took me about four years as a Ph.D. student in the TANGRAM project to come to this point. In addition, it took me seven more years in various test and integrator roles at ASML, before I started in the TANGRAM project. My ASML career started in November 1996, where I was one of four testers/integrators in the software validation and verification group. The number of engineers in the software development group approached fifty at that time, if I recall correctly. I had the opportunity to work in various integration and testing roles for many software releases.Integration and testing of these software releases has been a struggle despite the use of a standard software releasing process. Moreover, integration and testing of newly developed wafer scanners, including software and hardware, has been an even bigger struggle. A more structured integration and testing method was required, such that the time-to-market and quality requirements can still be met in the future. Developing such a method within the context of an organization like ASML is not easy. This was one of the reasons for ASML and the Embedded Systems Institute to start the TANGRAM research project in 2002.

My involvement in the TANGRAM project started in the beginning of 2003. TANGRAM was a welcome change of environment compared with the high pressure TWINSCAN reliability improvement project that I was involved in since 2001. In the beginning of 2003, I asked Koos Rooda (TU/e) and Tammo van den Berg (ASML) what the possibilities were to participate in the TANGRAM project with the goal to pursue a Ph.D. degree. I followed some courses to refresh my knowledge and eventually everybody agreed to proceed with this plan. The support of Tammo van den Berg, the many discussions and guidance of Koos Rooda, the review effort and the help with the mathematics of Asia van de Mortel-Fronczak and the critical questions of Tom Brugman contributed in many ways to this thesis. Therefore, I would like to show my deepest gratitude to Tammo, Koos, Asia and Tom for this opportunity, the support and all help in these years. I think that my involvement in the TANGRAM project was beneficial for all of us.

The progress made in our TANGRAM sub-project was only possible, because of the cooperation with Roel Boumen. It was clear from the beginning that we both would write our own thesis, while we both had the advantage of a common way of reasoning about integration and testing. Thank you, Roel, for your sharp mind. Your focus and dedication kept me going. In addition, thank you for the fun while working and traveling with you!

Many people were involved in the TANGRAM project. Some only in the beginning, some only in the end. Thank you *all* for your cooperation. Special thanks

are reserved for Niels Braspenning for the last 6 months where we both were writing our thesis and the fun we had in Xi'An and New York. Thank you, Jan, Jurryt, Luud, Michiel, René, Will and the other TANGRAM members for your cooperation. Many thanks and gratitude go out to the students that worked with us. Thanks to Rolf Theunissen for developing the first version of the 'flexible' simulator. Thanks Pieter Verduijn, Marcel van der Heijden and Martijn van Campenhout for developing the publish-subscribe based LONETTE tool-set and many thanks to Hans Ekelmans for all the work on test model partitioning.

I also would like to thank the members of the committee, Jos Baeten, Ed Brinksma, Tom Brugman, Wan Fokkink and Arjan van Gemund, for the helpful suggestions and the questions raised. In addition, I would like to thank Jan Tretmans for reviewing this thesis and several papers. Thanks also to the many persons at ASML that were involved in any way in my research. Thanks to the members of the Systems Engineering group of the TU/e for their support and the friendly and cooperative environment.

Last but not least, this work would not have been possible without the constant support, love and interest of my family and friends. Especially, Robin and Wilma for proof reading my thesis.

Finally, I cannot offer more than my very special thanks and gratitude to my wife Nellie and our two sons, Thijs and Tom, for their love and support.

Ivo de Jong
Oosterhout, January 2008.

SUMMARY

Integration and test strategies for complex manufacturing machines

Complex manufacturing machines, like ASML wafer scanners, consist of thousands of components like electronic boards, software, mechanical parts and optics. These components of multiple disciplines are assembled or integrated into modules. The modules are integrated into sub-systems forming the system, according to an integration plan. Components as well as modules, sub-systems, and systems, can be tested, diagnosed and fixed, according to a test-diagnose-fix plan. An increase in the number of components results in an increase of the number of tasks in these plans. Moreover, the effort required to obtain a sequence that describes in which order the tasks should be executed also increases. The duration and the cost of a sequence depends on the quality of the system. In this project we introduce a method to analyze the duration and the cost of sequences of integration and test-diagnose-fix tasks. The method uses test-diagnose-fix models to analyze the performance of sequences. The basic elements in such a model are: a) test, diagnose and fix tasks with their costs and durations, b) fault states, c) the coverage of test tasks on fault states, d) failure probabilities of fault states. These elements can be obtained for components, modules or sub-systems of multiple disciplines. Three case studies have been performed using this method. The outcome of the analysis indicates that choosing a different test sequence can reduce the test duration by 30% to 70%. In addition, three techniques have been developed to improve integration and test-diagnose-fix sequences:

- To reduce the execution time of test-diagnose-fix sequences an algorithm has been developed to determine a new test task with an optimal coverage w.r.t. the fault states. The algorithm selects the new test task based on the maximum information gain. A test sequence, including the new test case, improves the test duration of the test-diagnose-fix task, because faults can be detected earlier.

- To reduce the execution time of test-diagnose-fix sequences an adapted hypergraph partitioning algorithm has been developed. The algorithm partitions a test-diagnose-fix task into smaller tasks which can be executed in parallel. The result of a case study is a reduction of the test duration by 30% with a concomitant increase of 30% in the test cost.

- The impact of the choice of the system architecture on the execution time and planning effort of integration and test-diagnose-fix sequences is investigated.

SAMENVATTING

Complexe fabricagemachines, zoals de waferscanner ontwikkeld door ASML, bestaan uit duizenden componenten zoals electronische borden, mechanische onderdelen, software en optica. Deze componenten worden geassembleerd, c.q. geïntegreerd, in modules. Modules worden volgens een integratieplan geïntegreerd tot sub-systemen, die samen het systeem vormen. Componenten, modules, sub-systemen en het systeem kunnen worden getest, gediagnosticeerd en problemen kunnen worden opgelost, volgens een *test-diagnose-fix*-plan. Een toename van het aantal componenten resulteert in een toename van het aantal taken in dit plan. Daarnaast neemt ook de inspanning toe die nodig is om een integratievolgorde met daarin test-diagnose-fix-taken te verkrijgen. In dit project wordt een methode geïntroduceerd om de tijdsduur en kosten van integratie- en test-diagnose-fix-taken te analyseren. Deze methode maakt gebruik van een test-diagnose-fix-model om de prestatie te analyseren van een integratie- en test-diagnose-fix-volgorde. De elementen van dit model zijn a) test-, diagnose- en fixtaken met de bijbehorende kosten en tijdsduur, b) mogelijke fouten, c) de dekking van de testtaken ten aanzien van de mogelijke fouten, d) de kans dat een mogelijke fout aanwezig is. Deze elementen kunnen worden bepaald voor componenten, modules en sub-systemen uit meerdere disciplines. Drie casussen zijn uitgevoerd, gebruikmakend van deze methode. Het resultaat van de analyse van de test-diagnose-fix-volgorde is dat een andere volgorde kan leiden tot een reductie van de test-diagnose-fix-duur van 30% tot 70%. Daarnaast zijn in dit project drie technieken ontwikkeld die integratie- en test-diagnose-fix-volgorden verbeteren:

- Om de tijdsduur van een test-diagnose-fix-volgorde te verkorten is een algoritme ontwikkeld dat een nieuwe testtaak welke een optimale dekking heeft kan bepalen. Dit algoritme selecteert deze nieuwe testtaak gebaseerd op de toename in informatie van deze taak.

- Om de tijdsduur van een test-diagnose-fix-volgorde te verkorten is een aangepast *hyper-graph* partitioneringsalgoritme ontwikkeld. Dit algoritme verdeelt de taken van een test-diagnose-fix-taak over meerdere test-, diagnose- en fixtaken die parallel uitgevoerd kunnen worden. Het resultaat van een casus met deze methode is een verkorting van de tijdsduur van een test-diagnose-fix-volgorde van 30% met een toename van de kosten van 30%.

- De invloed van de keuze van de systeemarchitectuur op de duur en planningsinspanning van een integratie- en test-diagnose-fix-volgorde is onderzocht.

CONTENTS

Preface v

Summary vii

Samenvatting ix

1 Introduction 1
- 1.1 ASML . . . 3
- 1.2 The Tangram research project . . . 7
- 1.3 Research questions . . . 8
- 1.4 The outline of this thesis . . . 10

2 Modeling for integration and test plans 13
- 2.1 Integration and test plans in different organizations . . . 14
- 2.2 Integration and test tasks . . . 27
- 2.3 System integration and test models . . . 31

3 Integration and test sequencing 45
- 3.1 Integration strategies . . . 46
- 3.2 Test positioning strategies . . . 49
- 3.3 Integration sequencing . . . 52
- 3.4 Integration and test sequencing . . . 53

4 Integration and test planning 55
- 4.1 Analyzing integration and test sequences . . . 56
- 4.2 Planning and analysis of test-diagnose-fix tasks . . . 64
- 4.3 Planning and analysis of reliability test-diagnose-fix tasks . . . 88

5 Improving integration and test sequences 101
- 5.1 Partitioning test-diagnose-fix tasks . . . 102
- 5.2 Developing new test cases . . . 125
- 5.3 Updating constraints and/or objectives . . . 144
- 5.4 Selecting a system architecture and design . . . 145

6 Conclusions 157

A Appendices 163

Bibliography 173

About the author 181

1 INTRODUCTION

Research performed by Microsoft and other parties [Boehm, 1981; Engel et al., 2004] report that during system development 30-50% of the effort is spent on integrating and testing of their systems. These numbers match with the duration spent on integrating and testing ASML wafer scanners [Brugman, 2003]. Wafer scanners are used to manufacture integrated circuits (ICs). Nowadays, ICs consist of millions of transistors and this number of transistors is growing. Since 1970, the number of transistors on an IC doubles every 18 months according to Moore's Law [Moore, 1965]. Moore's Law, in a way, dictates the semiconductor industry. Wafer scanners are manufacturing machines that perform the most critical step in the development process of ICs. Because of this, these wafer scanners enable Moore's Law.

Doubling the number of transistors in an IC every 18 months requires a considerable reduction in IC size in the same time period. This reduction corresponds with a, so called, 'shrinking node'. Completely new manufacturing technology could be required for every 'shrinking node'. For ASML, this means that new types of wafer scanners need to be developed, integrated and tested at a constant pace.

The wafer scanners provided by ASML enable IC manufacturers to follow Moore's Law.

The wafer scanners provided by ASML enable IC manufacturers to follow Moore's Law. A new type of wafer scanner enables the IC manufacturer to be ahead of the competition. Delivering a new type of wafer scanner as early as possible to customers is therefore important for ASML. One could say that Moore's Law results in very high time-to-market demands for companies like ASML.

Shrinking the IC patterns, such that the number of transistors can be doubled, requires that new technology needs to be introduced into IC factories. New wafer scanners or even new wafer scanner platforms are developed for this purpose. These new wafer scanner types become more complex with each shrinking node. This, in return, results in a growing number of components and a growing complexity of the individual components. The capabilities of new wafer scanners increase on a par with the list price of these systems. Figure 1 depicts the increase in the relative list price as function of time. The *wavelength*, depicted as I-Line, KrF, ArF, ArFI and EUV? along the time-axis, indicates the type of light source that is used. Each new light source emits light with a shorter wavelength resulting in better imaging capabilities i. e. smaller transistor features. The *aperture*, in the range 0.4 to 1.35, also has an impact on the imaging capability of a wafer scanner, i. e. the size of the transistor features. Higher apertures lead to smaller lines. The *platform*, described as Stepper, Step & Scan and Dual Stage, indicates when a new wafer scanner platform is introduced to enable future developments. The *wafer size*, 150mm, 200mm or 300mm, is the

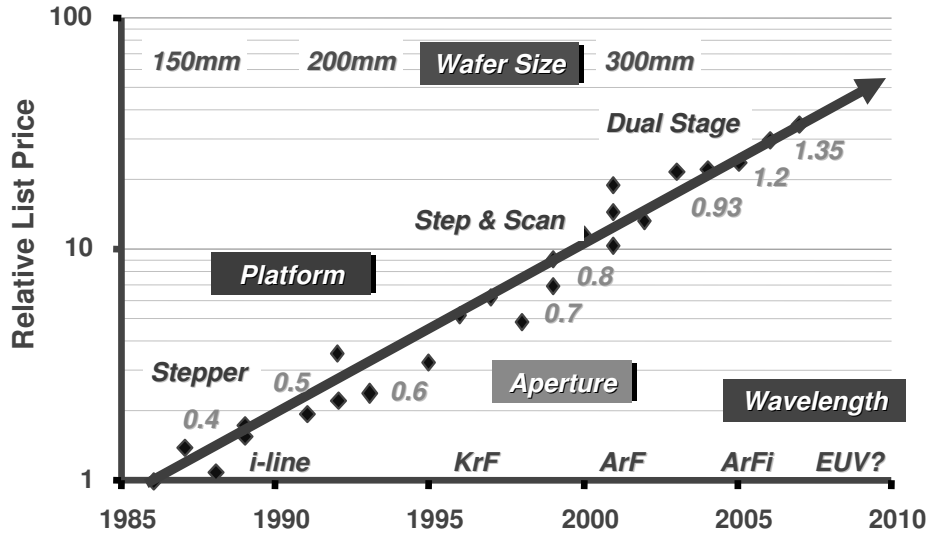


Figure 1. The relative list price versus the year of shipment Source: ASML at the Back of America Annual Investment Conference, San Francisco, Sept 18th, 2007

Each new type of light source, aperture (lens), platform and wafer size results in an increase of the number and the complexity of the components in a wafer scanner.

increasing size of the silicon wafer that enables the imaging of more ICs in a single production run. Each new type of light source, aperture (lens), platform and wafer size results in an increase of the number and the complexity of the components in a wafer scanner.

Adding more components, made of multiple disciplines, to the design of a wafer scanner also has an impact on integration and testing, because these components need to be tested, integrated and the integrated components also need to be tested. Adding a single component results in an additional integration task. Moreover, additional, so called test-diagnose-fix tasks[1], are required to qualify the component. The quality of the component and how it can be integrated into the complete system dictates whether adding this new component leads to a time-to-market increase. A lot of planning and re-planning effort is spent in preventing additional components from becoming critical in the sequence of integration and test tasks.

The planning effort increases when the number of components and the complexity of the components in a system increases. Testing, consisting of test execution, diagnosing problems and developing solutions, is an inherently stochastic process, because it is unknown beforehand what faults are present in the system. Therefore, a set of test cases needs to be selected that finds these possible faults. Moreover, the actual duration varies depending on the test strategy used. The variability in combination with the large number of components

[1] A *test-diagnose-fix task* is the testing task where test cases are executed, failed test cases are diagnosed and diagnosed problems are fixed.

and their multi-disciplinary nature initiated our research into *integration and test strategies* in the TANGRAM project.

Model-based techniques for integration and testing are the focus of the TANGRAM research project. The TANGRAM project has been split up into five subprojects. The work described in this thesis has been performed in the subproject concerning *integration and test strategies*. The resulting integration and test planning method utilizes these integration and test strategies. This planning method is explained in detail in this thesis, including the models and algorithms used. The remainder of this introduction describes the integration and testing background at ASML. Then the context of the TANGRAM project is introduced, followed by the research questions and a thesis outline. The remainder of this thesis contains four chapters, Chapter 2 to Chapter 5, each describing a detailed step of the integration and test planning method. Conclusions are drawn and recommendations are made in Chapter 6.

The integration and test planning method is explained in detail in this thesis, including the models and algorithms used.

1.1 ASML

ASML is the industrial partner in the TANGRAM project. This means that ASML is the primary source of case studies for the research performed by the academic project partners. The project partners use the industrial case studies to verify and validate existing and new theories.

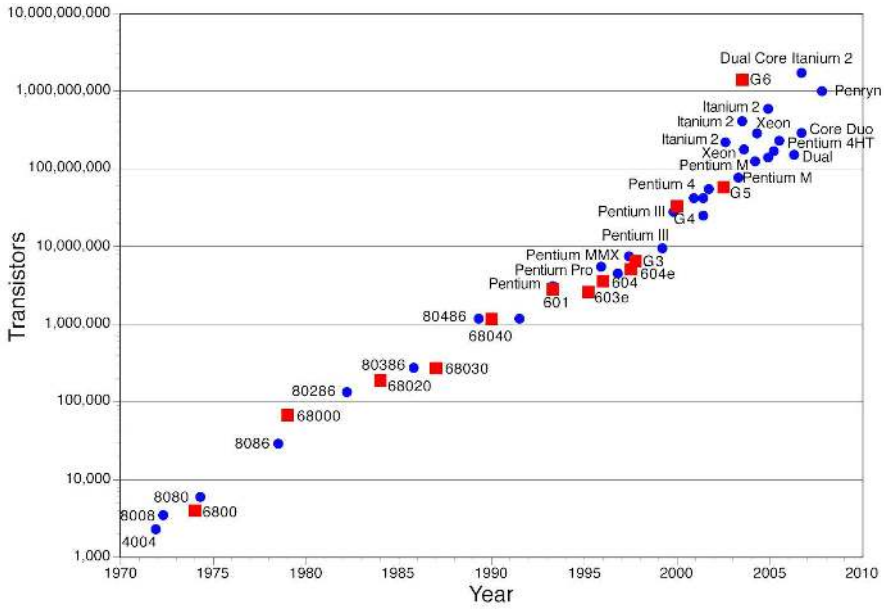


Figure 2. Moore's Law Means More Performance. Processing power, measured in millions of instructions per second (MIPS), has steadily risen because of increased transistor counts

ASML is a world leading manufacturer of advanced technology systems for the semiconductor industry. The company offers an integrated portfolio for manufacturing complex integrated circuits. ASML designs, develops, integrates, markets and services these advanced systems. Global semiconductor manufacturers use ASML wafer scanners to manufacture ICs for computers, mobile phones, PDA's, MP3 players and many other devices. Typical types of chips include microprocessors and memory chips. The basic elements of an IC are transistors. Many transistors can be used to form complex ICs, like Intel™ Core 2 Quad microprocessors or SAMSUNG™ 64Gigabit NAND chips announced in October 2007. More transistors in general means more processor power or more available memory. Figure 2 illustrates Moore's Law by depicting the number of transistors for Intel™ processors released in the time frame 1970 - 2006.

Figure 3. The ASML XTIII:1900i wafer scanner

Nowadays, IC manufacturers are able to place billions of transistors on an IC, because of efforts to continuously shrink the transistor size over the last 35 years. ICs currently manufactured using the ASML TWINSCAN™ XT:1900i wafer scanner, depicted in Figure 3, contain structures with a linewidth of 45nm or less. Note that the linewidth of the Intel 4004 processor, manufactured 35 years ago, was 10 μm.

ICs are manufactured on a silicon wafer. The structure of the transistors is 'imaged' on this wafer in a similar fashion to the way that light reaches the 'negative' in a photo camera. A transistor is composed out of five to thirty or more of these 'images'. The 'imaging' process needs to be repeated for each of the 15-30 layers that are placed on top of each other to form an IC. Between the imaging steps, other steps are performed, like baking, etching and coating. A simple overview of the IC manufacturing process of a single layer is depicted in Figure 4. ASML wafer scanners are used for Step 5 in the IC manufacturing process: the exposure of the structure on the silicon wafer. The image of the structure that needs to be placed on the wafer, the reticle or mask, is placed in the wafer scanner. Then, a wafer is loaded into the wafer scanner and laser emitted light is sent through a series of lenses and through the reticle and again through a series of lenses for shrinking purposes. The resulting image reaches

ASML wafer scanners are used for Step 5 in the chip manufacturing process: the exposure of the structure on the silicon wafer.

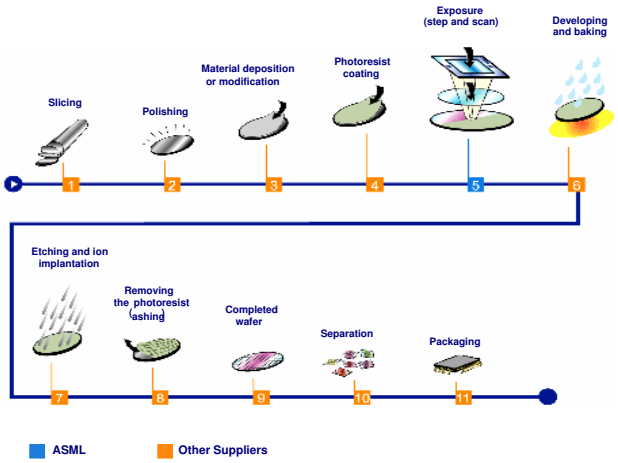


Figure 4. The IC manufacturing process

the wafer surface that contains a photo-resistant coating. The coating reacts with the light that reaches the wafer. The coating does not react at the places where the light does not pass the lines placed on the reticle. Multiple images are exposed next to each other, until the wafer is fully exposed. Then the wafer is unloaded from the wafer scanner. The photo-resistant coating is developed in the next processing step (not in the wafer scanner). Additional steps, such as material deposition or edging, can be performed on the wafer by other manufacturing machines. A new layer of photo-resistant coating starts the next imaging cycle using another reticle. An example reticle, which contains the design of the IC, and a schematic overview of the imaging process in the wafer scanner are given in Figure 5.

The three most important performance characteristics of a wafer scanner are: throughput, overlay and imaging.

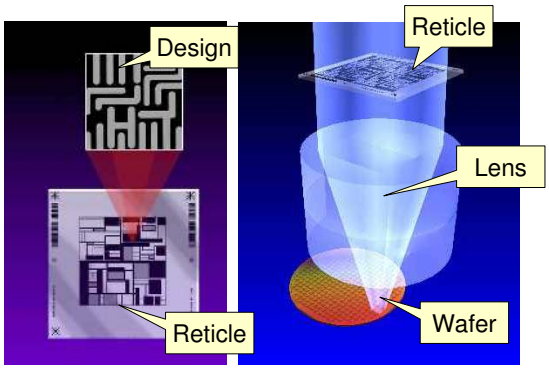


Figure 5. From design onto reticle and a reticle that masks the light to pattern the image on the wafer

The three most important performance characteristics of a wafer scanner are: throughput, overlay and imaging. Throughput determines how fast the wafer scanner can process wafers. Current throughput requirements exceed 150 wafers per hour, depending on the system type. The second important characteristic is overlay. Overlay is the measure of how well two layers are imaged on top of each other in two subsequent process steps: the placement of subsequent images forms the three dimensional structure of a transistor. A misplaced image can result in a non-functional IC. The overlay capability of a wafer scanner also determines how good two images can be placed on top of each other and therefore influences the number of transistors per cm^2. The third important characteristic of a wafer scanner is the imaging capability. The imaging capability is the minimal line width that can be imaged using the wafer scanner. A transistor built up using smaller lines results in more transistors per cm^2.

1.1.1 *Integration and testing of wafer scanners*

The integration and testing activities of wafer scanners can be found throughout the research, development, manufacturing, customer installation and operational phases of wafer scanners. During wafer production, at the customer site, periodic maintenance is performed to test and calibrate (fix) the performance of the wafer scanner. The duration of the assembly and testing phase when manufacturing and installing new wafer scanners at customer sites is measured in days to weeks. The hardware of the complete system is assembled in this period and then tested and calibrated until the nanometer performance is met. The manufacturing phase is the most expensive period for ASML, because all hardware is present, while the system is not signed off by the customer. Subsystems are partly manufactured and tested at ASML or tested and delivered by suppliers. The focus of development testing is on design qualification of new hardware, electronics, software and optics. These test-diagnose-fix tasks are performed on testrigs, a part of a wafer scanner, and on prototype systems. During the research phase, new sensors and stages are studied and the feasibility of new concepts is tested. These studies are performed together with partners and ASML subsidiaries worldwide. Often, special test equipment is developed for this purpose.

Many integration and testing activities at ASML are on the critical path.

Many integration and testing activities at ASML are on the critical path to the delivery of new wafer scanners to customers. Nowadays, several new types of wafer scanners are developed, integrated and tested concurrently. The pressure on integration and testing of these new wafer scanners increases. Integration tasks and test-diagnose-fix tasks are mixed such that the minimal time-to-market is obtained. This approach has resulted in impressive time-to-market achievements in the past. The increase in the number of components and their complexity, together with the parallel development of new types of wafer scanners, requires considerable improvements in the current integration and testing way of working.

Maturity models (CMMi)[SEI, 2007], (TMMi)[E. van Veenendaal, 2006] and

test process improvement (TPI)[Koomen and Pol, 1999] are natural ways to improve the quality of the system and the integration and test process. Next to these process oriented methods, product oriented formal methods are available or should be investigated. Theories, often state-of-the-art at research institutes, solve specific integration and testing problems for specific cases. A combination of the continuous process improvement effort and state-of-the-art theories and tooling is required for short term improvement and long term acceleration of the integration and test process. This is one of the reasons why the TANGRAM project was started by ASML together with the Embedded Systems Institute (ESI). The next section introduces the goals of the TANGRAM project and the partners involved.

A combination of the continuous process improvement effort and state-of-the-art theories and tooling is required for short term improvement and long term acceleration of the integration and test process.

1.2 THE TANGRAM RESEARCH PROJECT

The goal of the TANGRAM research project is to reduce the duration of the integration and test phase of large complex embedded systems, like the ASML wafer scanner. The cost and product quality should not be affected. TANGRAM has been initiated by the Embedded Systems Institute and ASML . The increase in the number of components, number of new system types to be developed and delivered in parallel, and the complexity of the components resulted in the need for a large scale research project in the area of integration and testing. Integration and testing these complex systems always remains necessary, because it is unknown beforehand if the components are of perfect quality.

In Figure 6 there are two test durations t_1 and t_2 depicted that can be used to illustrate the goal of the TANGRAM project. The integration and testing task starts in parallel with the development task and progresses until the shipment date of the product. Development is finished before the integration and testing task is finished. The time between the end of the development task and the shipment date is marked by t_1. Fixing problems found during testing is considered part of testing. The duration of the testing task after product shipment is marked by t_2. The goal of the TANGRAM project is to reduce both t_1 and t_2. Note that the actual shipment date is not relevant for the reduction of t_1 and t_2.

The work in the TANGRAM project was divided into four, so-called, 'Lines of attention'. Line of attention 1 (LoA1) focused on integration and test strategies.

The work in the TANGRAM project was divided into four so-called 'Lines of attention' or LoAs. Line of attention 1 (LoA1) focused on integration and test strategies. Line of attention 2 (LoA2) focused on integration and test infrastructure. Line of attention 3 (LoA3) focused on model-based testing. Line of attention 4 (LoA4) focused on model-based diagnosis. Along the way, a fifth line of attention was defined that focused on early model-based integration and testing. These five lines of attention focused on the five most important integration and testing problems of ASML. This thesis describes a part of the research performed on integration and test strategies in LoA1. The other part focused on integration and test sequencing. The partners that cooperated in LoA1 are ASML and Eindhoven University of Technology, Department of Mechanical Engineering, Systems Engineering Group.

The expertise in the Systems Engineering section of the Eindhoven Univer-

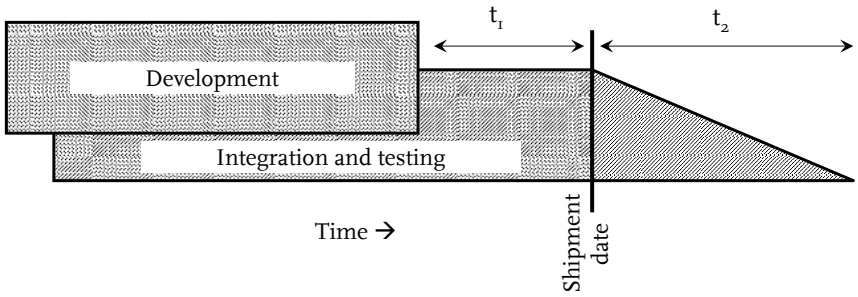


Figure 6. The goal of the TANGRAM project is to reduce both t_1 and t_2, the integration and test activities between end of development and product shipment and the integration and test activities after shipment.

sity of Technology is in the analysis of manufacturing machines on the one hand and on the other hand the supervisory control of embedded systems. The techniques, tools and methods developed for the analysis of manufacturing machines have been used for the analysis of the integration and test process. The initial LoA1 framework that described integration and testing as a four step process with a modeling step, a sequencing step, a scheduling step and an execution step was based on a scheduling framework seen in [Pinedo, 2001]. This framework evolved during the course of the project in the integration and test planning method described in this thesis. An alternative approach was followed in LoA1 relative to the other LoAs. This approach was different for two reasons. Firstly, the initial goal was rather broad and secondly the methods that were to be used to solve the integration and test strategy problem were not known. In addition, these methods should be applicable in a broad problem area and for components of multiple disciplines as opposed to the methods in the other LoAs, which are often applicable for a single problem area for components of one or a few disciplines. The initial objectives for line of attention 1 were translated into three research questions explained in the next section.

1.3 RESEARCH QUESTIONS

The TANGRAM project plan [Brugman and Beenker, 2003] summarizes the goals for LoA1. Parts of the original project plan are used here to illustrate the original goals: *In the current integration and test strategy the main test problems occur when the various disciplines integrate their results into the final system. Integration is in practice not always driven by the functional dependencies and the risks that changes are supposed to carry with them. Very often development resources, delivery schedules, a predefined timetable of machine and/or software integration/test slots, and a number of other constraints determine the integration and test order. Due*

to this, currently various system aspects are not adequately tested before first product release. The objectives for line of attention 1 are:

- *Develop an integral multidisciplinary test strategy such that:*
  - *Testable system equals sum of testable sub-systems, etc.*
  - *Components can be tested and diagnosed in sub-system environment.*
  - *Minimum granularity of testable component equals replace spare parts*
  - *Sub-systems can be early tested in isolation and in joined system and simulation context, the same sub-system can be tested in its environment.*
- *Allow for and define a growth path from current system architecture to a testable system architecture*
- *Trade-off analysis for testability requirements (e.g. on cost and performance) based on risk analysis.*

The initial objectives from the TANGRAM project plan resulted in a rough direction that Line of Attention 1 should follow, but not a clear set of research questions. The first period in the project was spent on investigating integration and test strategies within ASML and at different organizations. The company visits are described in Section 2.1. Investigating integration and test plans in different organizations and mapping these results to the ASML case led to the following research questions. Research question 1 relates to the difference in integration and test approach observed in various types of organizations.

Research question 1

Which organizational factors have an impact on the integration and test plan for systems developed by that organization?

The second research question relates to the form and performance of integration and test plans.

Research question 2

What are the basic elements of integration and test plans? What are the key performance indicators of integration and test plans? How can these key performance indicators be measured and used to compare different integration and test plans with each other?

Research question 3 relates to improvement techniques for integration and test plans. Developing a single integration and test plan is step one, comparing this plan with a number of plans is the second step. Choosing the best plan and improving this plan is the third step.

Research question 3

Which improvement techniques for integration and test plans are beneficial for complex manufacturing machines?

1.4 THE OUTLINE OF THIS THESIS

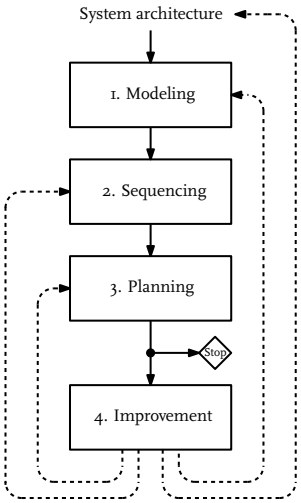


Figure 7. Overview of the integration and test planning method

The integration and test planning method consists of four steps: 1) modeling, 2) sequencing, 3) planning and 4) improvement.

The structure of the integration and test planning method has been used to organize this thesis. The integration and test planning method consists of four steps: 1) modeling, 2) sequencing, 3) planning and 4) improvement, as shown in Figure 7. The dashed lines are the feedback loops originating from the improvement step. A summarized integration and test planning method can be found in [Tretmans, 2007].

Chapter 2 focuses on the modeling step in the integration and test planning method. The chapter introduces the models, that are used in the other chapters: *system test models* and *system integration models*. System test models can be used by test case developers, reviewers, test planners and test executors, since the model represents all information used for test-diagnose-fix tasks in a structured form. Integration models describe the elements required for integration sequencing and planning. The integration models are relevant for integration and test planners.

Chapter 3 describes how an integration and test sequence is obtained using two strategies, an *integration strategy* and a *test positioning strategy*. This chapter corresponds with the sequencing step depicted in Figure 7. A few strategies are discussed including advantages and disadvantages of these strategies.

Chapter 4 describes the planning step of Figure 7. Planning, analysis and improvement for single test-diagnose-fix tasks is described in this chapter. Planners of test-diagnose-fix tasks can use these techniques to create and analyze test-diagnose-fix plans. The effect of a different *test strategy* is evaluated, such that the best test-diagnose-fix sequence can be selected.

Chapter 5 describes the improvement step of the integration and test planning method. Improvement techniques are defined, that improve the integration and

test sequence as a whole. Integration and test planners can analyze the effect of these improvement techniques on the duration, cost and remaining risk[2] of the integration and test plan, such that the best plan can be selected.

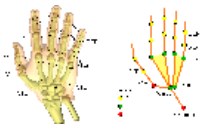
The conclusions of this thesis are presented in Chapter 6.

Every chapter starts with a picture describing the details of the step in the method, which is explained in that chapter.

Each chapter starts with a picture describing the details of the step in the method, which is explained in that chapter. The steps, which are explained in each chapter are depicted as ovals, while the inputs and outputs are depicted as rectangles. The steps, inputs and outputs are uniquely numbered.

[2] The definition of risk is based on [Kaplan, August 1997] and used for risk-based testing in [Amland, 2000; Pfleeger, 2000]. Risk is calculated by multiplying the probability (or likelihood) that a failure occurs with the impact (or consequence) of that failure. The failure probability and impact are introduced in detail in Section 2.3.2.

2 MODELING FOR INTEGRATION AND TEST PLANS



The first step in the integration and test planning method is the modeling step. The elements of the modeling step in the method are depicted in Figure 8. The *system architecture* is modeled such that it can be used for integration and test planning. The inputs of this step are the system architecture (0.A) and objectives and constraints (0.B) of the integration and test sequence. First, the *system integration model*, a model of the system architecture, is made in step (1.1). Based on this model, *system test models*, containing fault states, test cases and their properties are defined in Step (1.2). Section 2.3 describes these models, their elements and the elements that are used to compose integration and test sequences. These elements are a result of the investigation into integration and test sequences at different organizations. The next section describes the influence of different business drivers on the integration and test sequences that are used by the visited organizations.

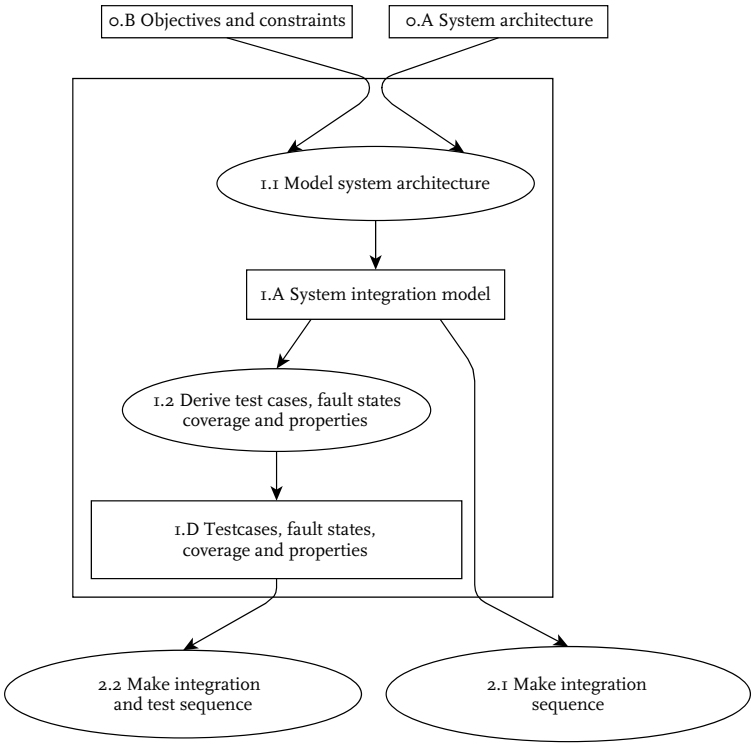


Figure 8. Overview of the modeling step

2.1 AN OVERVIEW OF INTEGRATION AND TEST PLANS IN ORGANIZATIONS WITH DIFFERENT BUSINESS DRIVERS

Before the system integration model and the system test model are introduced an overview of different integration and test plans in different organizations is given. This overview shows that organizations with different business drivers handle integration and testing differently. The elements of an integration and test sequence are equal for these organizations. This section is based on [I. de Jong et al., 2007a] and [Tretmans, 2007].

Planning an integration and test task is often done by experts in the organizations visited. These experts have a thorough knowledge of the system, integration and testing and the business drivers of an organization. An integration and test sequence developed for an airplane is different from the integration and test sequence for a wafer scanner. Safety (quality) is most important for an airplane, while time-to-market is most important for a wafer scanner. These important aspects are reflected in the integration and test sequence. A number of companies have been visited in order to investigate the influence of the business drivers on the resulting integration and test sequences. An integration and test sequence describes the tasks that have to be performed to integrate individual components into a system. Test tasks are performed between these integration tasks. Note that integration is sometimes called assembly. Integration and testing is performed in early development phases and also in a manufacturing environment. Business drivers describe what are the most important drivers for an organization. Business drivers are defined in terms of time, cost or quality. The hypothesis is that the order in which the importance of business drivers is perceived in an organization determines the way of working and therefore the integration and test sequence.

An integration and test sequence describes the tasks that have to be performed to integrate individual components into a system. Test tasks are performed between the integration tasks.

The goal of the investigation into different integration and test plans at different organizations is to determine what are the common elements of such an integration and test sequence. In addition, the differences are investigated. A number of aspects of an organization in addition to the business drivers are recorded, for example: company size, product volume, number of components in the system, technology used and the sub-contractor model. Note that much of the data is obtained directly from the organization or from publicly available resources.

The structure of this section is as follows. First, the business drivers and organizational aspects, which we consider to be of influence, are discussed. Then, the different organizations, business drivers, organizational aspects and integration and test plans are discussed in detail followed by a summary and conclusions.

2.1.1 *Business drivers*

Business drivers are the requirements that describe the goal of an organization. The business drivers *Time*, *cost* and *product quality* are known from manufacturing management [Laugen et al., 2005; Pawar et al., 1994]. We will use these business drivers to characterize the investigated organizations.

We use the business drivers *Time*, *cost* and *product quality* to characterize the investigated organizations.

An organization with *time* as the key business driver is focused on delivering products as quickly as possible to the market. An organization with *cost* as key business driver is focused on delivering products as cheaply as possible to the market. Finally, an organization with *product quality* as key business driver is focused on delivering products to the market that satisfy the customer as much as possible. The order of importance determines the way of working in the organization. For example, an organization with T-C-Q (Time first, cost second and quality least important) as business drivers delivers products of different quality and production cost than an organization operating with T-Q-C as the order of its business drivers. Organizations of these types are described in more detail in the next sections. Both deliver products as quickly as possible to the market. The first organization develops, manufactures and services these products as cheaply as possible. Product quality is least important. The focus of the second organization is on product quality (after fast delivery). Cost is least important.

2.1.2 *Organizational aspects*

The integration and test sequences of very different organizations were investigated. Sometimes a specific department was visited. The observed integration and test sequence was probably only type of sequence in that organization, while the business drivers are identified for the entire organization. Therefore, additional aspects of the organization are recorded to determine the possible effect of these aspects. The organizational aspects recorded are:

1. The number of products shipped per year and number of end users; both influence the required product quality and maintenance cost. A higher product quality is required when a high number of products is shipped, otherwise the repair cost would be too high.

2. More complex products result in more complex integration and test sequences. Complexity can be the result of many components, resulting in many integrations and possible test-diagnose-fix tasks. Complexity can also be the result of the use of complex technology resulting in complex test cases.

3. Using many different sub-contractors for the development of components could result in many additional test-diagnose-fix tasks to test the delivered components. Next to that, political aspects could result in additional test-diagnose-fix tasks. For instance, sub-contractor test cases could be repeated to accept the delivered products, resulting in additional test-diagnose-fix tasks.

2.1.3 *Investigated organizations*

A number of different organizations have been visited to investigate the influence of business drivers on integration and test plans. A summary is given for each of the investigated organizations. The order of business drivers indicates the relative importance of the business driver, i. e. T-Q-C means that time-to-market is most important followed by quality and least important is cost. T-Q/C means that quality and cost are equally important. The order of the business drivers is determined by the authors after the visit or investigation. Next to that, relevant information like company size, product volume, number of components, technology used and the number of sub-contractors was recorded.

Semiconductor (ASML and others):

Company size	Medium, 6000 employees
Product volume	200-300 systems/year
Business drivers	T-Q-C
Number of components	average – very large
Technology used	New technology
Sub-contractors	Many, cooperating

Table 1. Semiconductor equipment manufacturer characteristics

A typical semiconductor equipment integration and test sequence (Figure 9) consists of development tasks (*dev*) executed at suppliers, followed by a supplier test-diagnose-fix task and a system assembly task (*asm*). The assembly task of each system is followed by two test-diagnose-fix tasks: the calibration test tdf_C [1] and acceptance test tdf_A [2]. Chuma [Chuma, 2006] investigated the duration of the assembly phase (*asm*) and the durations of tdf_C and tdf_A for lithographic equipment manufactured at ASML, Canon and Nikon[3]. The average duration of the assembly phase is 9.8 days while the average duration of the calibration and acceptance test are respectively 34.5 and 32.5 days in 2005 according to the report.

ASML develops semiconductor equipment using platforms. The integration and test sequence of the first wafer scanner of a new platform is developed specifically for this system (see product development later). Subsequent system types in a new platform are integrated and tested based on a previous system type. First, a previous system type is manufactured as in Figure 9. New sub-systems are developed. The old sub-systems are replaced by the new versions.

[1] A calibration test-diagnose-fix task is a task where test cases and calibration tasks are interchanged. Test cases are executed on the system to determine the performance of the system. If the system is 'out of specification', calibrations are performed and testing continues.

[2] An acceptance test is the test executed to determine if the customer accepts the system.

[3] ASML, Canon and Nikon are the main suppliers of lithographic equipment to the semiconductor market.

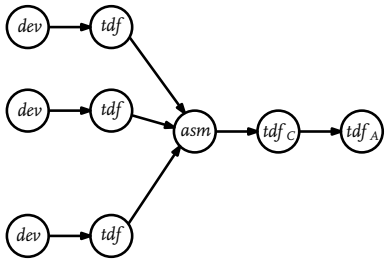


Figure 9. Typical semiconductor manufacturing integration and test plan

Figure 10 depicts this integration and test plan. The previous system type is assembled after the first assembly task. Modules, like M_1, are disassembled (*das*) and a newly developed module M_1' is assembled. Module M_2 is replaced similarly by M_2' in Figure 10.

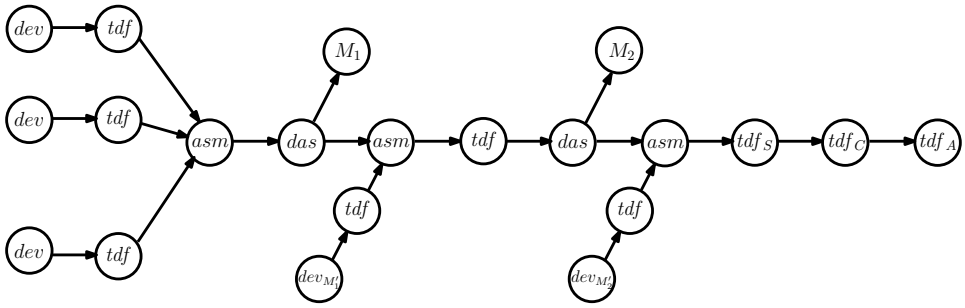


Figure 10. Semiconductor development integration and test plan

A typical aspect in this time-to-market driven organization is that the newly developed sub-systems M_1' and M_2' are not tested thoroughly. Integration progress is more important than testing the sub-systems. Remaining risk in the system is covered in higher level (later) test-diagnose-fix tasks. The final acceptance test is a combination of a thorough, system level, design qualification tdf_S and the normal final calibration and acceptance test-diagnose-fix tasks tdf_C and tdf_A. The test cases in the final test-diagnose-fix tasks tdf_S, tdf_C, and tdf_A are often mixed such that a faster test sequence is obtained.

Automotive:

A typical assembly line (Figure 11) for cars consists of a number of assembly tasks (*asm*) followed by a short final acceptance test-diagnose-fix task tdf_A. Sup-

Company size	large, 30000 employees
Product volume	100000 systems/year
Business drivers	C-T/Q
Number of components	Medium
Technology used	Proven technology
Sub-contractors	Many, cooperating

Table 2. Automotive manufacturer characteristics

pliers develop (manufacture) and test the parts that are assembled into a car. Testing is standardized and focused on quality (for instance measurement techniques for electrical systems are described in IEC 61508 Part 7 [International Electrotechnical Commission, 2005]).

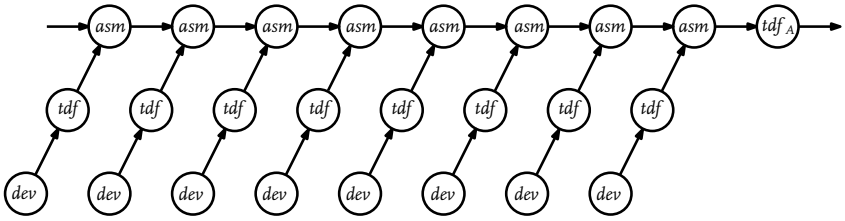


Figure 11. A typical 'assembly line' for cars

Communication:

Company size	large, 30000 employees
Product volume	120000000 systems/year
Business drivers	Q-C/T
Number of components	Small
Technology used	Proven technology and new software
Sub-contractor	Few/none

Table 3. Communication equipment manufacturer characteristics

A mobile phone communicates with other mobile phones via the (GSM/GPRS/3G) network. The communication protocol between a mobile phone and the infrastructure is standardized [ETSI, 1999-2007]. A single test-diagnose-fix task of a few weeks qualifies if a mobile phone operates according

to the standard. The visited organization developed such a standard test set and this test set is used by different mobile phone developers to test newly developed mobile phones. This test-diagnose-fix task is repeated if problems are found and fixed until the phone operates according to the standard. A generic representation of this retest task with test-diagnose-fix tasks is depicted in Figure 12 in the form of a recurring test-diagnose-fix task and development (fix) task. Note that 120000000 mobile phones have been shipped in the USA only in the year 2005 [McQueen et al., 2006]. The estimated number of shipped units in 2011 is 1.25 billion worldwide. The technology used in mobile phones consists of relatively proven hardware technology. The application (software) is new in this type of products.

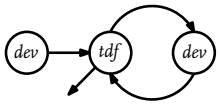


Figure 12. Specific example of a mobile phone test-diagnose-fix task

Avionics/Department of defense (DoD):

Company size	large, 30000 employees
Product volume	300 systems/year
Business drivers	Q-C-T
Number of components	High
Technology used	Proven technology
Sub-contractors	Many, regulated

Table 4. Avionics/DoD manufacturer characteristics

Airplanes and systems developed for the department of defense (DoD) are integrated and tested using a strict process, for example the integration and test process for the 777 flight controls [Buus et al., 1997]. All sub-systems are tested in the supply chain to ensure a short final test phase. To accommodate this, interfaces between sub-systems are thoroughly described and do not introduce new problems. An integration and test sequence for an airplane or DoD system

is similar to the sequence depicted in Figure 9. Sub-systems are tested completely before integration. The duration of the final calibration test phase tdf_C for an airplane, like an Airbus A320, is only a few days, including a test flight. Assemblies are performed between the final calibration test phase and acceptance test phase. For instance, the engine of an airplane is assembled when all other parts have been assembled and calibrated. The reason for this is safety and cost. Assembling an engine is done in a special area and the engine is costly, so it is assembled as late as possible.

Space (satellites):

Company size	medium, 5000 employees
Product volume	10 systems/year
Business drivers	Q-C-T
Number of components	Medium
Technology used	Proven technology
Sub-contractors	Few, cooperating

Table 5. Space/satellite manufacturer characteristics

Development of a satellite or other space vehicles results in a single unique system. The integration and test sequence is very similar to an integration and test sequence of a newly developed system. The assembly phases are executed as concurrently as possible. Test tasks are planned after each development and each assembly task such that the risk in the system is minimal at all times. An overview of international verification and validation standards for space vehicles, including the main differences between standards, is described in [Giordano and Messidoro, 2001]. A planning and scheduling method for a space craft assembly, integration and verification (AIV) is described in [Arenthoft et al., 1991].

Machine builders

Company size	Medium, 5000 employees
Product volume	1000 systems/year
Business drivers	C-Q-T
Number of components	Medium
Technology used	Proven technology
Sub-contractors	Many, cooperating

Table 6. Machine manufacturer characteristics

A number of machine building organizations has been visited. The developed systems varied from manufacturing equipment to large office equipment.

A variety of integration and test plans has been observed in the different organizations. Most of the organizations use an integration sequence that is similar to the sequence used in the automotive industry. Some use a 'fixed rate' assembly, e. g. each assembly task is performed in a fixed 20 minute time slot by a single operator. Some calibration tests are performed between assembly tasks. Configuring the system for a customer is done just before the acceptance test tdf_A. An example of a sequence with a customer specific configuration in the last assembly task is depicted in Figure 13.

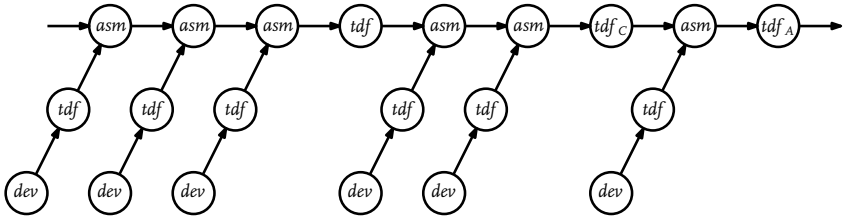


Figure 13. Example manufacturing sequence for machine builders

Drug industry:

Company size	large, 20000 employees
Product volume	Millions of tablets/year
Business drivers	Q-C-T
Number of components	Small
Technology used	New technology
Sub-contractors	None

Table 7. Drug developing company characteristics

Finally, the drug testing industry is discussed based on [Raven, 1997, 1998]. The products in this industry are different compared to the technical products discussed before. Testing of medical drugs is quite different. Figure 14 depicts an integration and test sequence for medical drugs.

The development of a potential new drug is a combination of chemical design and a structured search. The integration and test sequence starts if a new chemical entity (NCE) is discovered. A screening test (tdf_S) is performed to test the potential of the new chemical. The new chemical is then 'integrated' into tablets (dev_T) or dissolved in liquid (not depicted). What follows next are four

test phases in which the new drug is tested (tdf_A, tdf_I, tdf_{II}, tdf_{III}). The average total duration of the entire sequence is 14 years. Test phase tdf_A is performed on animals to test for toxicity and long term safety. Test phase tdf_I is performed mainly on healthy volunteers to determine the dose level, drug metabolism and bio-availability[4]. Test phase tdf_{II} is a test phase on a few hundred patients to test the efficacy of the dose and the absence of side effects. Test phase tdf_{III} is performed to test efficacy and safety on thousands of patients. Test phase tdf_{IV} is performed after the new drug has received a product license to test for rare adverse events and to gain experience with untested groups of patients. The conclusion of every test phase can be that testing will not be continued. The new drug will not be further developed and released, in contrary to the (technical) products of the other organizations where problems found can be fixed and testing continues.

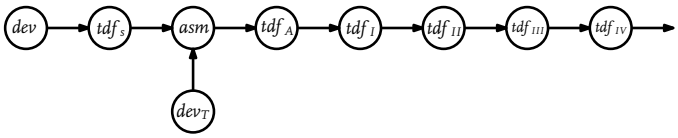


Figure 14. Integration and test sequence for medical drugs

Integration and testing of software baselines:

A special case of an integration and test sequence for product development is an integration and test sequence for software developments that are delivered into a single code base. All code ends up in a configuration management system. Testing is done on the code before delivery and on the 'release', a specific baseline in the configuration management system. Two example integration and test sequences are discussed. These types of integration and test sequences have been encountered at several visited companies, including ASML. Next to that, Cusumano describes a similar integration and test sequence that is used by Microsoft [Cusumano and Selby, 1997]. The first sequence, depicted in Figure 15, contains periodic test-diagnose-fix tasks. Integration continues when the result of the test-diagnose-fix task is *pass*.

The second sequence, depicted in Figure 16, contains a periodic test-diagnose-fix task executed in parallel with integrations of new code. A copy (*cpy*)

[4] How (and how fast) is the product entered in the body and bloodstream and how the product is excreted from the body.

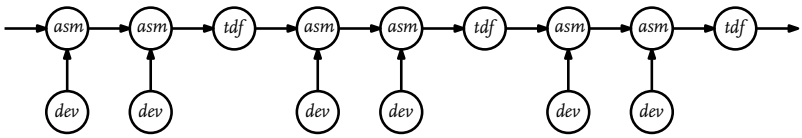


Figure 15. Software integration with periodic test-diagnose-fix tasks

of the software is made and used to test the (copied) software.

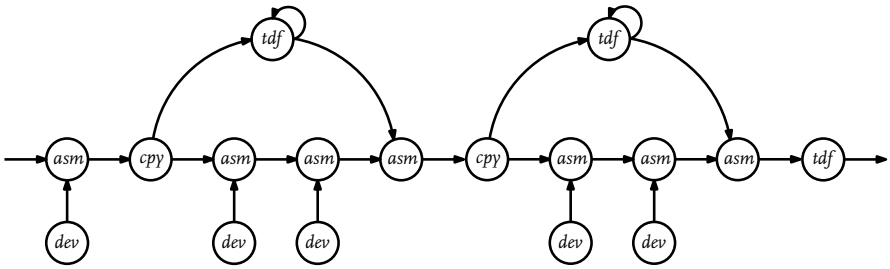


Figure 16. Software integration with parallel test-diagnose-fix tasks

The test-diagnose-fix task in the periodic case is on the critical path, while the test-diagnose-fix task in the parallel case is not. On the other hand, problems found in the periodic case are solved before new integrations are performed. Problem solving in the parallel case is more complex, because two baselines are to be maintained at any point in time. This is depicted in Figure 16 with an explicit 'self-loop' on the test process and an explicit assembly of solutions into the baseline.

2.1.4 *Overview of organizations and integration and test plans*

An overview of the organizational types and their influence on an integration and test sequence is depicted in Figure 17. The organizational types can be found in Table 8. Each circle indicates a visited or investigated organization. The size of the circle indicates the size of the organization (large circles correspond

with large organizations). The gray tone of the circle indicates the number of delivered end products. A darker circle indicates more shipments. Each circle contains the key business drivers (in order) for the visited organization. The organizations are placed in the graph in Figure 17 according to the integration and test planning approach on the x-axis (regulated or flexible) and the system complexity on the y-axis. The complexity is a combination of number of components and technology used. The type of organization is described in the bottom half of the circle. In some cases, multiple organizations of the same organizational type have been investigated.

A distinction is made between a regulated approach and a flexible approach, because these type of integration and test sequences were most different from each other in the observed organizations. The strategy of a *regulated approach* is focused on removing all risk as soon as possible. Consequently, test-diagnose-fix tasks are planned after each development and assembly task. The focus of each test-diagnose-fix task is on removing all possible risk. The flexible approach, on the other hand, is focused on maximal integration progress. Test tasks are planned after *some* of the development and assembly tasks. These test-diagnose-fix tasks are partially executed and the remaining risk is covered by later test-diagnose-fix tasks.

The *flexible approach* allows the improvement of test-diagnose-fix tasks by moving test cases from one task to another task. The regulated approach prescribes that specific test cases need to be performed in a specific test-diagnose-fix task. Optimization of a test-diagnose-fix task can only be done within the context of the test-diagnose-fix task itself in the regulated approach.

Semi	Semiconductor equipment
Avionics	Airplanes
Space	Satellites
DoD	Department of defense systems
Drugs	Medical drugs
Comm	Communication equipment
Machines	Machine equipment

Table 8. Legend of organizational types

2.1.5 *Conclusions and discussion*

Different organizations use different integration and test sequences to develop or manufacture their products. The elements of an integration and test sequence are the same for all investigated organizations. The key business drivers of an organization can be characterized by Time, Cost and Quality. An integration and test sequence is specific to an organization, the product and the business drivers. As a result, it can be concluded that a strategy to obtain an integration and test sequence for a specific organization cannot be copied to another organization just like that. The business drivers of both organizations

should match.

Two types of test approaches are distinguished: *regulated* and *flexible* integration and test sequences. The main differences between a regulated and flexible integration and test sequence are 1) the positioning of test-diagnose-fix tasks and 2) the type of test strategy that is used for each of the test-diagnose-fix tasks.

Optimizing an integration and test sequence could be beneficial in terms of time, cost and quality. A flexible integration and test sequence allows many improvement opportunities. Among these are the selection of a different integration sequence, test sequence, test positioning strategy and test strategy per test-diagnose-fix task.

A regulated integration and test approach results in a fixed integration sequence. Selecting a different (better) sequence is difficult. The cost of changing the regulations should be taken into account. This is also the case for the test positioning strategy and the chosen strategies for specific test-diagnose-fix tasks.

Regulated integration and test sequences are easier to sequence and control, which is a benefit. All parties involved know from the start what to expect and what to do. The test content is known in advance for all test-diagnose-fix tasks. A flexible integration and test sequence allows for the use of more improvement techniques to obtain a better plan. The cost of this flexibility is the organizational effort that is involved with the improvement cycle.

A combination of a regulated integration and test sequences with known 'control' points in the sequence and flexibility in the intermediate phases could be a good combination for organizations that either try to increase the quality levels and maintain the short time-to-market or try to reduce the time-to-market while maintaining the product quality.

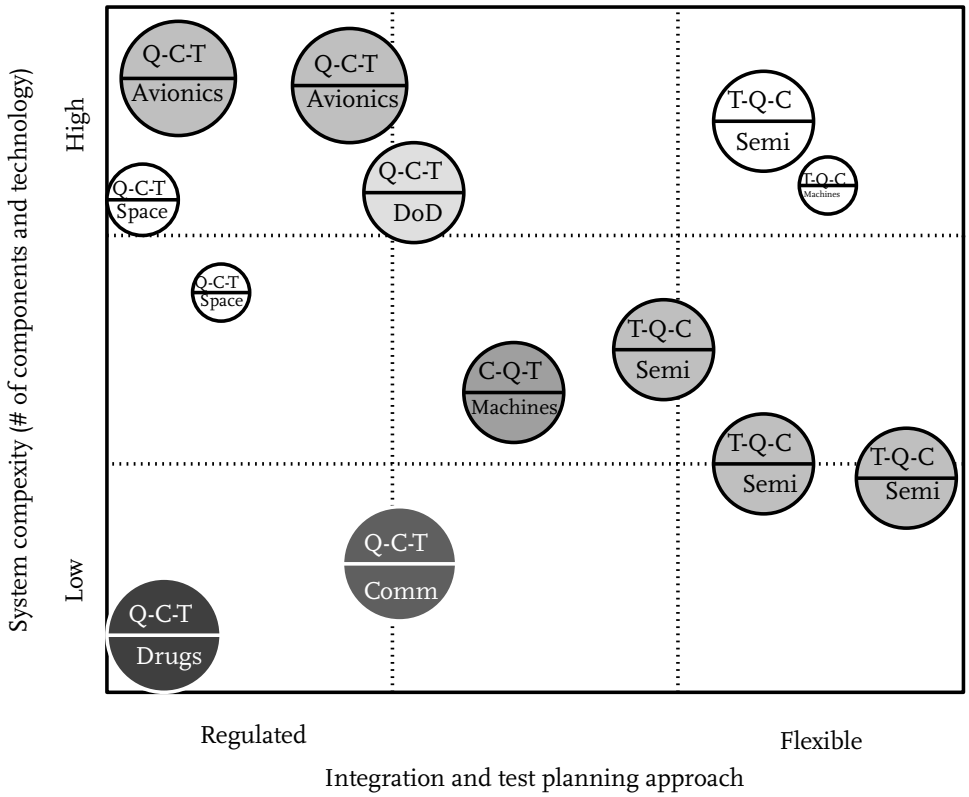


Figure 17. Overview of the visited organizations by system complexity and test strategy

2.2 INTEGRATION AND TEST TASKS

The investigation into integration and test plans in different organizations, described in the previous section, resulted in a set of five basic tasks that are used to form integration and test sequences. These five tasks are *develop*, *assembly*, *disassembly*, *copy* and *test-diagnose-fix*. This section describes each of these five tasks, as well as their inputs, outputs and properties.

The development task

The development task is the starting point in the integration and test sequence. The end point of a development task defines when the resulting component is ready for the next task in the sequence. How the development task is performed is not really of importance for integration and test sequencing. What is of importance is the remaining risk of a component when development is finished. The remaining risk determines what needs to be tested in the remainder of the integration and test sequence. The development duration and cost, φ_{dev} and C_{dev} respectively, are properties of the development task.

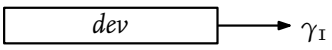


Figure 18. The development task

The development task is depicted in Figure 18 as a task with only a single component as output: γ_1.

The assembly task

The assembly task assembles two (combined) components and their interface(s) into a new combined component. The output of an assembly task of two components is a new combined component connected via the interface $X_{F\gamma_1,\gamma_2}$ between the two components. The risk of the combined component after assembly is the sum of the risk of the components and interfaces. Additional properties of an assembly task are the duration and cost that are involved with this task, respectively φ_{asm} and C_{asm}. The assembly task is depicted in Figure 19 as a task with three inputs (two components γ_1 and γ_2 and one interface $X_{F\gamma_1,\gamma_2}$) and a single output, the combined component.

The disassembly task

The disassembly task removes (disassembles) a combined component. The two components γ_1 and γ_2 are the outputs of this task depicted in Figure 20. The risk in the combined component is divided over the disassembled components

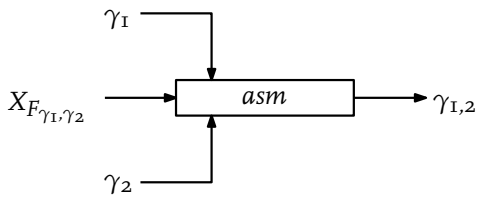


Figure 19. The assembly task

and their interface. The properties of a disassembly task are duration φ_{das} and cost C_{das}. The disassembly task is depicted in Figure 20 as a task with a single input, the combined component, and three outputs. The three outputs represent the combined component that is split up into two disassembled components and their interface.

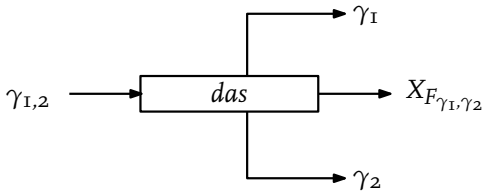


Figure 20. The disassembly task

The copy task

The copy task is specific for component that can be copied. Examples of components that can be copied are component designs and software components. Design (documents) and software components are components that can be copied, such that two identical versions of the same component are obtained. All faults present in the original component are present in the copy of the component. This is different for copies of mechanical and electrical components that can contain different faults, because each 'copy' is slightly different. The copy task can be used to describe integration and test sequences for software systems using a baseline integration strategy with parallel testing. Figure 21 depicts the copy task with a component as input and two equal components as output.

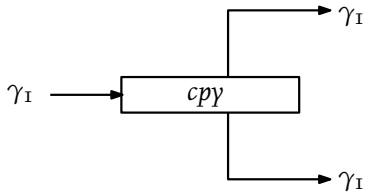


Figure 21. The copy task

The test-diagnose-fix task

The test-diagnose-fix task consists of a number of sub-tasks. The sub-tasks in the test-diagnose-fix task are: *test execution*, *diagnosis* and *fixing*. The sequence of sub-tasks is determined using a *test strategy*. The sub-tasks and elements of a test strategy are explained in detail in Section Section 4.2. Figure 22 depicts the test-diagnose-fix task that takes a component as input and outputs a (partly) tested, diagnosed and fixed component.

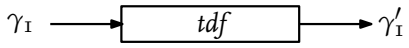


Figure 22. The test-diagnosis-fix task

Combined tasks

The five basic integration and testing tasks can be combined into sequences of integration and test-diagnose-fix tasks: integration and test sequences. Combinations of integration and testing tasks are often observed in industrial integration and test sequences. Four typical combinations, that are often observed, are described here.

The first combined task is a combination of two test-diagnose-fix tasks. The combination of two test-diagnose-fix tasks can be seen as a single test-diagnose-fix task. Subsequently, a single test-diagnose-fix task can be seen as two test-diagnose-fix tasks that are executed in sequence if more than one test case is executed in this test-diagnose-fix task.

The second combined task is a combination of an assembly task and a test-diagnose-fix task. This combination has been found in the industrial integration and test sequences that were analyzed. The industrial integration and test sequences contain activities that are described as 'integrate' the module γ_I and

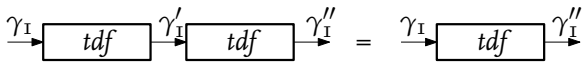


Figure 23. Combining two test-diagnosis-fix tasks

γ_2 with a duration that is much longer than the duration of the assembly task only. Integration of module γ_1 and γ_2 means assembly of γ_1 and γ_2, followed by a test-diagnose-fix task of the combined module $\gamma_{1,2}$. Figure 24 depicts the combination of an assembly and test task in a single task.

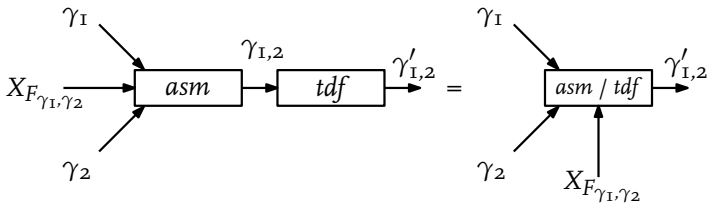


Figure 24. Combining an assembly task and a test-diagnosis-fix tasks

The third combined task is a combination of two or more assembly tasks. The combination of two up to n assembly tasks can be described as a single assembly task with $n + 1$ incoming components and a single combined component as output. A prerequisite of this combination is that all components that are to be assembled are available at the start of the combined assembly task. Figure 25 depicts the combination of two assembly tasks into a single assembly tasks with three modules and their interfaces as input.

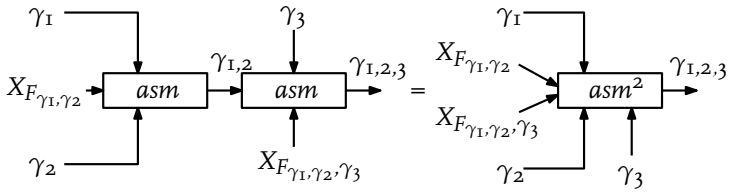


Figure 25. Combining two assembly tasks

The fourth combined task is a combination of a disassembly task and an

assembly task into a *replacement* task. Taking a component out of the system and placing the next version of the component back into the system, i.e. replacing a component, can be modeled using a single task this way. Figure 26 depicts the replacement task as a combination of a disassembly and an assembly task.

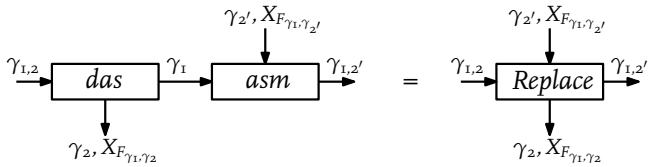


Figure 26. Replacing a component with a new version, a combination of a disassembly and assembly task

2.3 SYSTEM INTEGRATION AND TEST MODELS

The sequence of integration (assembly) and test-diagnose-fix tasks results in a complete and functional system

The result of an integration and test sequence is a *complete* and *functional* system. Components are integrated and tested in this sequence. The sequence of component integrations depends on the components that are present in the system and the possible interfaces between these components, since components that are not connected via an interface cannot be integrated together. Knowledge of the components and interfaces in the system is used by integration planners to determine an *integration sequence*. Test-diagnose-fix tasks are placed between (some of) the integration tasks afterwards.

Two models will be introduced in this chapter to support planning, analysis and improvement of integration and test sequences. The first model is a model of the system architecture usable for integration and testing: the *system integration model*. The second model is used to model the details of a single component or interface: the *system test model*. Together, these models are used for performance analysis of single test-diagnose-fix tasks and complete integration and test sequences.

2.3.1 *System integration model*

The system integration models describe a system. These models are used for sequencing and improvement in the integration and test planning method. The architecture of a system consists of components and interfaces. Both the components and interfaces are represented by the system integration model.

The system integration models should fulfill the input-output requirements of the integration and test-diagnose-fix tasks. The output of an integration and

test-diagnose-fix task is the input of the next task. Integration and test-diagnose-fix tasks are combined into integration and test sequences. For instance, an assembly task can be followed by another assembly task or a test-diagnose-fix task. A component in the system integration model that is the output of the assembly task should be of the same type as the input of the next task. System integration models should enable performance modeling of the integration and test sequence. Analyzing the performance of an integration and test sequence is one of our goals. Therefore, the integration models should support this goal. System integration models should support abstraction if a system contains many components and interfaces. Integration and test sequencing is a problem for large systems in particular. These systems consist of many components and interfaces. The integration and test models should support the combination of smaller models into large models, such that large systems can be modeled easily.

The system integration model A is defined as a five-tuple containing: models of the components in the system Γ, models of the interfaces in the system $\mathcal{X}_F$, the relation between the components and interfaces $R_{\gamma,\mathcal{X}_F}$, a so-called *layering* $\mathcal{L}$ and the objective and constraints of the integration and test sequence *Obj*. Thus, $A = (\Gamma, \mathcal{X}_F, R_{\gamma,\mathcal{X}_F}, \mathcal{L}, Obj)$, where Γ represents a set of component models γ. The elements of the system integration model are described in detail below.

The component model γ
A component in the system is modeled as a *system test model* D introduced in Section 2.3.2, and properties: $\gamma = (D, \varphi_{dev}, C_{dev})$. The system test model D describes the component in terms of test cases and fault states and is described in Section 2.3.2. The duration and cost of developing a component γ is modeled as φ_{dev} and C_{dev} respectively. The development duration is modeled as the duration between the start of the integration and test sequence and the moment that component development is ready. The end moment defines when the integration and test sequence can progress. The cost of development can also be modeled. This cost is added to the overall cost of the integration sequence. Often, the cost is set to 0, because only the cost of integration and test tasks is of interest when integration and test sequences are analyzed and compared and not the cost of developing a module that is just an offset in the total cost.

The interface model X_F
An interface is also modeled as a system test model. The duration and cost of developing the interface is also taken into account. The interface is modeled as a triple: $X_F = (D, \varphi_{dev}, C_{dev})$.

The interface risk is modeled explicitly. Even for interfaces with very low failure probabilities and risk, the risk needs to be modeled. The combined risk of many low risk interfaces results in an overall, system level, remaining risk that could be too high. The low risk interfaces are also modeled by explicitly modeling the interfaces. Note that a broad multi-disciplinary definition of

interfaces is considered.

Relationship between components and interfaces R_{γ,χ_F}
The relationship between components and interfaces is modeled as a set of triples that contain the models of two components and their interface $R_{\gamma,\chi_F} \subseteq \Gamma \times \mathcal{X}_F \times \Gamma$, which is a set of triples representing the connections that can be made between the components in the system. Two components can be connected by two or more interfaces. This is modeled either as separate relations or by combining the system test models of the interfaces into a single model. The interfaces between one component and many other components are modeled as separate relations.

Layering $\mathcal{L}$
The minimal set of models that is required to create integration and test sequences consists of the component model, the interface model and the model of the relationship. The model of the relationship between components is used to obtain the sequence. In this way, any combination of tasks can be made. The number of sequences obtained using these five tasks and the component models is infinite, because of the possibly infinite combinations of disassembly and assembly tasks, the copy task and the repeatable test-diagnose-fix tasks. Even if the copy and disassembly tasks are not taken into account and it is assumed that test-diagnose-fix are not repeated, then the number of possible sequences leading to a complete and tested system is huge for systems with many components and interfaces. The number and type of possible sequences can be constrained by using *layering*.

For this reason, a *layering* $\mathcal{L}$ in the system, consisting of sets of components Γ, is defined: $\mathcal{L} \subseteq \mathcal{P}(\Gamma)$. The relation between the components in the groups and between the groups are derived using the component-interface relation R_{γ,χ_F}. A set of components in a layering can contain a component that is also present in another set of components. In other words, sets of components can overlap. A layering is often used to describe a functional clustering opposed to the hierarchical or mechanical clustering. The physical components can contribute to one or more of these functional clusters. In this way, functional integration and test sequences can be derived from the system integration model.

Objectives and constraints *Obj*
The objective for an integration and test sequence describes whether the resulting system should be integrated and tested quickly (T), cheaply (C) or with a high quality (Q). The constraints of the integration and test sequence describe if the executed integration and test sequence should not exceed a fixed duration, a fixed cost level or a risk level. Objectives and constraints are both expressed in terms of time, cost and remaining risk. The objectives are expressed as a single objective function where time, cost and quality are weighted according to their relevance, when these objectives are used for optimization as in Chapter 5. The

time, cost and remaining risk objectives are described relative to each other, such that it is indicated which objective is most important. The constraints of an integration and test sequence determine the limits that may not be exceeded. A constraint on duration means that the sequence should be finished at a certain deadline. A constraint on cost means that a certain cost level may not be exceeded and a constraint on quality means that the integration and test sequence should lead to a certain quality level. Some examples of objectives for a single test-diagnose-fix task are described in Sections 4.2.1 and 5.3. The objectives of a complete integration and test sequence are described in a similar fashion.

The next section describes the details of the *system test model* that is used to model components and interfaces.

2.3.2 *System test models*

Components and interfaces in a system are modeled as a *system test model*.

Components and interfaces in a system are modeled as a *system test model*. A system test model describes a component or interface as a combination of possible fault states, test cases, the coverage of the test cases on the fault states and the properties of the fault states and test cases. In this section, the system test model is described in detail. The practical implication of every modeling element is discussed. The model completeness is described and stepwise improvement of the model is explained.

The test cases, fault states and their properties are modeled as a so called *system test model*. The *system test model* is a ten-tuple consisting of those elements that are relevant for a test-diagnose-fix task. The ten-tuple is derived from a basic system test model used by [Pattipati et al., Jan 1991] for sequencing diagnosis tasks. Boumen [Boumen et al., Jan. 2008] uses a system test model, based on the basic test model, for probabilistic test sequencing. The basic test model is defined as $D = (T, S, C, P, R_{ts})$. Our *system test model* D is defined as a ten-tuple $(T, S, (C_T, \varphi_T), (C_D, \varphi_D), (C_F, \varphi_F), (C_{AF}, \varphi_{AF}), P, I, U, R_{ts})$, where:

- T is a finite set of k tests.
- S is a finite set of l fault states.
- $C_T : T \rightarrow \mathbb{R}^+$ gives for each test in T the associated cost of performing that test.
- $\varphi_T : T \rightarrow \mathbb{R}^+$ gives for each test in T the associated duration of performing that test.
- $C_D : T \rightarrow \mathbb{R}^+$ gives for each test in T the associated cost of diagnosing the failed test.
- $\varphi_D : T \rightarrow \mathbb{R}^+$ gives for each test in T the associated duration of diagnosing the result of the test if the result is *fail*.

- $C_F : S \to \mathbb{R}^+$ gives for each fault state which is to be fixed the cost of fixing that fault state.
- $\varphi_F : S \to \mathbb{R}^+$ gives for each fault state in S the associated duration of developing a fix for the fault state.
- $C_{AF} : S \to \mathbb{R}^+$ gives for each fault in S the associated cost of applying the fix on the system under test.
- $\varphi_{AF} : S \to \mathbb{R}^+$ gives for each fix which is to be applied the associated duration.
- $P : S \to [0..1]$ gives for each fault state in S the *a priori* probability that the fault state is present.
- $I : S \to \mathbb{R}$ gives for each fault state in S the impact of the fault state if the fault state exists in the system under test.
- $U : S \to [0..1]$ gives for each fault state in S the uncertainty that the fault state is present, which is 1 if no test case has been executed that covers the fault state.
- $R_{ts} : T \times S \to [0..1]$ gives for each test t and fault state s the coverage of the test t on fault state s.

Impact can be a value relative to each fault state or a specific value related to a property of the system under test.

Fault states

Fault states describe *possible* faults in the system, including the associated failure probability and impact. Fault states are assumed to be independent of each other. Additional properties are modeled per fault state, like the uncertainty about the fault state (used for reliability qualification), the duration and cost of fixing the fault state and the duration and cost of applying the fix on the system under test. A set of fault states can be modeled as a single combined fault state with adjusted properties [Boumen, 2007] for the combined fault state, like for instance a higher failure probability. A broad definition of fault states is used when modeling a system. Fault states could be based on existing requirements, known failures from previous system releases, failure mode effect analysis (FMEA/FMECA) [Bowles, 19-22 Jan 1998], expert knowledge and even more detailed fault state sources like manufacturing failure databases or static code analysis tools for software systems.

Test cases

A test case is described in terms of its coverage on the defined fault states. The coverage of the test case on the fault states determines the capability of the test case on system level. Note that the detailed test procedure is a valuable source to determine the coverage of the test case. The other properties that are modeled for a test case are execution duration, execution cost, diagnosis duration and cost. A set of test cases can be modeled as a single test case with a higher coverage, a longer duration and higher cost.

The detailed test procedure is a valuable source to determine the coverage of the test case.

Coverage

The coverage is modeled as the probability that a test case t can detect a fault state s. The coverage of a test case on a fault state can be estimated by counting for a test case how many of the functions are covered by the test case. The number of covered functions is divided by the total number of functions in the system that this test case covers. The coverage is used in Chapter 4.3 to determine the reduction of the failure probabilities of the fault states that are covered by a test case when the test case *passes*.

Uncertainty

The uncertainty is defined as the so-called *subjective* uncertainty [Helton, 1997] and describes the lack of knowledge about the system. This lack of knowledge of the analyst concerns the analysis of the test-diagnose-fix task or integration and test task. The *stochastic* uncertainty, which is another definition of uncertainty that is often used, is a property of the test-diagnose-fix task itself and describes the variability of the behavior of the task. The stochastic uncertainty of a test-diagnose-fix task (or a complete integration and test sequence) is measured in terms of the performance indicators: duration, cost and remaining risk as defined in Section 4.1. The subjective uncertainty is used in the reliability test planning method in Section 4.3.

Risk

Our measure for product (or system) quality is risk. The risk that is present in a system is the sum of the risk of the fault states that are possibly in the system. The risk that a fault state is in the system is defined as the product of the probability that the fault state is in the system and the impact of the fault state when it is present: $R(s) = P(s)I(s)$ [Amland, 2000; Kaplan, August 1997; Pfleeger, 2000]. Since fault states are assumed to be independent, the risk of all fault states in the system can be added such that the system risk is obtained: $R = \sum_{s \in S} R(s)$.

The required probability can be estimated or obtained from historical data.

The required probability, a property of the product, can be estimated or obtained from historical data. The required impact value is not a property of the product. The impact of the presence of a fault state is estimated, in case the fault state occurs after the (integration) and test sequence is finished. If the product is delivered to customers once the integration and test sequence is finished, then the impact for customers is estimated. If the product is released for the next task in the integration and test sequence, then the impact for the remainder of the integration and test sequence is estimated.

The impact of the existence of a fault state can be estimated in terms of a physical property. The impact of the existence of the other fault states is then also estimated in terms of the same physical property. The combined risk in terms of this physical property can then be determined. For instance, if a system is modeled for overlay testing, then the impact of the existence of all fault states

in the system is modeled in terms of nanometers overlay. In this way, test cases can be selected that reduce the overlay risk as much as possible.

Example telephone system

As an example, a test model is defined for a common telephone. The telephone consists of a handset, a cable and a device. Each of these modules can contain a fault. The interfaces between the modules can also contain a fault. Hence, in total, 5 fault states are defined. Additionally, 6 test cases are defined covering the 5 fault states. A graphical view of the telephone is given in Figure 27.

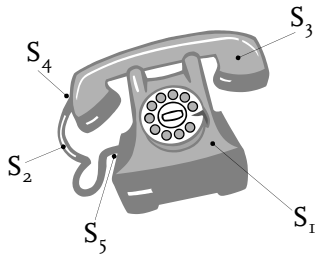


Figure 27. Telephone system

The set of five possible fault states in the telephone is: $S = \{s_1, s_2, s_3, s_4, s_5\}$.

- s_1 the device contains a fault
- s_2 the cable contains a fault
- s_3 the handset contains a fault
- s_4 the interface between the cable and the device contains a fault
- s_5 the interface between the handset and the cable contains a fault

The set of six tests is defined to cover these fault states is: $T = \{t_0, t_1, t_2, t_3, t_4, t_5\}$.

- t_0 tests the complete phone system
- t_1 tests the device
- t_2 tests the cable
- t_3 tests the handset
- t_4 tests the device and the cable
- t_5 tests the handset and the cable

A matrix representation of the system test model D, including the properties per fault state and test, is given in Table 9. The coverage of a test case on a fault state is placed in the matrix.

S / T	t_0	t_1	t_2	t_3	t_4	t_5	P	I	U	C_F	φ_F	C_{AF}	φ_{AF}
s_1	0.2	0.5	0	0	0.3	0	10%	7	1.0	2	2	1	1
s_2	0.2	0	0.5	0	0.3	0.3	10%	5	1.0	2	2	1	1
s_3	0.2	0	0	0.5	0	0.3	10%	3	1.0	2	2	1	1
s_4	0.2	0	0	0	0.3	0	10%	3	1.0	2	2	1	1
s_5	0.2	0	0	0	0	0.3	10%	3	1.0	2	2	1	1
C_T	3	1	1	1	2	2							
φ_T	6	2	2	2	4	4							
C_D	3	1	1	1	2	2							
φ_D	10	1	1	1	6	6							

Table 9. A test model for the telephone example

2.3.3 *System test modeling in practice*

The *system test model* can be obtained in two ways. The first approach starts with the available set of test cases. The second approach uses the available set of fault states. If the available set of test cases is taken as a starting point, then one single fault state 'the system contains a fault' is added to the model and all test cases have 'imaginary' coverage on this fault state. The remainder of the model (fault states and coverage) is obtained by making each test case unique, i. e. no two test cases may cover the same set of fault states.

The system test model can be obtained in two ways. The first approach starts with the available set of test cases. The second approach uses the available set of fault states.

A test case can be made unique by adding a fault state to the model that is covered by only one of the test cases in the set. The coverage of the other test cases on the new fault state should be estimated also. This process repeats until all test cases are unique. It is assumed that the current set of test cases does not contain test cases that are the same and it is assumed that the available set of test cases covers the system sufficiently.

The second approach starts with a set of fault states that must be covered by the test cases. These fault states typically represent the existing (customer) acceptance criteria or negated requirements. A requirement that is *not* met is called a negated requirement. For instance, the throughput requirement of a wafer scanner for example is 150 wafers per hour. The corresponding fault state is: the wafer scanner does *not* meet its 150 [wph] throughput requirement. Once the set of fault states is defined, test cases are added to the system test model until all fault states are covered. Additional specific test cases could be added for fault states with high failure probabilities, because it is more efficient to test fault states with high failure probabilities with specific test cases instead of test cases that cover a range of fault states [Boumen et al., Jan. 2008].

Model completeness and stepwise improvement

One could ask if the derived *system test model* is complete. At least the *relevant* fault states and test cases should be modeled. To answer this question, all *possible* fault states and *possible* test cases have been drawn as a large box covering the

four rectangles depicted in Figure 28. All possible fault states in the system are depicted along the y-axis, while all possible test cases are depicted along the x-axis. The rectangle marked as 1 in Figure 28 depicts the test cases that are modeled in the system test model. The fault states in rectangle 1 are all fault states that are covered by test cases in rectangle 1. In other words, rectangle 1 represents the system test model as it is currently known.

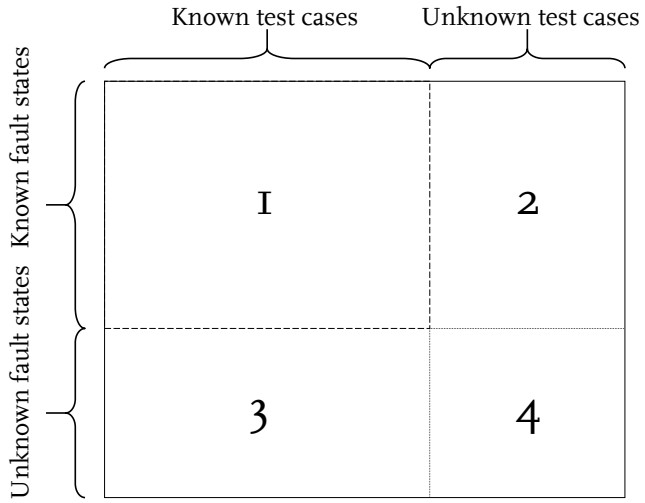


Figure 28. Model completion described using four rectangles

Rectangle 2 represents all test cases that could be in the system test model, but are not. The fault states covered by the test cases in rectangle 2 are (partly) also covered by the test cases in rectangle 1. Discovering which test cases could be added to the model could be beneficial for the performance of a test-diagnose-fix task. The next-best-testcase algorithm, described in Section 5.2, exploits the possibility to define new test cases based on the known fault states.

Rectangle 3 represents all fault states that are not covered by the known test cases of rectangle 1. The fault states in rectangle 3 represent unknown system risk and are therefore relevant if a certain remaining risk (quality) level is to be reached. The problem is that these fault states are out of sight of the modeler/tester. A simple two step approach to discover these fault states is: 1) add a fault state called 'remaining faults/risk' to the model and estimate what the coverage is of the known test cases on the fault states, 2) try to identify the specific coverage of the known test cases on the 'remaining faults/risk' by adding a new fault state to the model and transferring the coverage of the test case on the 'remaining faults/risk' fault state to the specific fault state. In this way, the coverage of the other test cases on the new specific fault state needs to be estimated. Continue this process until the risk of the 'remaining faults/risk' fault state is acceptable.

Rectangle 4 contains both the fault states that are unknown and also the un-

known test cases. The problem with the fault states in rectangle 4 is that these unknown fault states are to be covered by test cases that are also unknown. It is obvious that the set of fault states in rectangle 4 should be as small as possible. This can be obtained by reducing the unknown test cases in rectangle 2 and reducing the unknown fault states in rectangle 3.

A system test model should contain as many fault states and test cases as possible to ensure that all risk in the system is taken into account when selecting and executing test cases based on this model. Adding as many test cases as possible to the system test model has a similar impact.

A system test model should contain as many fault states and test cases as possible to ensure that all risk in the system is taken into account when selecting and executing test cases based on this model.

Tools and other system test model representations

A modeling tool has been developed in the TANGRAM project that supports the modeling steps in the method. The modeling tool, called LONETTE , also serves as a framework from which different analysis and improvement algorithms can be started and results can be analyzed. The model of the telephone example, modeled in LONETTE , is depicted in Figure 29. Additional properties are calculated automatically in LONETTE based on the model. These properties are for each test case: the covered risk, the failure probability and the gained information. For each fault state the risk involved with the fault state is calculated. How these properties are derived is explained in the subsequent chapters.

	Faultstates	1	2	3	4	5	6					
Tests		t0	t1	t2	t3	t4	t5	Probability (%)	Impact	Fix cost [euro]	Fix time [h]	Fault state risk
1	s1	0.2	0.5			0.3		10	7	2	2	0.7
2	s2	0.2		0.5		0.3	0.3	10	5	2	2	0.5
3	s3	0.2			0.5		0.3	10	3	2	2	0.3
4	s4	0.2				0.3		10	3	2	2	0.3
5	s5	0.2					0.3	10	3	2	2	0.3
	Test Time [h]	3	1	1	1	2	2					
	Test Cost [euro]	6	2	2	2	4	4					
	Diagnose Time [h]	10	1	1	1	6	6					
	Diagnose Cost [euro]	10	1	1	1	6	6					
	Repeatable	Yes	Yes	Yes	Yes	Yes	Yes					
	Test Risk	0.42	0.35	0.25	0.15	0.45	0.33					
	Fail Probability (%)	10	5	5	5	9	9					
	Information Gain	0.4564	0.2864	0.2864	0.2864	0.4275	0.4275					

Figure 29. System test model of a telephone in LONETTE

A different view of a system test model is a connected graph: a *test graph*. The fault states and the test cases are depicted as circles and diamonds respectively. The relations between the test cases and the fault states are depicted as edges. Figure 30 depicts the example telephone system. The thickness of the edges represents the coverage of the test case on the fault state. The grayness of the fault state is a measure for the failure probability of that fault state.

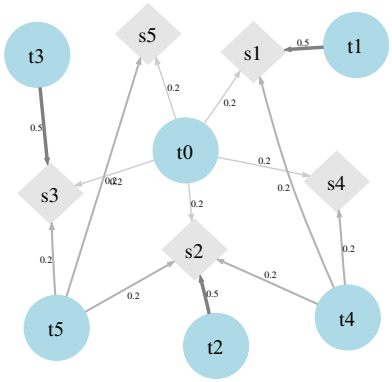


Figure 30. The telephone system depicted as connected graph. Circles describe test cases, diamonds describe fault states. The grayness of an edge describes the coverage of the test case on a fault state.

The test-fault state graph for a bigger model, used in one of the cases, is depicted in Figure 31. This test-fault state graph depicts a number of fault states that are covered well by the test cases, depicted with black edges, on the edge of the graph. Some fault states are, however, only partially covered by test cases. These are depicted in the center of the graph. A second example of a larger system test model is depicted in Figure 32.

The system test models, introduced in this chapter, are used in the next chapters for sequencing, planning and improvement of test-diagnose-fix tasks. The system test models can be used for a wide range of test problems, because the models contain only information that is relevant to testing and not the information that is specific to components of a single discipline. Moreover, the system test models contain information that a test engineer would consider implicitly when planning a test-diagnose-fix task. No new information is required. The known information is merely made explicit.

The system test models can be used for a wide range of test problems, because the models do not contain that is specific for components of a single discipline.

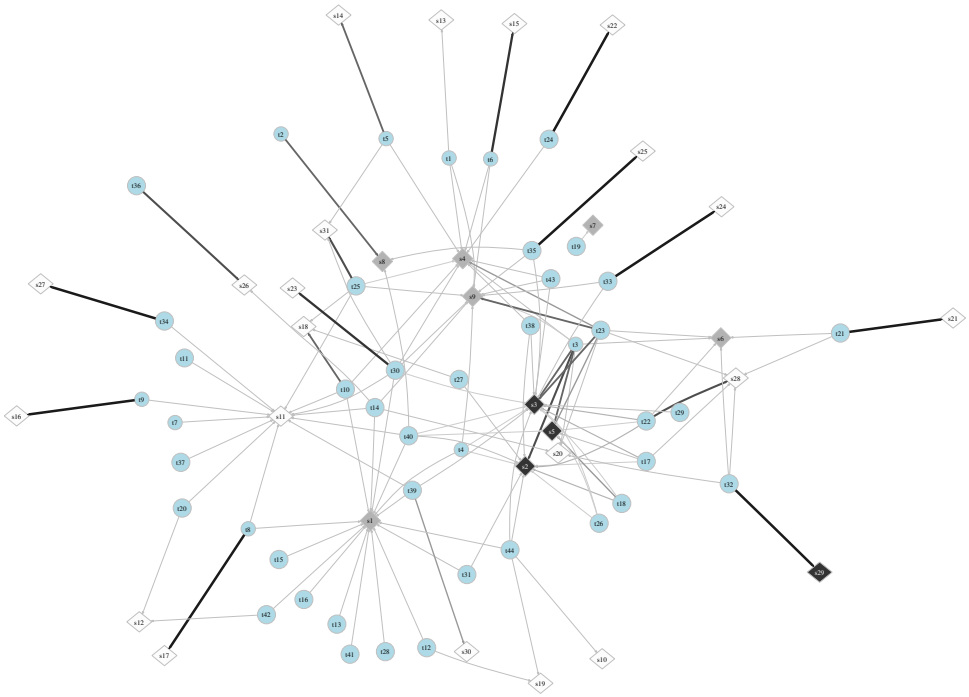


Figure 31. Graph of system test model that is used for regression testing of an ASML sub-system

Figure 32. Large graph of system test model that is used for alpha testing of an ASML software release

3 INTEGRATION AND TEST SEQUENCING

The goal of integration sequencing is to form a sequence of integration tasks, that results in a complete system. Integration and test sequencing positions test-diagnose-fix tasks between integration tasks, such that a combined integration and test sequence is formed. The start moment and duration of these test-diagnose-fix tasks is determined from the position in the integration sequence. The components that are integrated at a certain moment in the sequence determine which faults can be present in the system and can be tested. The performance of the resulting integration sequence is measured in terms of time (duration), cost and remaining risk (quality). An overview of the sequencing step is given in Figure 33. An integration sequence is created in step (2.1) of the method using the system integration model (1.A) and an integration strategy (2.A). Then, test-diagnose-fix tasks are positioned in this integration sequence in step (2.2) using a test positioning strategy (2.B). The integration strategy and test positioning strategy are inputs for the integration and test sequencing step. Both are explained below.

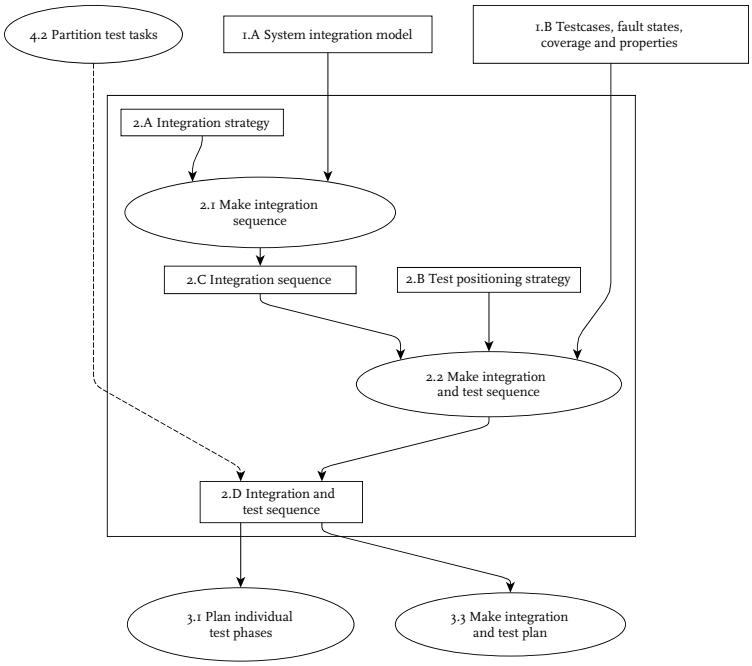


Figure 33. Overview of the sequencing step

3.1 INTEGRATION STRATEGIES

An integration strategy is used to create an integration sequence.

An *integration strategy* (2.A) is used to create an integration sequence. Common examples of integration strategies are *bottom-up integration*, *top-down integration* and *big-bang integration* [Beizer, 1990]. Other examples of integration strategies originate from concurrent engineering and baseline development. Seven integration strategies are described in this section. Note that, for large systems, integration strategies can be combined, such that two (or even more) integration strategies are used in a single integration sequence.

3.1.1 *Bottom-up integration*

The bottom-up integration strategy integrates components starting with the lowest level components in the system. The lowest level components are integrated first. Then, the combined components are integrated into modules that are further integrated into sub-systems. This process continues until the complete system is formed. A benefit of this method is that the system functionality and quality is gradually built up. No models to simulate components during integration need to be developed (as opposed to the the top-down integration strategy). A disadvantage of this strategy is that it is not known until the last integration task whether the complete system can actually be integrated, i. e. it is unknown if the components actually fit together and can perform the system level requirements until the last moment. Some of the interfaces are formed late in the process. If these interfaces contain faults, these faults are discovered, diagnosed and fixed late in the process, resulting in additional cost and time being spent very late in the process.

3.1.2 *Top-down integration*

The controlling component is developed first in the top-down integration strategy. Executable simulators or stubs of lower level components are integrated with the controller component, such that a system consisting of all components is formed early in the project. The simulated components, sometimes called *stubs*, are replaced with realizations of the components when these become available. This process continues until the complete system consists of realizations of components. An advantage of a top-down integration strategy is that a (simulated) complete system is available very early in the integration sequence. This fully functional system can be used from that point on to perform system level test cases. These system level test cases are used to qualify the interfaces with minimal support from the simulated components. A disadvantage of the top-down integration strategy is that simulated components or stubs need to be developed first. Additional effort is spent on developing and maintaining these stubs. This stub development activity can become critical in the integration sequence. The trade-off between spending additional modeling effort and the benefit of finding interface problems earlier is often difficult to make and

A stub replaces the real component in the system. A stub has typically limited functionality.

system specific. A case study where this trade-off analysis has been investigated is performed within the TANGRAM project and described in [Braspenning et al., 2007].

3.1.3 *Big-bang integration*

In a big-bang integration first all components are developed, while none of the components are integrated. Integration of all components is performed in a single big-bang integration. After the big-bang integration, the system is complete. The difference with the top-down integration is that no simulated components are used. An advantage of this integration strategy is that no long, multistage, integration sequence is required that integrates all components into a complete system. The integration duration is short. A disadvantage of this strategy is that this approach only works if the quality of each of the components is high. The quality of the interfaces between the components is high and limited risk is introduced when these interfaces are created.Components or interfaces of less quality result in faults that are found *after* the big-bang integration when the system is completely integrated. Diagnosing and fixing these faults is therefore costly and on the critical path to system delivery.

3.1.4 *Concurrent engineering*

The goal of concurrent engineering is that the components in the system are developed as much as possible in parallel and independent manner. The resulting integration sequence contains many assembly tasks in parallel. A benefit of concurrent engineering is that the development and integration cost and effort is spent in a shorter time frame, because of parallelism in the sequence. In general, it can be stated that development and integration should be performed in parallel, within the cost limit, because this results in the shortest possible duration. A disadvantage of concurrent development and integration is that more resources and cost are spent on integration and testing per time unit.

3.1.5 *Standard enforced strategy*

An integration and test strategy enforced by the department of defense framework results in an integration and test sequence with a standardized form.

This integration strategy is enforced by a standard, framework or policy. An overview of this type of integration standards, frameworks and policies can be found in [Stavridou, 1997]. Integration strategies based on standards or policies are often found in large organizations, where sub-systems are developed by different companies or organizational units. See Section 2.1 for the characteristics of some of the organizations using a standard enforced strategy. An advantage of a standard enforced strategy is that all parties involved know what the strategy will be and how changes to the sequence are dealt with. Communicating a change in the sequence is relatively easy. A disadvantage of this strategy is that faster or cheaper, integration sequences are not considered because they do not conform to the standard.

3.1.6 *Baseline development*

A baseline development integration strategy is often used in software development, where the entire code base is available from a previous release in a configuration management system.

Baseline development integrates components in a baseline system. A complete baseline system exists before integration of new components can be started. A previous version of the component is replaced with a newly developed component. A baseline development integration strategy is often used in software development, where the entire code base is available from a previous release in a configuration management system. The old code is replaced (disassembled and assembled) by overwriting the old code by new code. This strategy is also followed for the development of a new type of wafer scanner provided that it is based on a previous system type. The previous system type is built first. Then, new components replace existing components in the system and parts of the performance test cases are re-executed. An advantage of the baseline development strategy is that integration into the baseline is a natural point to perform quality checks. In this way, the quality of the baseline is protected. The strategy is simple to explain and to follow. A disadvantage of this strategy, seen within ASML, is the way of working that is more or less fixed. For instance, a late delivery of a single component to the baseline might not be optimal when the component contains much risk. An incremental delivery, where the high risk component is split up into smaller low risk parts, might be better for the integration sequence as a whole. A baseline development strategy should allow deliveries of partial components, such that the high risk component can be delivered earlier.

3.1.7 *Optimal integration*

An optimal integration strategy, based on an assembly by disassembly algorithm, is described in [Boumen et al., 2006a]. The optimal integration sequencing algorithm determines all possible sequences in which the complete system can be *dis*assembled into single components. The sequence that disassembles the system with a minimal duration, cost or remaining risk is selected. This optimal disassembly sequence is inverted such that the optimal assembly sequence is obtained. The algorithm uses a model of the components and interfaces as input, which is very similar to the model described in Chapter 2. The main benefit of this strategy is that all possible (dis-)assembly sequences are considered and the best sequence is chosen. An optimal solution can be calculated using an algorithm that is executed automatically with every change in the delivery dates of components. The disadvantage of the algorithm is that only systems of limited size can be handled. The algorithm can calculate all possible (dis-)assembly sequences for systems of limited size. A heuristic is required if the system contains too many components, resulting in sub-optimal solutions, although, sub-optimal solutions still perform better than integration sequences that are derived manually for the cases described in [Boumen et al., 2006a]. Moreover, these sub-optimal solutions can still be derived automatically when the inputs change.

In practice, combinations of integration strategies are used. A system consisting of three aggregation levels, system, sub-system and component level, could be integrated using a big-bang combined with a bottom-up integration strategy. The components could be integrated into sub-systems using a bottom-up integration strategy and the sub-systems could be integrated using a big-bang integration strategy. Combinations of integration strategies are also used for different aspects in the system. An example of this combination was seen in the development of a new wafer scanner platform at ASML. A number of prototype systems were built when a new wafer scanner platform was developed. Each prototype system had a specific focus. For instance, one prototype system was built to prove that the overlay requirements are met. For this purpose, a big-bang integration was applied, followed by a single system level qualification phase and the overlay qualification test. Another prototype system was built to test the functional performance and reliability. Components, modules and sub-systems were integrated using a bottom-up integration strategy. Specific functional and reliability test cases were performed to qualify the functional and reliability requirements. These test cases were planned in between the assembly tasks that followed from the bottom-up integration strategy. It can be seen that the integration and test sequences of this new wafer scanner platform utilized several integration strategies. Strategies that position test-diagnose-fix tasks between the integration tasks are discussed in the next section.

3.2 TEST POSITIONING STRATEGIES

A *test positioning strategy* is used for integration and test sequencing to determine where test-diagnose-fix tasks are placed between the other tasks. The *test positioning strategy* determines when risk is reduced (by testing, diagnosing and fixing), while an *integration strategy* determines how components are assembled i. e. risk is built up. Four test positioning strategies are described below.

The test positioning strategy determines when risk is reduced (by testing, diagnosing and fixing), an integration strategy determines how components are assembled i. e. risk is built up.

3.2.1 *The 'maximum integration progress' test positioning strategy*

The 'maximum integration progress' test positioning strategy places test-diagnose-fix tasks between the other tasks when test time is available between the previous task and the next task. Figure 34 describes a small part of an integration and test sequence as a Gantt chart. Test-diagnose-fix task tdf_{γ_2} is a test-diagnose-fix task that is positioned in the available time window between the development of component γ_2 and the assembly task $asm_{\gamma_1-\gamma_2}$. No test-diagnose-fix task is positioned for component γ_1 and the risk of component γ_1 is reduced in the last test-diagnose-fix task $tdf_{\gamma_1-\gamma_2}$. An advantage of the 'maximum integration progress' test positioning strategy is that integration is given priority, such that the complete system is available as soon as possible, while still some risk is reduced by testing. A disadvantage of this strategy is that the risk of some components is not tested until late in the sequence. For example,

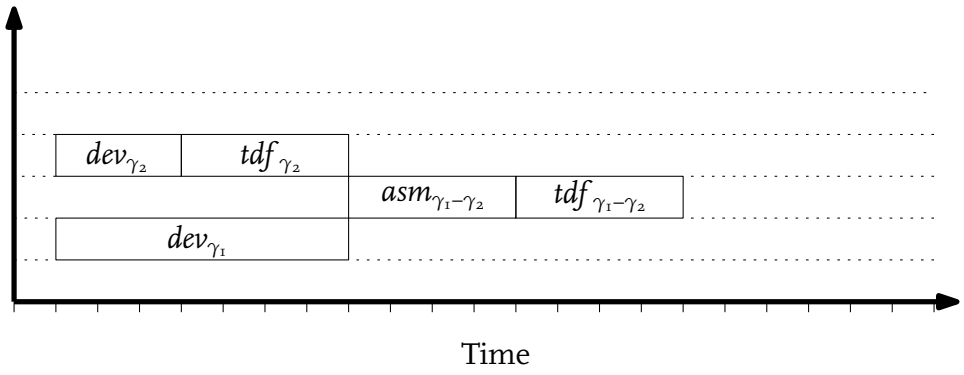


Figure 34. Gantt chart of the 'maximum integration progress' test positioning strategy

the risk in component γ_1, in Figure 34, is transfered to test-diagnose-fix tasks on a higher level. This strategy could result in a large number of test cases that still need to be executed once the system is completely integrated, because that is the first moment that these test cases can be executed. Components with the highest risk that are on the critical path are not tested early in the process if this test positioning strategy is applied.

3.2.2 *The 'test all' test positioning strategy*

Solving problems as soon as possible is the cheapest solution, while leaving problems for later test-diagnose-fix tasks results in an increase of the test duration.

The 'test all' test positioning strategy places a test-diagnose-fix task after each develop, assembly and copy task. The goal of each test-diagnose-fix task is to reduce the risk in the system as much as possible. This way, problems are solved as soon as possible, which is the cheapest solution because diagnosis and fix cost are spent early in the sequence. A disadvantage of this test positioning strategy is that all components are always thoroughly tested even if the risk involved is minimal. An advantage of this test positioning strategy is that it is easily communicated and planned. Figure 35 depicts a resulting Gantt chart of a 'test all' test positioning strategy. Note that the duration of the test-diagnose-fix tasks does not need to be equal for all tasks.

3.2.3 *The 'fault state once' test positioning strategy*

The 'fault state once' test positioning strategy plans a test case that covers a fault state after the last introduction of this fault state, i. e. each fault state is tested once. A disadvantage of this method is that the overall risk in the system can increase rapidly when many reoccurring fault states exist, because risk reduction is done late in the process. An advantage of this strategy is that test cases are only executed once, preventing double execution of the same test cases. Detailed information about the fault states in the system and when these fault states are introduced is required for this strategy. Moreover, it should be known which test

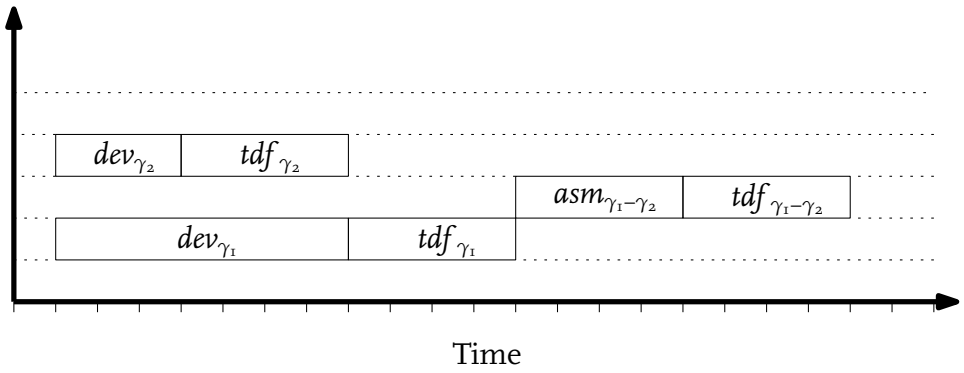


Figure 35. Gantt chart of the 'test all' test positioning strategy

cases cover which fault states, such that it can be determined when test cases are allowed to be executed.

3.2.4 *The 'risk profile' test positioning strategy*

A final qualification (test-diagnose-fix) task is often the last task in an integration and test sequence. The system risk after integration determines what the duration is of this final qualification phase. Testing reduces this failure probability and by this the risk is reduced. Figure 50 in Chapter 4 depicts the relation between the failure probability of a system and the test duration. It shows an increase of the duration of a test-diagnose-fix task with an increase of the failure probability. The goal of the 'risk profile' test positioning strategy is to reduce the risk in the integration and test sequence such that a certain risk level is reached and the final qualification phase is as short as possible. The 'risk profile' test positioning strategy plans a test-diagnose-fix task if the risk in the system has reached a certain upper risk limit. Risk is reduced, by testing, until a certain lower risk limit is reached. A more advanced strategy uses a 'risk profile' where the upper and lower risk limits are described as function of time. The upper risk limit in the beginning of the integration sequence could be set to a higher level than the risk limit at the end of the sequence. The same approach is followed for the lower limits. Figure 36 illustrates the risk profile that 'bounces' between the upper and lower risk limits that are both functions of the test duration.
An integration strategy and a test positioning strategy are used to create an integration and test sequence in steps 2.1 and 2.2 of the integration and test planning method.

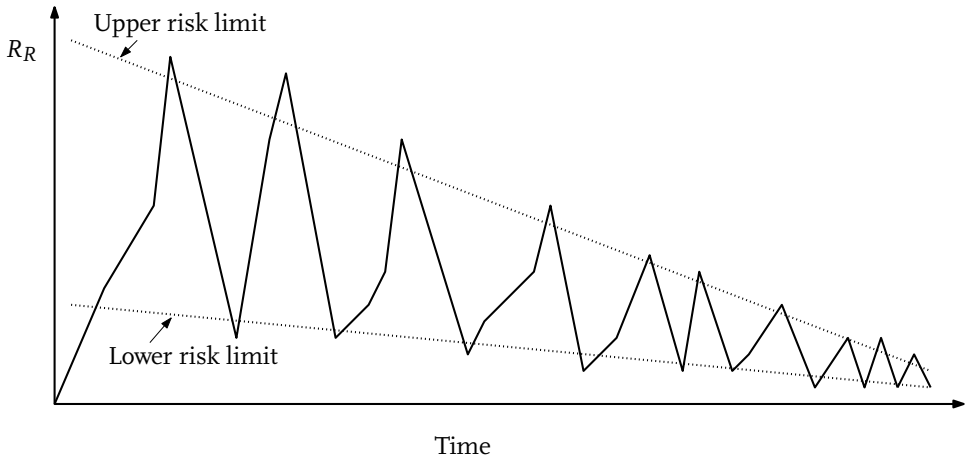


Figure 36. A 'risk profile' test positioning strategy depicted

3.3 INTEGRATION SEQUENCING

An integration sequence is created in this step, using an integration strategy (2.A) and a *system integration model* (1.A). Integration sequences can be created by hand or using integration sequencing algorithms like the algorithm proposed in [Boumen et al., 2006a]. The number of *possible* integration sequences is large for reasonably sized systems. Therefore, it is highly unlikely that the optimal integration sequence is chosen by hand. Next to that, changes are bound to happen and the integration sequence should be changed accordingly. This causes the integration sequence to become gradually less optimal. An algorithmic approach in general leads to integration sequences that are faster, cheaper or reduce more risk.

An integration sequence is described by a graph $G^I = (V, E)$, where the vertices contain one of the four integration tasks, (develop, assembly, disassembly and copy), that were previously introduced. The edges describe a precedence relation between tasks. The algorithmic approach to integration sequencing has been described in detail in [Boumen, 2007]. Many manual approaches exist, most based on expert knowledge or a standard way of working in a specific organization. In general, it could be stated that the manual approaches lead to good first integration sequences for a newly developed product. Changes to the manually derived sequence, however, are difficult to justify and result in many difficult discussions between integration experts. Comparing and analyzing these integration and test sequences such that the best sequence could be chosen is necessary, but is seldom done.

3.4 INTEGRATION AND TEST SEQUENCING

Integration and test sequencing places test-diagnose-fix tasks in between the integration tasks (development, (dis)assembly and copy) according to a test positioning strategy. Integration sequencing and test sequencing are often executed in a single step by experts. A combined integration and test sequence is formed based on an integration and test sequence from a previous project or using a *template* sequence. A *template* sequence describes an integration and test sequence of a particular form that is often re-used in an organization. The integration and test sequencing algorithm proposed in [Boumen, 2007] also combines the integration sequencing and test positioning steps in one step. Heuristics must be used for larger systems, because of the computational complexity of the algorithm. Practical methods rely on template integration and test sequences and the algorithmic approach relies on heuristics. The practical use of sub-optimal template sequences and the lack of optimal algorithms justifies that the integration and test sequencing process is split up into a two step process: first integration sequencing followed by the positioning of test-diagnose-fix tasks. Integration sequencing is performed using the cited integration and test sequencing algorithm or using the integration strategies that are described in this chapter. Positioning test-diagnose-fix tasks is performed using the test positioning strategies that are described. The result of the sequencing step, described in this chapter, is an integration and test sequence: a graph, where the vertices contain development, assembly, disassembly, copy or test-diagnose-fix tasks and the edges describe the relations between these tasks.

The practical use of sub-optimal template sequences and the lack of optimal algorithms justifies that the integration and test sequencing process is split up into a two step process: first integration sequencing followed by the positioning of test-diagnose-fix tasks.

The timing of the tasks in the integration and test sequence and the dependencies determine the overall duration of the integration and test sequence. The duration and cost of each task are estimated. The duration of the integration and test sequence is calculated by summing up the durations of all task on the critical path. The cost of the integration and test sequence is determined by summing up the cost of all tasks. The estimated timing of the test-diagnose-fix tasks is used as input for the detailed planning step that is explained in the next chapter.

4

INTEGRATION AND TEST PLANNING

The expected duration, cost and remaining risk of a test-diagnose-fix task depends on the selected test-diagnose-fix strategy and the possible faults that are present in the system under test.

The previous sequencing chapter describes how a sequence of integration and test tasks can be formed. The duration and the order of tasks defines what the estimated duration, cost and remaining risk of the entire integration and test sequence is. However, the duration of individual test-diagnose-fix tasks depends on the possible faults that are present in the components and interfaces and the selected test strategy. A detailed sequence of test, diagnose and fix tasks is cre-

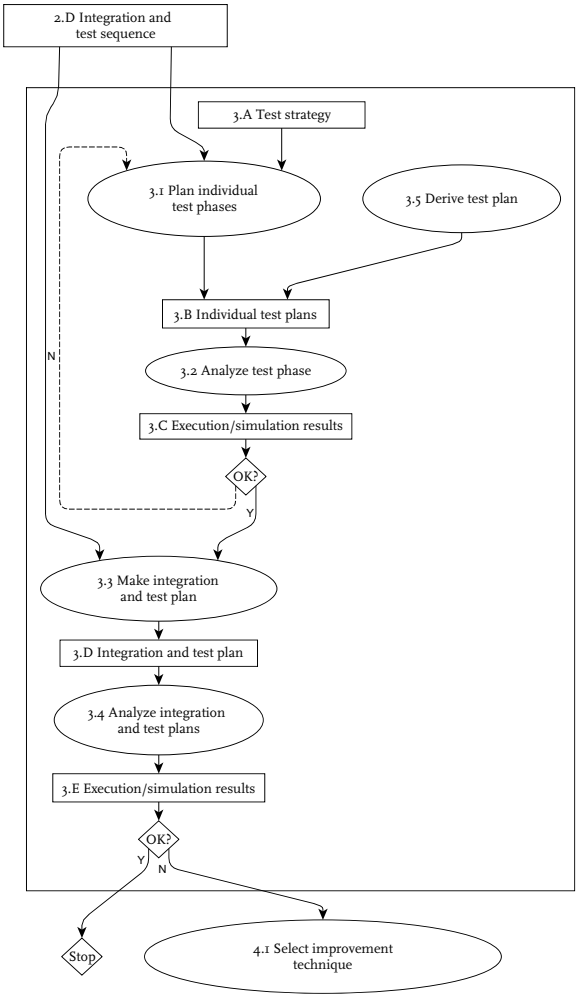


Figure 37. Overview of the planning step

ated in this planning step for each test-diagnose-fix task. The duration, cost and remaining risk of the resulting test-diagnose-fix sequence is determined and analyzed. A different test strategy can be selected when the analysis results do not satisfy the constraints. An overview of the planning step is given in Figure 37.

The resulting duration, cost and remaining risk for all test-diagnose-fix tasks in the integration and test sequence are fed back into the integration and test sequencing process. The performance of the integration and test sequence is then again analyzed in step (3.4). The integration and test planning step is finished, if the results are according to the objectives and constraints of the integration and test sequence. Otherwise, a number of improvement techniques, described in the next chapter, can be applied.

This chapter contains three sections. The first section describes the details of the planning and analysis of a single test-diagnose-fix task. Section 4.3 describes the modeling, planning and analysis of reliability test-diagnose-fix tasks. Section 4.1 describes step (3.4) of the method, the planning and analysis of an integration and test sequence.

4.1 ANALYZING INTEGRATION AND TEST SEQUENCES

This section is based on the paper 'Integration and test strategies for semiconductor manufacturing equipment' [I. de Jong et al., 2006] and defines the key performance indicators (KPI) of an integration and test sequence and how these KPI can be analyzed.

Currently, integration and test sequences are created manually based on expert knowledge. Defining an integration and test sequence can be an easy task for small mono-disciplinary systems. Experts can manually adapt integration and test sequences such that time-to-market and cost are taken into account. Comparing sequences is done using known mono-disciplinary criteria, like the amount of code coverage for software systems and the number of tested inputs and outputs for electronic boards. Mono-disciplinary methods often cannot be used to compare integration and test sequences for large multi-disciplinary systems. Therefore, integration and test sequences are still developed by experts, because methods to create and evaluate integration and test sequences of multi-disciplinary systems hardly exist. In this section, we present a method to describe, analyze and evaluate multi-disciplinary integration and test sequences. The structure of this section is as follows. First, the key performance indicators of an integration and test sequence are described in Sub-section 4.1.1. Sub-section 4.1.2 defines how these key performance indicators can be measured over time and for many (simulated) executions of integration and test sequences. The results of a case study related to a performance measurement of the integration and test sequence of an XT:850E wafer scanner are discussed in Sub-section 4.1.4.

4.1.1 *Key performance indicators of an integration and test sequence*

Integration and test sequences should be compared using criteria applicable to a variety of systems. The high level criteria (key performance indicators) characterizing integration and test sequences are:

- Φ: Total integration and test duration, defined as the time from the start of the integration and test task until the moment the stop criterion is met.
- C: Integration and test cost, defined as the sum of the costs of all assembly, disassembly, copy and test-diagnose-fix tasks. The cost of developing components can also be taken into account.
- R_R: Remaining risk, which is our measure for quality, defined as the risk that remains in the system when the test stop criterion is met. The risk for each possible fault is calculated using Equation 4.19 in Sub-section 4.2.1.

4.1.2 *Analyzing integration and test sequences*

The key performance indicators, defined above, are used to analyze and compare integration and test sequences with each other. The duration of an integration and test sequence is calculated by summing up the duration of the tasks on the critical path. The cost of an integration and test sequence is calculated by summing up the cost of all tasks in the sequence. The remaining risk is calculated using the failure probability of the remaining faults in the system and the impact of these fault states. Consequently, the risk at any point in time can be calculated also. The key performance indicators are illustrated by an example telephone system consisting of three modules: a handset, a cable, a device and the interfaces between handset and cable and cable and device. Figure 27 in Sub-section 2.3.2 is a graphical overview of the telephone system.

The first integration and test sequence, depicted by Figure 38, is an integration sequence where each assembly task is followed by a test-diagnose-fix task. The stop criterion of each test task is $R_R = 0$, meaning that testing stops when all remaining risk is removed.

The key performance indicators for this integration and test sequence can be derived as follows:

$$\Phi = \Phi_{asm_1} + \Phi_{tdf_1} + \Phi_{asm_2} + \Phi_{tdf_2} \tag{4.1}$$

$$C = C_{asm_1} + C_{tdf_1} + C_{asm_2} + C_{tdf_2} \tag{4.2}$$

$$R_R = 0 \tag{4.3}$$

The duration and cost of developing or manufacturing the individual components are not taken into account. The assumption is that all components are available when integration and testing is started. A *risk profile* of the integration

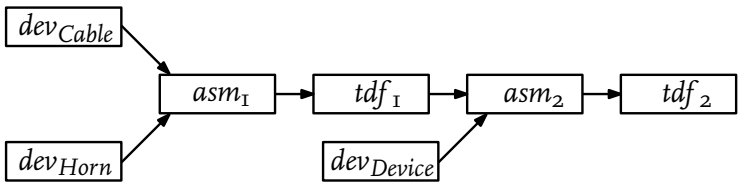


Figure 38. Example integration and test sequence 1

and test sequence is depicted in Figure 39. This risk profile depicts the risk as a function of time. At $t = 1$ the risk increases with 1 risk unit per developed module. At $t = 2$ additional risk is introduced by the assembly of the cable and the handset. The reason for this risk are the interface faults introduced by assembling the cable and handset. At $t = 3$ testing reduces the risk of the cable, handset and interface between the cable and handset to 0. The remaining risk in the system at that point is 1 risk unit from the device. At $t = 4$ risk is again introduced due to the interfaces between device and rest of the phone. This risk is reduced by the test task at $t = 5$. The remaining risk after test task tdf_2 is 0.

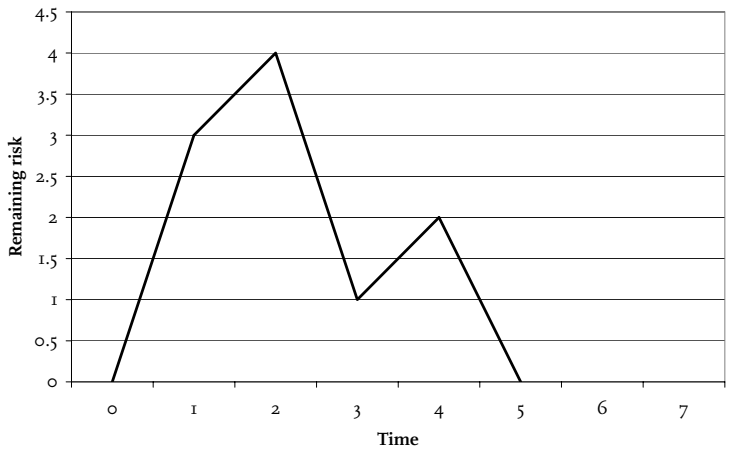


Figure 39. Risk profile of the first integration sequence

In a second integration and test sequence, each module is tested first. Every assembly is tested also. The stop criterion for each test task is now $R_R = 20\%$ of the initial risk of the test task. So, 80% of the risk is reduced with each test task. Figure 40 depicts this strategy. The risk after each task is denoted on the edges between the tasks. Now the key performance indicators for the second

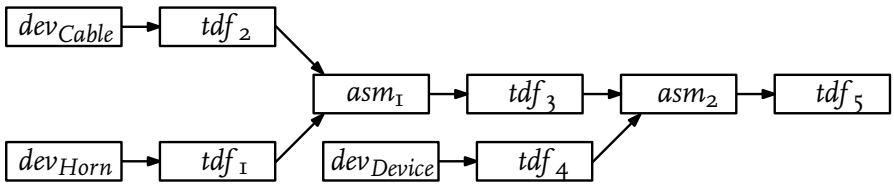


Figure 40. Example integration and test sequence 2

integration and test task can be derived using Equations (4.4), (4.5) and (4.9).

$$\Phi = \max((\max(\Phi_{tdf_1}, \Phi_{tdf_2}) + \Phi_{asm_1} + \Phi_{tdf_3}), \Phi_{tdf_4}) + \Phi_{asm_2} + \Phi_{tdf_5} \tag{4.4}$$

$$C = C_{tdf_1} + C_{tdf_2} + C_{asm_1} + C_{tdf_3} + C_{tdf_4} + C_{asm_2} + C_{tdf_5} \tag{4.5}$$

$$\begin{aligned} R_R &= R_{Rtdf_5} && (4.6)\\ &= 0.2R_{Rasm_2} \\ &= 0.2(R_{Rtdf_3} + R_{Rtdf_4} + R_{R\mathcal{X}_{FC,H,D}}) \\ &= 0.04R_{Rasm_1} + 0.04R_{RDevice} + 0.2R_{R\mathcal{X}_{FC,H,D}} \\ &= 0.04(R_{Rtdf_1} + R_{Rtdf_2} + R_{R\mathcal{X}_{FC,H}}) + 0.04R_{RDevice} + 0.2R_{R\mathcal{X}_{FC,H,D}} \\ &= 0.04(0.2R_{RHorn} + 0.2R_{RCable} + R_{R\mathcal{X}_{FC,H}}) && (4.7)\\ &\quad +0.04R_{RDevice} + 0.2R_{R\mathcal{X}_{FC,H,D}} \\ &= 0.008R_{RHorn} + 0.008R_{RCable} + 0.04R_{R\mathcal{X}_{FC,H}} && (4.8)\\ &\quad +0.04R_{RDevice} + 0.2R_{R\mathcal{X}_{FC,H,D}} \end{aligned}$$

The corresponding risk profile is depicted in Figure 41. Note that for this example it is assumed that the components in the system are independent of each other, such that the risk can be added in Equation (4.9).

Again this is a fairly straightforward derivation of the key performance indicators. The complexity of the derivation increases with the increase of the system size. Additionally, the assumption is that the values for each of the key performance indicators are deterministic, which is not the case for testing in most cases. The test process is a stochastic process, because of two reasons: the actual set of faults in the system under test can only be estimated, but is not known in advance (otherwise testing would not be necessary). The second reason is that the selected test strategy, described in Sub-section 4.2.1, influences the duration, cost and remaining risk of the test-diagnose-fix task.

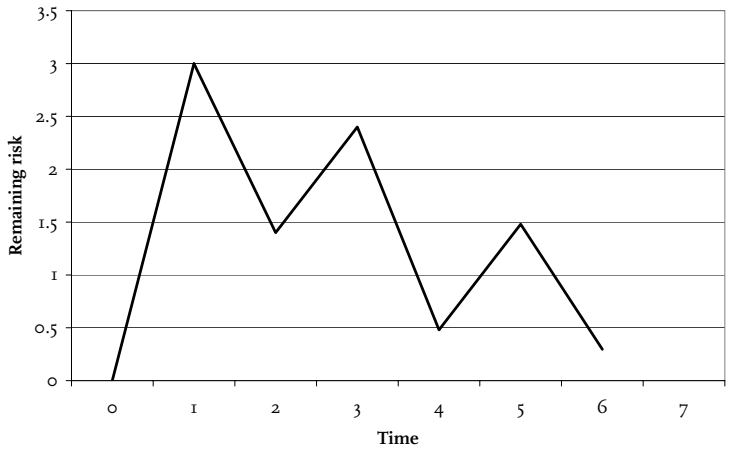


Figure 41. Risk profile of the integration sequence of Figure 40

4.1.3 *Analysis of integration and test strategies*

The result of a single simulation of an integration and test sequence is an *integration and test profile*. This profile consists of profiles for risk and cost as function of the integration and test time. These profiles can be used to analyze a single integration and test execution. High peaks in risk become evident and the integration and test sequence can be adjusted to minimize these high peaks if required. Another way of measuring and analyzing key performance parameters of an integration and test task is by simulating numerous executions of an integration and test sequence. The actual faults in the system under test are randomly selected for each simulation. Each simulation results in a total test duration, cost and risk, and an integration and test profile. A histogram of the key performance indicators describes what the minimal, maximal and expected values for the KPI are. The risk profile, cost profile as function of time and a histogram of the test duration are illustrated by the case study described in Subsection 4.1.4.

4.1.4 *Case: Analysis of an integration and test sequence of the XT:850E wafer scanner*

In this case study, the performance of a complete integration and test sequence is measured and analyzed. The integration and test sequence is derived from the actual Microsoft Project plan of the XT:850E wafer scanner released a few years ago. The Microsoft Project plan was not made with this case in mind but still it was easy to convert the tasks in the plan into an integration and test sequence. The only required adjustment was the separation of combined inte-

gration and test-diagnose-fix tasks into separate integration tasks, followed by a test-diagnose-fix task as explained in Section 2.2.

The full analysis of this integration and test sequence requires that a system test model for every component in this wafer scanner must be defined. At the time that this case study was performed, not all models were available, so simple abstracted models replace the missing models. For this purpose, 45 simple models were made. These parameters were derived from the system level risk estimation. Two prototype wafer scanners were assembled for the XT:850E development project. One critical component was assembled and tested first in prototype 2 and then disassembled and assembled into prototype 1. Prototype 1 was then fully qualified and the product was released to the manufacturing department and customer.

An integration and test process simulator has been used to simulate the components that are used to form a wafer scanner. Two thousand different instances were generated for each component. The execution of the derived integration and test sequence was simulated using these 'developed' 2000 components. The duration, cost and remaining risk for each of the simulations has been recorded. Additionally, the number of (simulated) faults found and the number of faults excluded has been recorded. Excluded fault states are faults that are diagnosed not to be present, fixed or faults excluded by *passed* test cases.

The histogram of the total test duration is depicted in Figure 42. The percentages on the righthand side of the graph indicate the cumulative percentage of the simulations. The median lies around 82 days. This means that if the deadline is 82 days, this deadline is met with a 50% probability. The range of the total test duration lies between 70 and 96 days, which is a difference of around 1 month.

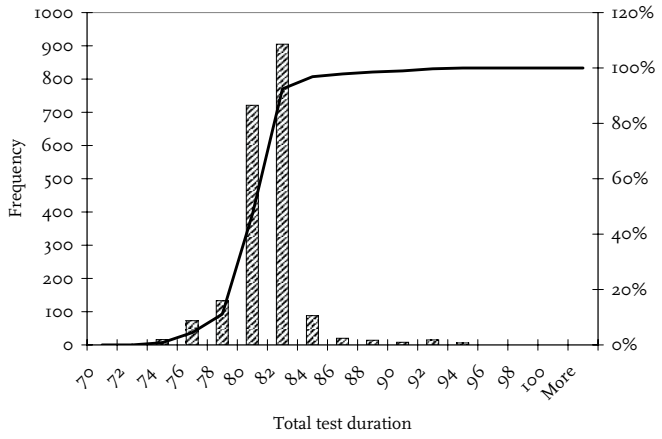


Figure 42. Histogram of the total test duration for prototype 1 of the XT:850E wafer scanner

The detailed risk, cost and failure profiles are depicted in Figure 43. The top

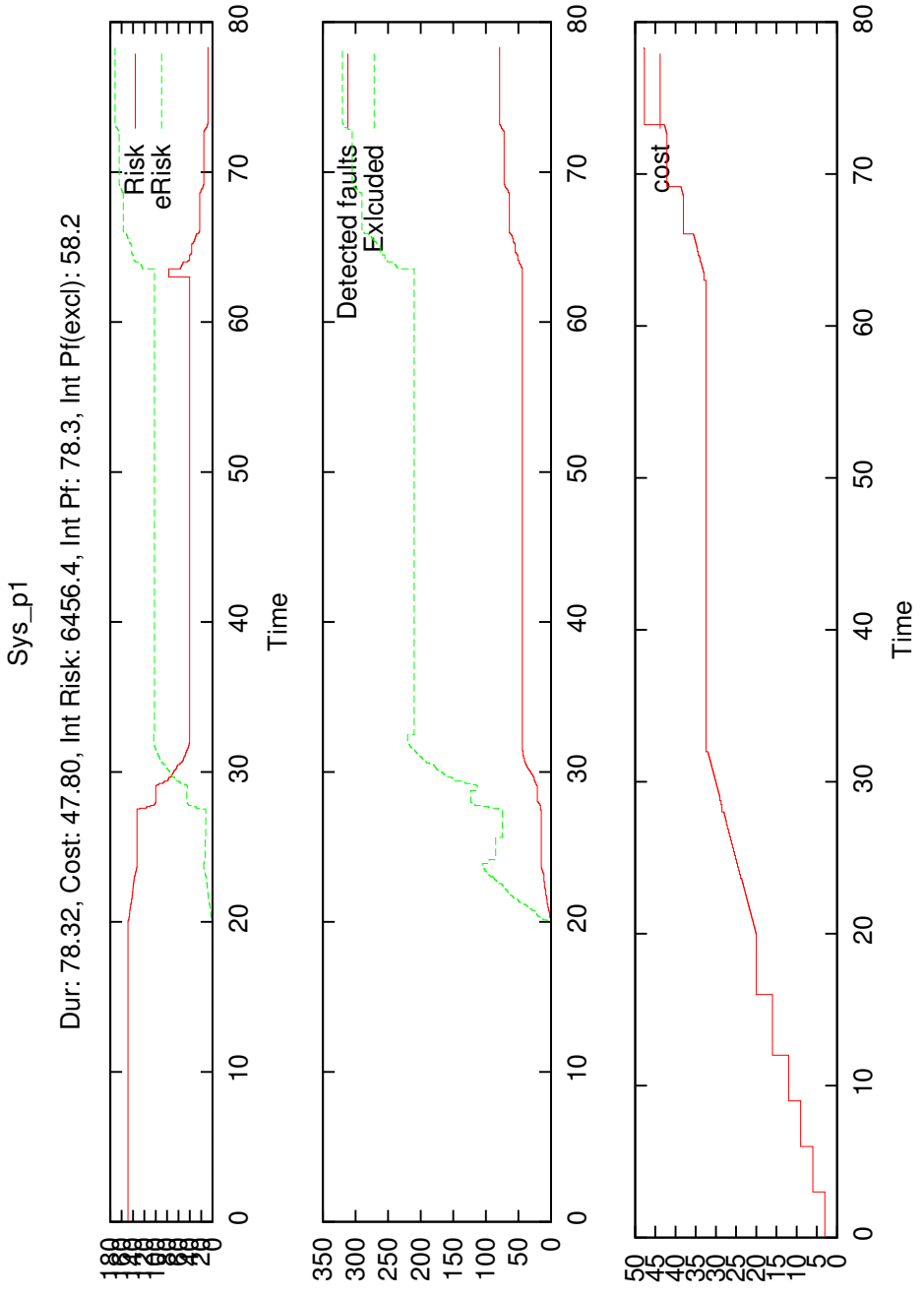


Figure 43. Detailed risk, cost and failure profile as function of time

graph depicts the remaining risk in the system and the excluded risk. The remaining risk is the risk that is still in the components and interfaces of the system. The excluded risk is the risk removed by passed test cases, diagnosed failed test cases and fixes. The excluded risk remains the same if no test-diagnose-fix tasks and integration tasks are performed. This meant that no test cases were planned, while in practice test cases that were planned on other, more busy, systems were scheduled on this machine. This is not depicted. This is the case between day 0 and day 20 and day 33 and 64. A complete system is first built between day 0 and day 20, followed by a qualification phase between day 20 and 33, where the system is qualified in this period. Then the risk remains at the same level until the critical component is delivered on day 64. The risk increases because of the assembly of the critical component and then the risk is reduced until the target level is reached on day 78.

The middle graph depicts the number of faults that are actually found in the system and the number of faults excluded by passed test cases. Excluding and detecting faults is only performed when test-diagnose-fix tasks are performed. This explains the flat line between day 0 and day 20 and between day 33 and 64, where no test cases are executed. The decrease in the number of excluded faults is caused by a removal of components from the system, because a new component version was introduced. The number of detected faults only increases and does not decrease when a component is removed, because the number of faults found is a KPI of the process and not a KPI of the components in the system itself.

The bottom graph depicts the cost that is spent on integration and testing. The cost increases with each executed integration task or test-diagnose-fix task. The cost of each task is added to the system level cost at the *end* of each task. This explains the stepwise increase of the cost between day 0 and day 20, where relatively long assembly tasks are performed.

4.1.5 *Conclusions*

In this section, we presented a method to analyze and compare system-level integration and test strategies. The key performance indicators of such integration and test strategies are defined as total integration and test duration Φ, cost C and remaining risk R_R. The method is supported by a set of techniques enabling designers to model and simulate integration and test strategies.

Following this method, the integration and test sequence of a newly developed wafer scanner has been modeled and analyzed. The analysis results indicate large flat areas in the risk graph, indicating the possibility for additional test-diagnose-fix tasks and the criticality of the component delivered on day 64.

4.2 PLANNING AND ANALYSIS OF TEST-DIAGNOSE-FIX TASKS

This section is based on 'Analysis of test-diagnose-fix strategies for complex manufacturing machines' [I. de Jong et al., 2007e] and describes the elements of a test strategy and the analysis method for single test-diagnose-fix tasks.

A test-diagnose-fix task is often planned by the executer of the test cases: the tester. The tester determines the sequence in which test, diagnose and fix tasks are performed, based on the available set of test cases, the goal of the test-diagnose-fix task, the available resources, the chosen test process and sometimes the results of previous test cases. The resulting sequence of test, diagnose and fix tasks is executed on the system under test. Obtaining the sequence of test, diagnose and fix tasks is increasingly difficult, because the number of available test cases increases with an increase in the complexity and size of the system. An increase in the number of test cases results in an increase of the number of diagnosis and fix tasks that are possibly executed. Consequently, the number of *possible* test-diagnose-fix sequences increases. Additionally, the test-diagnose-fix *sequence* becomes more important, because this sequence directly influences the duration and cost of the test-diagnose-fix *task*. The duration and cost of a test-diagnose-fix task are important parameters for an integration and test sequence, because of the pressure to deliver new products of high quality, faster and cheaper. A more structural integration and test sequence selection method is required, such that the best integration and test sequence is selected.

Common performance analysis techniques of test-diagnose-fix tasks only focus on the quality of the product and how the best quality is reached. Time does not matter in this context. Furthermore, these analysis techniques use specific mono-disciplinary techniques to measure the performance of test-diagnose-fix task. Examples of a mono-disciplinary performance technique are so-called 'software test techniques' as defined in [BCS/SIGIST, 2001]. The applicability of the analysis method for test-diagnose-fix tasks on systems of different disciplines is important, because nowadays integration and test sequences assemble and test components of multiple disciplines. Equally important is the applicability of the analysis method across the integration and test plan. This includes the lowest level component test-diagnose-fix tasks and the highest level system test-diagnose-fix tasks. In this way, test-diagnose-fix tasks can be compared with each other and the results can be taken into account in the overall integration and test plan.

4.2.1 *Test-diagnose-fix strategy*

A test-diagnose-fix task executes test, diagnose and fix tasks according to a test-diagnose-fix sequence. The restrictions of a test-diagnose-fix sequence are: 1) a diagnose task must be preceded by a test task , 2) a fix task must be preceded by a diagnosis task. All other combinations of tasks are allowed in a test-diagnose-fix sequence. However, not all combinations are typically considered for every

system under test. A test-diagnose-fix strategy, or *test strategy* in short, reduces the number of combinations that are considered. In other words, a test strategy determines how a sequence of test, diagnosis and fix tasks is formed. Many definitions of a test strategy exist in literature. The most used definitions include the test environment [Sasidhar et al., 1997], the individual test cases [Tai, 1990; BCS/SIGIST, 2001], multi-chip module testing [Flint, 20-25 Oct 1996], test sequencing [Cai et al., 2005], test-diagnose-fix task analysis [Benkahla et al., 1996; Mahoney, 22-25 Sep 1997] and the positioning of test-diagnose-fix tasks [Kung et al., 1995]. The most comprehensive definition of a test strategy is given in the first table in [Farren and Ambler, 1995]. This definition of a test strategy includes almost everything somehow related to a test-diagnose-fix task.

In other words, a test strategy determines how a sequence of test, diagnosis and fix tasks is formed.

Our definition of a test strategy is more restricted if compared with the general definition in [Farren and Ambler, 1995]. Not everything related to a test-diagnose-fix task is included into our definition. Only the elements that have an effect on the cost, duration and remaining risk of a test-diagnose-fix task are included. A test strategy determines how the test, diagnose and fix tasks are scheduled, how test, diagnose and fix tasks are sequenced and when the test-diagnose-fix sequence is stopped. All three elements of a test-diagnose-fix strategy are discussed in detail next.

Test process configuration

The test process configuration defines how test, diagnose and fix tasks are scheduled. The test process configuration influences the total test duration, cost and remaining risk of a test-diagnose-fix task and is therefore an important aspect to consider in a test-diagnose-fix strategy.

The first test process configuration discussed here is the *parallel test process configuration*. This test process configuration is often used in a time-to-market driven environment, like test-diagnose-fix tasks of an ASML wafer scanner. Testing, diagnosing and fixing is done in parallel. Consequently, fixes are applied and the system under test is changed while the test-diagnose-fix task is not finished yet. A Gantt chart of an example test-diagnose-fix sequence, scheduled using a parallel test-diagnose-fix process configuration, is depicted in Figure 44. The duration of the test-diagnose-fix task is depicted along the x-axis. The re-

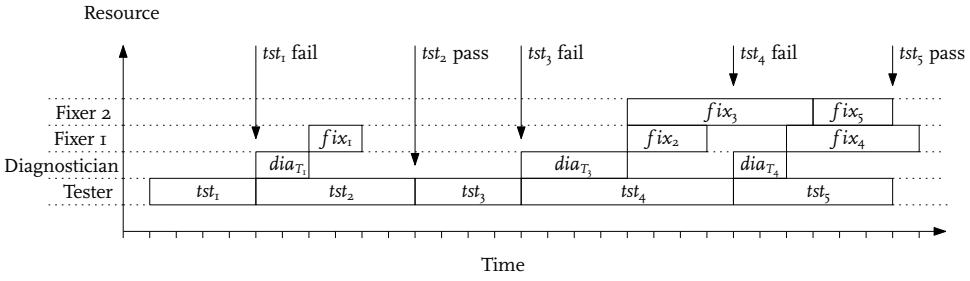


Figure 44. Gantt chart of the parallel process configuration

sources used for test execution, diagnosing and fixing problems are positioned along the y-axis. In this case, two fixers are available. Test execution task tst_1 is executed first and this test fails. A diagnosis task dia_{T_1} is started in parallel with the next test execution task. The diagnose task is succeeded by a fix task fix_1 that fixes the problems found in the diagnose task. The sequence continues until test 5 is executed, all failing test cases are diagnosed and fixes are developed. No re-testing is performed.

The second test process configuration, the *sequential test process configuration*, is a test process configuration where diagnosing, fixing and applying fixes is performed *after* the execution of all test cases. The system under test is not changed while test cases are executed. This test process configuration is used when the execution of test cases is outsourced to another organization. Test cases are executed and the results are passed to the development organization. This strategy is also used for test-diagnose-fix tasks that are executed automatically during the night or weekend. The results are analyzed on the next day. An example Gantt chart of the sequential test process configuration is depicted in Figure 45.

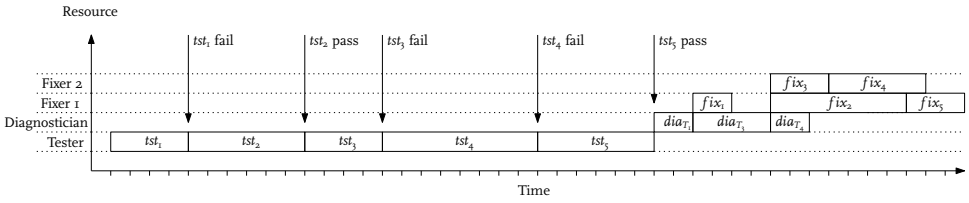


Figure 45. Gantt chart of the sequential process configuration

A third test process configuration discussed here is the *interleaved test process configuration*. This test process configuration executes diagnosis and fixes between, not in parallel with, test tasks. An example of this test process configuration can be observed during the system level design qualification of ASML wafer scanners. The nanometer performance of a wafer scanner is measured for the first time in this test-diagnose-fix task. When a test case fails, testing is stopped and not continued until the problem is diagnosed and fixed. Diagnosis and fixing is *interleaved* with test execution. An example Gantt chart of the interleaved test process configuration is depicted in Figure 46.

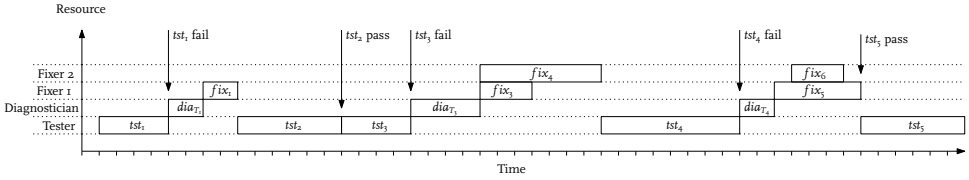


Figure 46. Gantt chart of the interleaved process configuration

Test sequencing

The second element in the test-diagnose-fix strategy is the test sequence. A test sequence G^T is created using a test sequencing algorithm Π^T. We divide the test sequencing algorithms into two groups: *off-line* and *on-line* test sequencing.

The first group, *off-line test sequencing*, does not take the result of the previous test into account when the next test case is selected i. e. the test sequence is derived before the test-diagnose-fix task starts. Examples are *risk-based test sequencing* [Rothermel and Harrold, 1996] [Harrold et al., 2001] [Amland, 2000] and *random test sequencing*. A framework for sequencing algorithms is defined first. The sequencing algorithms that are used further on are defined using this framework.

A sequencing algorithm Π^T that sorts the test cases can be defined as a map function according to Equation (4.9). The map function, ${\Pi^T}_n$, returns for each j-th element in the test sequence G^T the corresponding index of test case t in T

$$ {\Pi^T}_n : \{1, 2, ..., n\} \rightarrow \{1, 2, ..., n\} \tag{4.9} $$

where, ${\Pi^T}_n(1) = \pi_1$, ${\Pi^T}_n(2) = \pi_2$, ... ,${\Pi^T}_n(n) = \pi_n$, with the integers $\pi_1, \pi_2, ..., \pi_n$ all distinct, and

$$ X_{\pi_1} \geq X_{\pi_2} \geq ... \geq X_{\pi_n} \tag{4.10} $$

Sorting function X_{π_j} represents the jth smallest value for function X_π of a test. The calculation of function X determines the resulting sorted test sequence.

The second group, *on-line test sequencing*, takes the result of each individual test case into account when the next test case is selected. The next test case can be determined when the result of the previous test case is available. In *exploratory testing* [Kaner et al., 1999] and *adaptive testing* [Levendel, 2002], this technique is used to manually adapt the direction of testing while performing tests. The sequencing algorithm Π^T is in these cases the tester who selects the test case that is executed next. The remainder of this section describes five functions to determine the sorting function as introduced in Equations (4.9) and (4.10). A telephone example is used to illustrate some of the sorting properties. The telephone example is a system test model depicted in Table 10.

Random test sequencing

The random sequencing technique selects test cases randomly from the available test set T and can be seen as a sorting algorithm Π_n as defined in Equations (4.9) and (4.10). The sorting function X_{π_j} is calculated using Equation (4.11) with the resulting integers $X_{\pi_1}, X_{\pi_2}, ..., X_{\pi_n}$ all distinct.

$$ X_{\pi_j} \rightarrow \textbf{random}(n) \tag{4.11} $$

One of the possible random test sequences for the telephone example is $G_{Ra} = [t_1, t_3, t_0, t_2, t_5, t_4]$.

Ordered test sequencing

The ordered test sequencing technique selects test cases in the order of test case

S / T	t_0	t_1	t_2	t_3	t_4	t_5	P	I	φ_F	φ_{AF}
s_1	0.2	0.5	0	0	0.3	0	10%	7	2	1
s_2	0.2	0	0.5	0	0.3	0.3	10%	5	2	1
s_3	0.2	0	0	0.5	0	0.3	10%	3	2	1
s_4	0.2	0	0	0	0.3	0	10%	3	2	1
s_5	0.2	0	0	0	0	0.3	10%	3	2	1
C	3	1	1	1	2	2				
φ_T	6	2	2	2	4	4				
φ_D	10	1	1	1	6	6				

Table 10. A test model for the telephone example

definition. The order of definition is defined as a list O consisting of distinct integers. The function X_{π_j} can be determined using Equation (4.12):

$$X_{\pi_j} \rightarrow O(j) \tag{4.12}$$

The ordered test sequence for the phone example would be $G_O = [t_0, t_1, t_2, t_3, t_4, t_5]$ when the order of definition is defined as $O = [0, 1, 2, 3, 4, 5]$.

Risk-based test sequencing

This technique sorts test cases with the highest risk first using Equations (4.9) and (4.10). Risk-based testing is frequently used in practice. Test cases that cover the highest amount of risk are executed first if a risk-based test sequence is used. This approach is beneficial if the available test time is limited (less than the required test time) and the risk in the system is relatively low.

Again, this sequencing algorithm can be seen as a sorting algorithm as defined in Equations (4.9) and (4.10). The sorting function is the risk for each test case, $R(j)$, as calculated using Equation (4.13):

$$X_{\pi_j} = R(j) = \sum_{s \in S} P(s) I(s) R_{ts}(j, s) \tag{4.13}$$

Note that the *system test model* contains all information that is required to calculate the risk reduction of a test case. The resulting risk-based test sequence for the phone example is G_{Ri} is $[t_4, t_0, t_1, t_5, t_2, t_3]$, where $X_{\pi_1} = 0.45$, $X_{\pi_2} = 0.42$, $X_{\pi_3} = 0.35$, $X_{\pi_4} = 0.33$, $X_{\pi_5} = 0.25$ and $X_{\pi_6} = 0.15$.

Risk-cost-based test sequencing

This technique sorts test cases according to the highest risk/cost ration. The normal risk-based test sequencing technique does not take cost into account. Test cases that cover a lot of risk *and* are very costly, compared with the other test cases, are placed first in the normal risk-based test sequencing technique. A combination of cheaper test cases that together cover the same amount of risk could be more beneficial.

The risk/cost-ratio is calculated using Equation (4.14).

$$X_{\pi_j} = \frac{R(j)}{C_T(j)} = \frac{\sum_{s \in S} P(s) I(s) R_{ts}(j, s)}{C_T(j)} \tag{4.14}$$

The resulting risk-cost-based test sequence for the phone example is G_{RC} is $[t_1, t_2, t_4, t_5, t_3, t_6]$, where $X_{\pi_1} = 0.35$, $X_{\pi_2} = 0.25$, $X_{\pi_3} = 0.23$, $X_{\pi_4} = 0.17$, $X_{\pi_5} = 0.15$ and $X_{\pi_6} = 0.14$.

Information gain-based test sequencing

This technique sorts test cases based on the information gained per test case using Equations (4.9) and (4.10) and based on [Raghavan et al., Jan 1999]. Information in this context is maximal when the pass probability of a test case is closest to 50%. Hence, a good test case is a test case that covers fault states with a failure probability of 50%. This way, the pass probability of the test case is also 50%, i. e. the test case has a 50% probability to detect a fault state and 50% probability to prove the absence of certain failt states in the system. This *information gain* is corrected for test cost, such that the cheapest test with the highest information gain is selected. The function used for sorting, the cost corrected information gain for each test case, is calculated using Equation (4.15),

$$X_{\pi_j} = \frac{IG(j)}{C_T(j)} = \frac{-\left(p_p(j) log_2 p_p(j) + p_f(j) log_2 p_f(j)\right)}{C_T(j)} \tag{4.15}$$

where $p_p(j)$ is the *pass*-probability of test j and $p_f(j)$ is the fail probability. The pass probability is defined by

$$p_p(j) = \prod_{s \in R_t(j)} \left(1 - P(s) R_{ts}(j, s)\right) \tag{4.16}$$

where $R_t(j)$ is defined as follows:

$$R_t(j) = \{s \mid s \in S \wedge R_{ts}(j, s) > 0\} \tag{4.17}$$

and the corresponding *fail*-probability as

$$p_f(j) = 1 - p_p(j) \tag{4.18}$$

One of the resulting information gain-based test sequences for the phone example is G_{IG} is $[t_1, t_2, t_3, t_4, t_5, t_6]$, where $X_{\pi_1} = X_{\pi_2} = X_{\pi_3} = 0.286$, $X_{\pi_4} = X_{\pi_5} = 0.214$ and $X_{\pi_6} = 0.152$. The algorithm could result in other solutions as well, because the information gain and cost of test 1 through 3 is the same, which is also the case for test 4 and 5.

Test sequencing - On-line test sequencing

The on-line test sequencing techniques use the same algorithms to sequence test cases as off-line test sequencing. The difference is an update of the failure probabilities of the fault states after each test, diagnosis and fix task. The update of the failure probability is described in Equation 4.23. The updated failure probabilities are taken into account for the selection of the new test case in the sequence instead of the initial model.

Test stop criteria

The next element of a test-diagnose-fix strategy is the test stop criterion. The test stop moment influences the total test duration Φ, the total test cost C and the remaining risk in the system R_R. The remaining risk R_R in the system is calculated using Equation (4.19).

$$R_R = \sum_{s \in S} P(s) I(s) \tag{4.19}$$

The duration and cost of the executed test cases contribute to the overall test duration and cost. The actual faults in the system and when these faults are detected result in diagnosis and fix tasks that introduce additional cost and duration. The duration, cost and remaining risk of the test, diagnosis and fix sequence depends on the test stop criterion. Figure 47 describes the remaining risk and cost in a test-diagnose-fix task as function of the test duration including these test stop criteria for an example test-diagnose-fix task:

- Stop testing when a certain risk level is reached, e. g. $R_R = 25$.
- Stop testing when a certain deadline is reached, e. g. $\Phi = 200$.
- Stop testing when a test cost level is reached, e. g. $C = 125$.

In practice, it is difficult to determine if a test stop criterion based on remaining risk is reached, because it is based on the estimation of the risk for each fault state. A combination of estimated risk and other test metrics should be used to determine if the stop criterion, based on risk, is reached.

4.2.2 *Analysis of test-diagnose-fix tasks*

The performance of a test-diagnose-fix task is described by three key performance indicators (KPI): Time Φ, Cost C and remaining risk R_R. These general KPI are independent of the type of system under test, abstraction level of the test case and level of aggregation in the system. Time is a measure for the duration of the test-diagnose-fix task. Time is spent whilst executing test cases, diagnosing problems, fixing faults. Additional waiting time could be spent if test, diagnose or fix resources are not available. Cost is a measure for the cost involved in executing the test-diagnose-fix task and is built up by executing test

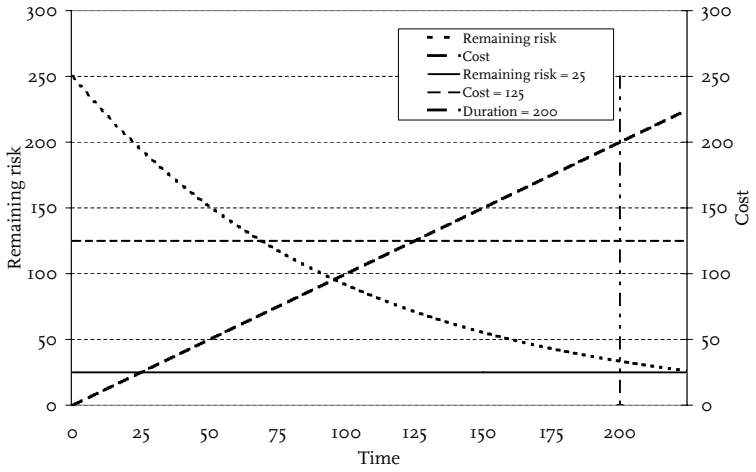


Figure 47. Test stop criteria depicted

cases, diagnosing failed test cases and fixing results. Remaining risk R_R is a measure of the quality of the system after test execution. Executing test cases, diagnosing and fixing problems all reduce the failure probability of the covered fault states. In this way, remaining risk is reduced. A test-diagnose-fix task that quickly reduces all risk at low cost is considered good, while a test-diagnose-fix task that does not reduce risk at high cost slowly is considered a test-diagnose-fix task of bad quality.

The execution of a single test-diagnose-fix task leads to a single value for the test duration, cost and remaining risk. However, the faults actually present in the system influence the duration, cost and remaining risk of a single execution of a test-diagnose-fix task. Any combination of faults can be present in a system. Therefore, the combination of possible faults needs to be taken into account when the KPI of a test-diagnose-fix task are determined. Taking into account the actual faults in the system can be done in various ways. First, the test-diagnose-fix task can be executed many times on many faulty systems and the KPI can be measured and a statistical evaluation of the test duration, cost and remaining risk can be performed. Large numbers of different systems under test are required for this approach. This is possible in a volume manufacturing environment, but not in a development environment with only a few systems under test.

A different approach to determine the KPI is using a model of the system under test and a model of the test process. Many 'systems under test' are generated for a newly developed system that is not yet available. The different test process configurations, test sequencing techniques and stop criteria can be selected in the test process model. The expected values of the KPI are determined as well as the distribution of test duration, cost and remaining risk.

The variation in test duration, cost and remaining risk is influenced by the

The actual faults in the system under test that can be present in any combination, results in 2^l fault state combinations for a system of l fault states.

test, diagnose and fix sequence and the actual faults in the (modeled) system under test. The actual faults in the system under test that can be present in any combination, results in 2^l fault state combinations for a system of l fault states. Executing a test-diagnose-fix task for each combination of fault states is too computationally expensive for large systems. Therefore, a simulation model of the test process has been developed that creates faulty systems under test using a binomial distribution and executes test cases according to a predefined strategy on these created systems under test. A schematic overview of the test-diagnose-fix process is depicted in Figure 48. The sub-processes in the test-diagnose-fix process use the system test model as input and perform sub-tasks on this system test model. All sub-processes are explained in detail. A process is depicted as a circle, while the inputs and outputs of the processes are depicted as edges.

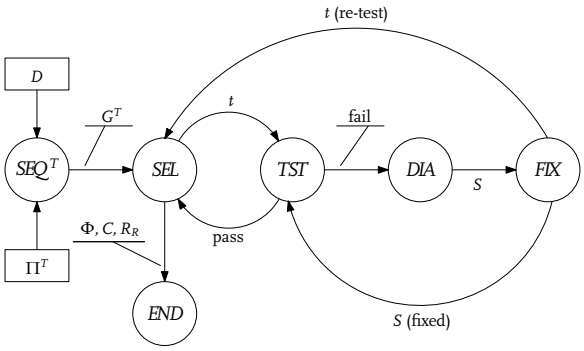


Figure 48. Overview of the test-diagnose-fix process

The SEQ^T process sequences a set of test cases using a sequencing algorithm Π^T and system test model D. The resulting test sequence G^T is passed to the *Test selection* process *SEL*. The *Test selection* process *SEL* dispatches the first test case in the sequence to the test case execution process *TST* and waits for the result of the test case. When the result is received, the next test case in the sequence is dispatched to the test execution process. Meanwhile, test cases can be received that require re-execution. These (re-)test cases are placed first in the remainder of the test sequence. The diagnosis process *DIA* diagnoses failed test cases. Diagnosis is assumed to be perfect. This means that diagnosing a set of candidate fault states results in two sets: a set of fault states that are present and a set of fault states that are not present. The faulty states are fixed in the *FIX* process and the non-faulty states are excluded from the set of possible fault states. The fixed fault states are buffered and the fix is applied on the system under test. Next the details of the test execution, diagnosis and fix process are described using:

- a sequence of test cases G,
- the initial set of present fault states: S_S,

- the set of detected fault states: S_D,
- the set of candidate sets of possibly present fault states: S_c,
- the set of fault states that are not present (excluded): S_E,
- the set of fault state sets that are to be fixed: $\mathcal{X}_f$.

Test execution TST

The execution of test cases in the *TST* process is described next. Executing a test case t takes $\varphi_T(t)$ time and costs $C_T(t)$. Both are properties of the test case. The test execution process is described as a set of equations that take the sequence of executed test cases and the current test case t as input. The sequence of test cases, combined with the current test case t is described as Gt.

The coverage of test case t on the fault states in the system determines if a test case can *detect* these fault states. The fault states that can possibly be detected by test case t are described using:

$$S_D(t) = \{s \mid s \in S \wedge sample(dist) < R_{ts}(t, s)\} \tag{4.20}$$

The set of candidate fault sets contains a set of fault states for each failed test case and is updated if the detectable fault states are actually present in the system after executing test sequence G:

$$S_c(Gt) = \begin{cases} S_c(G) & \text{if} \quad S_D(t) \cap S_S(G) = \emptyset \\ S_c(G) \cup \{R_t(t) \setminus S_E(G)\} & \text{if} \quad S_D(t) \cap S_S(G) \neq \emptyset \end{cases} \tag{4.21}$$

where $R_t(t)$ describes the fault states that are covered by test case t:$R_t(t) = \{s|s \in S \wedge R_{ts}(t, s) > 0.0\}$. The set of fault states that are known to be absent in the system is modeled as $S_E(Gt)$, i. e. the *excluded* fault states. If a test case fails, all covered fault states are added to the candidate set, except those fault states that are already known to be absent in the system. Initially, the candidate set is empty: $S_c(\epsilon) = \emptyset$.

The set of excluded fault states contains those fault states that are known to be absent. A fault state is known to be absent if: 1) The fault state is fixed, 2) the fault state is diagnosed not to be present or 3) the failure probability of the fault state is below a minimal level. The first and third case are described in this *test execution* process, the second case is described in the *diagnosis* process. The initial set of excluded fault states is empty: $S_E(\epsilon) = \emptyset$. The excluded fault state set is updated according to:

$$S_E(Gt) = S_E(G) \cup \{\bigcup_{X_f \in \mathcal{X}_f(Gt)} X_f\} \cup \{s \mid s \in S \wedge P(s, Gt) < 10^{-7}\} \tag{4.22}$$

where $S_E(G)$ is the previous set of excluded fault states. The second part excludes fixed fault states and the third part excludes fault states with a failure probability below 10^{-7}.

The failure probability of every fault state, after executing a sequence of test cases G, is calculated to determine if fault states can be excluded. This failure probability is also used to determine the next test case to be executed when an on-line test sequencing method is used. The proper way to determine the failure probability after executing a sequence of test cases requires that all combinations of fault states are taken into account. This is computationally intensive, therefore the probability estimator of [Boumen et al., Jan. 2008] is used. The failure probability is estimated using:

$$P(s, G) = (1 - P(s, G)) \prod_{S' \in S_c(G) \wedge s \in S'} (1 - \frac{P(s, G)}{\sum_{s_i \in S'} P(s_i, G)}) \tag{4.23}$$

Initially, the probabilities are set to the a-priori probabilities: $P(s, \epsilon) = P(s)$. The probabilities of the fault states that are excluded (by any of the three methods) is set to 0.0:

$$P(s, G) = 0.0 \text{ for all } s \in S_E(G) \tag{4.24}$$

Finally, the set of fault states that are present in the system after executing test sequence Gt is updated by removing the fault states that can be detected by the test $S_D(t)$ and that are present in the system $S_S(G)$ after executing sequence G:

$$S_S(Gt) = S_S(G) \setminus (S_D(t) \cap S_S(G)) \tag{4.25}$$

The initial set of fault states $S_S(\epsilon) = S_S$ is equal to the set of fault states that are present in the beginning.

Diagnosis DIA

The diagnosis of failed test cases is assumed to be perfect. This means that fault states that are present in the system are correctly diagnosed to be present. Fault states that are not present in the system are diagnosed not to be present. Diagnosing a failed test case t takes $\varphi_D(t)$ time and costs $C_D(t)$.

The set of excluded fault states is updated with the fault states that are diagnosed not to be present in the system, according to:

$$S_E(Gt) = S_E(G) \cup (S_D(t) \setminus S_S(G)) \tag{4.26}$$

The candidate set is updated as well as the failure probabilities of the fault states that are diagnosed.

$$S_c(Gt) = \{s_c \setminus (S_D(t) \setminus S_S(G)) \mid s_c \in S_c(G)\} \tag{4.27}$$

$$P(s, Gt) = 0.0 \text{ for all } s \in (S_D(t) \setminus S_S(G)) \tag{4.28}$$

$$P(s, Gt) = 1.0 \text{ for all } s \in (S_D(t) \cap S_S(G)) \tag{4.29}$$

Fixing FIX

Fixing fault states is modeled as an update of the candidate set and a placement of the fix in the buffer. Fixing a fault state s takes $\varphi_F(s)$ time and costs $C_F(s)$. The set of fixed fault states is added to the fix buffer $\mathcal{X}_f$:

$$\mathcal{X}_f(Gt) = \mathcal{X}_f(G) \cup \{S_D(t) \cap S_S(G)\} \tag{4.30}$$

Then, the candidate set is updated according to:

$$S_c(Gt) = \{s_c \setminus S' \mid s_c \in S_c(G) \wedge S' \in \mathcal{X}_f(Gt)\} \tag{4.31}$$

4.2.3 *Illustration: Comparing test strategies*

The analysis method as introduced in the previous sections is illustrated in this section. For this purpose, a large number of system test models are generated. The KPI of test-diagnose-fix tasks performed on these models are determined for different test strategies. The system test model is varied in this illustration. The system test models are randomly generated to exclude effects of specific modeling techniques. A number of parameters are varied to analyze the effect of these parameters on the test duration. First, the number of fault states and test cases are varied to determine what the minimum model size is. Models that are too small lead to unstable results, whilst models that are too large result in long simulation durations. The probability of the fault states and the test coverage are also varied. The cost and remaining risk are not considered in this experiment, because the effect of different systems under test on the KPI of a test-diagnose-fix task can be explained by using test time only.

The size of the model

The first experiment is performed to illustrate the effect of the size of the system test model on the test duration. The minimum model size that is used for further experiments is derived using this experiment. The failure probability of the fault states is varied in the range 10%, 25%, 50% and 90%. The size of the model is varied in two directions: the number of test cases is varied and the number of fault states is varied. A number of reference models have been generated for this purpose. The reference models contain either 4, 12 or 20 test cases with a total duration of 10.0 time units. The number of fault states in the model is also 4, 12 and 20. The resulting models are large enough to illustrate the effect of the varied parameters and these models are small enough to minimize the computational effort.

Test models have been created with a random coverage for each of the test cases on the fault states with an average density, ρ (Equation (4.32)), of 0.25 for all models. Five models for each combination of number of fault states, number of test cases and failure probabilities are created. The results are averaged over these five models. The remaining risk stop criterion is set to 0.2 to ensure that a reasonable number of test cases is executed. No stop criterion is set for total

test duration and cost. A random test selection technique is used. A random test sequence results in many different test sequences and by this results in the highest variance in test duration, cost and remaining risk. A random test sequencing technique is the worst-case situation. One thousand simulations were performed with these settings for each test model, because one thousand simulations leads to a stable average test duration.

$$\rho = \frac{\sum_{t \in T, s \in S} R_{ts}(t, s)}{|T| * |S|} \tag{4.32}$$

The resulting average test duration $\overline{\Phi}$ and the average standard deviation $\overline{\sigma_\Phi}$ for each of the failure probabilities are presented in Tables 11 to 15.

(a) $\overline{\Phi}$

$\|S\| / \|T\|$	4	12	20
4	7.73	2.58	1.56
12	22.93	7.13	3.57
20	52.71	8.83	4.86

(b) $\overline{\sigma_\Phi}$

$\|S\| / \|T\|$	4	12	20
4	3.103	1.097	0.618
12	7.473	2.486	0.950
20	12.617	2.297	1.093

Table 11. Average duration (a) and standard deviation (b) for $\overline{P} = 0.1$

(a) $\overline{\Phi}$

$\|S\| / \|T\|$	4	12	20
4	19.34	5.33	3.03
12	46.23	9.28	5.29
20	64.93	12.87	6.77

(b) $\overline{\sigma_\Phi}$

$\|S\| / \|T\|$	4	12	20
4	9.056	2.426	1.313
12	13.168	3.333	1.823
20	19.741	4.524	2.139

Table 12. Average duration (a) and standard deviation (b) for $\overline{P} = 0.25$

(a) $\overline{\Phi}$

$\|S\| / \|T\|$	4	12	20
4	36.03	6.33	3.69
12	84.55	13.07	6.28
20	137.15	18.12	7.82

(b) $\overline{\sigma_\Phi}$

$\|S\| / \|T\|$	4	12	20
4	19.545	3.040	1.837
12	39.641	6.163	2.263
20	72.568	7.811	2.761

Table 13. Average duration (a) and standard deviation (b) for $\overline{P} = 0.5$

The following conclusions can be drawn from these results.

(a) $\overline{\Phi}$

$\lvert S\rvert / \lvert T\rvert$	4	12	20
4	25.03	6.23	3.66
12	49.77	14.41	7.24
20	53.84	19.28	10.93

(b) $\overline{\sigma_\Phi}$

$\lvert S\rvert / \lvert T\rvert$	4	12	20
4	16.775	3.911	2.187
12	43.615	7.782	3.677
20	33.604	13.434	6.881

Table 14. Average duration (a) and standard deviation (b) for $\overline{P} = 0.9$

(a) $\overline{\Phi}$

$\lvert S\rvert / \lvert T\rvert$	4	12	20
4	22.03	5.12	2.98
12	50.87	11.18	5.60
20	77.15	14.77	7.59

(b) $\overline{\sigma_\Phi}$

$\lvert S\rvert / \lvert T\rvert$	4	12	20
4	12.120	2.619	1.489
12	25.974	5.070	2.178
20	34.633	7.017	3.218

Table 15. Average duration (a) and standard deviation (b) for all failure probabilities

- The average duration of the test-diagnose-fix task decreases with an increase of the number of test cases in the model. The sum of test durations of the test cases, $\sum_{t \in T} \varphi_T(t)$, is equal for all models with 4, 12 and 20 test cases. Consequently, the cost of a single test in the 20-test case model is $\frac{1}{5}$-th of the cost of a test case in the 4-test model, while the coverage is roughly the same. This results in a reduction of a factor 7 to 10 in average test duration. The main cause of this effect is the last executed test case that brings the remaining risk of the test-diagnose-fix task below the target. This last test case is relatively lengthy in the 4-test case models. This is illustrated in Figure 49, where the duration of t_1 is less than t_2. The total test duration is longer if t_2 is selected and the remaining risk target is reached by this test case.

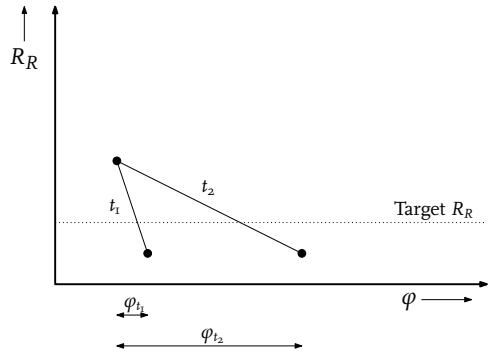


Figure 49. Two last test cases with different durations affecting the total test duration

- The average duration of the test-diagnose-fix task becomes higher when the number of fault states in the test model increases. The reason for this effect is the increase of the number of fault states in the system causing an increase of the system risk. The 20-fault state system contains five times more risk than the 4-fault state system. This results in an increase of the test duration of a factor 2,5 to 3,5 on average (see Table 15(a) for details).

- Further analysis reveals some combination effects of the above. On the other hand, the used test models also have an effect of the test duration. In practice, the model is given and not considered a variable as it is in this illustration.

The number of test cases and fault states to be used for the remainder of this illustration is set to $|S| = |T| = 12$. The usage of models with four test cases or four fault states leads to unstable results. Using more than 12 fault states increases the simulation time and does not increase the stability of the results further.

The failure probability

This experiment is performed to illustrate the effect of an increasing failure probability on the total test duration. A system test model of 12 test cases, 12 fault states is used. Sixteen test models are generated for each of the failure probabilities in the list: 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80% and 90%. The *average* density of all models is 25%. Small variations in coverage between models can be seen, because every model is newly created. The results of 1000 test process simulations for each of the models are depicted below in Table 16 and Figure 50. Figure 50 only contains the average total test durations, while Table 16 also contains the standard deviation, minimum and maximum values. The following conclusions can be drawn from these results:

P	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$\overline{\Phi}$	6.39	9.52	11.18	12.24	12.98	12.42	15.98	13.63	14.01
$\overline{\sigma}_{\varphi}$	1.974	3.315	4.465	5.289	5.363	5.357	8.577	7.478	7.870
φ_{min}	1.67	2.50	2.50	2.50	2.50	2.50	2.50	2.50	2.50
φ_{max}	21.67	40.83	71.67	112.50	110.00	64.17	248.33	128.33	132.50

Table 16. Total test duration results for varying failure probabilities

- An increase in the failure probability of the fault states in the test model results in an increase of the total test duration. This increase in total test duration is not linear. A logarithmic curve fit is depicted in Figure 50. A small increase of the lower failure probabilities leads to a large increase in total test duration. This effect is less apparent for the higher failure probabilities. Test time reduction can be obtained by decreasing the failure probability. A trade-off needs to be made between the cost of decreasing

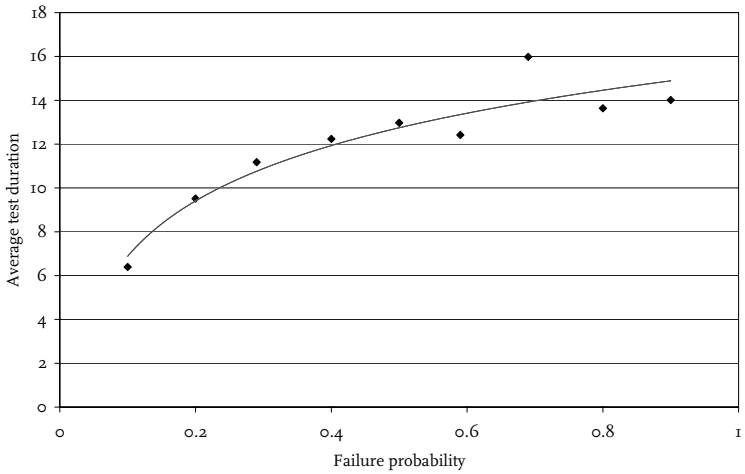


Figure 50. System test model failure probability versus total test duration

the failure probability and the gain in total test duration. Decreasing the failure probability can be done by executing additional test cases in an earlier (parallel) test-diagnose-fix task or by increasing the quality of the system under test. The additional duration involved in these tasks needs to be taken into account when comparing the total test duration.

- The total test duration for system test models with a failure probability of 70% contains an outlier. One test model is present in the dataset, that can result in a total test duration of 248.33. One of the fault states in this test model is covered by a single test. The coverage of this single test on this fault state happens to be very low. One (randomly selected) test sequence, consists of 149 times this test only and results in a very long test duration. Removing this test model from the dataset results in more consistent results (not depicted).

The test coverage (density)

This experiment is performed to illustrate the effect of the test coverage on the total test duration. The coverage of the test cases in the test model is measured by the density ρ of the test model according to Equation (4.32). The range of densities used for this experiment is (0.1, 0.2, 0.3, 0.5, 0.7, 0.8, 0.9). Sixteen models were generated for each density in the range. Because the generation of models does not result in a model with an exact density as required, an error in the density of 0.05 is allowed for each model. The resulting models have a minimal density of the target density and maximal density of the target density +0.05. The results of this experiment, the average total test duration $\overline{\Phi}$ and standard deviation of the total test duration $\overline{\sigma_\Phi}$, are depicted in Table 17. Figure 51 graphically depicts the average total test duration as function of the density of the test

model. Additionally, the fitted power function is depicted in the graph. The

ρ	$\overline{\Phi}$	$\overline{\sigma_\Phi}$
0.1	55.09	17.455
0.2	17.81	7.261
0.29	8.55	3.403
0.5	4.28	1.470
0.69	2.66	1.115
0.8	2.07	0.932
0.9	1.46	0.807

Table 17. Result of the density experiment (average test duration and standard deviation)

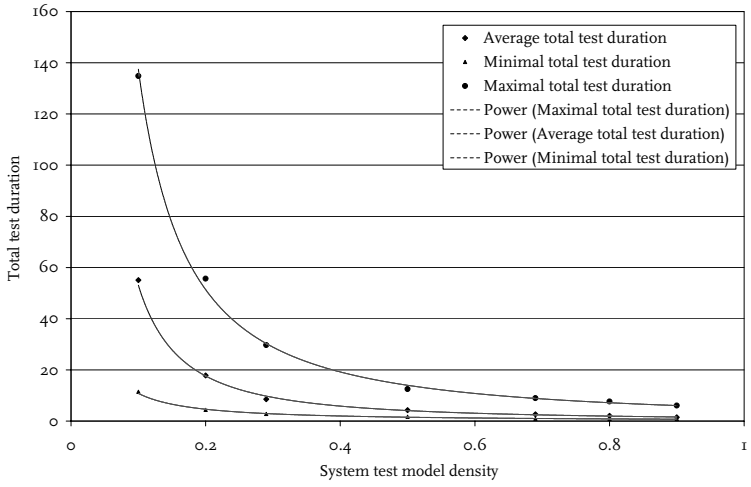


Figure 51. System test model density versus total test duration

following conclusions can be drawn based on these results:

- A higher density (coverage of the test cases on the fault states) leads to a lower total test duration. The reason for this is twofold. First, a test case with a high coverage on a fault state s results in a high reduction of the failure probability of a fault state when the test case *passes*. The target remaining risk is reached earlier this way. Second, a test case with a high coverage on a fault state has a high probability of detecting the fault state if the fault state is actually in the system. Fault states are found early in the test sequence and can therefore be diagnosed and fixed earlier. Diagnosing and fixing a fault state also reduces the risk.
- A density increase of 10% to 30% is very beneficial for the total test duration. A factor three in test coverage results in a factor six reduction in total test duration. The cost of increasing the test coverage from 10% to 30%

is not taken into account here. Increasing the density (coverage) beyond 30-40% seems not beneficial.

- The target risk in the system is kept at the same level for all experiments. Systems with a very low target risk (systems with very high quality requirements) should benefit from an increase in test coverage. This is not investigated in detail. In general, it depends on the system test model, the number of test cases, the fault states and the length of the test cases, whether increasing the coverage is beneficial.

The results as presented in this illustration are created using generated models and illustrate the effect of the modeling parameters on the KPI. The results of the illustration match with intuition about testing and the relationship between the system under test, the test cases and the KPI. The effect of the test sequence is not illustrated, nor are, the combination effects of different parameters. For this, it is better to perform a case study. Many combinations of parameters lead to different results, while the goal of a case study is clear from the start. The case studies presented in this section are all executed to evaluate an existing test-diagnose-fix task, including the test sequencing technique. Furthermore, the influence of an improved diagnosis and fix duration is investigated by two out of the three case studies.

4.2.4 *Cases*

Three case studies have been performed with the test-diagnose-fix task analysis method in three different domains. The focus of the case studies is on selecting the best test-diagnose-fix strategy and investigating the benefit of improving the diagnosis and fix duration. The first case study is performed on a software sub-system. The test-diagnose-fix task of the second case study concerns a system level functional qualification. The third case study has been performed in the ASML manufacturing department for the final qualification of wafer scanners.

Case 1: Software sub-system test-diagnose-fix strategy

This software sub-system is part of an ASML wafer scanner. The sub-system that is used in this case study is the controlling software for Lot Production (LP). LP consists of roughly 350.000 lines of code in around 450 files. The modeled test-diagnose-fix task is a system level test-diagnose-fix task consisting of 54 test cases that cover 31 fault states. LP is a mono-disciplinary sub-system consisting of software only. The average test case duration is 76 minutes with a standard deviation of 129 minutes. The main reason for the high standard deviation are 3 test cases with a duration of 6 hours or more. The average diagnose duration for failed test cases is 60 minutes. The fix duration is 60 minutes also. The system test model is depicted in Table 40 in the Appendix.

Test execution is normally planned during the weekend and fault diagnosis and fixing failures is performed on the following Monday. The normal test se-

quencing technique is the off-line, ordered test sequencing technique with serial diagnosis and fixing. Normally, testing is stopped when all test cases are executed once. The average remaining risk reached by the off-line test sequencing techniques is used as the test stop criterion for the on-line test sequencing techniques. This way, a comparison on total test duration can be made between the off-line and on-line test sequencing techniques and the best test sequencing technique can be chosen.

The effect of different test sequencing techniques on the performance parameters is determined, as well as the effect of different diagnosis and fix durations for the chosen test sequences. A number of test sequencing techniques, as described in Section 4.2.1, are used. Some of these techniques are used in an off-line and on-line setting indicated with *Of* or *On*. A parallel process configuration is indicated with *P* and a serial process configuration with an *S*. For example, the off-line, risk-cost based test sequencing technique is indicated with [Of/RC/P] for the parallel test process configuration and [Of/RC/S] for the serial test process configuration. Table 18 gives an description of the combination of sequencing techniques and their abbreviation as used here and in the other case studies.

Test sequencing technique	Description
[Of/O/S]	Off-line ordered test sequencing, serial diagnosis and fix process
[Of/O/P]	Off-line ordered test sequencing, parallel diagnosis and fix process
[Of/RC/S]	Off-line risk-cost-based test sequencing, serial diagnosis and fix process
[Of/RC/P]	Off-line risk-cost-based test sequencing, parallel diagnosis and fix process
[Of/Ra/P]	Off-line random test sequencing, parallel diagnosis and fix process
[Of/Ri/S]	Off-line risk-based test sequencing, serial diagnosis and fix process
[Of/Ri/P]	Off-line risk-based test sequencing, parallel diagnosis and fix process
[Of/iR/P]	Off-line inverse risk-based test sequencing, parallel diagnosis and fix process
[Of/IG/P]	Off-line information-gain-based test sequencing, parallel diagnosis and fix process
[On/Ri/P]	On-line risk-based test sequencing, parallel diagnosis and fix process
[On/RC/P]	On-line risk-cost-based test sequencing, parallel diagnosis and fix process
[On/Ra/P]	On-line random test sequencing, parallel diagnosis and fix process

Table 18. Table of test sequencing techniques and their abbreviation

For this case study, the on-line test sequencing techniques perform better in terms of the total test duration than the off-line test sequencing technique. See Table 19 for the average test duration $\overline{\Phi}$, average test cost $\overline{C}$ and the average remaining risk $\overline{R_R}$ and see Table 20 for the standard deviations. The normal approach, *[Of/O/S]*, is emphasized in both tables. The average duration of an off-line ordered test sequence in combination with a serial diagnosis and fix process is 60.87 hours. While the duration of the on-line, risk-based test sequence with parallel diagnosis and fixing, [On/Ri/P], is 17.3 hours. The standard deviation of the total test duration is higher for the on-line test sequencing techniques resulting in more variation at the test stop moment. Consequently, the standard deviation for the cost of the test-diagnose-fix task is also larger for the on-line test sequencing techniques.

The standard deviation of the remaining risk is much lower, 0.207 for the original test-diagnose-fix strategy compared to 0.038 for the optimal ([On/Ri/P]) test-diagnose-fix strategy, because the target remaining risk is approached more closely. The risk reduction of the last test cases in the original test-diagnose-fix strategy is large, resulting in a higher standard deviation of the remaining risk. The risk reduction of the optimal test-diagnose-fix strategy in the end phase of testing (close to the remaining risk target) is small and results in a smaller standard deviation of the remaining risk.

	[Of/NO/S]	[Of/RC/P]	[Of/RC/S]	[Of/Ra/P]	[Of/Ri/P]	[Of/iR/P]
$\overline{\varphi}$	60.87	61.48	60.82	61.60	60.64	62.73
$\overline{C}$	93.85	89.27	93.99	89.39	88.95	90.04
$\overline{R_R}$	0.94	0.87	0.92	0.87	0.88	0.85

	[Of/IG/P]	[Of/Ri/S]	[Of/NO/P]	[On/Ri/P]	[On/Ra/P]
$\overline{\varphi}$	61.81	60.84	60.75	17.30	44.17
$\overline{C}$	89.42	93.19	89.02	31.10	60.24
$\overline{R_R}$	0.88	0.94	0.88	0.87	0.86

Table 19. Results for case study 1: test duration, cost and remaining risk

	[Of/NO/S]	[Of/RC/P]	[Of/RC/S]	[Of/Ra/P]	[Of/Ri/P]	[Of/iR/P]
$\overline{\sigma}_{\varphi}$	1.336	1.855	2.328	1.947	1.590	2.194
$\overline{\sigma}_{C}$	3.971	2.455	5.136	2.178	1.936	2.334
$\overline{\sigma}_{R_R}$	0.207	0.151	0.232	0.151	0.138	0.159

	[Of/IG/P]	[Of/Ri/S]	[Of/NO/P]	[On/Ri/P]	[On/Ra/P]
$\overline{\sigma}_{\varphi}$	1.905	1.352	1.633	3.03	9.964
$\overline{\sigma}_{C}$	2.182	3.738	1.925	5.038	10.943
$\overline{\sigma}_{R_R}$	0.146	0.198	0.139	0.038	0.026

Table 20. Results for case study 1: standard deviation

The reduction in average total test duration for the on-line, risk-based, parallel test-diagnose-fix strategy is 71% if compared with the off-line, ordered, serial

test-diagnose-fix strategy. The standard deviation of the total test duration is increased by more than a factor 2 for the online risk-based parallel strategies.

Case 2: System qualification

In this case study, a specific system function of a newly developed wafer scanner is qualified. This qualification period currently consists of 53 test cases that are executed in the order that these test cases are specified. The system function is split up into 60 sub-functions that need to be tested. The sum of the test durations of the individual test cases is 6560 minutes (109 hours). The average failure probability of the 60 fault states is 12%. The system test model is depicted in Table 42 in the Appendix. The normal approach during such a functional test-diagnose-fix task is to execute all available test cases. When test cases fail, a diagnosis is started and testing is continued. Fixes are applied to the system under test when the fixes become available. Testing is continued after these fixes are applied. The tester at the wafer scanner decides which test case is executed next. No guidelines are used for this selection. This test-diagnose-fix strategy corresponds with an off-line, random test sequencing, parallel test process configuration indicated as *[Of/Ra/P]*. The normal *combined* diagnosis and fix duration is 360 minutes in this system level test-diagnose-fix task. Developers are constantly available to diagnose problems and develop fixes.

A range of test sequencing techniques is investigated. In addition, it is investigated if it is beneficial, in terms of total test duration, to spend effort on minimizing the diagnosis and fix duration even further. The normal combined diagnosis and fix duration is 360 minutes. The combined diagnosis and fix durations, $\varphi_D + \varphi_F$, were set to 90, 180, 360 and 720 minutes.

$\varphi_D + \varphi_F$	[Of/Ri/P]	*[Of/Ra/P]*	[Of/RC/S]	[Of/Ri/S]	[Of/O/P]	[Of/O/S]
90	4405.42	*6724.27*	7280.80	7280.87	4856.50	2639.66
180	4942.63	*7320.19*	8042.43	7932.53	5366.35	2968.34
360	5999.05	*8544.28*	9493.41	9422.93	6448.33	3620.48
720	7583.41	*10414.23*	12295.11	12290.15	8057.89	4042.58

$\varphi_D + \varphi_F$	[On/Ra/P]	[On/RC/P]	[On/Ri/P]
90	21979.85	4059.59	6212.17
180	22902.46	4783.91	6937.39
360	23675.12	5633.23	8139.05
720	24745.46	7698.30	9520.02

Table 21. Results for case study 2: average test duration

Analysis of the results reveals an improvement of almost 30% when an off-line risk-based test sequencing with parallel diagnosis and fixing is used. Selection of the online, risk-cost-based test sequencing with parallel diagnosis and fixing improves the average test duration by 34%. However, the on-line test sequencing technique requires test sequencing to take place while executing test cases. The off-line test sequencing technique can be performed before test exe-

$\varphi_D + \varphi_F$	[Of/Ri/P]	*[Of/Ra/P]*	[Of/RC/S]	[Of/Ri/S]	[Of/O/P]	[Of/O/S]
90	0.096	*0.114*	1.779	1.798	0.175	4.638
180	0.097	*0.107*	1.818	1.771	0.175	4.640
360	0.096	*0.110*	1.747	1.784	0.175	4.582
720	0.098	*0.111*	1.735	1.811	0.175	4.675

$\varphi_D + \varphi_F$	[On/Ra/P]	[On/RC/P]	[On/Ri/P]
90	0.058	0.092	0.092
180	0.061	0.092	0.092
360	0.059	0.092	0.092
720	0.057	0.092	0.092

Table 22. Results for case study 2: Average risk

cution. It depends on the experience and knowledge of the test executor if the on-line test sequencing technique can be used. The average remaining risk for both methods is almost equal.

Case 3: Manufacturing system qualification

The manufacturing phase of a wafer scanner consists of two phases: the assembly phase and the test phase. The sub-systems are assembled in the assembly phase. The assembly phase is followed by the test phase, a test-diagnose-fix task. The system as a whole is tested and calibrated in this test-diagnose-fix task. This test-diagnose-fix task consists of a number of smaller test-diagnose-fix tasks, so called *job steps*. One particular job step has been investigated for this case study. This job step measures (tests) and calibrates (fixes) the performance of the alignment system. The alignment system of a wafer scanner aligns the mask and the wafer to each other. The required tolerance of the alignment results is between 4 and 15 nm [ASML, 2007], depending on the system type and specifications. The job step consists of 13 test cases and 29 fault states. The average failure probability is 37.6%. The density of this test model is 0.23. The system test model can be found in 39 in the Appendix.

Again the optimal test-diagnose-fix strategy is chosen and the diagnosis and fix duration is varied. The average diagnosis and fix duration in the test-diagnose-fix task of wafer scanners is $\varphi_D = \varphi_F = 70$ minutes. A parallel test configuration is used in all the experiments and testing is stopped when no risk remains in the system, i. e. $R_R = 0$.

The resulting average test durations for several test strategies are depicted in Table 23. The results of the reference test-diagnose-fix strategy *[Of/O/P]* are emphasized. Table 24 depicts the standard deviation and average remaining risk of the total test duration for the selected test strategies and diagnosis and fix durations. The test strategies that are not able to reach the test stop criterion of $R_R = 0$ are: [Of/Ra/P], [Of/Ri/S], [Of/O/S], [Of/RC/S] and [On/Ri/P]. The average remaining risk is not equal to 0 in these cases. The results of the test strategies that did not meet the required stop criterion are marked with paren-

$\varphi_D = \varphi_F$	[Of/Ra/P]	[Of/Ri/P]	[Of/iR/P]	[Of/IG/P]	[Of/Ri/S]
17.5	430.3	403.6	497.6	507.6	375.6
35.0	681.8	621.0	820.3	829.4	577.4
70.0	1182.3	1055.2	1461.5	1482.4	933.9
140.0	2182.1	1934.7	2760.8	2776.7	1684.0

$\varphi_D = \varphi_F$	*[Of/O/P]*	[Of/O/S]	[Of/RC/P]	[Of/RC/S]
17.5	*471.6*	371.4	400.5	365.8
35.0	*761.5*	537.2	616.4	535.4
70.0	*1344.5*	939.1	1045.6	914.3
140.0	*2481.3*	1577.5	1903.6	1641.8

$\varphi_D = \varphi_F$	[On/Ri/P]	[On/IG/P]	[On/Ra/P]	[On/RC/P]
17.5	2093.3	1017.1	658.6	1789.7
35.0	2190.1	1396.9	787.4	2092.8
70.0	2732.3	1826.4	1041.9	2470.5
140.0	3568.3	2819.2	1603.5	3373.7

Table 23. Results of case study 3: Test duration

theses and are not taken into account.

The results of this case study show that an improvement of the average test duration of more than 20% can be obtained by using a different test sequencing technique. The test process configuration should not be changed. Three test sequencing techniques lead to around 20% reduction of the average test duration for the normal case, where diagnosis and fixing problems both have a duration of 70.0 minutes. These test sequencing techniques are: off-line risk-based (-22%), off-line risk-cost-based (-22%) and on-line random (-23%) test sequencing. The on-line random test sequencing results in a slightly higher improvement of the average test duration when diagnosis and fixing costs 70 minutes. However, reducing the diagnosis and fix duration is not possible with this test sequencing technique, because reducing the diagnosis and fix duration results in an increase of the average test duration if compared with the normal test-diagnose-fix strategy. Furthermore, the standard deviation of the test duration for the on-line random test sequencing technique is much higher than the standard deviation of the average test duration of the normal case.

The standard deviation of the average test duration of the off-line risk-based and off-line risk-cost-based test sequencing techniques are similar to the standard deviation of the average test duration of the normal test-diagnose-fix strategy. The average test duration of the off-line risk-cost-based test-diagnose-fix strategy is slightly lower than the average test duration of the off-line risk-based test-diagnose-fix strategy.

$\varphi_D = \varphi_F$		([Of/Ra/P])	[Of/Ri/P]	[Of/iR/P]	[Of/IG/P]	([Of/Ri/S])
035.0	$\overline{\sigma}_\varphi$	(41.47)	31.9	30.8	30.4	(39.8)
070.0	$\overline{\sigma}_\varphi$	(78.68)	64.4	63.4	62.4	(108)
140.0	$\overline{\sigma}_\varphi$	(166.8)	122	122	126	(173)
280.0	$\overline{\sigma}_\varphi$	(319.9)	264	246	251	(380)

$\varphi_D = \varphi_F$		*[Of/O/P]*	([Of/O/S])	[Of/RC/P]	([Of/RC/S])
035.0	$\overline{\sigma}_\varphi$	*32.3*	(43.9)	30.7	(39)
070.0	$\overline{\sigma}_\varphi$	*64*	(88.1)	61.8	(76.5)
140.0	$\overline{\sigma}_\varphi$	*127*	(155)	119	(176)
280.0	$\overline{\sigma}_\varphi$	*256*	(275)	249	(299)

$\varphi_D = \varphi_F$		([On/Ri/P])	[On/IG/P]	[On/Ra/P]	[On/RC/P]
035.0	$\overline{\sigma}_\varphi$	(726)	278.5	228	502
070.0	$\overline{\sigma}_\varphi$	(526)	3109	239	616
140.0	$\overline{\sigma}_\varphi$	(731)	439.6	284	600
280.0	$\overline{\sigma}_\varphi$	(703)	492.3	402	685

Table 24. Results of case study 3: Standard deviation

4.2.5 *Conclusions*

A typical integration and test sequence for a complex manufacturing system consists of many test-diagnose-fix tasks between the integration (assembly) tasks. A tester selects a test-diagnose-fix strategy for a test-diagnose-fix task and executes the test-diagnose-fix task on the system under test. It is increasingly difficult to select the optimal test-diagnose-fix strategy when the number of components increases, the number of possible test cases increases and also the number of possible test-diagnose-fix sequences increases. A method is proposed that is able to analyze and compare sequences of test, diagnose and fix tasks, such that the best test-diagnose-fix sequence can be selected. The key performance indicators, duration, cost and remaining risk, are analyzed for a test-diagnose-fix sequence using this method. No specific mono-disciplinary knowledge is required for this analysis. This method is therefore applicable for test-diagnose-fix tasks at all levels in the integration and test sequence and even for manufacturing test-diagnose-fix tasks as shown in case study 3. Three case studies have been performed with this method. The goal of the case studies was to select the best test sequence for a test-diagnose-fix task and to investigate the effect of reducing the diagnosis and fix duration. The result of the first case study is a reduction of the average test duration of 71%, if an on-line, risk-based, parallel test sequence is used. The results of the second case study are: 1) a reduction of 30% by using an off-line risk-based test sequencing technique, 2) a reduction of 16% by reducing the diagnosis and fix duration by a factor two. The third case study results in an improvement of 54% in total test duration if an off-line risk-based test sequencing technique is used in combination with the reduction of the diagnosis and fix duration by a factor two.

4.3 PLANNING AND ANALYSIS OF RELIABILITY TEST-DIAGNOSE-FIX TASKS

This section is based on the paper 'Reliability qualification of semiconductor software releases' [I. de Jong et al., 11-12 June 2007] and describes the analysis of reliability test-diagnose-fix tasks. An uncertainty based test sequencing algorithm is used such that the duration of the test-diagnose-fix task can be reduced.

The reliability qualification period of a newly developed software release at ASML is planned according to SEMI standard E10-0600 [SEMI, 1986-2000]. The goal of the reliability qualification test-diagnose-fix task is to prove that the mean-time-between-failures (MTBF[1]) of a wafer scanner or software release is according to the agreed level. For this purpose, a reliability test-diagnose-fix task is planned. The SEMI-E10 standard defines the test duration that is required to reach a certain MTBF level with a certain level of confidence. One of the assumptions in the SEMI-E10 standard is that one system-level test case is used for reliability qualification. The standard contains a lookup table, Table A1-4, to determine the test duration. The values in the lookup table, the k-factors, are derived using an inverse χ^2 distribution. The k-factor that must be multiplied with the MTBF target to obtain the required test duration: test duration $= k \times MTBF_{target}$. Further details can be found in Section 7.6.5 and Table A1-4 in [SEMI, 1986-2000] and [NIST/SEMATECH, 2003-2006]. A higher test duration, due to increasing MTBF targets, results in an increase of the time-to-market of newly developed software releases.

Specific sub-system test cases could be beneficial if they utilize the subsystems much better than the system-level 'run production' test case. The benefit of specific sub-system test cases is that either the reliability target can be met in less time or a higher MTBF can be reached in the same amount of qualification time. Additionally, the sub-system test equipment is often much cheaper.

The problem with sub-system qualification is that the SEMI-E10 standard only supplies a test duration for a system-level qualification. Detailed methods from reliability engineering [NIST/SEMATECH, 2003-2006; Villacourt and Mahaney, 1994] are required to qualify the reliability at system level using subsystem test cases. These methods describe the components in the system with lifetime distributions. The structure of the system is used to combine the lifetime distributions into a system-level lifetime distribution. A failure distribution is used to model the effect of failures. The problem with this approach is the amount of work required to determine and maintain the system model.

This section introduces a simple and intuitive method for reliability qualification that can deal with sub-system test cases and actual failures. Section 4.3.1 describes the reliability qualification method as currently practiced by ASML and the normal reliability engineering approach that is capable of dealing with specific sub-system test cases. Section 4.3.2 introduces our method consisting

[1] MTBF is defined in SEMI-10 in Section 6.2.2 as the mean time between failures measured by productive time, $MTBF_p$

of a reliability test model and the method to determine the uncertainty about the MTBF on system level. Furthermore, a test sequencing algorithm and the effect of failing test cases and applying fixes to the system are described. Section 4.3.3 describes two performed cases including the results. Conclusions are presented in Section 4.3.4.

4.3.1 *Current method*

The current reliability qualification method at ASML is planned according to the SEMI-E10 standard. A target MTBF is chosen for a specific software release. The confidence level of 80% and five failures allowed in the qualification period are chosen based on previous results. The resulting k-factor is 7.9. The test cases are planned on one or more wafer scanners and the execution is monitored. A number of test cases is selected for reliability qualification. The production settings from these test cases differ from each other. For instance, the size of the image and the exposure settings are varied to mimic typical production settings. The reliability test cases are executed during the night and the results are gathered and analyzed to facilitate the diagnosis and fixing of reliability failures.

An extensive in-house alpha test and a beta test at selected customers are executed before new software is released to customers worldwide. The reliability qualification period starts with the start of the alpha test and continues throughout the beta test until the required target is met. The reliability target for consolidation releases is higher compared to software releases that are used for newly developed wafer scanners, because these consolidation releases replace the software on all systems worldwide. The duration of the reliability qualification period is therefore also higher. A reliability qualification period of a few weeks is planned for a new consolidation release. Functional test cases are performed during the normal operating hours and reliability qualification is performed during the nights. Nowadays, the duration of the reliability qualification becomes the bottleneck, i. e. the reliability qualification duration becomes longer than the available test time during the nights. Extending reliability qualification during the days would increase the time-to-market. Additional test resources could be assigned to prevent that the reliability qualification period is the bottleneck. The approach presented in this section increases the confidence on sub-system level faster, such that the system level confidence target is reached earlier.

Reliability engineering theory

The normal approach from reliability engineering [NIST/SEMATECH, 2003-2006] enables the use of sub-system test cases by the following approach. The components in the system and the interfaces between these components are identified. The components are connected *in series*, *in parallel*, *R out of N* or using more complex structures, which are combinations of the serial and parallel component structures. This structural information is then used to determine

the reliability equation. A lifetime distribution is chosen for each of the components from typical lifetime distributions, e. g.: exponential, Weibull, extreme value, log-normal, gamma, Birnbaum-Saunders or proportional hazards.

The next step is to estimate the parameters of the lifetime distributions. Typical parameters are the failure rate for an exponential distribution or shape and scale parameters for other distributions like the gamma and Weibull distribution. These parameters are used together with the structural information to calculate the system lifetime distribution. The duration required for system reliability qualification can be determined by using a χ^2 test on the system lifetime distribution. Determining the test duration at sub-system level and executing test cases at system level changes the lifetime distribution at sub-system level. Subsequently, the system level lifetime distribution is changed also, resulting in a shorter test duration at system level.

The approach to incorporate the effect of failures is based on the combination of two distributions. A system lifetime distribution is derived and its parameters are estimated. Additionally, an exponential distribution describing the failure rate is chosen. The combination of the two distributions leads to a new distribution, which incorporates the effect of the failure rate. This new distribution is then used to determine the test duration using the χ^2 test and a chosen confidence interval. This approach is described in [Villacourt and Mahaney, 1994] and [NIST/SEMATECH, 2003-2006].

The reliability theoretic approach is not used extensively for the reliability qualification of an ASML wafer scanner. The main reasons for not using the theoretical approach at system level are: the number of components is large, the many sub-system interactions are complex and the components change often. Furthermore, the multidisciplinary nature of the system requires the use of lifetime distributions of multiple types. The resulting reliability equation, probability density function and test duration are impossible to determine within the available time frame and budget. Simple models are required to bridge the gap between the high level SEMI standard and the low-level reliability engineering approach.

4.3.2 *Proposed method*

The proposed method uses a *reliability test model* to model the system and sub-system test cases. This model is used to determine a test sequence in which the reliability confidence is reached as fast as possible. The confidence as used in the SEMI-E10 standard is modeled as uncertainty per sub-system. The uncertainty on the system level is determined using the sub-system uncertainties. The effect of an executed test case (*passed* or *failed*) is fed back into this test model. The system test model, introduced in Chapter 2, is used for planning and analysis of reliability test-diagnose-fix tasks. An example matrix representation of a system test model (D), including the properties per fault state and test, is given in Table 25.

The relation between tests and fault states (R_{ts}) is represented by a 0 if a test

S / T	t_0	t_1	t_2	t_3	t_4	t_5	U
s_1	1/100	2/100	0	0	1.5/100	0	1.0
s_2	1/100	0	2/100	0	2/100	1.5/100	1.0
s_3	1/100	0	0	3/100	0	2/100	1.0
s_4	1/100	0	0	0	1.5/100	0	1.0
s_5	1/100	0	0	0	0	1.5/100	1.0
C_T	1	1	1	1	1	1	
φ_T	1	1	1	1	1	1	

Table 25. A reliability test model for the telephone example

does not cover the fault state. Otherwise, the coverage is larger than 0. In this example, the coverage is $\frac{1}{100}$, where 100 represents the target MTBF for the telephone. Test t_0 covers one hundredth of the reliability target if the test is executed. Note that for this example, the duration of each test is 1 hour indicating that 1 hour of testing covers $\frac{1}{100}$ of the reliability target.

The phone system can also be modeled as a 'single test-single fault state' test model as depicted in Table 26. This reliability test model describes the phone and the single 'run production' test case as required by the SEMI E10 standard. Test t_0, the 'run production' test, is the same test in both models. A single fault state s_0 describes that the reliability confidence target is not met. Note that the initial uncertainty is 1.0 when no test cases have been performed.

S / T	t_0	U
s_0	1/100	1.0
C_T	1	
φ_T	1	

Table 26. SEMI E10 modeled telephone example using a single test and a single fault state

Remaining uncertainty

The calculation of the *remaining uncertainty* U_R is straightforward for the 'single test-single fault state' reliability test model depicted in Table 26. The remaining uncertainty is equal to the uncertainty of the only fault state in the model, s_0. For a model with a set of fault states, the executed test sequence G is used to derive the uncertainty per fault state $U(s, G)$ after executing the test cases in G. Equation (4.33) is used to calculate the remaining uncertainty.

$$U_R(G) = \frac{1}{|S|} \sum_{s \in S} U(s, G) \tag{4.33}$$

In other words, the system-level uncertainty is the average of the fault state uncertainties. Both models must be modeled such that the coverage of test t_0

on fault state s_0 in the system model is the same as the coverage of t_0 on each of the fault states in the sub-system model, because both models represent the same system and test case. The rest of the test cases in the sub-system model can now be modeled *relative* to the test case t_0. For instance, test t_3 tests reliability fault state s_3 in the handset. This test case has a coverage that is three times higher than the system level test t_0, because the specific test case performs three times the number of cycles in the same time if compared with test t_0. Another reason could be that the coverage of the test case t_3 is three times higher than the coverage of test t_0.

Uncertainty reduction by executing a test

The uncertainty for fault state s decreases when the next test case t after sequence G *passes*. The uncertainty is calculated using Equation (4.35).

$$U(s, \epsilon) = 1.0 \tag{4.34}$$

$$\begin{aligned} U(s, Gt) &= U(s, G) - R_{ts}(t, s)\, U(s, G) \\ &= U(s, G)(1 - R_{ts}(t, s)) \end{aligned} \tag{4.35}$$

The initial uncertainty of a fault state as described in the system reliability model in Chapter 2 is $U(s, \epsilon) = U(s)$. The sequence G is empty in that case. Equation (4.36) describes the remaining uncertainty when a sequence G of test cases has been executed. Equation (4.37) describes the special case when the same test case is executed n times, i. e. the sequence G contains one test n times, $n = |G|$ and $G = t^n$.

$$U(s, Gt) = \prod_{t \in G}(1 - R_{ts}(t, s)) \quad \text{if } t \textit{ passes} \tag{4.36}$$

$$= (1 - R_{ts}(t, s))^n \quad \text{if } t \textit{ passes} \tag{4.37}$$

Equation (4.37) is the equation used for the SEMI reliability test model in Table 26. A *passed* test case results in an uncertainty reduction according to Equation (4.36). A *failed* test case on the other hand results in a diagnosis and a fix task. In the diagnosis task, the root cause of the failing test case is determined. A solution for the failure is developed and applied on the system. The duration of this diagnosis and fix task can range from minutes to weeks or longer. A loose cable can be diagnosed and connected within a few minutes, while a failure due to a faulty hardware design can take up to weeks to be fixed. A *failed* test case increases the uncertainty, because it is unknown at the moment of failure what the effect of the failure is on the overall system reliability. The uncertainty of the sub-system where the failure occurred is reset to 1.0:

$$U(s, Gt) = 1.0 \quad \text{if } t \textit{ fails} \tag{4.38}$$

Comparing SEMI-E10 with our method

Appendix A in the SEMI standard describes the method to determine the required test duration to reach a confidence level α. The system reliability model as depicted in Table 26 should lead to the same results. These results are shown below. In our method, the confidence level α is modeled using the remaining remaining uncertainty: $\alpha = 1 - U_R(G)$. The length of test sequence G can be determined for the special case, where the reliability test model contains a single test and a single fault state by rewriting Equation (4.37) into $|G| = \log_{(1-R_{ts}(t,s))} U_{target}$. Note that the duration of test sequence G depends on the target uncertainty U_{target}. The length of the test sequence can be calculated for the telephone system model in Table 26. The uncertainty target is 0.2 and corresponds with an 80% confidence limit. The length of the test sequence is $|G| = \log_{(1-\frac{1}{100})} 0.2 = 160.13$. The duration of a test case is 1 hour, so 161 hours of testing is required to reach the target of 20% uncertainty. The lookup value for the k-factor in the SEMI standard for the confidence level of 80% is 1.61. This value needs to be multiplied with the reliability target of 100 hours and also results in 161 hours of testing and is equal to our result. The details of the comparison of the two methods has been described in Appendix A, including the (numerical) proof that both methods yield the same results.

In our method, the confidence level α is modeled using the remaining remaining uncertainty: $\alpha = 1 - U_R(G)$.

The objective for the general case, where a model contains a number of test cases and a number of fault states, is to find an optimal sequence G^* with the minimal duration Φ^* from all possible sequences $\mathcal{G}$, such that the uncertainty target $1 - \alpha$ is met:

$$\begin{aligned} \Phi^* = \sum_{t \in G^*} \varphi_T(t) = \quad & \min_{G \in \mathcal{G}} \sum_{t \in G} \varphi_T(t) \\ s.t. \quad & U_R(G) = 1 - \alpha \end{aligned} \tag{4.39}$$

Algorithms to solve this test sequencing problem are described in [Boumen et al., Jan. 2008, 2006b]. These algorithms can be used to obtain an optimal test sequence G^* that meets the uncertainty target as soon as possible. These optimal algorithms are computationally intensive for large models, because both the *pass* and *fail* result are taken into account for each test case in the sequence. Risk is used as optimization criterion in these algorithms. Below, a more simple, less computationally intensive, algorithm is described that uses uncertainty as optimization criterion. Only *pass* results are taken into account. Failing test cases are handled by updating the uncertainties and recalculating the test sequence.

Sequence test cases

The goal of the *sequencing step* is to obtain a test sequence that reaches the required remaining uncertainty as quickly as possible. The sequencing method uses a reliability test model as input and the resulting test sequence G can be executed.

The goal of the *sequencing step* is to obtain a test sequence that reaches the required remaining uncertainty as quickly as possible.

Several general purpose test sequencing techniques are known in literature and practice, see [Rothermel and Harrold, 1996] [Harrold et al., 2001] [Amland,

2000] for some examples in the software domain. For example, risk-based testing results in a test sequence where the test cases are ordered such that the highest amount of risk is reduced first. The goal of reliability qualification is to reduce the uncertainty and not the risk. Moreover, a specific test case can be executed more than once to reduce the uncertainty. Hence, simple risk-based ordering is not sufficient. The sequencing method described below selects the test case such that the overall remaining uncertainty is reduced as much as possible. This procedure is repeated until the target uncertainty is reached. A single sequence is determined as opposed to the optimal sequencing algorithm in [Boumen et al., Jan. 2008, 2006b] where all possible test sequences are evaluated. The test sequencing algorithm is depicted in a flow chart in Figure 52.

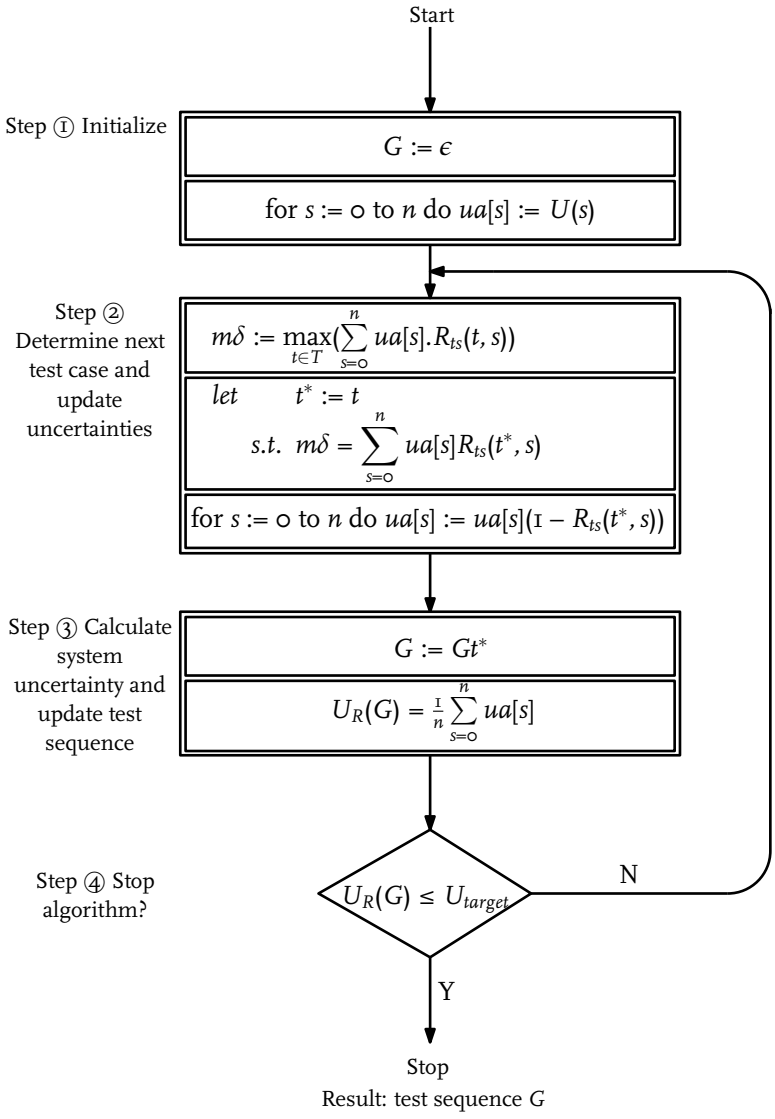


Figure 52. Flowchart of the test sequencing algorithm

Step ① is the initialization step. The test sequence G is initialized to an empty sequence and all uncertainties in ua are set to the uncertainties in U. The next test in the test sequence is determined in Step ②. First the delta uncertainty is calculated for all test cases in the test set T. Then, the test case that reduces the remaining uncertainty U_R the most is chosen as the next test case in the sequence, t^*. Step ③ updates the uncertainty of the fault states in the test model using the selected test case t^*. Then, the test sequence G is updated with the selected test case and the remaining uncertainty, U_R, is calculated using the updated fault state uncertainties.

Step ④ checks if the stop criterion is reached. The test sequencing algorithm continues if the stop criterion is not reached, otherwise the algorithm terminates resulting in a test sequence G.

Execute test sequence and update test model

With the method described above, sub-system test cases can be selected and executed. The selection of sub-system test cases leads to either a reduction of test duration or a reduction of risk in the same amount of test time. This effect is illustrated by Figure 53. Figure 53 depicts the uncertainty reduction by executing test cases (that all pass). The SEMI-E10 line depicts the normal uncertainty reduction using a system level test case. The sub-system modeling line depicts the usage of the sub-system test cases that are sequenced according to the described method. The target uncertainty is 20%.

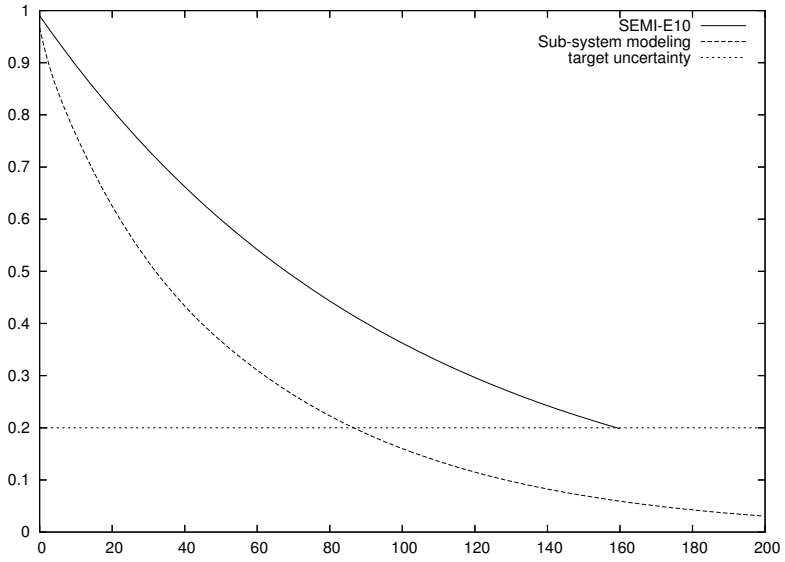


Figure 53. Uncertainty reduction leading to reduction of test duration or risk reduction

If the target uncertainty is to be reached faster, then the test duration is reduced by roughly 50% in this example. If more uncertainty is to be reduced in the SEMI-E10 test duration, then an uncertainty reduction of roughly a factor

three is obtained. It depends on the goal of the project as to whether a reduction of the test duration or an additional uncertainty reduction is required.

Assumptions

The assumptions in our method are:

- The failure rate is not taken into account, while the individual failure rates of sub-systems are directly related to the system reliability. It is assumed that sub-systems with higher failure rates will cause failures during testing and by this the uncertainty is increased. An increase of the uncertainty results in an increase of the test duration.
- The sub-system uncertainties are averaged to obtain the system level uncertainty according to Equation (4.33). This approach assumes that the probability density function of the uncertainty of each of the fault states is normal shaped. This is the case if the uncertainties of the fault states are reset due to failing test cases in the reliability qualification period. Otherwise, the uncertainty reduction follows an exponential decay function for which the average uncertainty is higher than the expected uncertainty. In this case, the uncertainty used to calculate the remaining uncertainty is too pessimistic.
- The system structure is not explicitly taken into account in this model. It is assumed that the test cases that are used in the model cover all aspects of the system that affect the system reliability. Next to that, a number of properties for the test cases and fault states in the model must hold:
  - All test cases should be unique. I. e., no two test cases may have the same coverage: $R_t(t_1) \neq R_t(t_2)$, where, $R_t(t) = \{s | s \in S \wedge R_{ts}(t, s) \neq 0\}$. Two test cases with the same coverage are either the same test case or a fault state is missing that makes these two test cases unique. The latter is often the case and adding the fault state makes the model more complete.
  - All fault states should be unique. I. e. no two fault states may have the same coverage: $R_s(s_1) \neq R_s(s_2)$, where, $R_s(s) = \{t | t \in T \wedge R_{ts}(t, s) \neq 0\}$. Two fault states with the same coverage describe the same subsystem. Two test cases should be added, in case of two equal subsystems that are present in the system. One testing the first subsystem and the other testing the second sub-system. This way, the reliability of both sub-systems can be qualified separately of each other.
  - The model does not contain fault states that are not covered by a test: $R_s(s) \neq \emptyset$ (for all $s \in S$). Fault states that are not covered by test cases are either not relevant for the system reliability or test cases must be added to cover these fault states.
  - The model does not contain test cases covering no fault state: $R_s(t) \neq \emptyset$ (for all $t \in T$). Test cases that do not cover fault states are

either not relevant or fault states must be added that are covered by these test cases.

4.3.3 *Case: software release qualification*

Two case studies have been performed using the described modeling and sequencing method. The first case study has been performed in addition to the normal SEMI approach. The second case study has been performed to monitor the reliability during the development of the new software release.

Case 1: Reliability qualification phase

The goal of the first case study was to decrease the uncertainty of the system reliability by executing additional test cases. Additional sub-system test cases have been selected using the sub-system reliability model of the system. The model used consists of 15 test cases and 12 sub-systems modeled as fault states. The reliability test model is depicted in Table 44. Every test case has a duration of six hours, since six hours are available for reliability qualification during the night. The initial uncertainty of all fault states is set to 1.0. Test t_1 is the standard 'run production' test case and all other test cases are modeled with test t_1 as the reference. The coverage of test t_1 on each of the fault states is $\frac{6}{150}$, where an MTBF-target of 150 hours is used together with the test duration of 6 hours.

The selected sub-system test cases have been executed in addition to the SEMI sequence. An additional 850 hours have been spent on testing and the uncertainty level at the end of the first phase was reduced more than the uncertainty reduction according to the SEMI-E10 approach. The result of the two approaches should be compared by performing the two corresponding test sequences on two equal independently developed systems. Then the number of issues found can be compared. This is too expensive in practice. Therefore, comparison is done by comparing the reliability hits of this software release and the previous release. The number of release specific issues, which are found at customers, is compared for this release and a previous software release. This comparison assumes that the quality of these two releases is the same.

The reliability Top-20[2] of the previous release contained eight software release specific MTBF failures. This Top-20 was created eight weeks after the software was released for customer installation. The eight release specific failures caused 60% of the MTBF failures. Two software release specific issues are found in the Top-20 of the release used for this case study. The Top-20 was created 12 weeks after the installation at customers started. The two release specific issues caused 11% of the MTBF failures in the twelve week measuring period. Factors other than the reliability qualification method probably also contributed to these results. The results and the simplicity of the modeling technique gave

[2] All lot aborts are registered in a database and related to a root cause. Root causes of the lot aborts are sorted according to the frequency of occurrence in 4 weeks. The 'Top-20' is the list of twenty root causes with the highest frequency in four weeks.

rise to the decision to continue this approach for reliability qualification of software releases.

Case 2: Reliability monitoring during development

The second case study has been performed to measure and monitor the uncertainty during the development of a new software release. The FMEA [Bowles, 19-22 Jan 1998] activity is succeeded by the reliability measurement program. Some of the failure modes need to be tested using a specific (sub-system) reliability test case. The resulting set of specific sub-system reliability test cases is put in a *test queue*. Two test systems were used to either execute the test queue or execute a multi-lot endurance test (MLET). The multi-lot endurance test consists

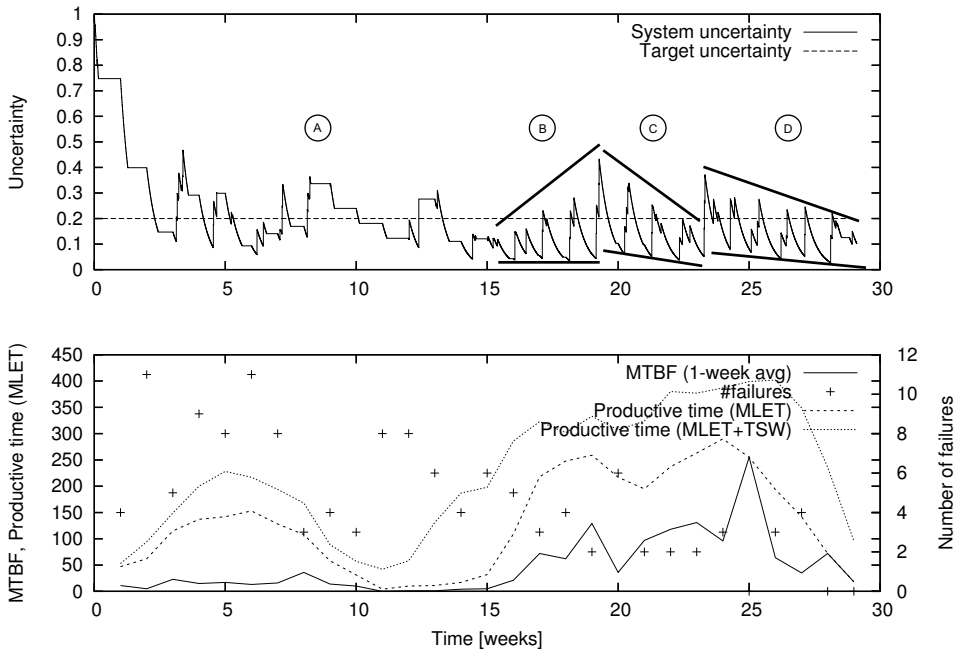


Figure 54. Uncertainty profile and MTBF during software development

of a set of test cases that together form the 'run production' test case. Whether the test engineer starts the test queue or the MLET-queue was not guided by a test sequence. This means that a test queue could be used for execution, while the MLET-queue would have reduced the uncertainty more and vice versa. A system reliability model of the involved sub-systems and the test cases has been defined. The model consists of nine fault states and twenty two test cases. The full model is defined in Table 45 in the Appendix. Note that the model is split up into two tables, since there are many test cases. The MLET duration and test-queue duration are recorded per week. The failures are recorded also each week, including the sub-system that caused the failure. These data is converted into

the format that can be used to determine the uncertainty profile. For this purpose, a sequence of test cases was created. Each hour of MLET testing results in a single MLET test (of one hour) as specified by the model by t_1. The sequence of the test queue is not recorded. However the set of test cases in the test queue is known and consists of t_2 to t_{22}. It is assumed that the duration of each test is one hour. In reality, some test cases have a longer duration and others have a shorter duration. Moreover, it is assumed that the test queue is restarted once it is finished. A sequence of test cases is created such that the recorded test queue duration is filled. The remainder of the time in the week is calculated and added as a single test with no coverage on fault states. This is necessary, because the data was recorded on a weekly basis and the uncertainty per week was determined. The failure data consist of a date and hour of failure and the sub-system causing the failure. The date and time are converted into hours since the start of the measurement, while the causing sub-system is mapped onto the related fault state in the reliability test model. An uncertainty profile, depicted in the upper graph in Figure 54, is calculated using the system test model, test sequence and failure data. The uncertainty profile is calculated for a period of 28 weeks. The initial uncertainty for all fault states is set to 1.0. Uncertainty is decreased when executed test cases result in a *passed* verdict. Test cases that result in a *failed* verdict reset the uncertainty for the covered fault states to 1.0. This behavior is reflected in the uncertainty graph by uncertainty reduction with passed test cases and jumps in uncertainty with failing test cases. The 'flat' areas in the graph indicate that no test cases have been executed. The bottom graph in Figure 54 depicts the productive time (MLET and MLET + test queue), the number of failures, indicated with a +, and the corresponding MTBF per week. The productive time of the test queues has not been taken into account for the MTBF calculation. Note that MTBF graph varies much due to the one-week average that is depicted. The other source of variation of the MTBF is the low productive time in certain weeks, that limits the MTBF. The following results have been obtained. Four distinct periods can be identified in the uncertainty profile. These periods are marked with Ⓐ, Ⓑ, Ⓒ and Ⓓ in the top graph of Figure 54.

- Period Ⓐ starts at the beginning of the measurement program and ends in week 15. This period is characterized by long flat areas where no testing takes place and some periods where the uncertainty is above the target uncertainty.

- Period Ⓑ shows an increase in peak uncertainties. The level of the uncertainty peaks becomes higher with each failure. Test execution is still efficient, since the minimal uncertainty levels are almost the same after each test period. In this period, the number of hits increases as well as the number of sub-systems causing these hits.

- Period Ⓒ shows a decrease in peak uncertainties and the minimal uncertainty level stays almost the same. The number of hits decreases in this period, while the productive time is high.

- Period Ⓓ shows a decrease in peak uncertainties and a decreasing minimal uncertainty. This is caused by the low number of failures (three weeks without failures) and the high number of test cases that have been executed in this period.

The analysis of the uncertainty profile can be used to adjust the test strategy for each of the four identified periods. Period Ⓐ can be improved by scheduling and executing more test cases, assuming that the required test resources are available. Period Ⓑ can be improved by speeding up the diagnosis and fix process. Period Ⓒ can be improved by selecting test cases with a high coverage on a specific fault state with a high uncertainty. Period Ⓓ could be optimized by executing only the MLET test case and only for a limited time per week. This can be done because the need to reduce much uncertainty does not exist any more.

4.3.4 *Conclusions*

The reliability qualification of ASML software releases is performed according to the SEMI-E10 standard. The required test duration is proportional to the target MTBF and the reliability qualification phase is on the critical path for software releases with high reliability requirements. The utilization of specific sub-system test cases could reduce the test duration or reduce the uncertainty of the system reliability. However, applying the current reliability engineering methods costs a lot of effort for a large, complex and changing system like a wafer scanner. This work introduces a method bridging the gap between the system level SEMI standard and the detailed reliability engineering methods. The model that is used to determine a reliability test sequence and duration is simple and intuitive. The method has been applied to two cases.

The first case study is related to the parallel execution of sub-system test cases during the reliability qualification phase for a software release. More than 850 hours of additional sub-system testing have been used, test cases have failed and problems were solved. The new software release has been rolled out on wafer scanners world wide. The number of failures caused by the new software release was 11% of the Top-20 failure count after 12 weeks. The number of failures caused by the previous release in the Top-20 after 8 weeks was 60%. This results from a combination of factors including the execution of additional and specific test cases.

The second case study concerns the monitoring of the uncertainty of the reliability during the development phase of a new software release. The main result of this case study is that the uncertainty can be monitored during development and the test plan can be adjusted using the monitoring results. Specific periods can be identified using the uncertainty profile and specific actions are proposed to improve the test plan. The advantage of this method is that a simple model is used to describe the system. The method to determine the remaining uncertainty is also straightforward, at the cost of losing part of the expressiveness of a full reliability engineering model.

5 IMPROVING INTEGRATION AND TEST SEQUENCES

This chapter describes the improvement step of the integration and test planning method. The improvement step currently consists of four techniques: test partitioning, development of new test cases, updating the constraints and objectives and redesigning the system. Figure 55 describes the selection step (4.1), where an improvement technique is selected, and four improvement techniques. The results of these four improvement techniques are fed back into the other three steps of the integration and test planning method. The feedback loops, starting as output of the improvement techniques, result in: another integration and test sequence (2.D), additional test cases (1.B), another integration model (1.A) or a change in the system architecture (0.A). Each of the improvement techniques is introduced in the next sections.

The main goal of partitioning a test-diagnose-fix task (4.2) is the reduction of the test duration by executing two resulting test-diagnose-fix tasks in parallel.

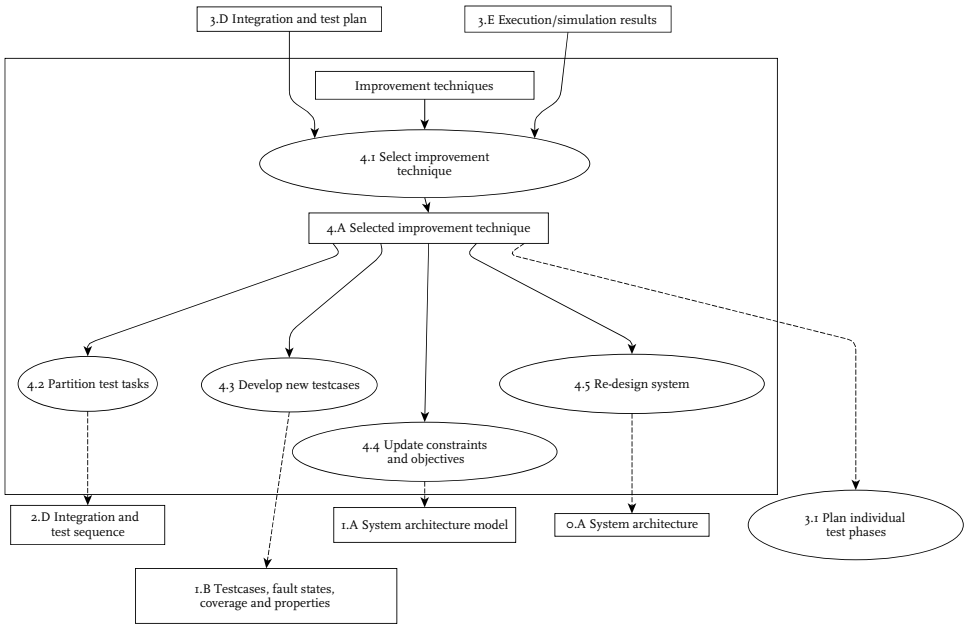


Figure 55. Overview of the improvement step

5.1 PARTITIONING TEST-DIAGNOSE-FIX TASKS

The next section is based on [I. de Jong et al., 2007b].

Parallel execution of test cases on two test systems can be the only option to meet the deadline in a time-to-market driven development environment, like the development of a new type of ASML wafer scanner. Purchasing an additional test system is considered for each test-diagnose-fix task on the critical path. Parallel execution reduces the duration of a test-diagnose-fix task with approximately 50%, depending on the duration of the individual test cases.

Dividing test cases over two test-diagnose-fix tasks can be done manually for small and simple test sets or with existing bin-packing algorithms [Coffman Jr. et al., 1997] for larger test sets with different test durations per test case. The overlap of test cases is not taken into account in these algorithms, only the test duration per test case is taken into account. It is shown in [Boumen et al., Jan. 2008] that this coverage information can be beneficial in terms of total test duration by selecting the optimal test sequence. The reduction of test time can be up to 20 – 30% if the coverage info is taken into account to determine a better test sequence. Executing a test-diagnose-fix task in parallel itself leads to a reduction of the duration of roughly 50%. An additional duration reduction of 20 – 30% per executed test-diagnose-fix task in parallel is expected, when the coverage information from the test cases is used. The total expected reduction of parallelizing test-diagnose-fix tasks in combination with the usage of the coverage information is therefore in the range of 60 – 65%.

A set of test cases can be divided into two smaller sets of test cases, such that these smaller sets can be executed in parallel or in series.

A set of test cases can be divided into two smaller sets of test cases, such that these smaller sets can be executed in parallel or in series. The advantage of executing two sets in parallel is that the duration of the test-diagnose-fix task can be reduced by more than a factor two. Dividing the test cases into two sets of test cases that are executed in series can be used as a heuristic for a sequencing algorithm as introduced in [Boumen et al., Jan. 2008]. Only parallel execution is considered in this paper.

A simple and straightforward division of test cases for parallel execution takes only the test duration into account. Test cases are divided such that two sets of test cases are obtained and the maximum duration of the two test sets of test cases is minimal. The duration of the set of test cases is determined by summing up the duration of the individual test cases in each test set. Several algorithms are available to solve this problem. However, the resulting test sets are equal in test duration if and only if all test cases result in a *pass* verdict, no faults are present in the system and no diagnosis and fix tasks are required. In practice, test cases *fail* and diagnosis and fixes are required. The diagnosis and fix duration, and the test sequence, influence the test duration. In fact, the total test duration Φ of a test set depends on the test cases, the test sequence, the actual set of problems in the system under test and the test stop criteria [I. de Jong et al., 2007e]. Hence, these four aspects need to be taken into account while partitioning a set of test cases.

Graph partitioning algorithms are used in our approach to partition a set of test cases into smaller sets of test cases. To be able to use this approach, we need to translate a set of test cases into a *test graph*. This is done in two steps. First, the set of test cases is represented as a system test model where test cases are related to possible faults in the system and relevant properties for test cases and fault states are given. Second, the system test model is translated into a test graph. The test graph is partitioned using a multi-level hypergraph partitioning method. This method originates from the domain of parallel computing [Hendrickson and Kolda, 2000]. The objective of partitioning in the domain of parallel computing is reducing the duration of a large calculation. The durations of the executed tasks can be summed up. In the test domain, the cost (test duration or cost) depends on the test cases, the test sequence, the actual system under test and the test stop criterion. Consequently, the total test duration is not a summation of the test case durations, but requires a more advanced calculation. Therefore, cost estimators have been developed.

The test graph is partitioned using a multi-level hypergraph partitioning method. This method originates from the domain of parallel computing.

This section is organized as follows: Section 5.1.1 is a general introduction into graph partitioning. The graph partitioning section is followed by Section 5.1.2 on system test model partitioning, where the details on partitioning the system test model are introduced as well as the adaption to the available graph partitioning algorithm. The objective functions and estimators are explained in Section 5.1.3. An example and an industrial case study are presented in Sections 5.1.4 and 5.1.5. In Section 5.1.6 conclusions and future work are presented. The system test model that is used by the algorithm has been introduced in Chapter 2.

5.1.1 *Matrix partitioning*

Originally, partitioning algorithms were used to partition computing jobs over multiple processors. For this purpose, jobs and inter-job communication are represented using a sparse matrix. The sparse matrix is partitioned over the processors, such that the computing time and the communication of data between processors is minimized. Usually, matrix partitioning is formulated as a graph partitioning problem that is NP-hard [Garey and Johnson, 1976]. The standard approach to partitioning graphs is explained in [Hendrickson and Kolda, 2000], including the discussion of the shortcomings of this approach. Available software packages like PaToH [Ü. Çatalyürek and Aykanat, 2002] and hMetis [Karypis and Kumar, 1998; Karypis, 2002] implement the partitioning algorithm using heuristics to generate solutions that approach the optimal solution in reasonable time. The standard partitioning problem has been extended with bipartite graph models and hypergraph models. Bipartite graph models allow the modeling of problems where the initial vertices are different from the final vertices. Hypergraph models are a more general form of bipartite models [Hendrickson and Kolda, 2000].

A hypergraph $G = (V, H)$ consists of a set V of vertices and a set H of *hyperedges*. Each hyperedge consists of a subset of vertices. In this way, the dependencies are represented as hyperedges and the partitioning goal is to equally

distribute the vertices and minimizing the hyperedges that cross the resulting partitions. The hypergraph model is used in this paper. Multi-level partitioning algorithms are proposed in the early 1990s by several researchers, such that large graphs could be partitioned faster. This approach has proven to be quite robust. The general idea behind the approach is as follows:

1. A large graph is grouped into a smaller graph, while preserving the essential properties of the original graph. This process is repeated up to a certain level.
2. The smallest graph is partitioned using a partitioning algorithm.
3. The partition is propagated back through the sequence of larger graphs and being refined along the way.

PaToH, hMetis and Mondriaan are three graph partitioning algorithms that follow this hypergraph approach. The performance of these algorithms are compared using the Rutherford-Boeing sparse matrix Collection [Duff et al., 1997] in [Riyavong, 2003]. The purpose of each algorithm is different, while the general performance of the algorithms is comparable. In our case, the generic hypergraph partitioning algorithm is applied for partitioning test-diagnose-fix tasks. For this purpose, the test problem needs to be transformed into a hypergraph model. Second, a specific objective function needs to be defined and implemented using one of the available implementations. The following sections describe the chosen model and algorithms for the partitioning of test-diagnose-fix tasks.

5.1.2 *System test model partitioning*

The approach taken for system test model partitioning is similar to the general partitioning approach. The relations between test cases and fault states are translated into a test graph. This test graph is coarsened into a hypergraph. This hypergraph is partitioned using a local search partitioning algorithm with a dynamic objective function. The reason for the dynamic objective function is that the cost of the test-diagnose-fix task in the partition depends on the coverage of the test cases in that partition. Finally, the matrix is uncoarsened at each coarsening level. The details of each step in this process are explained below.

The system test model represented as test graph

The edges in graph G represent the relations between test cases and fault states.

The system test model D is translated into a test graph $G = (V, E)$. The vertices in the test graph are the test cases from the system test model:

$$V = T \tag{5.1}$$

The edges in graph G represent the relations between test cases and fault states. An edge is a pair of a test case and a fault state where the coverage of the test

case on a fault state is > 0. These pairs follow from the relation R_{ts}:

$$E = \{(t, s) \in T \times S \mid R_{ts}(t, s) > 0\} \tag{5.2}$$

Using these definitions, a mathematical formulation of a graph partitioning problem is as follows:

Given: A directed graph $G = (V, E)$, $K \in \mathbb{N}$, $L \in \mathbb{N}$
Find: A partition $P = (V_1, ..., V_L)$ of V with $|V_l| \leq K$ for each $1 \leq l \leq L$, with the minimal cost J_P^*.

The minimal cost J_P^* for L partitions that are run in parallel is the minimum of the maximum cost for each partition P in the set of all possible partitions $\mathcal{P}$:

$$J_P^* = \min_{P \in \mathcal{P}}(\max_{l \in L}(J_P(V_l))) \tag{5.3}$$

Graph partitioning algorithms are able to find partitions for traditional applications, like processor load balancing, while minimizing communication volume. The traditional graph partitioning algorithm is explained first, starting with graph coarsening.

Graph coarsening

The graph is coarsened into a hypergraph $\mathcal{H}$ consisting of a set of vertices $\mathcal{V}$ and a set of hyperedges (nets) $\mathcal{N}$: $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. A set of vertices $\mathcal{V}$ consists of vertices V_l. Each element v of vertex V_l is only present once. So, $V_l \cap V_j = \emptyset$ holds for each vertex in $\mathcal{V}$. All elements in the original set of vertices V can be found in the set of vertices $\mathcal{V}$: $\bigcup_l V_l = V$. The initial hypergraph $\mathcal{H}_0$ consists of all vertices and edges of the original test graph. The initial hypergraph is coarsened into a sequence of smaller hypergraphs $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{N}_1)$, $\mathcal{H}_2 = (\mathcal{V}_2, \mathcal{N}_2)$, ..., $\mathcal{H}_m = (\mathcal{V}_m, \mathcal{N}_m)$. Each coarsened hypergraph is smaller than the previous in the sequence: $|\mathcal{V}_0| > |\mathcal{V}_1| > |\mathcal{V}_2| > ... > |\mathcal{V}_m|$, where $\mathcal{V}_0$ is the set of uncoarsened vertices: level 0, $\mathcal{V}_1$ is the first coarsening level and so on. Coarsening is achieved by combining disjoint subsets of vertices from the hypergraph. This way, multiple vertices form a single vertex in the hypergraph on the next level. The (combined) vertices in the next level can be combined again into a new vertex for the next level. This process stops when the number of the vertices in the hypergraph is below a user defined limit or when the ratio between the previous number of vertices and the new number of vertices is below a user defined stop ratio. Graph coarsening is performed using a matching-based clustering technique described in [Ü. Çatalyürek and Aykanat, 1999] as *Heavy Connectivity Matching*. This technique combines highly connected vertices into combined vertices. Highly connected vertices for system test models are vertices where test cases and fault states are highly connected. The Heavy Connectivity Matching algorithm visits vertices in a random order. The highest net connectivity

is calculated for all unmatched vertices v in V and the vertices that are *adjacent* to v. Two vertices are *adjacent* if both vertices share the same net. A net is defined as a set of fault states in a test graph. Two vertices v and w share a net if $nets(v) \cap nets(w) \neq \emptyset$, where $nets(v)$ returns the set of nets connected to vertex v. Vertices that share the same net are merged into a vertex V_l. Vertex v remains single if no adjacent unmatched vertices exist for vertex v. The highest net connectivity is the maximal net connectivity: $|nets(v) \cap nets(w)|$. Other matching criteria and strategies can be used. See [Ü. Çatalyürek and Aykanat, 1999] for a strategy called *Heavy Connectivity Clustering*, that is similar to *Heavy Connectivity Matching*. A vertex in *Heavy Connectivity Matching* is joined with only one other vertex (or none). A vertex in the *Heavy Connectivity Clustering* strategy can be joined with another vertex or with another group of vertices. In this case, no singleton vertices remain at the end. Which vertex or group of vertices is selected for joining a vertex is determined by a selection criterion. The selection criterion for the *Heavy Connectivity Clustering* strategy is based on the connectivity of the newly formed cluster divided by the weight of the newly formed cluster. The weight resembles the number of vertices in the newly formed cluster. The division by the weight is required to avoid the polarization towards very large clusters. See [Ü. Çatalyürek and Aykanat, 1999] for details on the calculation of the weight and connectivity for the newly formed cluster.

Hypergraph partitioning

Hypergraph partitioning can be performed in one or two dimensions. hMeTiS and PaToH are two software packages that implement one dimensional partitioning algorithms [Ü. Çatalyürek and Aykanat, 1999, 2002]. hMeTis uses an initial (random or guided) partition that is refined by the Kerninghan-Lin (KL) algorithm [Kernighan and Lin, 1970] followed by the Fiduccia-Matheyses (FM) algorithm [Fiduccia and Mattheyses, 14-16 June 1982]: the HKLFM-algorithm (hypergraph Kerninghan-Lin Fiduccia-Matheyses algorithm. PaToH uses a Greedy Hypergraph Growing algorithm [Ü. Çatalyürek and Aykanat, 2002] for partitioning. A two-dimensional algorithm has been proposed by Vastenhouw and Bisseling [Vastenhouw and Bisseling, 2005] and implemented in the Mondriaan software package. The two-dimensional algorithm partitions the vertices as well as the edges in an alternating algorithm where first the vertices are partitioned followed by a partitioning of the edges (or the other way around). The two-dimensional algorithm uses the HKLFM algorithm for each partitioning step in each of the two directions. Mondriaan can also operate as a one dimensional partitioning algorithm. We use the HKLFM algorithm as partitioning algorithm and the Mondriaan implementation as starting point for test graph partitioning. The objective function, which is specific for test-diagnose-fix tasks, is developed using the Mondriaan implementation.

The two-dimensional algorithm partitions the vertices as well as the edges in an alternating algorithm where first the vertices are partitioned followed by a partitioning of the edges

The hypergraph Kerninghan-Lin Fiduccia-Matheyses (HKLFM) algorithm is a refinement algorithm. First an initial partition is randomly selected. This random initial partition is then refined using the HKLFM algorithm. A single pass of the KLFM algorithm visits all vertices in the graph. Two vertices from differ-

ent partitions are selected and the cost is calculated assuming that these two vertices were swapped between partitions. The two vertices that result in the maximal gain are selected and these two vertices are placed in each others partitions (the two vertices are swapped). This process repeats for all vertices that are not swapped and stops when no vertices are available for swapping. The entire process can be repeated with the resulting partitions as input. Typically, a small number of passes is required to converge. The algorithm can be restarted a number of times with a completely different randomly selected initial partition. A modification to the KL algorithm, proposed by Fiduccia-Matheyses, improves the run time significantly without decreasing its effectiveness. In this algorithm, only one vertex is moved between partitions with each pass instead of swapping pairs. The gain of moving a vertex is calculated for each vertex in both partitions. The vertex with the highest gain is moved to the other partition and then excluded for further movement in this pass. A vertex is only moved when it does not violate the imbalance constraint as set by the user. The pass ends when no vertices can be moved, that is when no vertices are available to move or no vertex is able to move because of violation of the imbalance constraint. Again, this process can be repeated a number of times with the previous partition as input, such that a better result is obtained.

Uncoarsening and refinement

The coarsened partitioned hypergraph is uncoarsened in this phase. Uncoarsening means that the coarsened vertex is projected back to a set of vertices. A refinement step is performed after each uncoarsening step. The refinement step is again performed by the HKLFM algorithm using the uncoarsened partitions. This uncoarsening process is repeated until no coarsened vertices are present in the graph, i. e. level 0 is reached.

Parameters in the HKLFM algorithm

A number of parameters can be set to control the HKLFM algorithm. These parameters are discussed in this subsection. The first parameter, the minimal number of vertices (test cases) in a hypergraph $V_{\min}$ controls the allowed depth of the coarsening phase. The number of vertices in a hypergraph should exceed the minimal number of vertices: $|V_l| > V_{\min}$. Coarsening is stopped when the minimal number of vertices is reached. More coarsening levels lead to more executions of the KLFM algorithm. The optimal setting depends on how connected the graph is.

The second parameter, the fraction of reduction of the number of vertices η_{red}, is used to stop coarsening if subsequent coarsening phases are not beneficial enough. Coarsening is not beneficial enough if the number of vertices between two coarsening phases, $l-1$ and l, is not reduced enough. This parameter is expressed as a fraction: $\frac{|\mathcal{V}_{l-1}|-|\mathcal{V}_l|}{|\mathcal{V}_l|} > \eta_{red}$. The default setting of $\eta_{red} = 0.2$ was sufficient in our experiments.

The third and fourth parameters stop the heavy connectivity matching clustering method. The third parameter, *MaxNrVtxInCluster*, stops clustering when the maximal number of vertices (test cases) in a cluster C is reached:

$$|C| \leq MaxNrVtxInCluster \tag{5.4}$$

This way, the size of the cluster can be controlled and also the depth of the coarsening phase (depending on how the graph is connected). The fourth parameter, *CMaxWghtFrac*, stops clustering when the weight of a cluster is reached:

$$\frac{\sum_{v \in C_j} \varphi(v)}{\sum_{v \in \mathcal{V}_l} \varphi(v)} \leq CMaxWghtFrac \tag{5.5}$$

Where, C_j is the j-th cluster in vertex $\mathcal{V}_l$ on the i-th coarsening level. The weight of a cluster is approximated by the sum of the test durations in the cluster. This way, the size of the cluster in terms test duration (weight) can be controlled such that equally sized clusters are obtained.

Partition cost and communication volume

The original partitioning algorithms are developed to distribute calculations over multiple processors such that the communication volume between processors is minimized and the load on each processor is balanced. A fixed weight for communication overhead is used and the amount of calculations determines the processor load. Communication volume for partitioning system test graphs is similar. The overlap (communication) between test-diagnose-fix tasks should be minimized, because this is beneficial for the duration, cost and remaining risk of the combined test-diagnose-fix task. The main difference between partitioning for parallel distribution of processor load and parallel execution of test cases is how the cost of a partition is determined. For the distribution of computations, the sum of the tasks is equal to the load on the processor. In other words, the tasks are assumed to be independent of each other. This is not the case for test cases. For example, if two test cases cover the same fault state and the first test case fails because this fault state is actually in the system, then the second test case will also fail. The two test cases therefore depend on each other. The risk reduction and remaining risk in a test-diagnose-fix task depends on the executed test sequence. Test cases with very limited coverage on the fault states in a partition require much more time to reach the remaining risk target or cannot meet the target at all. Hence, the cost of a partition is not a simple summation of the test duration.

The main difference between partitioning for parallel distribution of processor load and parallel execution of test cases is how the cost of a partition is determined.

The graph partitioning algorithm from literature (HKLFM using the Mondriaan implementation) has been adapted such that the dynamic objective function is used to determine the 'partition cost'. The Mondriaan toolset is used as implementation instead of PaToH or hMetis, because partitioning fault states could also be beneficial. This extension can be developed easily using Mondriaan, because the basis of Mondriaan is a bipartition algorithm. The developed objective

functions are explained in detail in the next section, followed by a small example in Section 5.1.4.

5.1.3 *Objective functions*

The objective function of the *combined test-diagnose-fix task* is expressed in terms of the test duration Φ_T the cost C_T and the remaining risk after testing $R_{R,T}$. This is also the case for the objective function of a partitioned test-diagnose-fix task as depicted in Figure 56. Figure 56 shows the test duration, cost and remaining risk of the two partitioned test-diagnose-fix tasks T_1 and T_2 and the test duration, cost and remaining risk of the combined test-diagnose-fix task T. The system is copied first in the *cpy* task, such that test cases can be executed in parallel in the *tdf* tasks. The results are assembled in the *asm* task and the integration and test sequence can be continued. This assembly task assembles the risk in the system and is required to determine the complete duration of the parallel test-diagnose-fix task. The objective function of a test-diagnose-fix

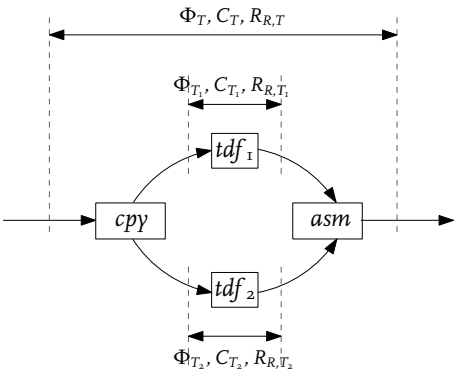


Figure 56. Key performance indicators of a partitioned test-diagnose-fix task

task is described as vector $\underline{c}$ in Equation (5.6). Each element in the cost vector $\underline{c}$ consists of two parts that are summed up. The first part is the expected value or maximum value of the test duration, cost or remaining risk. This way it is possible to optimize on the expected value or maximum value or combinations of these. The second part is a penalty function P used to penalize a solution which constraints are not met. Note that the solution is not excluded from the solution set to allow the algorithm to overcome local optima. The penalty function is defined in Equation (5.8) for the expected test duration. The penalty functions for the other parameters are defined in a similar fashion.

The objective function of the test-diagnose-fix task is described in terms of a weight vector $\underline{w}$. The elements of vector $\underline{w}$ in Equation (5.7) describe how important each of the performance measures are. The weight vector is later used in

the objective function.

$$\underline{c} = \begin{pmatrix} E(\Phi) + P_{E(\Phi)}, & \Phi_{max} + P_{\Phi_{max}}, \\ E(C) + P_{E(C)}, & C_{max} + P_{C_{max}}, \\ E(R_R) + P_{E(R_R)}, & R_{R,max} + P_{R_{R,max}} \end{pmatrix} \tag{5.6}$$

$$\underline{w} = (w_{E(\Phi)}, w_{\Phi_{max}}, w_{E(C)}, w_{C_{max}}, w_{E(R_R)}, w_{R_{R,max}}) \tag{5.7}$$

$$P_{(E_\Phi)} = \begin{cases} 10^5 (E(\Phi, target) - E(\Phi))^2 & \text{if } E(\Phi) > E(\Phi, target) \\ 0 & \text{if } E(\Phi) \leq E(\Phi, target) \end{cases} \tag{5.8}$$

The objective function of a single test-diagnose-fix task is defined according to Equation (5.9). The weight function typically contains 0's and a single 1 to indicate which performance indicator should be optimized. More complex weight vectors are possible, to accommodate a trade-off between performance indicators.

$$J = \underline{c} \cdot \underline{w}^T \tag{5.9}$$

The performance of the *combined test-diagnose-fix task* is also expressed in terms of test duration Φ_T, cost C_T and remaining risk $R_{R,T}$. The expected cost $E(C_T)$ is calculated using Equation (5.10).

$$E(C_T) = E(C_{T_1}) + E(C_{T_2}) \tag{5.10}$$

The test duration of the combined test-diagnose-fix task is the maximum of the test durations of the parallel test-diagnose-fix tasks:

$$\Phi_T = \max(\Phi_{T_1}, \Phi_{T_2}) \tag{5.11}$$

Note that this is a maximum of two discrete variables Φ_{T_1} and Φ_{T_2}. The maximal test duration of the combined test-diagnose-fix task is determined using Equation (5.12). Here, the maximum duration is the maximum of the maximal test duration of both parallel test-diagnose-fix tasks.

$$\Phi_{T,max} = \max(\Phi_{T_1,max}, \Phi_{T_2,max}) \tag{5.12}$$

The risk in the system is the summed product of the a-priory failure probability and impact for each fault state in the system:

$$R_{R,T} = \sum_{s \in S} P(s) I(s) \tag{5.13}$$

Executing test case t reduces the failure probability of a fault state s by $1 - R_{ts}(t, s)$. Therefore, executing test cases reduces the risk of certain fault states. The reduction of the failure probability due to test sequence G is expressed by:

$$\Delta_P(s, G) = 1 - \prod_{t \in G} (1 - R_{ts}(t, s)) \tag{5.14}$$

The calculation of the remaining risk in the combined test-diagnose-fix task is more involved, because a certain fault state can now be present in two (or more) parallel test-diagnose-fix tasks. Executing test cases in both parallel test-diagnose-fix tasks reduces the risk of the fault state in both test-diagnose-fix tasks. The reduction of the risk in both test-diagnose-fix tasks should be taken into account, because the combined remaining risk is lower, hence, the remaining risk stop criterion could be reached earlier. Therefore, the remaining risk of the combined test-diagnose-fix task depends on the fault states and the sequences executed in each of the parallel test-diagnose-fix tasks. The fault states in the parallel test-diagnose-fix tasks can be overlapping, i. e. one fault state can be covered by two test-diagnose-fix tasks. The set of fault states S is split up into subsets to determine the remaining risk and maximal remaining risk for the parallel test-diagnose-fix tasks. Three subsets can be distinguished for a test-diagnose-fix task with two parallel test-diagnose-fix tasks: S_1, S_2 and S_o. Where, the subset S_1 are the fault states that are covered in test-diagnose-fix task 1 and S_2 are the fault states covered in test-diagnose-fix task 2. The overlapping fault states, S_o, are the fault states that are covered in both test-diagnose-fix task 1 and 2: $S_o = S_1 \cap S_2$.

The executed test sequence is of importance, because the sequence determines how much risk is reduced during testing [I. de Jong et al., 2007e]. The remaining risk of the combined test-diagnose-fix task therefore depends on the test sequences of both parallel test-diagnose-fix tasks, G_1 and G_2, and the fault states in the partitions. The remaining risk for *two* parallel test-diagnose-fix tasks is:

$$\begin{aligned} R_R(G_1, G_2) &= \sum_{s \in S_1 \setminus S_o} \Delta_P(s, G_1) P(s) I(s) \\ &+ \sum_{s \in S_o} \Delta_P(s, G_1) \Delta_P(s, G_2) P(s) I(s) \\ &+ \sum_{s \in S_2 \setminus S_o} \Delta_P(s, G_2) P(s) I(s) \end{aligned} \tag{5.15}$$

Above, the test sequences of both test-diagnose-fix tasks are denoted by G_1 and G_2. The maximum remaining risk for a parallel test-diagnose-fix task i is calculated using Equation (5.16). All possible execution sequences $\mathcal{G}_i$ for test-diagnose-fix task i need to be taken into account to determine the maximal remaining risk for a test-diagnose-fix task. The sequence of test cases is fixed, however test cases can pass or fail resulting in a diagnosis of the problem and eventually a fix. Therefore, $\mathcal{G}_i$ represents all possible pass and fail sequences and this way the maximum remaining risk is calculated.

$$R_{R,max,i} = \max_{G_i \in \mathcal{G}_i} \sum_{s \in S_i} \Delta_P(s, G_i) P(s) I(s) \tag{5.16}$$

The maximal remaining risk of the combined test-diagnose-fix task is the maximum of the maximal remaining risk of the parallel test-diagnose-fix tasks if these parallel test-diagnose-fix tasks are independent. In general, this is not the case and parallel test-diagnose-fix tasks are dependent because of the overlap in

fault states. Determining the maximum remaining risk for the combined test-diagnose-fix task requires that all possible combinations of fault states in $\mathcal{P}(S)$ need to be taken into account for both sequences. To reduce the complexity of the calculation the maximal remaining risk for two test-diagnose-fix tasks is estimated using Equation (5.17).

$$\begin{aligned} R_{R,max} = & \max_{G_1 \in \mathcal{G}_1, G_2 \in \mathcal{G}_2} \Big(\\ & + \sum_{s \in S_1 \setminus S_o} \Delta_P(s, G_1) P(s) I(s) \\ & + \sum_{s \in S_o} \Delta_P(s, G_1) \Delta_P(s, G_2) P(s) I(s) \\ & + \sum_{s \in S_2 \setminus S_o} \Delta_P(s, G_2) P(s) I(s) \Big) \end{aligned} \tag{5.17}$$

Test stop criteria of the partitioned test-diagnose-fix tasks

The hypergraph partitioning algorithm splits a test-diagnose-fix task into two tasks and combines the cost vector of the two test-diagnose-fix tasks into a combined cost vector. The test stop criterion should still be met after combining the results of the partitioned test-diagnose-fix tasks. Meeting the test stop criterion for (maximal) cost can be assured by setting the stop criteria for the partitioned test-diagnose-fix tasks such that the sum of stop criteria equals the overall stop criterion. The stop criterion for (maximal) duration is obtained in a similar fashion. The stop criterion for duration for the partitioned tasks is equal to the overall stop criterion, because they are executed in parallel. The stop criterion for the remaining risk is more involved, because the remaining risk of the partitioned test-diagnose-fix tasks influences the overall remaining risk, because overlapping fault states are covered in more than one partitioned test-diagnose-fix task. Moreover, the effect of all test-diagnose-fix tasks on these overlapping fault states need to be taken into account to determine if the remaining risk stop criterion is met. The remaining risk of the combined test-diagnose-fix task needs to be determined after every step in the KLFM algorithm, to check if the stop criterion on system level is met using Equations (5.16) and (5.17). This is computationally intensive. Therefore, another approach is followed. The remaining risk stop criterion for the parallel test-diagnose-fix tasks is calculated in advance by multiplying the remaining risk stop criterion on system level with a remaining risk factor η_{R_R}. A suitable value for the remaining risk factor needs to be selected for this purpose. A remaining risk factor that is too low leads to less optimal but feasible partitions, while a remaining risk factor that is too high leads to infeasible partitions. The risk reduction in the partitions takes longer than required because a low risk factor is used. This way, the risk stop criterion is reached and feasible solutions are obtained. However, a better solution could be available. A high risk factor on the other hand, results in shorter durations for the partitions, so a better solution could be obtained. However, the combined risk could exceed the risk stop criterion, leading to infeasible solutions.

Estimated objective function

The idea behind the estimator of the test duration, cost and remaining risk is explained first, followed by the definitions. The interleaved test process configuration can be depicted as a directed binary graph. The initial state of this binary graph is the first test in the off-line test sequence. Each test can *pass* or *fail*. A passed test case results in the selection of the next test in the sequence. This results in the pass transition from the initial state. The failing test case is the second transition. This way, a binary directed graph is built up. A failing test case results in a diagnosis and fix task and the next test in the sequence is selected after diagnosis and fixing is completely finished. The directed binary graph of the interleaved test process configuration is depicted in Figure 57. A state is depicted as a circle, a transition as an edge and a leaf node as a triangle. A leaf node indicates that the stop criterion is reached. Multiple leaf nodes can exist, since the stop criterion could be reached via multiple paths through the binary graph. A state, depicted as a circle in Figure 57, is defined as a five-tuple

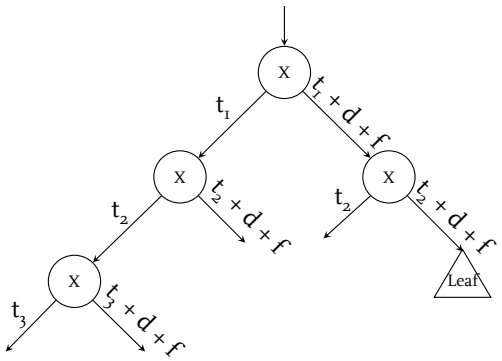


Figure 57. Directed binary graph representation of the interleaved test process configuration

$X = \left(p, \Delta_P, \varphi, c, r_R\right)$ with a domain $\mathbb{R} \times (S \to \mathbb{R}) \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}$. The elements in the state X are:

- p, the probability that the state is reached during the test-diagnose-fix task,
- Δ_P, the reduction of the failure probability of each fault state $s \in S$,
- φ, the time passed since the start of the test-diagnose-fix task,
- c, the cost of testing since the start of the test-diagnose-fix task,
- r_R, the remaining risk in the system.

A leaf node is reached if the following condition holds: $((\Phi > \Phi_{target}) \vee (c > C_{target}) \vee (r_R < R_{R,target}))$. The leaf nodes of the directed binary graph are used to

determine the expected and maximal cost, remaining risk and duration of the combined test-diagnose-fix task. Two transitions are possible from each state in the directed binary graph: a transition if a test passes and a transition if a test fails. The pass transition results in state $X_p(t, X)$ derived using Equation (5.18). A fail transition results in state $X_f(t, X)$ derived using Equation (5.19).

$$\begin{aligned} X_p(t, X) \quad = \quad & (p_p(t, \Delta_P)p, \delta_{pass}(t, \Delta_P), \varphi + \varphi_T(t), \\ & c + C_T(t), r_R(\delta_{pass}(t, \Delta_P))) \end{aligned} \tag{5.18}$$

$$\begin{aligned} X_f(t, X) \quad = \quad & ((1 - p_p(t, \Delta_P))p, \delta_{fail}(t, \Delta_P), \\ & \varphi + \varphi_T(t) + \varphi_D(t) + E(\varphi_{Fix}), \\ & c + C_T(t) + C_D(t) + E(C_{Fix}), \\ & \Delta r_R(\delta_{fail}(t, \Delta_P))) \end{aligned} \tag{5.19}$$

Both transitions result in a new state according to the pass or fail result of the test case. The transition of the passed test case uses the pass probability p_p of the test case, while the fail transition uses the fail probability $1 - p_p$. A different reduction of failure probability δ_{pass} and δ_{fail} is used accordingly. The other difference in the state transitions is the update of the duration and cost with the diagnosis and estimated fix duration for the fail transition due to the interleaved test process. The pass probability of a test t is defined according to Equation (5.20). The reduction of the failure probability for all fault states is given as argument δ to this function.

$$p_p(t, \delta) = \prod_{s \in S} (1 - \delta(s)P(s)) \tag{5.20}$$

The reduction in failure probability due to a passing test δ_{pass} is described as a function $\delta_{pass} : T \times (S \to \mathbb{R}) \to (S \to \mathbb{R})$ and defined as follows, for $s \in S$:

$$\delta_{pass}(t, \delta)(s) = \delta(s)(1 - R_{ts}(t, s)) \tag{5.21}$$

The remaining risk can be calculated using a certain reduction in failure probability δ according to Equation (5.22).

$$\Delta r_R(\delta) = \sum_{s \in S} \delta(s)P(s)I(s) \tag{5.22}$$

The reduction in failure probability due to a failing test δ_{fail} is described as follows. For $s \in S$:

$$\delta_{fail}(t, \delta)(s) = \begin{cases} \delta(s) & \text{if } R_{ts}(t, s) = 0 \\ 0.0 & \text{if } R_{ts}(t, s) > 0 \end{cases} \tag{5.23}$$

The expected fix duration, $E(\varphi_{Fix})$, is calculated using Equation (5.24). A number of fixes can be executed in parallel when a diagnosis finishes. The expected fix duration depends on the fix duration of each fault state, as well as the probability that the fix duration is spend. The fix duration of each fault state φ_F is known

from the system test model. The probability that the fix duration is spent p_{Fix} is calculated using Equation (5.25).

$$E(\varphi_{Fix}) = \sum_{s \in S} \varphi_F(s) p_{Fix}(s) \tag{5.24}$$

The idea behind the calculation of the fix duration probability p_{Fix} is that the fix duration is spent depending on the probability that the fault state is actually in the system and the probability that the fix duration of another fault state that is to be fixed in parallel takes longer. The probability that a fault state is actually in the system is $\Delta_P(s)P(s)$, while the probability that the fix duration of another fault state takes longer is determined using a sorted list of fault states that are to be fixed. A sorted list, S_{sorted}, is created with the fix duration and fault state as elements. The list is sorted such that the fault state with the highest fix duration is placed first, followed by the fault state with the second highest fix duration and so on. The probability that a fix duration is spend can be calculated for each $s \in S_{sorted}$:

The probability that a fault state is actually in the system is $\Delta_P(s)P(s)$, while the probability that the fix duration of another fault state takes longer is determined using a sorted list of fault states that are to be fixed.

$$p_{Fix}(s) = \frac{1}{1 - p_p} \Delta_P(s) P(s) \prod_{sl \in S_L} (1 - \Delta_P(sl) P(sl)) \tag{5.25}$$

Where, the list S_L contains the fault states that are already visited. The current element is added to S_L after $p_{Fix}(s)$ is determined.

A single recursive definition for the expected fix duration with signature $E(\varphi_{Fix}) : S_{sorted} \times S \rightarrow \mathbb{R}$ that replaces Equation (5.24) is given in Equation (5.26).

$$E(\varphi_{Fix})(S, S_L) = \begin{cases} 0 & \text{if } S = \emptyset \\ \varphi_F(S.0) \frac{1}{1-p_p} \Delta_P(S.0) P(S.0) & \text{if } S \neq \emptyset \\ \prod_{sl \in S_L} (1 - \Delta_P(sl) P(sl) + \\ E(\varphi_{Fix})(tl(S), S_L \cup \{S.0\}) \end{cases} \tag{5.26}$$

Where $S.0$ is the first element in the sorted list and $tl(S)$ returns the sorted list except the first element of the sorted list, i. e. the remainder of the sorted list or *tail*. The expected fix cost, $E(C_{Fix})$, is calculated using Equation (5.27).

$$E(C_{Fix}) = \sum_{s \in S} (1 - \Delta_P(s) P(s)) C_F(s) \tag{5.27}$$

A state in the binary graph represent, among the probability that the state is reached, the time passed since the start of the test-diagnose-fix task and the cost and remaining risk. The end states in the binary graph of nodes represent the states that reached the test stop criterion. The time, cost and remaining risk reached in these end states are used to determine the expected duration, cost and remaining risk for the entire test-diagnose-fix task. The set of all sequences leading to an end state in the binary graph $\mathcal{G}$ is used in Equations (5.16) and (5.15). The expected cost and duration of a single test-diagnose-fix task are calculated by taking the average duration and cost of all the sequences in $\mathcal{G}$.

S / T	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	P
s_1	1	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0.1
s_2	1	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0.1
s_3	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0.1
s_4	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0.1
s_5	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0.1
s_6	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0.1
s_7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0.1
s_8	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0.2
s_9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.1
s_{10}	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0.3
s_{11}	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0.1
s_{12}	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0.1
s_{13}	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0.1
s_{14}	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0.1
s_{15}	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0	0.01
s_{16}	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0.1
s_{17}	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	1	0	0.1
s_{18}	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	0.1
s_{19}	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	1	0.1
s_{20}	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0.1
s_{21}	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0.1
s_{22}	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0.1
C_T	3	1	1	1	2	2	5	3	3	5	5	3	1	1	1	2	2	

Table 27. System test model (relevant elements) of the example VOIP system

5.1.4 *Example*

An extension of the telephone example is used to illustrate the partitioning of system test models. The simple telephone system is extended with a Voice-Over-IP (VOIP) network and another telephone on the other side. So the example system consists of two telephones connected to each other via a VOIP system. The system test model is depicted in Table 27. The impact I of all fault states is set to 1.0. The diagnosis, fix duration and apply fix duration is set to 0.0 for all fault states and test cases. The test graph of this VOIP system is presented in Figure 58. Two types of nodes can be seen in Figure 58. The circular nodes represent test cases. The grey diamonds represent fault states. The a-priori fault state probability, P, is represented by the grey color. A darker tint of grey indicates that the failure probability is higher. The edges in Figure 58 represent the relations, R_{ts}, between test cases and fault states. The system test model of the VOIP system consists of 17 test cases with a total test duration of 41 time units. The test graph of the VOIP system is first parallelized with a bin-packing algorithm. The test duration is the only required parameter per test case. A result of bin-packing (bp) are the test sets T_1^{bp} and T_2^{bp} :

$$T_1^{bp} = \{4, 5, 6, 7, 9, 12, 13, 14, 17\} \tag{5.28}$$

$$T_2^{bp} = \{1, 2, 3, 8, 10, 11, 15, 16\} \tag{5.29}$$

The test duration of T_1^{bp} is the sum of all individual test cases, $\Phi_{T_1^{bp}} = \sum_{t \in T_1^{bp}} \varphi_T(t)$.

The test duration of T_2^{bp} is determined in a similar fashion. The duration of both test-diagnose-fix tasks executed in parallel is $\Phi^{bp} = max(\Phi_{T_1^{bp}}, \Phi_{T_2^{bp}})$. For

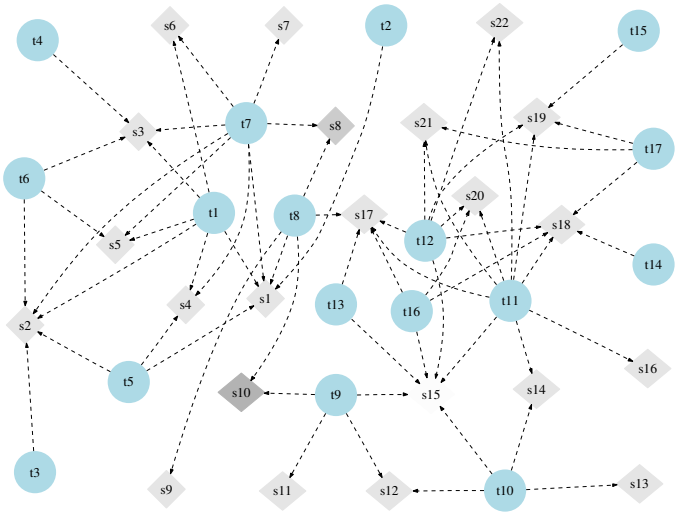


Figure 58. Test graph of the VOIP system

the VOIP system, $\Phi_{T_1^{bp}} = 20$ and $\Phi_{T_2^{bp}} = 21$, so the test duration of both test-diagnose-fix tasks executed in parallel is $\Phi^{bp} = 21$. A graphical overview of the test cases, fault states and relations in the VOIP system is depicted in Figure 59. The resulting test sets T_1^{bp} and T_2^{bp} are represented by darker and lighter circles respectively. The fault states and relations are presented for completeness purposes. These relations are not taken into account in the bin-packing algorithm. The dotted edges indicate that the connected fault state is covered by only one of the two test partitions. The black solid edges indicate that the connected fault state is covered by tests from two test partitions. This illustrates that the bin-packing algorithm does not take test coverage into account. This graph can be compared with the resulting graph of the test graph partitioning algorithm later on. It can be concluded based on Figure 59, that the bin-packing algorithm does not take the coverage of each test case and the timing related properties of each test and fault state into account, since the selected test cases for each test-diagnose-fix task are scattered. The overlap between the test-diagnose-fix tasks obtained via bin-packing is large, as can be seen by the solid lines in Figure 59.

The overlap between the test-diagnose-fix tasks obtained via bin-packing is large, as can be seen by the solid lines in Figure 59.

The adapted HKLFM algorithm does take the coverage and timing properties into account. The step by step adapted HKLFM can be visualized using the VOIP test model of Figure 58. First the test graph is coarsened, i. e. test cases are grouped together. Seven groups are created for the VOIP example: H_1, H_2, ..., H_7. These groups are graphically presented in Figure 60, where it can be seen that test cases are grouped with neighboring test cases.

The initial partitioning after coarsening is represented with darker and lighter circles. Initially, one partition consists of one test group with one test case (t_{15}), while the other partition contains six groups of test cases. Now, groups of test cases are moved to the other partition if this is beneficial for the total cost. The

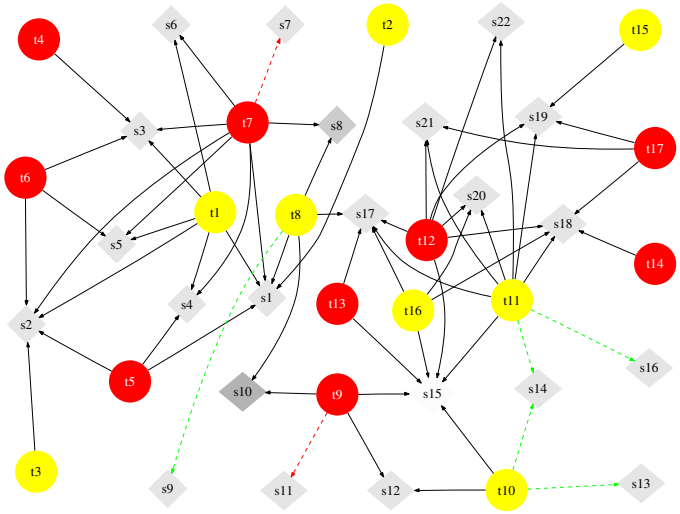


Figure 59. Test graph of the bin-packed VOIP system

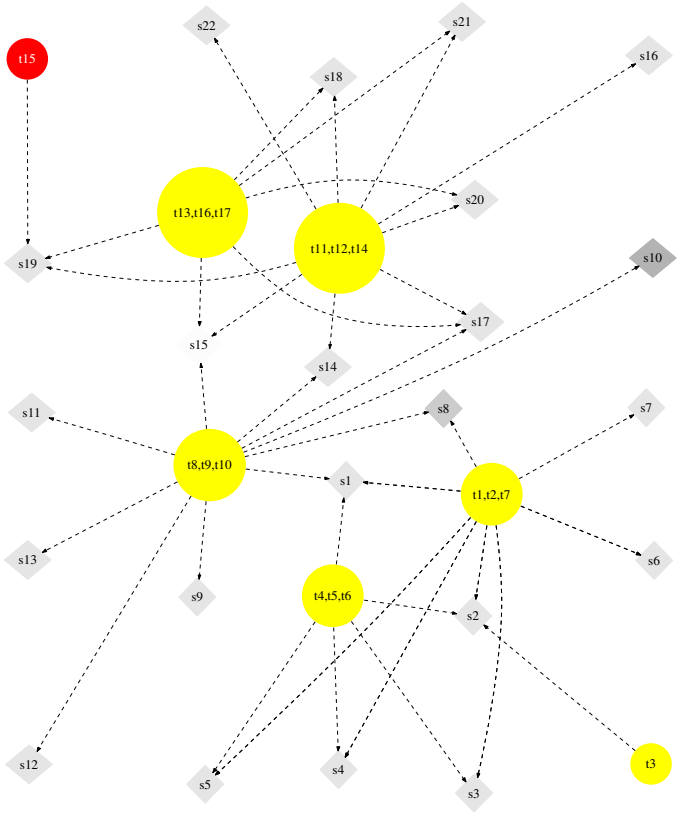


Figure 60. Coarsened test graph of the VOIP system

scheme that has been followed in this case is:

Step 1: Move t_3 to partition 2, $\Phi^{HKLFM} = 29.0$.
Step 2: Move test cluster with t_8,t_9,t_{10} to partition 2, $\Phi^{HKLFM} = 13.0$.
Step 3: Move t_{15} to partition 1, $\Phi^{HKLFM} = 12.0$.
Step 4: Move t_3 to partition 1, $\Phi^{HKLFM} = 11.0$.
Step 5: Trying the other test groups results in no benefit, so terminate.

Table 28. Partitioning scheme for the VOIP system

Then, the groups of test cases are uncoarsened into single test cases, such that partition $T_1 = \{1, 2, 3, 4, 5, 6, 7, 11, 12, 13, 14, 15, 16, 17\}$ and partition $T_2 = \{8, 9, 10\}$. The same HKLFM procedure is now repeated on the uncoarsened test cases. The resulting test case swaps are:

Step 1: Move t_{13} to partition 2, $\Phi^{HKLFM} = 11.0$.
Step 2: Move t_2 to partition 2, $\Phi^{HKLFM} = 11.0$.
Step 3: Trying the other test cases results in no benefit, so terminate.

Table 29. Uncoarsened partitioning scheme for the VOIP system

Step 2 is beneficial because the number of links between the test-diagnose-fix tasks is lower when t_2 is moved to partition 2. The total cost remains the same however.

The final result is depicted in Figure 61. The cost of these parallel test-diagnose-fix tasks is 11.0. If this result is compared with the initial bin-packing result, then a cost improvement of 47% is obtained for this example. If the resulting cost is compared with the initial (non-parallel) test set with total test cost of 41 time units, then an improvement of 73% is obtained.

5.1.5 *Case: Lot operations sub-system*

In this section, a case study is discussed where the test partitioning algorithm and proposed objective functions are applied. The case study is performed with the lot operations sub-system of an ASML wafer scanner. This sub-system consists of software only. The test-diagnose-fix task that is considered in this case can be executed on a standard SUN Solaris system without requiring additional hardware. Moreover, the fault states in this sub-system are configuration independent, because the same software can be copied and installed on more than one test system. A wafer scanner is used in the critical (lithographic) step in the manufacturing process of integrated circuits (IC's). The manufacturing execution system in an IC factory sends a so-called 'lot' to a wafer scanner. The lot describes the type of job that the wafer scanner needs to perform. These lots are queued in the lot operations (LO) sub-system. The LO sub-system then

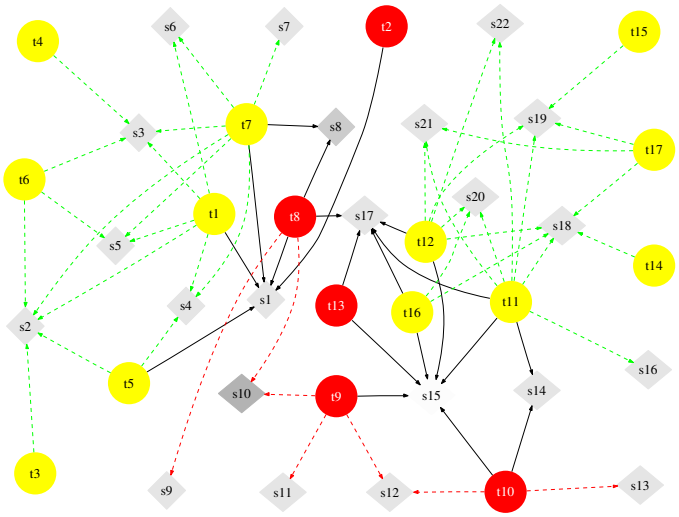


Figure 61. Test graph of the VOIP system partitioned by the adopted HKLFM algorithm

performs the required setup actions in the wafer scanner and waits until all required material is available to process the lot. The LO sub-system is the main controller of the wafer scanner.

ASML test engineers have defined a system test model of the lot operations sub-system with the following properties:

- Number of test cases: 44,
- Number of fault states: 31,
- Cumulative test duration: 56.3 [h],

The test cases that are described in the system test model are developed over a period of a few years. The test cases first developed test the normal operation of LO, while test cases developed in a later stage test additional functionality and special cases where problems were found over the years. The set of 44 test cases, 'test chapters', in the system test model contain a few up to hundreds automated test cases. The set of fault states in the system contains fault states for individual components in the system as well as fault states for sub-functions and special high-risk areas. The diagnosis cost and duration is set to 1 hours. This is also the case for the fix cost and duration. We assume here that a test engineer is present when the test cases are executed. Problems found are analyzed and fixed immediately and then testing is continued. This test process matches with the interleaved test process as discussed in Section 4.2. The test-diagnose-fix task that is partitioned is the final test of a redesign of the LO sub-system. The LO sub-system has been redesigned to accommodate future developments. Additionally, the internal design required an update, because the changes over time in the system resulted in a degradation of the current design. The test-diagnose-fix task is planned to test if the redesigned code still complies with the

normal expected behavior of the LO sub-system. This test-diagnose-fix task is parallelized to gain time-to-market. Cost is less important, because enough test resources and developers are available. Three groups of failure probability of the fault states can be distinguished: $P(s) > 80\%$ (4 fault states), $80\% \geq P(s) > 30\%$ (6 fault states) and $P(s) \leq 30\%$ (21 fault states).

Reference case

First, the duration of the single (non-parallel) test-diagnose-fix task has been determined for reference purposes, because currently all test cases are executed in a weekend, without interleaved diagnosis and fix activities. The current test duration is therefore 56.3 hours. A reference case, with interleaved diagnosis and fixing, needs to be compared with the parallel execution of test-diagnose-fix tasks. The test stop criterion is set to a remaining risk of 1.20, the interleaved test process and an off-line information gain-based test sequencing method are used. No stop criterion on cost and time is defined. The expected duration of the single test-diagnose-fix task is 16.0. The maximal test duration is 37.6 and the expected and maximal cost are 18.18 and 40.13, respectively. The expected remaining risk is 1.08, while the maximal remaining risk is 1.20 according to the target. The test cases are capable of reducing the remaining risk to a level of 1.15, when no faults are present in the system. If faults are present in the system, it depends if the faults are fixed if the remaining risk level of 1.15 is reached. Therefore a slightly higher target remaining risk of 1.2 is chosen, resulting in more feasible sequences.

Experiments

Several experiments are performed. The performance of the HKLFM algorithm was tested by varying the minimum number of vertices $V_{\min}$, the max weight factor *CMaxWghtFrac* and the number of vertices in a cluster *MaxNrVtxInCluster*. Furthermore, the remaining risk factor η_{R_R} is varied to determine the optimal remaining risk factor for this case.

The settings for $V_{\min}$, *CMaxWghtFrac* and *MaxNrVtxInCluster* are summarized in Table 30. The number of vertices in each different coarsening level are summarized under $|\mathcal{V}_0|$ for level 0 through $|\mathcal{V}_2|$ for level 2. The experiments are marked as C1 through C7. C1 is the experiment where the HKLFM algorithm is used without coarsening and uncoarsening the test-diagnose-fix tasks. The results of the experiments with varying *CMaxWghtFrac*, *MaxNrVtxInCluster* and $V_{\min}$ are presented in Table 31. The number of performed replications is indicated in the # repl. column. The number of replications is 10 for C1 through C3 because the results of these experiments were very close to each other. This way, it is excluded that an accidentally bad or good solution influences the results. Experiments C4 through C7 have been performed 5 times. The results of each experiment C1 through C7 have been used to calculate the average duration μ_Φ, standard deviation σ_Φ and the minimal duration $\hat{\Phi}^*_{\min}$. The difference between the minimal duration and the minimal duration of experiment C1 is calculated

Exp.	*CMaxWghtFrac*	*MaxNrVtxInCluster*	$V_{\min}$	$\lvert\mathcal{V}_0\rvert$	$\lvert\mathcal{V}_1\rvert$	$\lvert\mathcal{V}_2\rvert$
C1	-	-	∞	45	-	-
C2	0.01	2	28	45	28	-
C3	0.5	2	24	45	24	-
C4	0.1	2	14	45	24	14
C5	0.006	3	30	45	30	-
C6	0.12	3	18	45	18	-
C7	0.025	3	13	45	20	13

Table 30. The settings of the performed experiments

Exp.	# repl.	μ_Φ	σ_Φ	$\hat{\Phi}^*_{\min}$	$\frac{\hat{\Phi}^*_{\min}-\hat{\Phi}^*_{C1}}{\hat{\Phi}^*_{C1}}$	CPU [hr]
C1	10	12.33	1.31	11.50	0.00	10×15
C2	10	12.05	0.78	11.66	0.01	10×7
C3	10	11.79	0.00	11.79	0.02	10×4
C4	5	14.89	1.18	12.53	0.09	5×4
C5	5	13.14	1.33	12.05	0.05	5×8
C6	5	18.89	1.10	16.81	0.46	5×6
C7	5	16.32	0.00	16.32	0.42	5×6

Table 31. Results of the different experiments

using $\frac{\hat{\Phi}^*_{\min} - \hat{\Phi}^*_{C1}}{\hat{\Phi}^*_{C1}}$. The CPU [hr] column (10 × 15) indicates that a single replication costs 15 hours and this replication is repeated 10 times. The trade-off between the computational effort and obtained minimal test duration can be seen for the different settings of the algorithm. The obtained minimal test duration is higher for all experiments than the best solution of experiment C1. The best solution of experiment C1 also took the longest time to calculate.

The remaining risk factor η_{R_R} has been varied to determine the optimal remaining risk factor and the effect of varying η_{R_R} on the end result. Experiment C3 has been used for this purpose, since experiment C3 delivers the best results in the shortest calculation time. The result of experiment C3 differs 2% from the optimal solution and is determined in 10 × 4 hours, while experiment C2 differs 1% and is determined in 10 × 7 hours. The remaining risk factor values used are 0.25, 0.3, 0.325, 0.34, 0.35, 0.36, 0.375. Figure 62 depicts the remaining risk factor versus the minimal test duration. The minimal duration is obtained

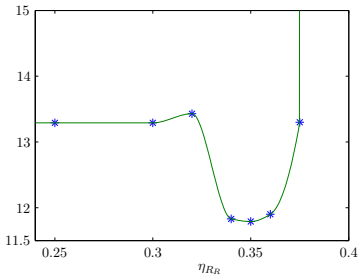


Figure 62. Relation between the expected test duration $E(\Phi)$ and factor η_{R_R}

with a remaining risk factor of 0.35. Infeasible solutions are obtained when the remaining risk factor exceeds 0.375.

Results

Partitioning the test-diagnose-fix task of the LO sub-system results in a reduction of the expected test duration from 16 hours to 11.50 hours, which is a reduction of around 30%. The maximal test duration is also reduced by around 30%. The expected test cost is increased with roughly 30%, because test cases are executed in parallel. The detailed results are presented in Table 32. The computational effort can be reduced by using coarsening and clustering as set up in experiment C3.

5.1.6 *Conclusions*

This section extends the well-known Hypergraph Kerninghan-Lin and Fiduccia-Matheyses (HKLFM) partitioning algorithm, such that it can be applied to partition test-diagnose-fix tasks. The extension includes a domain specific objective function, where the presence of an edge in a partition influences the outcome

	Expected test duration		
Exp.	$E(\varphi_S)$	$E(\varphi\|\Pi^*)$	Change
C1	16.00	11.50	-28.1%
C3	16.00	11.74	-26.6%

	Maximal test duration		
Exp.	$\max(\varphi_S)$	$\max(\varphi\|\Pi^*)$	Change
C1	37.6	25.60	-31.9%
C3	37.6	25.10	-33.2%

	Expected test cost		
Exp.	$E(C_S)$	$E(C\|\Pi^*)$	Change
C1	18.18	23.50	29.3%
C3	18.18	23.82	31.0%

Table 32. Detailed results of partitioning the LO test-diagnose-fix task into two parallel tasks. $R_{R,max,target} = 1.2$ and $\eta_{R_R} = 0.35$

of the objective function. The extension of the HKLFM algorithm enables the partitioning of test-diagnose-fix tasks into parallel test-diagnose-fix tasks, while taking the important elements of a test-diagnose-fix task into account: the test sequence, test process, test stop criteria and the system under test itself. An estimator for the objective function has been developed for this purpose and this estimator has been incorporated in the Mondriaan toolset.

The hypergraph partitioning algorithm for test-diagnose-fix tasks has been applied to a case study at ASML. A test-diagnose-fix task of a software subsystem has been partitioned and an improvement in terms of expected test duration of around 30% is obtained. Different parameters of the partitioning algorithm have been varied to investigate the effect of these parameters on the expected test duration and computational effort. An optimal setting for the case study has been determined this way.

A number of possible improvements can be made. The current objective function can be extended with additional test sequencing techniques and test process configurations. This extension broadens the applicability for other types of test-diagnose-fix tasks. Three extensions related to partitioning are proposed. First, the current algorithm supports partitioning a test-diagnose-fix task into two tasks. More than two test-diagnose-fix tasks could be beneficial. This can be done using a recursive algorithm or a flat method algorithm as proposed by [Lim and Cong, 1998]. The recursive method can be applied manually using the current toolset.

The second extension relates to the partitioning of fault states instead of test cases. The current partitioning method is limited to the partitioning of test cases only, while the partitioning of fault states could also be chosen. The partitioning of fault states should lead to the same results with the current objective

functions. It could be possible that more simple estimators of the KPI could be defined if the fault states are partitioned.

The third extension related to partitioning uses the hypergraph partitioning algorithm to partition a large test-diagnose-fix task into two test-diagnose-fix tasks that are executed in series. This way, a rough test sequence, of large groups of test cases, is obtained. These groups of test cases could be sequenced in detail using an optimal test sequencing algorithm [Boumen et al., Jan. 2008].

Communication between two parallel test-diagnose-fix tasks is currently not supported, resulting in less optimal results. Communication enables that probability information about fault states present in both test-diagnose-fix tasks is exchanged. Fixing a fault state in one test-diagnose-fix task is then beneficial for the other test-diagnose-fix task(s) also. Communication on the other hand complicates the estimator of the test duration of a single test-diagnose-fix task and could result in an increase of the computational effort.

We would like to thank Kirsten van de Meer (ASML) and Rudi Pendavingh (Eindhoven University of Technology) for their participation in this work.

5.2 DEVELOPING NEW TEST CASES

The next section is based on [I. de Jong et al., 2007d].

The complexity of wafer scanners increases with every new system type. The effort involved with designing and developing new test cases for components in a wafer scanner therefore also increases. A set of test cases, a test set, for an existing component is the basis for the test set for the new version of the component. Changes in the new version of the component require that new test cases are developed. Developing new test cases is a process where component experts and test experts are involved. First, the high-risk areas are defined. Second, the available test cases are reviewed. Third, missing test cases are identified. Finally, these missing test cases are developed and can be executed. High-risk areas result in new test cases that have a specific coverage on that high-risk area.

The areas that are of high-risk change for subsequent versions of the component. A specific test case with high coverage on a high-risk area is not optimal in the next version of the component when this risk is reduced, because this test case does not reduce much risk in the next version of the component. A test case covering many low risk areas is much more beneficial for the new version of the component. Often, test cases are developed when the risk in a specific area is high and then these test cases are not reviewed and changed anymore. One of the characteristics of platform development is that the majority of the components remain unchanged between two system versions. In other words, the risk in these unchanged components remains at the same low level as the previous component version. The selected set of test cases executed in test-diagnose-fix tasks for two versions of a system type should therefore be adjusted based on the risk in the system.

One of the characteristics of platform development is that the majority of the components, used in a version of the system, remain unchanged in the next system version.

This section presents a method that defines the so-called *next-best-test-case*, using the coverage of the existing test cases and the failure probability of possible faulty areas in the system. The problem of defining the next-best-test-case is defined in Section 5.2.1. A basic algorithm is presented in Section 5.2.2. This algorithm is able to determine optimal next-best-test-cases by selecting the coverage of the test case such that the highest information gain is obtained. This algorithm is able to determine the next-best-test-case for small systems. Industrial-size systems contain many possible faulty areas (*fault states*) and large test sets. Therefore, Section 5.2.3 presents an algorithm to select the next-best-test-case using a combination of a clustering technique and the optimal next-best-test-case algorithm. The performance of the optimal next-best-test-case-algorithm and the clustered next-best-test-case algorithm are illustrated in Section 5.2.4. Section 5.2.5 describes two cases that have been performed using this method to improve the test set of large test phases at ASML.

5.2.1 *Problem definition*

The next-best-test-case problem is a selection problem, where the best test case, out of all possible test cases, needs to be selected according to some objective function. The test *signature* describing the *coverage* of the test case is what defines a test case. The coverage is described in terms of faulty areas in the system or *fault states*. The next-best-test-case algorithm determines the best signature according to an objective function. Available test cases are taken into account in the objective function assuming that these test cases are executed first. Consequently, the coverage of the available test cases on the fault states is taken into account. The available test cases, the fault states in the system and the coverage of the test cases on these fault states are modeled in a *system test model*. Information gain is used as performance criterion and is introduced first using the system test model introduced in Chapter 2. Sections 5.2.2 and 5.2.3 describe the next-best-test-case algorithm that uses the system test model and objective function to calculate the next-best-test-case.

System test model

The system test model is introduced earlier in Chapter 2. Table 33 depicts a system test model for version *v*1.0 of an example component. Six test cases are defined and five fault states. The failure probabilities and impact are defined per fault state. The *IG* row represents the information gain per test case that is derived from the elements of the model. The information gain per test case is described later in this section. The coverage of each test case on a fault state is represented by a value between 0 and 1, where 0 means that the test case does not cover the fault state and 1 means that the fault state is 100% covered by the test case. A value between 0 and 1 represents partial coverage.

Table 34 represents a system test model for the second version of the same component. The differences between both models are the failure probabilities and the additional fault state s_6. The failure probabilities are reduced due to the

S / T	t_0	t_1	t_2	t_3	t_4	t_5	P	I
s_1	0.5	0.5	0	0	0.5	0	80%	1
s_2	0.5	0	0.5	0	0.5	0.5	80%	1
s_3	0.5	0	0	0.5	0	0.5	10%	1
s_4	0.5	0	0	0	0.5	0	10%	1
s_5	0.5	0	0	0	0	0.5	10%	1
IG	0.89	0.97	0.97	0.29	0.93	0.99		

Table 33. An example system test model of component version 1.0, $D_{v1.0}$

test cases, diagnosis and fix tasks that are executed for version $v1.0$ or increased because of changes in the system. Fault state s_6 represents the specific changes for version $v2.0$ of the component.

S / T	t_0	t_1	t_2	t_3	t_4	t_5	P	I
s_1	0.5	0.5	0	0	0.5	0	8%	1
s_2	0.5	0	0.5	0	0.5	0.5	8%	1
s_3	0.5	0	0	0.5	0	0.5	8%	1
s_4	0.5	0	0	0	0.5	0	8%	1
s_5	0.5	0	0	0	0	0.5	70%	1
s_6	0.5	0	0	0	0	0.5	80%	1
IG	0.92	0.24	0.24	0.24	0.52	0.94		

Table 34. An example system test model of component version 2.0, $D_{v2.0}$

The next-best-test-cases for model version $v1.0$ and $v2.0$ are expected to be different because of the failure probabilities of the fault states in the two versions of the model and the coverage of the test cases on the fault states (s_6).

Objective function

The objective function, used to evaluate possible next-best-test-cases, is based on the information gain of a test case. The information gain for a test case, defined in Equation (5.30), is maximal if a test case has a failure probability of 50%. A test case with a failure probability of 50% has a 50% probability that a fault is found. Fault detection is one of the purposes of a test case. This test case also has a 50% probability that it passes. A passed test case means that the covered fault states do not exist in the system. A failing test case is an opportunity to improve the system (and reduce the failure probability). A passing test case reduces the failure probability of the covered fault states. A test case is represented by its test *signature*. A test signature represents the set of fault states that is covered by the test case. The information gain for a *single* test case with signature *sig* is calculated using Equation (5.30).

The objective function, used to evaluate possible next-best-test-cases, is based on the information gain of a test case.

$$IG(sig) = -\left(p_p(sig)log_2 p_p(sig) + p_f(sig)log_2 p_f(sig)\right) \tag{5.30}$$

where $p_p(sig)$ is the *pass* probability of a test with signature *sig* and $p_f(sig)$ is the *fail* probability. The pass probability is defined by

$$p_p(sig) = \prod_{s \in sig} \left(1 - P(s) R_{ts}(s) \right) \tag{5.31}$$

and the corresponding *fail* probability as

$$p_f(sig) = 1 - p_p(sig) \tag{5.32}$$

The objective function used in the next-best-test-case algorithm takes the coverage of the previously executed test cases into account. For this purpose, it is assumed that the test cases, currently present in the test model, are executed before the new next-best-test-case. The exclusion of fault states by previous test cases results in a change of the failure probability of the covered fault states. Therefore, the pass and failure probability of a possible new test case is affected. This means that Equation (5.31) to determine the pass probability for a test case needs to be revised such that the previously covered test cases are taken into account.

The optimal method to calculate the pass probability for a test case takes all combinations of covered fault states for each test case into account. This method is computationally intensive, because of the calculation of all possible fault state combinations for each test. Therefore, an estimator of the pass probability is required. This *probability estimator* estimates the pass probability for a test case with a high accuracy. The optimal method to determine the pass probability of a test case and the probability estimator are investigated in [Boumen et al., Jan. 2008]. The probability estimator, defined in Equation (5.33), is used to calculate the pass probability of a test case.

$$p_p(sig) = \prod_{s \in sig} \left((1 - P(s)) \prod_{S' \in S_c \wedge s \in S'} (1 - \frac{P(s)}{\sum_{s_i \in S'} P(s_i)}) \right) \tag{5.33}$$

5.2.2 *Optimal next-best-test-case algorithm*

The next-best-test-case function NBTC : $\mathcal{P}(S) \times \mathcal{P}(\mathcal{P}(S)) \rightarrow \mathcal{P}(S)$ takes a set of fault states as input and the set of candidate sets S_c of fault states. The function returns the set of fault states covered by the next-best-test-case, the *signature* of the next-best-test-case. The set of fault states describes what could be covered by the next-best-test-case. The candidate set S_c describes what is already covered by the test cases that are previously executed. The candidate set contains the signature of all test cases that are currently in the test model.

The candidate set S_c is the set of test signatures for all test cases that are currently present in test set T, using the coverage relation $R_t(t) : \mathcal{P}(T) \rightarrow S$: $R_t(t) = \{s \mid R_{ts}(t, s) > 0\}$.

$$S_c = \{R_t(t) \mid t \in T\} \tag{5.34}$$

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

$$\begin{aligned}&\text{NBTC}(S, S_c) = \\ &\{sig^* \mid \forall sig \in \Theta(S, S_c) : IG(sig^*) \geq IG(sig)\}\end{aligned} \tag{5.35}$$

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

The idea behind the optimal next-best-test-case algorithm is that the signature of a test case is a set of fault states.

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

$$\Theta(S, S_c) = \text{setcombine}(S) \setminus \{\emptyset \cup S_c\} \tag{5.36}$$

The optimal next-best-test-case algorithm is able to determine the optimal test case for a given system test model, because all possible test cases are examined. The problem with the optimal next-best-test-case algorithm is that determining all possible test cases using Equation (5.36) is computationally intensive. Algorithms to generate all possible combinations are fairly simple, but the memory usage or duration of the calculation increases exponentially with the number of fault states in the system. Therefore, the optimal next-best-test-case algorithm cannot be used to determine the next-best-test-case for large system test models. The current maximum number of fault states lies around 17. The clustered next-best-test-case algorithm has been developed to overcome this disadvantage. The clustered next-best-test-case algorithm is explained in Section 5.2.3.

5.2.3 *Clustered next-best-test-case algorithm*

The clustered next-best-test-case algorithm replaces fault states with a new fault state, such that the information of the replaced fault states is represented by the new fault state. Clustering of fault states continues until $|S| < maxS$, where $3 \leq maxS \leq 15$.[1] The next-best-test-case for the clustered system test model can now be calculated using the optimal next-best-test-case algorithm. This process is repeated on the resulting signature such that the result is refined. A flowchart of the clustered next-best-test-case algorithm NBTC_{cl} is depicted in Figure 63.

In a formal definition the clustered next-best-test-case algorithm is defined as a function $\text{NBTC}_{cl} : \mathcal{D} \times \Pi^{cl} \times \mathbb{N} \rightarrow S$ that takes a system test model D, a clustering

[1] The maximum number of fault states depends on the available computer processing power. A safe choice of 15 fault states is chosen here.

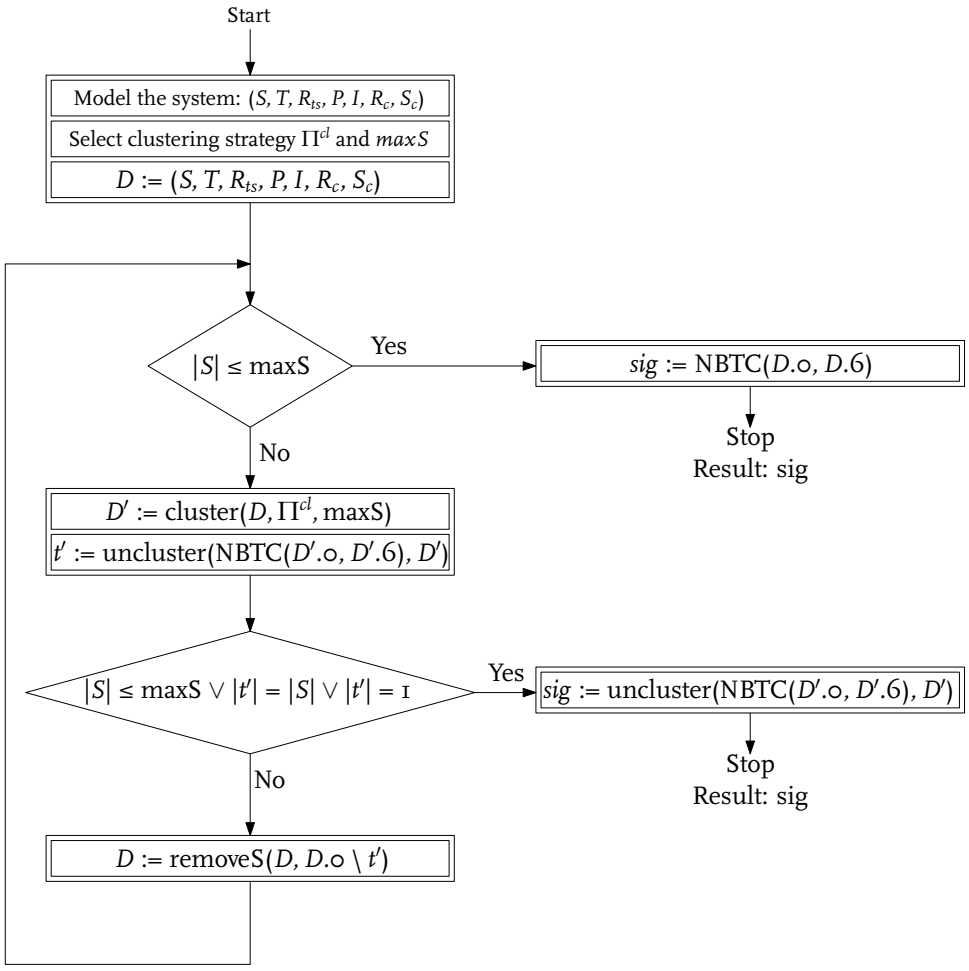


Figure 63. Flowchart of the clustered next-best-test-case algorithm

strategy $\Pi^{cl} = \{\textbf{minmin}, \textbf{minmax}, \textbf{maxmin}, \textbf{maxmax}\}$ and a maximum number of fault states in the cluster $maxS$ as input and outputs the signature of the next-best-test-case. The corresponding algorithm is described in Equation (5.37).

The system test model, used in the clustering algorithm, is based on the system test model defined in Chapter 2: $D = (S, T, R_{ts}, P, I, R_c, S_c)$. The elements R_c and S_c are specific for the clustered next-best-test-case algorithm and used to maintain additional information required in the algorithm. The relation R_c defines for a fault state in which cluster the fault state is clustered or $\{\emptyset\}$ if the fault state is not clustered. This clustering relation is defined as $R_c : S \rightarrow \mathcal{P}(\mathcal{S})$. The clustering relations defined for the set of all possible fault states $\mathcal{S}$ are empty initially. Note that the domain of clustering relation is larger than the domain of the fault state set S for the purpose of clustering and unclustering using Equations (5.52), (5.60) and (5.61). The candidate set S_c is used to maintain the

signatures of the executed test cases.

$$\text{NBTC}_{cl}(D, \Pi^{cl}, maxS) = \begin{cases} \text{NBTC}(D.o, D.6) & \text{if } |D.o| \leq maxS \\ t' & \text{if } |D.o| \leq maxS \vee |t'| = |D.o| \vee |t'| = 1 \\ \text{NBTC}_{cl}(removeS(D, S \setminus t')) & \text{if } |D.o| > maxS \wedge |t'| \neq |D.o| \wedge |t'| > 1 \end{cases} \tag{5.37}$$

Where variable t' stands for:

$$t' = uncluster(NBTC(D'.o, D'.6), D') \tag{5.38}$$

and

$$D' = cluster(D, \Pi^{cl}, maxS) \tag{5.39}$$

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

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

$$\begin{aligned} &\text{removeS}(D, sig) = \\ &(S', T', D.2 \upharpoonright (S' \times T'), D.3 \upharpoonright S', D.4 \upharpoonright S', D.5 \upharpoonright S', \{x \setminus sig \mid x \in D.6\} \setminus \{\emptyset\}) \end{aligned} \tag{5.40}$$

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

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

clustered fault states, without reusing the properties of existing fault states for this purpose.

$$\begin{aligned}&\text{uncluster}(sig, D') = \\ &\{s \mid s \in D'.\mathrm{o} \wedge ((\exists s' : s' \in sig : s' \in D'.5(s)) \vee (\exists s' : s' \in sig : s' \in D'.\mathrm{o}))\}\end{aligned} \tag{5.41}$$

The clustering technique determines what the performance is of the clustered next-best-test-case algorithm.

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

Clustering

The clustering technique used in the clustered next-best-test-case algorithm determines the performance of the algorithm, because a bad clustering algorithm results in next-best-test-cases that are not optimal at all. A general clustering algorithm is defined in Equation (5.42), and utilizes an objective function to cluster fault states. Three different clustering methods are defined. Additionally, a clustering strategy Π^{cl} needs to be selected.

$$\begin{aligned}&\text{cluster}(D, \Pi^{cl}, maxS) = \\ &\begin{cases} D & \text{if} \quad |D.\mathrm{o}| \leq maxS \\ \text{cluster}(\text{mrg}(D, IG^{-}(x), IG^{-}(y))) & \text{if} \quad |D.\mathrm{o}| > maxS \wedge \Pi^{cl} = \textbf{minmin} \\ \text{cluster}(\text{mrg}(D, IG^{-}(x), IG^{+}(y))) & \text{if} \quad |D.\mathrm{o}| > maxS \wedge \Pi^{cl} = \textbf{minmax} \\ \text{cluster}(\text{mrg}(D, IG^{+}(x), IG^{-}(y))) & \text{if} \quad |D.\mathrm{o}| > maxS \wedge \Pi^{cl} = \textbf{maxmin} \\ \text{cluster}(\text{mrg}(D, IG^{+}(x), IG^{+}(y))) & \text{if} \quad |D.\mathrm{o}| > maxS \wedge \Pi^{cl} = \textbf{maxmax} \end{cases}\end{aligned} \tag{5.42}$$

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

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

A fault state with the *lowest* information gain is selected using Equation (5.43). A fault state with the *highest* information gain is selected using Equation (5.44). Fault states are not selected twice for merging, by removing the first fault state

from the set of fault states as in variable y. Clustering using information gain as objective function is referred to as CTMIG in the remainder of this section.

$$IG^{-}(S) = \{x \in s^* \mid s^* \in S \wedge \forall s \in S : IG^s(s^*) \leq IG^s(s)\} \tag{5.43}$$

$$IG^{+}(S) = \{x \in s^* \mid s^* \in S \wedge \forall s \in S : IG^s(s^*) \geq IG^s(s)\} \tag{5.44}$$

Where:

$$IG^s(s) = -\left(p_p^s(s) log_2 p_p^s(s) + p_f^s(s) log_2 p_f^s(s)\right) \tag{5.45}$$

where $p_p^s(s)$ is defined as:

$$p_p^s(s) = \prod_{t \in T} \left(1 - P(s) R_{ts}(t, s)\right) \tag{5.46}$$

and:

$$p_f^s(s) = 1 - p_p^s(s) \tag{5.47}$$

The second, similar to the first, clustering objective function clusters two fault states using failure probability instead of the information gain. Equations (5.48) and (5.49) are used to combine two fault states for the different strategies. Clustering using failure probability as objective function is identified with CTMP in the remainder of this section.

$$P^{-}(S) = \{x \in s^* \mid s^* \in S \wedge \forall s \in S : P(s^*) \leq P(s)\} \tag{5.48}$$

$$P^{+}(S) = \{x \in s^* \mid s^* \in S \wedge \forall s \in S : P(s^*) \geq P(s)\} \tag{5.49}$$

The third clustering objective function clusters two fault states using the information gain per fault state together with the risk involved with this fault state. The use of risk in addition to the information gain could result in test cases with less information gain in favor of the risk covered. The usage of an objective function that is a combination of two functions has an advantage when determining what fault states can be clustered. The usage of risk could lead to better solutions. The information gain risk is calculated using Equations (5.50) and (5.51). Clustering using information gain and risk as objective function is identified as CTMIGR in the remainder of this section.

$$IGR^{-}(S) = \{x \in s^* \mid s^* \in S \wedge \forall s \in S : IGR(s^*) \leq IGR(s)\} \tag{5.50}$$

$$IGR^{+}(S) = \{x \in s^* \mid s^* \in S \wedge \forall s \in S : IGR(s^*) \geq IGR(s)\} \tag{5.51}$$

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

The actual combination, or merging, of fault states is performed by the **mrg** function, defined in Equation (5.52). This function merges s' and s'' into a new fault state $newS$ and removes s' and s'' from the system test model D. Note that, the new fault state $newS$ can be uniquely identified. Next to that, the properties related to s' and s'' are updated. The mrg function is defined as:

$$\begin{aligned} &\text{mrg}(D, s', s'') = \\ &((D.\text{o} \cup \{newS\}) \setminus \{s', s''\}, D.\text{i}, R'_{ts}, P', I', R'_c, S'_c) \end{aligned} \tag{5.52}$$

Where:

- R'_{ts} is defined as $R'_{ts} : D.\text{i} \times (D.\text{o} \cup \{newS\}) \setminus \{s', s''\} \rightarrow \mathbb{R}$, such that: for every $t \in D.\text{i}$ and $s \in D.\text{o}$:

$$R'_{ts}(t, s) \text{ if } s \neq s' \wedge s \neq s'' \tag{5.53}$$

and

$$R'_{ts}(t, newS) = (R_{ts}(t, s') + R_{ts}(t, s''))/2 \tag{5.54}$$

- P' is defined for every $s \in D.\text{o}$ as :

$$P'(s) = P(s) \text{ if } s \neq s' \wedge s \neq s'' \tag{5.55}$$

and

$$P'(newS) = 1 - ((1 - P(s'))(1 - P(s''))) \tag{5.56}$$

- I' is defined for every $s \in D.\text{o}$ as :

$$I'(s) = I(s) \text{ if } s \neq s' \wedge s \neq s'' \tag{5.57}$$

and

$$I'(newS) = (I(s') + I(s''))/2 \tag{5.58}$$

- R'_c is defined for every $s \in D.\text{o}$ as :

$$R'_c(s) = R_c(s) \text{ if } s \neq s' \wedge s \neq s'' \tag{5.59}$$

and

$$R'_c(s') = R_c(s') \cup \{newS\} \tag{5.60}$$

and

$$R'_c(s'') = R_c(s'') \cup \{newS\} \tag{5.61}$$

- S'_c is defined for every $s \in (D.\text{o} \cup \{newS\}) \setminus \{s', s''\}$ as :

$$S'_c = \{R'_{ts}(t, s) \mid t \in D.\text{i}\} \tag{5.62}$$

5.2.4 *Illustration*

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

The following system test model parameters have been varied: the number of test cases in the model [8 or 16], the number of fault states in the model [5,7,9,11 and 13], the failure probability for all fault states in the models [0.1, 0.25, 0.5, 0.75 and 0.9] and the density of the model [0.1, 0.25, 0.5, 0.75 and 0.9]. All these parameters are self explanatory, except the density that is defined as: $\rho = \frac{1}{|T|\times|S|} \sum_{t\in T, s\in S} R_{ts}(t,s)$. The density of a system test model is a measure for the coverage of all test cases on all fault states.

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

The first result that is evaluated is the effect of the clustering method and clustering strategy versus the average failure probability of the fault states and the density of the system test model. The average information gain of each next-best-test-case is compared with information gain of the optimal next-best-test-case. Table 35 depicts the results of this analysis. The information gain for the clustered next-best-test-cases is an average over 38 next-best-test-cases calculated for each of the different settings. These 38 next-best-test-cases were calculated for a combination of different models and maximum fault states $maxS$. The optimal information gain is an average of 10 optimal next-best-test-cases calculated for 10 models with different settings.

Table 35 contains 81 results. The effect of the clustering strategy is investigated. Therefore, the best clustering strategy is selected for the 27 combinations of clustering method, failure probability and density. The **minmin** clustering strategy performs best in 23 of the 27 cases. The **maxmax** clustering strategy performs better in two of the 27 cases. The **minmax** clustering strategy performs

$\overline{IG}$		P								
		0.1			0.5			0.9		
Clustering		ρ								
method	strategy	0.1	0.5	0.9	0.1	0.5	0.9	0.1	0.5	0.9
CTMIG	maxmax	0.703	0.705	0.719	0.355	0.534	0.578	0.552	0.492	0.616
	minmax	0.713	0.717	0.829	0.578	0.616	0.710	0.616	0.607	0.604
	minmin	0.892	0.845	0.894	0.797	0.828	0.916	0.831	0.812	0.783
$\overline{IG}_{CTMg}$		0.770	0.755	0.814	0.577	0.659	0.735	0.666	0.637	0.668
CTMIGR	maxmax	0.668	0.793	0.719	0.441	0.505	0.682	0.547	0.506	0.613
	minmax	0.709	0.763	0.882	0.588	0.627	0.705	0.610	0.607	0.597
	minmin	0.936	0.798	0.900	0.801	0.810	0.887	0.828	0.762	0.766
$\overline{IG}_{CTMIGR}$		0.771	0.785	0.833	0.610	0.647	0.758	0.662	0.625	0.659
CTMP	maxmax	0.691	0.832	0.747	0.572	0.674	0.786	0.611	0.756	0.775
	minmax	0.712	0.786	0.892	0.518	0.581	0.691	0.611	0.579	0.602
	minmin	0.914	0.766	0.892	0.832	0.849	0.735	0.820	0.816	0.810
$\overline{IG}_{CTMPg}$		0.772	0.794	0.844	0.641	0.701	0.737	0.681	0.717	0.729
Opt	Opt	0.990	0.983	0.997	0.996	1.000	1.000	0.998	1.000	0.999

Table 35. The average information gain for different clustering methods and strategies versus the average failure probability and model density

better in one of the 27 cases. The **minmin** and **minmax** clustering strategy lead to the same results in one of the 27 cases.

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

The second investigated clustering setting is the use of the refinement step, defined in Equation (5.37) as recursive call to the $NBTC_{cl}$ algorithm. For this purpose, a number of experiments has been conducted with and without the use of the refinement step in the algorithm. The results of these experiments for the different models are depicted in Table 36. The table depicts the average information gain. The models are described as 16x11 if the model contained 16 test cases and 11 fault states. Three rows of results are presented: *don't refine* if no refinement step was used, *optimal* for the optimal result and *refine* if the refinement step was applied.

A 19% improvement of the information gain is obtained if the refinement step in the algorithm is used. The data in Table 36 includes all clustering strategies. If the results of the **maxmax** and **minmax** clustering strategies are removed from the dataset, then the average improvement due to refinement is 7%. Both results justify the application of the refinement step in the clustered next-best-test-case algorithm.

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

$\overline{IG}$	Refinement setting		
D	don't refine	optimal	refine
16x5	0.740	0.995	0.877
16x7	0.701	0.999	0.843
16x9	0.647	0.998	0.807
16x11	0.608	0.999	0.759
16x13	0.522	1.000	0.748
8x5	0.765	0.979	0.898
8x7	0.739	0.998	0.874
8x9	0.655	0.998	0.861
8x11	0.639	0.997	0.829
8x13	0.528	0.998	0.823
Average $\overline{IG}$	0.632	0.996	0.820

Table 36. The average information gain for different system test models and the application of a refinement step in the algorithm.

5.2.5 *Case*

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

Case 1: Weekly validation test

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

The properties of the system test model are:

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

- Average coverage of the test cases on the fault states: 0.244

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

The maximum number of fault states in a cluster was set to 13, such that the largest possible clusters were used and the results were obtained quickly. The used clustering techniques are: CTMIG, CTMP and CTMIGR. The **min-min** and **maxmax** clustering strategies were both used for all three clustering techniques. The resulting information gain profiles for the three techniques are depicted in the Figures 64, 65 and Table 37 for the **minmin** and **maxmax** strategy respectively.

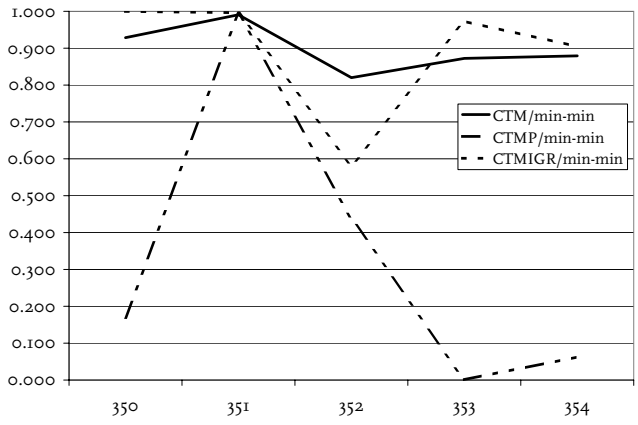


Figure 64. The information gain profile for the 5 test cases determined using the **minmin** strategy

It can be concluded from the Table 37 and Figures 64 and 65 that the **maxmax** clustering strategy performs worse than the **minmin** clustering strategy. The first next-best-test-case (350) of the CTMIGR clustering method, with **minmin** clustering strategy, performs best, i. e. the information gain for this test case is $IG(t_{350}) = 1.0$.

The test experts analyzed the newly generated test case t_{350}. Test case t_{350} covers a single fault state s_{37} with a failure probability of 96.6%. Two other test cases t_{210} and t_{313} that were already present in the system test model and covered s_{37} both with a coverage of 0.3. The failure probability of s_{37} was reduced by the two existing test cases into a failure probability of: $P(s_{37})_{t_{210},t_{313}} = (1 - R_{ts}(t_{210}, s_{37})(1 - R_{ts}(t_{313}, s_{37}) = 0.966(1-0.3)(1-0.3) = 0.473$. The information

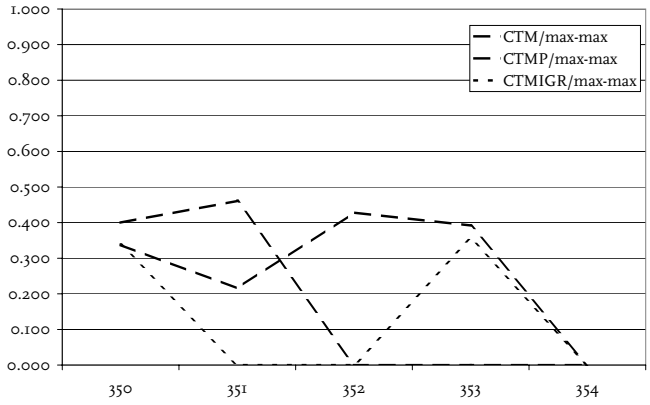


Figure 65. The information gain profile for the 5 test cases determined using the **maxmax** strategy

	CTMIG	CTMP	CTMIGR	CTMIG	CTMP	CTMIGR
	minmin			maxmax		
350	0.929	0.169	1.000	0.400	0.337	0.339
351	0.991	0.994	0.996	0.461	0.215	0.000
352	0.820	0.438	0.582	0.000	0.428	0.000
353	0.872	0.001	0.973	0.000	0.392	0.355
354	0.879	0.063	0.904	0.000	0.000	0.000

Table 37. Information gain of the 'new' weekly validation test cases for the different clustering techniques and strategies

gain of a test case that only covers this fault state with a failure probability of $P(s_{37}) = 0.473$, leads to an information gain close to 1.

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

Case 2: Software alpha test

ASML releases a software baseline to customers when a new type of wafer scanner is released. Additional 'consolidation' software releases are released to enable customers to upgrade all wafer scanners in the IC factory to the same software release. The number of different software releases delivered to customers is four or more per year. Every software release is qualified in an alpha test and beta test before it is ready to be rolled out at customers world-wide. Alpha testing is performed on different types of wafer scanners at the ASML premises.

	CTM	CTMP	CTMIGR	CTM	CTMP	CTMIGR
	minmin			maxmax		
77	0.460	0.269	0.460	*1.000*	0.725	0.457
78	0.748	0.177	0.541	0.164	0.525	0.977
79	*1.000*	0.381	*1.000*	0.999	0.999	0.815
80	0.028	0.823	0.028	0.998	0.000	0.002
81	0.098	0.927	0.098	0.941	0.055	0.000

Table 38. Information gain of the 'new' alpha test cases for the different clustering methods and strategies

Beta testing is typically performed at wafer scanners that are already running production at customers worldwide.

The alpha test of a new software release has a number of goals. First, it needs to be tested if the performance of the wafer scanners is equal or better with the new software release. Second, the high-risk areas in the new software release are tested. Third, problems found during the alpha test period are analyzed, solved and retested. A standard set of performance test cases is available to test the performance of the new release. This set is always executed. Another set of test cases is selected to meet the second goal. This selection process is continued throughout the test execution and problem solving process to meet the third goal. A system test model has been created to support the selection process of the alpha test cases. The system test model had the following characteristics at the start of the alpha test:

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

The same modeling parameters as used in the previous case study were applied: five 'new' test cases were defined using the three clustering methods and the **minmin** and **maxmax** clustering strategy. The information gain profile of all clustering methods and strategies is depicted in Figure 66. The detailed results are depicted in Table 38.

The test case resulting in the best information gain (1.0) is derived using the CTM clustering method with a **maxmax** clustering strategy. For this case study, the **maxmax** clustering strategy leads in general to better results. The fault

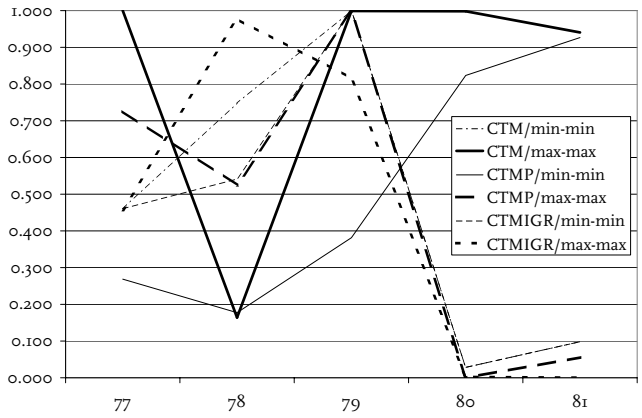


Figure 66. Information gain profile for the different clustering methods and strategies

states covered by the best test case are: $\{s_5, s_{11}, s_{13}\}$. The properties of these fault states are $P(s_5) = 82.85\%$, $P(s_{11}) = 0.1\%$ and $P(s_{13}) = 30\%$. All three fault states are already covered by other test cases. Fault state s_{11} describes the possibitity that the software operating system (OS) is faulty and is the lowest level fault state. Fault state s_{13} describes the possibility that the measurement sensors are faulty, a mid-level fault state. Fault state s_5, a high-level fault state, models the possible fault that a complete system is not calibrated. A test graph for this model, depicting the relations between fault states and test cases, can be found in Figure 67 and is used by the test experts for detailed analysis of the (new) test cases. The new test case, and its coverage on the fault states, was drawn (by hand) in the figure. This way, the result was checked with the expectations of the test experts. The resulting test case seems good, however, it was not possible to find better test cases using this method. The measure for the information gain per test case is required for that purpose.

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

5.2.6 *Conclusions*

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

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

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

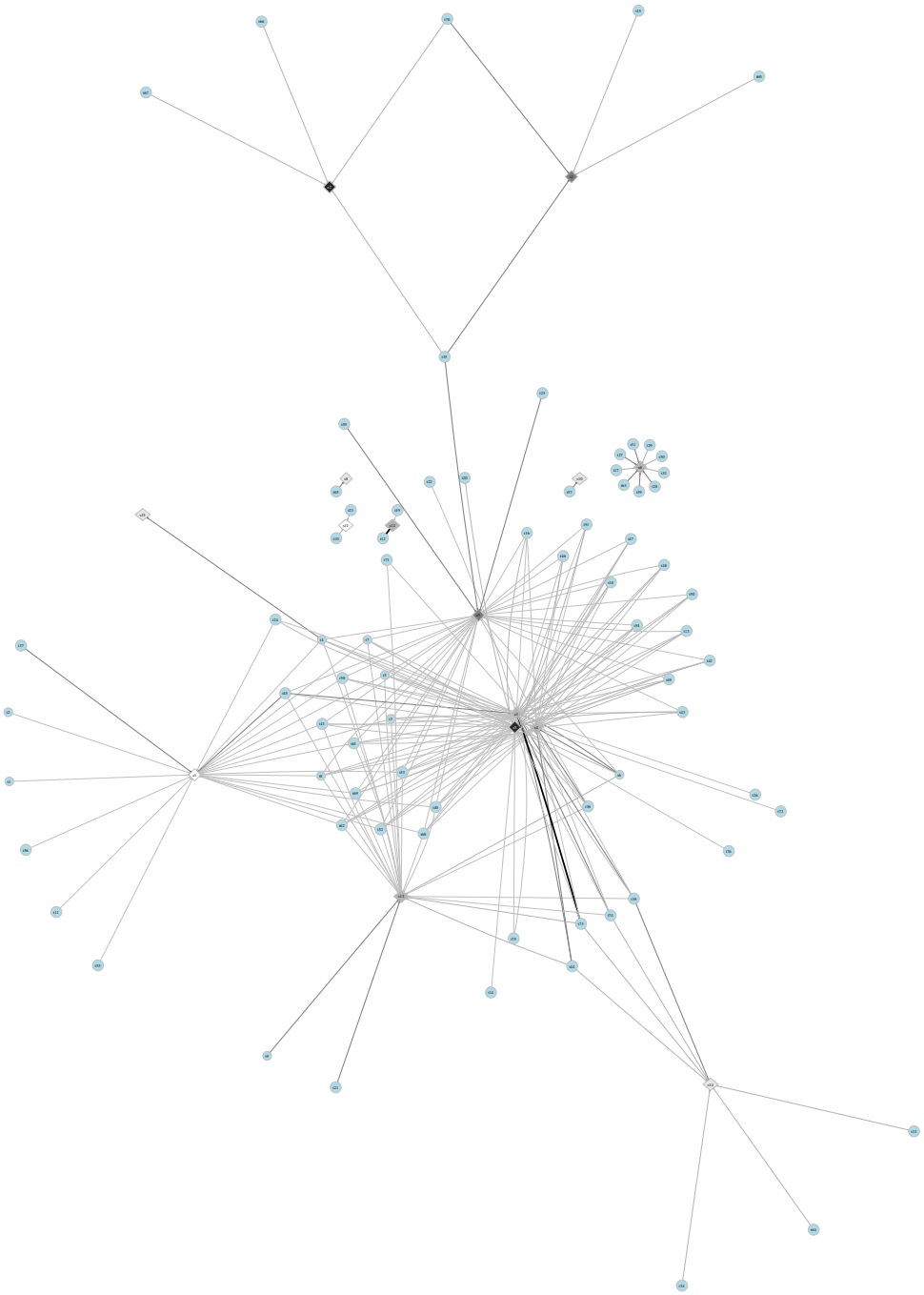


Figure 67. Test graph of the alpha test model (excluding new test cases)

5.3 UPDATING CONSTRAINTS AND/OR OBJECTIVES

Updating the high-level constraints and objectives for integration and test sequences is in the list of improvement techniques, because this could be the only remaining option if other improvement techniques did not lead to feasible solutions.

Updating the high-level constraints and objectives for integration and test sequences is in the list of improvement techniques, because this could be the only remaining option if other improvement techniques did not lead to feasible solutions. Updating the constraints or objectives is a different type of improvement technique than the other three improvement techniques described in this chapter. The objectives of an integration and test sequence are described in terms of time, quality and cost (TQC). These objectives are defined relative to each other. One of the objectives is most important, another is second most important and the last objective is least important. See Section 2.3.1 for more details on these objectives and their relation.

The constraints of an integration and test sequence are described in terms of maximal duration, maximal cost and minimal remaining risk. Integration and test sequences that are obtained using the described planning method should satisfy these constraints, otherwise the integration and test sequence is not feasible. The goal of this 'improvement' technique is to increase the number of feasible integration and test sequences that are considered in the planning method. Next, the constraint updates and objectives updates are discussed.

5.3.1 *Updating constraints*

Two improvement techniques are discussed here: updating the relative importance of the constraints and updating the constraint order. The first improvement technique changes the relative importance of the constraints, not the order. The relative importance can be expressed by means of weight factors for each constraint. These weight factors are used to weigh the results of the expected duration, cost and remaining risk of an integration and test sequence, such that a single value is obtained that can be used to compare integration and test sequences. Equation (5.63) describes such an objective function. The relative importance of the constraints is defined by choosing appropriate weights. For instance, an organization with an extreme focus on time, some focus on quality and less focus on cost (T-Q-C)[2] could be described by weights of $w_\Phi = 10000$, $w_{R_R} = 100$, $w_C = 10$ for time, quality and cost respectively.

$$J = \Phi \cdot w_\Phi + C \cdot w_C + R_R \cdot w_{R_R} \tag{5.63}$$

Updating the relative importance of the constraints means that the weights are adjusted relative to each other, resulting in different integration and test sequences.

The order of importance of the constraints can also be changed. An example T-Q-C order can be changed into T-C-Q or otherwise by changing the weight values. This update also leads to different integration and test sequences with a different performance. However, changing the order also means that another

[2] The order of the business drivers time, quality and cost (T-Q-C) is described in detail in Section 2.1.

business driver has become more important. It is highly unlikely that this situation happens during the course of the project, or due to the fact that the integration and test sequences do not meet the initial constraints and objectives.

5.3.2 *Updating objectives*

Three types of updates are discussed: updating the maximal duration, maximal cost and minimal remaining risk. Combinations of these three updates are possible, but not discussed here. The maximal duration objective can be relaxed, such that a feasible integration and test sequence is obtained. Increasing the maximal duration means that the deadline is shifted.

The maximal duration objective can be relaxed, such that a feasible integration and test sequence is obtained. Increasing the maximal duration means that the deadline is shifted.

The maximal cost objectives can be relaxed, such that more cost can be spent. Spending more cost on model-based integration could result in less faults in a later stage [Braspenning et al., 2007]. Spending more cost on parallel testing decreases the duration and/or remaining risk as described in Section 5.1.

The remaining risk objective can be relaxed, such that less testing is required. Less test, diagnosis and fix tasks to be executed results in less cost and a decrease of the integration and test duration. The integration and test sequence is finished earlier and the system can be released earlier to customers. The remaining risk that is still in the system should be reduced after shipment using a costly diagnosis and fix process. Still it can be beneficial to be first in the market with a new product.

In practice, a trial and error approach is followed when the constraints and objectives are changed, such that a feasible integration and test sequence is obtained. Structured methods to analyze and compare the effect of a change in constraints and objectives on the performance of an integration and test sequence are based on designs of experiments. More optimal methods as developed in the TANGRAM project and described in [Boumen, 2007] still require heuristics such that solutions are obtained in feasible time. To determine the values for the heuristics a design of experiments is required, because the combination of models, algorithm and heuristics influences the performance of integration and test sequences. Despite the great progress that has been made in this area, some work is to be done such that the effect of model parameters on the performance of integration and test sequences can be predicted.

5.4 SELECTING A SYSTEM ARCHITECTURE AND DESIGN

The next section is based on [I. de Jong et al., 2007c].

System architecting [Muller, 2007] is the process of creating an architecture for a system. The resulting architecture is a trade-off between the most important architectural views, like functionality, maintainability, extendability, etc. A list of these architectural views, so called quality attributes, that need balancing is given in the ISO-9126 standard [ISO-9126-1, 2000-03-20]. Two of these

quality attributes are *testability* and *manufacturability*. Testability fits within our definitions and defines if a system is well testable or not. Manufacturability in the context of ISO-9126 is a much broader term than the term integratability[3], which we prefer. Only a limited number of architectural views is considered when an architecture and design are defined. Testability and integratability are often not the most important views that are considered. Consequently, the resulting architecture does not reflect testability and integratability very well. Two solutions exist for this problem. The first solution defines integratability and testability as the most important architectural views of a system and the entire system architecture and design are centered around this view. Although, this solution results in a system that is designed for integration and testing, it requires a specific business case where this is beneficial. For most systems, integration and testing are not the most important architectural views.

The second solution uses an existing architecture, design and resulting integration and test sequence. The architecture and design are then improved based on the performance of the integration and test sequence. This, more iterative, solution is the basis for this section. The structure of this section is as follows. First, definitions of an 'architecture' are discussed in Section 5.4.1. Then, Section 5.4.2 defines methods and guidelines for the selection of components, interfaces and layerings. These guidelines are illustrated with example systems. Conclusions are given in Section 5.4.6.

5.4.1 *Architectures*

According to the Oxford English Dictionary, the term *'Architecture'* is defined as: 1) the art or practice of designing and constructing buildings, 2) the style in which a building is designed and constructed, 3) the complex structure of something. The first definition describes the *process* of designing buildings, or systems, also called *architecting* in [Muller, Jan. 2004]. The second definition reflects the most commonly known definition of an architecture that is the style of a building. The, most relevant, third definition broadens the second definition, such that it is applicable to complex structures of 'something'. If we start with the first, building oriented, definition, then we see that the low-level components of most buildings are equal. For instance, bricks, mortar, nails and glue are used to build any type of house. The architectural 'style' defines if a building is functional and beautiful.

For systems, as defined as 'something' in the third definition, the same holds. The components (bricks) can be defined for a family of systems. This is also the case for the interfaces (mortar, nails, glue, etc.). Complex manufacturing machines are also built up using components and interfaces as defined in Chap-

[3]The word integratability is first used by SUN microsystems to indicate the easy integration capabilities of the Java™ platform. The word integratability is not defined in a dictionary at this moment. Other words describing integratability could be synthesizability or assembleability. Both words have different meanings in different domains and are also not defined in a dictionary.

ter 2. The *style*, if applied to these systems determines if the system is functional and beautiful. The style determines how the components interact and are able to function together. The architectural style also determines how well the system can be split up into smaller sub-systems and how well this work-break-down-structure can be communicated and maintained throughout the development process: the beauty of the architecture. Our definition for architectural style, which is presented earlier in Chapter 2, is called *Layering*. The layering splits up the set of components and interfaces of a system into groups of components and interfaces. These groups of components are used for integration and test planning, such that the planning effort is reduced and the integration and test sequences are kept more or less the same after replanning.

The architectural style also determines how well the system can be split up into smaller sub-systems and how well this work-break-down-structure can be communicated and maintained throughout the development process: the beauty of the architecture.

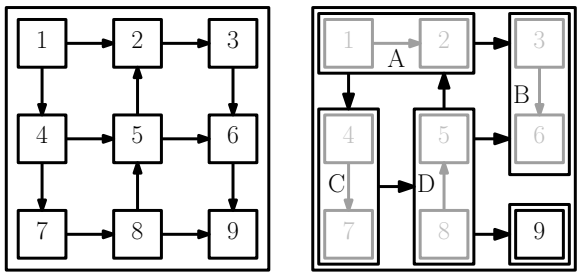
Components, interfaces and a layering are the elements of an architecture that are relevant for integration and testing. The remainder of this section describes how components, interfaces and a layering can be selected such that they are suitable for integration and testing.

5.4.2 *Selection of a suitable set of components*

A suitable set of components, in the context of integration and testing, means that the components are integratable and testable. The aspects that influence the integratability and testability of an architecture are the number of components and the size of the components. The break-down of the system into components determines how many components need to be integrated. Not all components need to be taken into account for the system level integration and test sequence. Small risk, low-level components could be ignored in the system level integration and test sequence. It is then assumed that these components are integrated and tested on sub-system level. How to select the number of components suitable for integration and testing is discussed below.

The number of components in the system determines how many integration tasks are required to build the complete system. More integration tasks could lead to longer integration and test sequences, depending on the possible parallelism in the sequence. An example system is broken down into nine components in Figure 68(a) and into five components in Figure 68(b). The resulting integration sequences for the nine and five component system are depicted in Figure 69 and Figure 70 respectively. The number of assembly tasks in the sequence of the nine-component system is larger and therefore probably takes more time, depending on the duration of the individual tasks.

Splitting up a system into more components leads to longer integration sequences. A small number of components results in components that are too complex, too large, hence contain too much risk. Higher risk results in lengthy and costly test-diagnose-fix tasks, because the risk in the system needs to be reduced. Exclusion of risk can be done in two ways. First, *passed* test cases reduces the risk in the system because it is proven, by a passed test case, that some risk is not in the system. Second, *failed* test cases result in a diagnosis of the failure and a fix of the fault. Fixing a failure also reduces the risk, so does



(a) Example system split up into nine components

(b) Example system split up into five components

Figure 68. An example system split up into nine and five components

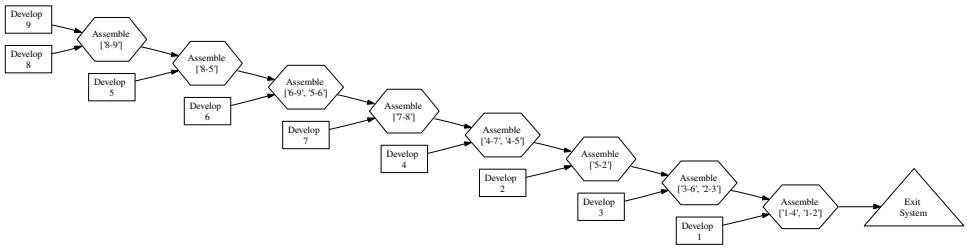


Figure 69. Integration sequence of the system depicted in Figure 68(a)

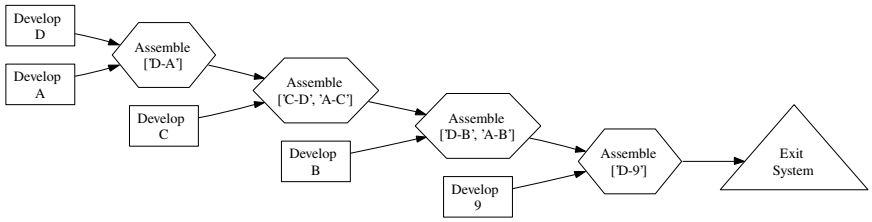


Figure 70. Integration sequence of the system depicted in Figure 68(b)

the diagnosis that a fault is not in the system. Both ways of reducing risk cost more time/money when the risk in the system is high. Consequently, a suitable number of components is a balancing act between the number of components to select and the size of the resulting integration sequence. Some guidelines for this balancing act are given below.

Guidelines for selecting components

Selecting the number of components to accommodate the integratability and testability is a balancing act. An algorithm that determines the optimal integration and test sequence, given the components and interfaces is NP-hard [Boumen et al., 2006a, 2007]. Comparing a number of component selections is therefore also NP-hard. Nevertheless, some guidelines for component selection are given below. The first guideline is most important, the other guidelines are exceptions on the first guideline.

- The risk of a component should guide the split up of a component. High risk components should be split up to accommodate: parallel integration, parallel testing and a shorter test-diagnose-fix task because less faults are found.

- Parallel integration and testing is only beneficial when the resulting integration and test sequence has a shorter duration than the original sequence. The resulting integration and test sequence can be longer in one of the following cases:

  - The development duration of both new components is longer, because additional time is spent on interface definition and development. This can be the case if two development groups are involved and/or the interface decisions are more difficult to make.
  - The assembly of the two components is difficult (because the interface agreements are more difficult to make).
  - The combination of components and interfaces is the risky part of the development, while the individual components are relatively simple. The additional parallel test-diagnose-fix tasks have no benefit in this case, because reducing the risk is done in the last test-diagnose-fix task and is therefore on the critical path in the integration and test plan.

- Splitting up components is only beneficial if the interfaces between the new components 'allow' splitting up these components. The properties of interfaces that 'allow' splitting up components are:

  - The number of (different) interfaces between the split up components should not increase, because the increase in interfaces could result in an increase of the duration of the assembly phase.

– The interface risk after splitting up a component into two components should be low. A component split-up that results in high risk interfaces results in a longer test phase after assembly. Examples of high risk interfaces after component split up are interfaces that are to be developed newly, because of the component split up, or interfaces with extreme requirements in terms of speed, tolerance, etc.

5.4.3 *Selection of a suitable set of interfaces*

Two approaches can be followed to select the suitable (integratable/testable) set of interfaces. The first approach is directly related to the selection of components, while the second approach involves the selection of a different interface paradigm. The first approach is a result of the component selection as discussed in the previous section. Splitting up components leads in general to more interfaces. E. g. dividing a single component with two external interfaces into two components leads to at least one additional interface between the two components. The selection criteria for suitable interfaces are similar for component selection in this setting. Dividing a component into two smaller components reduces the risk in the component. Dividing the component into two smaller components with (too) many interfaces between these components increases the risk, because of possible problems with definition, implementation and utilization of these interfaces.

Testability is increased by reducing the number of interfaces, interface usage and reducing the risk involved with this type of interfaces. Integratability is increased by reducing the time required to connect, disconnect and reconnect two components to each other, i. e. the assembly time and cost are minimal.

The second approach involves the selection of a different *interface paradigm* that increases the integratability and testability. Testability is increased by reducing the number of interfaces, interface usage and reducing the risk involved with this type of interfaces. Integratability is increased by reducing the time required to connect, disconnect and reconnect two components to each other, i. e. the assembly time and cost are minimal.

Example: A mechanical interface in the ASML wafer scanner

An example of a *mechanical* interface in an ASML wafer scanner that is specially chosen to increase the testability and integratability is the so called 'cable slab' between the body of the wafer scanner and a wafer stage. Figure 71 depicts the mechanical interface. The wafer stage is a sub-system of the wafer scanner, that needs to move in six degrees of freedom. Any contact of the wafer stage with the environment (the body) results in vibrations in the wafer stage system resulting in overlay and imaging problems. The required power, signal and airflow to control and move the wafer stage is supplied via the 'cable slab'. A special interface has been designed such that the stage can easily be replaced and tested. A specially designed interface imposes additional risk, because no common-of-the-shelf (COTS) interface is used. Additional 'cable slab' and sub-system testing was required for the 'cable slab', while the cable slab itself reduced risk of overlay and imaging problems on system level. The cable slab is an example of an interface where additional time and cost is spent to reduce the higher level system risk.

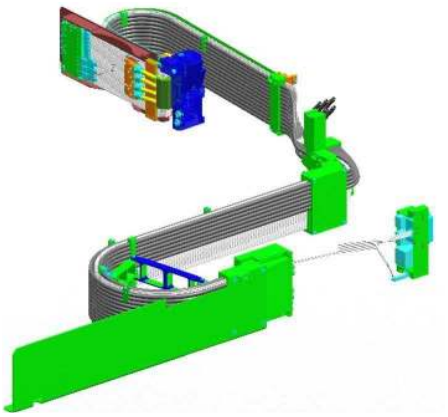


Figure 71. Cable slab

Next, some guidelines for selecting an interface paradigm are given. An interface paradigm that is suitable for integration and testing:

- reduces the number of interfaces or the risk involved with interfaces,
- reduces the bandwidth used by the interfaces,
- reduces the assembly or disassembly duration and cost,
- improves the testability and diagnosability by adding additional measurement capabilities,
- improves the ability to assemble and disassemble components.

5.4.4 *Selection of a suitable set of layerings*

Components and interfaces are the only elements that are necessary to create an integration and test plan. However, the number of possible integration and test plans increases dramatically when the number of components (and interfaces) increases. The selected *layering* for a system reduces the complexity of creating an integration and test plan. The functional and organizational layering can differ from the layering that is chosen for the benefit of integration and testing. Selecting a layering suitable for integration and testing means that the set of components and interfaces are grouped such that a good integration and test sequence can be created.

The functional and organizational layering can differ from the layering that is chosen for the benefit of integration and testing.

Example: Layering a car axle

An example car axle with wheels is depicted in Figure 72 and modeled as a four component architecture, depicted in Figure 73. The architecture consists of two

wheels (W_1 and W_2) and two steering knuckles (SK_1 and SK_2). The axle itself is modeled as the interface between the two steering knuckles. The functional

Figure 72. Example car axle system

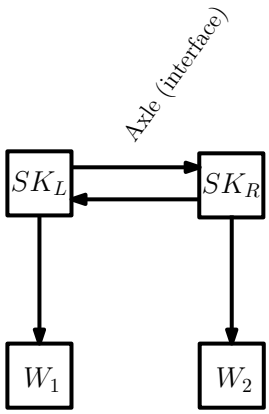
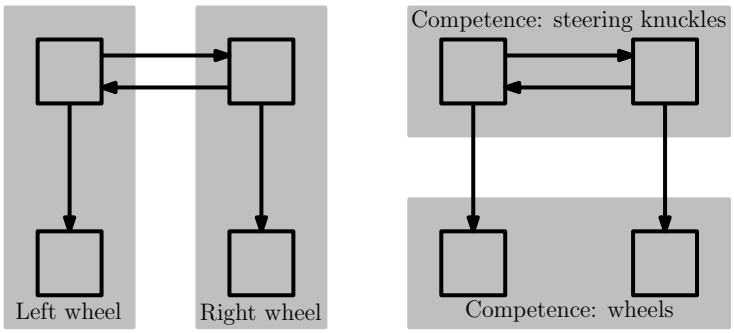


Figure 73. Car axle architecture

layering for this system could be defined as a layer for each side of the car as depicted in Figure 74(a). An organizational layering could be grouped according to the two competences involved in this system: wheels and steering knuckles. The organizational layering is depicted in Figure 74(b). The integration sequences for the functional and organizational layering have a different duration, assuming that assembling each interface costs the same time in this example. Figure 75 depicts the resulting integration sequence for the layering according to the competence. The first two assembly tasks assemble a wheel and a steering knuckle. The last assembly task assembles both wheels and steering knuckles with their common interface, the axle. A total of four interfaces is created. However, the first two interfaces are created in parallel. If the creation duration of



(a) Functional layering of the car axle system

(c) Organizational layering of the car axle system

Figure 74. Two layerings of the car axle system

each interface is one time unit, then the total duration of this sequence is three time units.

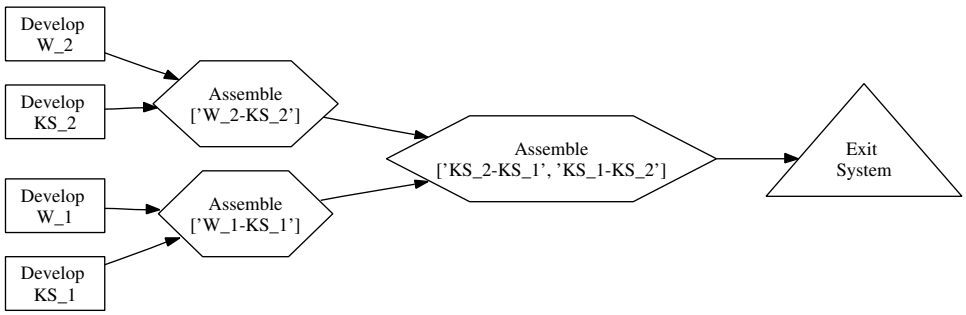


Figure 75. Integration sequence of the axle according to the functional layering

The integration sequence according to the organizational axis is depicted in Figure 76. Now, the steering knuckles are assembled first, followed by an assembly of the wheels to the steering knuckles. The duration of this integration sequence is four time units, assuming again that the creation of *each* interface costs one time unit and testing is not taken into account.

This example does not include test-diagnose-fix tasks. It is assumed that the components are available at the start of the integration sequence. Development durations and cost of components need to be taken into account as well as the test-diagnose-fix tasks.

Some guidelines are given such that a suitable layering can be chosen.

- The layering chosen during design of the system is not necessarily the best layering for integration and testing.
- Selecting a new layering for integration and testing can be done late in the

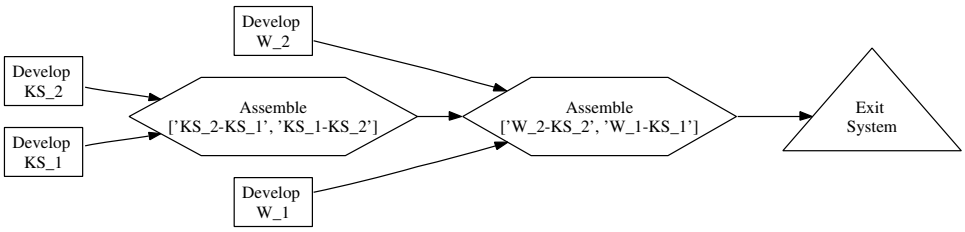


Figure 76. Integration sequence of the axle according to the organizational layering

process. See the next sub-section for details.

- If all possible sequences of components and interfaces that are able to form a system are considered, the optimal sequence can be selected. However, this approach makes planning an integration and test sequence for large systems complex and time consuming. To reduce this complexity, layering can be applied.

- Take the interfaces between components into consideration when choosing a layering. A layering that divides the system, such that the integration and test sequence can be executed as much as possible in parallel is beneficial.

- Layers containing components that are not connected result in less optimal integration and test plans.

5.4.5 *Illustration: TWINSCAN wafer scanner integration*

The TWINSCAN™ wafer scanner platform of ASML was designed and developed from 1997 onwards. In 2000-2001, the first TWINSCAN™ systems were shipped to customers. The initial design was set up such that the (relatively large) sub-systems are easily recognizable within the system and that these sub-systems are evolvable. The sub-systems are recognizable for the purpose of system level training and understandability. Evolvability was important for the purpose of creating a platform that served as a basis for the quick development of new system types. The technical design and layering was centered around the most important aspect of the TWINSCAN™ system at that time: productivity.

The TWINSCAN™ system was designed using a sub-system oriented architecture/layering. The system level specifications were broken down into sub-system level specifications and so on. The sub-systems could be identified easily in the system architecture. This architecture was suitable for breaking the high level problem into smaller problems and for managing this work-break-down structure.

However, system level integration requires cooperation of all sub-systems, such that the system function, run production, could be performed. A single

sub-system contributes to the performance of many other sub-systems that together result in a system level performance. The goal at integration time, is to combine all these sub-systems, with their individual performance requirements, into a complete and performing system.

'Life-of-a-wafer' describes in terms of *sub-functions* how production is run on a wafer scanner. The *sub-systems* in the wafer scanner contribute to one or more of the *sub-functions* in the 'Life-of-a-wafer' process.

Around one year before the first shipment, the intended function, run production, of the wafer scanner was split up in around 15 sub-functions. These functions were related to each other by a process description called 'life-of-a-wafer'. 'Life-of-a-wafer' describes in terms of *sub-functions* how production is ran on a wafer scanner. The *sub-systems* in the wafer scanner contribute to one or more of the *sub-functions* in the 'Life-of-a-wafer' process. The focus during integration and testing shifted from sub-system development to sub-function and function integration and qualification. The integration and test sequence reflected this new layering, because milestones for these functions were planned and function owners were assigned.

Shifting from a sub-system development approach to sub-function integration, a new layering was not an easy task. Many developers were convinced or forced to use the 'new' way-of-working. Functional milestones did not seem like progress for upper management and customers who wanted to see the results in terms of throughput, overlay or imaging performance. However, the developed systems were shipped on time and a record breaking number of additional system types were shipped in the next year. Furthermore, the system reliability and availability was brought to the required levels twice as fast if compared with the previous platform. Afterward, it can be concluded that changing the layering very late in the process, with the purpose of creating a better integration and test plan, was successful for the TWINSCAN™ platform.

5.4.6 *Conclusions*

The selection of an architecture that is suitable for testing and integration is important, because this choice influences the integration and test sequences that can be created. By this selection, the performance of an integration and test sequence is influenced. Often, a single architecture is considered for a system under test. Testability and integratability aspects are balanced together with other architectural aspects. A suitable architecture for integration and testing is chosen, if testability and integratability are aspects of high importance. Otherwise, the resulting architecture leads to a sub-optimal integration and test sequence. Selecting an architecture that is suitable for integration and testing requires a component selection, an interface selection and a selection of layers.

Component selection is a balancing act between the number of selected components and the risk of these components. Selecting too many components leads to a large integration and test sequence, while selecting too few components results in an integration and test sequence with too much risk in the individual components and long and unpredictable test-diagnose-fix tasks. Component selection determines what level of abstraction is used for the system. Consequently, an integration and test sequence is obtained with more or fewer

integration tasks.

Interface selection is related to component selection, because the selection of components determines what interfaces are selected. Interface selection by selecting a different interface paradigm could reduce the number of interfaces and the interface usage. Additionally, selecting an interface that does not fit in the interface paradigm could increase the number of interfaces and the usage of the interfaces.

Component and interface selections are, in principle, the only required aspects to consider for a suitable integration and test architecture. The components and interfaces are the inputs for the integration and test planning process. However, the number of possible integration and test sequences for real life systems is large. Selecting a layering for a system reduces the number of possible integration and test sequences that are to be considered. Therefore, layering is an important selection mechanism.

Component, interface and a layering can be selected late in the development process, when the system architecture does not change because of the selection. Additionally, selecting a different interface paradigm should be done as early as possible, because a change in the interface paradigm often requires that the individual components need to be changed as well. Implementing a new interface paradigm could introduce additional risk. The additional risk can be minimized by selecting an implementation of the interface paradigm that is stable already.

6 CONCLUSIONS

This chapter concludes the work performed in a four year project on 'Integration and test strategies for complex manufacturing machines'. This study has been performed at ASML and at Eindhoven University of Technology in the context of the TANGRAM research project. The results described in this thesis can be characterized as broad and methodological as well as specific and in-depth.

Three main contributions can be distinguished. The first contribution is the high-level *integration and test planning method* introduced in the Chapter 1. The chapters in this thesis correspond to the steps in the *integration and test planning method* and explain in detail the steps that need to be performed to create an integration and test plan. Where possible, existing or new models, algorithms and strategies are applied in the steps in the method. The used system integration and test models are explained in Chapter 2. The models and algorithms have been applied to practical integration and test planning problems as illustrated in the case studies performed. Not all models and improvement algorithms are mandatory to apply the integration and test planning method in a real-life context. The integration and test planning method serves as a framework for this thesis and as a framework for integration and test planning in practice.

The second main contribution of this thesis are the models that are defined and used for integration and test planning and improvement. The models that were selected were easy to define and well suited for the planning and improvement algorithms. The basic test model introduced by [Pattipati et al., Jan 1991] served as a starting point. The enhanced basic test model, the *system test model*, is used to model, analyze and improve real-life test-diagnose-fix tasks, whilst the simplicity of the system test model was not sacrificed. The simplicity of the system test models enabled the roll out of the developed methods and algorithms within ASML. Apparently, the system test models only formalize the knowledge of system integration and test engineers, because the roll out of the integration and test planning method would not have proceeded as it has until now. The system test model contains enough information for the analysis and comparison of single test-diagnose-fix tasks. In addition, system test models are used to model components and interfaces in the *system integration model*, the second model used in this method. Both models are used to analyze integration and test sequences as explained in Chapter 4. Additionally, the improvement algorithms, described in Chapter 5, use the system test model as basis.

The third main contribution of this thesis are the developed analysis and improvement techniques and algorithms. Two levels of analysis and improvement can be distinguished: analysis and improvement of test-diagnose-fix tasks, and analysis and improvement of integration and test sequences. Analysis and im-

provement of test-diagnose-fix tasks results in a reduction of the duration of test-diagnose-fix tasks from 61 to 17 hours and 142 to 94 hours, a gain of 71% and 34% respectively. The improvements are measured in terms of total test duration, because this is the most important business driver for ASML, whilst similar results for cost or quality are expected when the methods are applied in other organizations. Although, this has not been investigated. The actual results depend, of course, on the models that are made for the case studies, i. e. the system under test. A good example of the application of the modeling and the analysis techniques is the analysis of reliability test-diagnose-fix tasks as described in Section 4.3. This technique maps the SEMI-E10 reliability qualification standard onto the system test model, including the use of uncertainty as a measure for reliability *confidence*. This mapping enables the application of lower level reliability test cases, such that the system level reliability is qualified faster. Moreover, the system level uncertainty can be monitored throughout the integration and test sequence. The application of this reliability qualification method leads to a six-fold quality improvement for the performed case study. The analysis method for integration and test sequences leads to cost and remaining risk profiles as function of time. This way, the performance of the sequence as a whole is evaluated. The expected duration, cost and remaining risk for an integration and test sequence can be used to compare integration and test sequences. Partitioning test-diagnose-fix tasks into two parallel test-diagnose-fix tasks resulted in a reduction of the test duration from 16 to 11.5 hours. This is a reduction of 30% in test duration with a cost increase of 30%, because of parallel test execution.

Practical contributions

Four years of study into integration and test strategies also led to a number of practical contributions at ASML that did not end up in this thesis. Some of these practical contributions are described briefly here:

- The test process model (and simulator) were used to analyze the cycle time, also called flow time or sojourn time, of a new wafer scanner platform in the ASML factory. Existing cycle time measurements were used to calibrate the models of the current platform. The expected change in diagnosis and fix durations for the new platform was estimated. The new cycle time was estimated by simulation. This way, the change in cycle time of the new platform could be predicted. As a spin-off, the calibration of the current platform, using the test process model, provided valuable insight into the current cycle time. Cycle time improvements for the current platform are identified based on these results.

- A utilization measurement program on wafer scanners in the ASML factory and prototype systems was set up using the integration and test process model as basis. Measuring the utilization of wafer scanners in a *customer* manufacturing environment requires that productive time (run production) and nonproductive time (idle, down, maintenance, etc.) is

measured. For wafer scanners in the ASML factory, the productive time is the time that a system performs integration, test, diagnosis and fix tasks. A 'state model', based on the state model used to determine the effective process time of semiconductor equipment [A. de Ron and Rooda, Feb. 2005], representing the states in the integration and test process, has been developed to measure the performance of the wafer scanners in the ASML factory and for ASML prototype systems. These measurement results are currently used to guide utilization improvement projects.

- The integration and test planning method describes how a test sequence can be created and analyzed. An existing test sequence can also be taken as a starting point. For this purpose, test sequences, executed to manufacture ASML wafer scanners, are obtained for a number of newly manufactured wafer scanners. The analysis of these logs, obtained from these wafer scanners, revealed if the test sequences were actually executed in the specified order. Additionally, the duration of each step could be measured and matched with the target duration. For this purpose, a research project in cooperation with Van der Aalst and Rozinat of the Department of Information Systems of the Eindhoven University of Technology was performed [Rozinat et al., 2007]. Business process modeling algorithms were applied to the logs of the executed test sequences. The 'reference' test process model has been compared with the process model derived from the test logs and a number of improvements to the reference sequence were suggested. The application of algorithms to derive the test sequence as executed from available logging information can be seen as a formalization of step (3.5) of the integration and test planning method.

RESEARCH QUESTIONS

The research questions from Section 1.3 are answered in this section using the results of the previous chapters. Research question 1 relates to the observation that integration and test sequences are not the same across organizations.

Research question 1

- Which organizational factors have an impact on the integration and test plan for systems developed by that organization?

A large number of organizations have been visited and investigated throughout the TANGRAM project to answer research question 1. These organizations have been visited with the goal to investigate real-life integration and test sequences. Additionally, the different aspects of organizations have been recorded. Chapter 2 describes the different organizations and the observed integration and test sequences. Each integration and test sequence at each visited organization is unique.

Integration and test sequences observed in the different organizations can be characterized by two aspects: the flexibility in the organization and the complexity of the system, where complexity is measured in terms of number of components and the technology used for the components. The investigated organizations have been classified according to these two aspects in Figure 17. The order of the business drivers is one of the factors of an organization depicted in this figure. The order of the business drivers, denoted by Q-C-T (quality-cost-time) or T-Q-C for the organizations in Figure 17, determines if an organization is primarily quality driven or time-to-market driven. Note that the (integration and testing) flexibility of an organization corresponds with the classification of Q-C-T or T-Q-C in Figure 17. Therefore, the underlying answer to research question 1 is that the organizational factor that impacts the integration and test plan of an organization can be described in terms of the business drivers quality (Q), cost (C) or time (T). The priority of these business drivers determines how the integration and test plan is made.

Research question 2 relates to the difference in the observed integration plans, the elements in these integration plans and the performance indicators of an integration and test plan.

Research question 2

- What are the basic elements of an integration and test plan?
- What are the key performance indicators of an integration and test plan?
- How can these key performance indicators be measured and used to compare different integration and test plans with each other?

An integration and test plan consists of a sequence of integration and test tasks: an *integration and test sequence*. An *integration and test sequence* for a product in an organization is unique. All integration and test sequences are, however, composed of the same set of elements: *develop*, *assemble*, *disassemble*, *copy* and *test-diagnose-fix* . The answer to the first question is that these five elements are the basic elements of an integration and test plan. The details of each element and some typical combinations of elements are described in Section 2.2.

An integration and test planning method, which utilizes three strategies, is used to create these integration and test sequences. Many integration and test sequences can be created using this integration and test planning method. The performance of these integration and test sequences is evaluated based on: duration, cost and remaining risk. The remaining risk is our measure for product quality. The duration, cost and remaining risk can be analyzed for the complete integration and test sequence and for each of the tasks in the sequence. The duration, cost and remaining risk for the *develop*, *assembly*, *disassembly* and *copy* tasks can be estimated and are more or less deterministic. The duration, cost and remaining risk of test-diagnose-fix tasks depend on the faults in the system and the used test strategy. Therefore, testing is an inherently stochastic process.

The answer to the second question is that the performance indicators of an integration and test sequence are: duration, cost and remaining risk, where it should be noted that these performance indicators are stochastic variables, because of the stochastic nature of testing.

The answer to the third question is given in Chapter 4, where an integration and test process simulator is used to measure the performance of one or many simulated executions of an integration and test sequence. A simulator is used, because real-life evaluation of the performance of an integration and test sequence requires a large number of systems that are to be integrated. This situation only occurs in a manufacturing environment and not for integration and test sequences executed in a product development environment. The integration and test sequences are compared by comparing the expected duration, cost and remaining risk for the simulated integration and test sequences.

The ability to analyze and compare integration and test sequences is important, because in this way the results of the integration and test planning method can be evaluated. Improving an existing integration and test sequence, such that a better sequence is obtained is also important, because in this way shorter, cheaper sequences or products of higher quality can be obtained. Research question 3 relates to improving integration and test sequences.

Research question 3

- Which improvement techniques for an integration and test plan are beneficial for complex manufacturing machines?

Four improvement techniques have been investigated in this work. The first technique, updating the objectives and constraints, is not really an improvement technique. It is described only because it completes the framework with a technique that is often applied in practice. The benefit of this technique is that the deadline, cost limit or required product quality is adjusted. This adjustment could result in shorter or cheaper integration and test sequences or higher product quality.

The second improvement technique describes how a single test-diagnose-fix task can be improved by defining new test cases. The 'next-best-test-case' is derived from the system test model. The benefit of this technique is that test case development is guided, such that the test with the best test coverage can be developed.

The third improvement technique splits test-diagnose-fix tasks into two test-diagnose-fix tasks, which are executed in parallel. Parallel execution of test cases results in a reduction of the test-diagnose-fix duration by roughly a factor 2 if the coverage of the test cases on the fault states is not taken into account. An additional test duration reduction can be obtained by taking the coverage of the test into account. The benefit of this improvement technique is shown in the performed two case studies, which show a reduction of the test duration of 30%.

The fourth and last improvement technique describes how the system architecture can be changed, such that a better integration and test sequence is obtained. This technique, described in Section 5.2, presents guidelines to split

components and interfaces. Additionally, guidelines for the definition of a so called *layering* are presented. The benefit of this improvement technique is that the effect of architectural choices on the integration and test sequence are made explicit.

DIRECTIONS FOR FURTHER RESEARCH

Detailed directions for further research have been proposed in the different sections in this thesis. These directions for further research are not repeated here. Instead, three most relevant research direction are described in more detail.

The first research direction continues the work for planning test-diagnose-fix tasks described in Section 4.2. Additional test sequencing algorithms could be defined that lead to faster or cheaper test-diagnose-fix sequences or lower remaining product risk. This could also be the case for other test process configurations. The performance of new test sequencing algorithms and test process configurations should be investigated further, such that the test planning method can be used to analyze more test-diagnose-fix tasks.

The second research direction that could be investigated is the application of *on-line* integration strategies. The integration strategies described in Chapter 3 determine the sequence of integration tasks beforehand. These integration strategies are considered to be *off-line integration strategies* analogous to the off-line test sequencing algorithms described in Section 4.2. It could be investigated if *on-line integration strategies* are beneficial for the performance of integration (and test) sequences.

The third research direction that could be investigated further is the application of business process modeling to derive business process models for *integration* and test sequences. For this purpose, logs need to be produced by systems that are integrated and tested. A research project has been performed to analyze the logs of executed *test sequences* in the ASML factory [Rozinat et al., 2007]. In this project, a business process model of the executed test sequence was derived and compared with a 'reference business process model'. The diagnose, fix tasks and integration tasks were not recorded and therefore not taken into account. The test-business process model was rather complex and large. Abstractions on the test-business process model were required, such that the results could be analyzed. Further research should focus on the analysis of logs where integration, diagnosis and fix tasks are included. The resulting business process models are expected to be even more complex than the derived business process models from test logs. Consequently, additional abstractions and analysis techniques need to be investigated.

APPENDICES

A

APPENDIX A

This appendix explains the relation between the reliability qualification method according to the SEMI-E10 standard and the qualification method that is introduced in Section 4.3.

SEMI-E10 reliability qualification method

The reliability qualification method in the SEMI-E10 standard relies on Table A1-4 to determine the test duration. The so-called k-factor in Table A1-4 must be multiplied with the target mean-time-between-failure to obtain the test duration: The test duration Φ that is required to reach the confidence level is determined according to:

$$\Phi_{SEMI} = k \cdot MTBF_{target} \tag{A.1}$$

The k-factors in Table A1-4 can be derived manually using a χ^2 distribution [NIST/SEMATECH, 2003-2006]. The required 'confidence'-level and the number of degrees of freedom are used as inputs for the χ^2 distribution. The number of degrees of freedom ν is determined using the maximum allowed number of failures r. Thus, $\nu = 2(r+1)$.

The relation between the SEMI-E10 standard and our reliability qualification method is defined for the maximum allowed number of failures of 0 (and any confidence level). In this specific situation, the number of degrees of freedom equals $2(0+1) = 2$. The situation where $r > 0$ is explained at the end of this appendix.

A χ^2 distribution with 2 degrees of freedom is equal to the exponential distribution. The relation between the remaining uncertainty $U_R = 1-\alpha$ and the k-factor for an exponential distribution is:

$$1 - \alpha = U_R = \lambda e^{-\lambda k} = e^{-k} \tag{A.2}$$

because the initial uncertainty equals 1, $\lambda = 1$. Note that the remaining uncertainty level is described here as $1-\alpha$, where α is the confidence level that is to be reached. The k-factor that is required such that the remaining uncertainty level is reached is derived from Equation (A.2):

$$k = -\ln U_R \tag{A.3}$$

The test duration Φ_{SEMI} that is required to reach the confidence level is determined according to:

$$\Phi_{SEMI} = k \cdot MTBF_{target} = -\ln U_R \cdot MTBF_{target} \tag{A.4}$$

Reliability qualification method according to Section 4.3

We relate the SEMI-E10 standard to our method using a system test model D that contains a single reliability test case, which corresponds to the 'run production' test case in the SEMI-E10 standard. A single fault state is used to model that the system possibly does not meet the reliability specification. This single test case is repeated n times, such that the remaining uncertainty level is reached.

The uncertainty reduction due to the single execution of a test case is defined in Equation (4.35):

$$U(s, Gt) = U(s, G)(1 - R_{ts}(t, s))$$

where Gt describes test sequence G followed by test case t and $R_{ts} = \frac{1}{MTBF_{target}}$ as described in Section 4.3.2. We assume that all test cases have a duration of 1 hour. If this is not the case, the coverage relations of the test cases need to be updated to model the uncertainty reduction such that the test case duration equals one hour. This way, the duration of n executions of a single test case is equal to the test duration Φ_D. Where, Φ_D represents the duration of the reliability qualification phase using *system test model D*. The uncertainty reduction due to n executions of the same test case according to Equation (4.37) equals :

$$U(s, G) = (1 - R_{ts}(t, s))^n \tag{A.5}$$

The number of test cases to execute, such that the uncertainty level is reached is derived by rearranging Equation (A.5) into:

$$n = log_{1-\frac{1}{MTBF_{target}}} U(s, G) = \Phi_D \tag{A.6}$$

Relating SEMI-E10 to our method

The SEMI-E10 standard and our method for the single test case system test model are related if Equations (A.4) and (A.6) are equal:

$$\Phi_{SEMI} = \Phi_D \tag{A.7}$$

$$-\ln\beta \cdot MTBF_{target} = log_{1-\frac{1}{MTBF_{target}}}\beta \tag{A.8}$$

where, $\beta = U_R = U(s, G)$, because the same target uncertainty level is to be reached for both methods.

The following steps rewrite Equation (A.8):

$$-\ln\beta = \frac{\ln\beta}{MTBF_{target}\ln(1 - \frac{1}{MTBF_{target}})} \tag{A.9}$$

$$-1 = \frac{1}{MTBF_{target}\ln\left(1 - \frac{1}{MTBF_{target}}\right)} \tag{A.10}$$

$$\ln\left(1 - \frac{1}{MTBF_{target}}\right) = -\frac{1}{MTBF_{target}} \tag{A.11}$$

After rewriting Equation (A.10) into Equation (A.11), it appears that the target uncertainty is not relevant anymore for the relation of both functions.

Applying Taylor series ($\ln(1-z) = -z - \frac{z^2}{2} - \frac{z^3}{3} - \frac{z^4}{4} - \ldots$) to the left side of the equation results in:

$$-\frac{1}{MTBF_{target}} - \frac{(\frac{1}{MTBF_{target}})^2}{2} - \frac{(\frac{1}{MTBF_{target}})^3}{3} - \frac{(\frac{1}{MTBF_{target}})^4}{4} - \ldots = -\frac{1}{MTBF_{target}} \tag{A.12}$$

Therefore the difference between both methods is:

$$\Phi_{SEMI} - \Phi_D = -\frac{(\frac{1}{MTBF_{target}})^2}{2} - \frac{(\frac{1}{MTBF_{target}})^3}{3} - \frac{(\frac{1}{MTBF_{target}})^4}{4} - \ldots \tag{A.13}$$

For higher MTBF targets this difference between both methods can be neglected. A typical MTBF target for production systems is 300 hours. The difference between both methods is $-5 \cdot 10^{-6}$ in this case.

Failures during test execution

SEMI-E10 handles failures during test execution in Table A1-4. The number of 'allowed' failures can be chosen and the corresponding *k*-factor can be derived using χ^2 distribution or looked up in Table A1-4. The degrees of freedom are increased when more fault states are allowed. This way the χ^2 distribution shifts, resulting in higher *k*-factors and therefore in higher test durations.

Failures are modeled explicitly in our method. Once a failure occurs, the uncertainty of the corresponding fault state is increased to 1.0 indicating that the previous knowledge is not valid anymore. Modeling the failure explicitly is conservative if compared with the χ^2 distribution with $r > 0$, because the SEMI-E10 method assumes that *r* failures occur according to a certain failure distribution. In the worst case situation this is an exponential, or memoryless, distribution. Our method does not assume a distribution, but in the best case failures occur with the same exponential distribution as the SEMI-E10 method. In the worst case the failures occur at the end of the test period, which corresponds with a very skewed Gamma distribution. In this worst case, more test cases need to be executed in our method to reduce the uncertainty. Our approach is therefore more conservative and could require more executed test cases until the uncertainty target is met.

APPENDIX B

SYSTEM TEST MODELS USED IN SECTIONS 4.1 AND 4.2

S / T	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	P	I	C_F	φ_F
s_1	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0.1	100.0	70.0	70.0
s_2	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0.01	100.0	70.0	70.0
s_3	0	0	0	0	0	0	0	0	1.0	1.0	0	0	0	0.02	100.0	70.0	70.0
s_4	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0.05	100.0	70.0	70.0
s_5	0	0	0	0	1.0	0	0	0	1.0	1.0	0	0	0	0.05	100.0	70.0	70.0
s_6	0	0	0	0	1.0	1.0	1.0	1.0	0	0	0	1.0	0	0.2	100.0	70.0	70.0
s_7	1.0	0	1.0	0	0	0	0	0	0	0	0	0	1.0	0.05	100.0	70.0	70.0
s_8	0	1.0	0	0	0	1.0	0	0	0	0	0	0	1.0	0.02	100.0	70.0	70.0
s_9	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0.02	100.0	70.0	70.0
s_{10}	1.0	1.0	1.0	0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.07	100.0	70.0	70.0
s_{11}	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0.02	100.0	70.0	70.0
s_{12}	0	0	0	0	0	1.0	0	0	0	0	0	0	1.0	0.99	100.0	70.0	70.0
s_{13}	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.1	100.0	70.0	70.0
s_{14}	1.0	0	0	0	0	1.0	1.0	1.0	1.0	1.0	0	0	0	0.01	100.0	70.0	70.0
s_{15}	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0.99	100.0	70.0	70.0
s_{16}	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.99	100.0	70.0	70.0
s_{17}	0	1.0	0	0	0	1.0	0	0	0	0	0	0	1.0	0.99	100.0	70.0	70.0
s_{18}	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0.5	100.0	70.0	70.0
s_{19}	0	0	0	1.0	1.0	0	0	0	0	0	0	0	0	0.02	100.0	70.0	70.0
s_{20}	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0.99	100.0	70.0	70.0
s_{21}	0	0	0	0	0	1.0	0	1.0	0	0	0	0	0	0.99	100.0	70.0	70.0
s_{22}	0	0	0	0	0	0	1.0	1.0	0	0	0	0	0	0.99	100.0	70.0	70.0
s_{23}	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0.1	100.0	70.0	70.0
s_{24}	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0.99	100.0	70.0	70.0
s_{25}	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0.25	100.0	70.0	70.0
s_{26}	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0.99	100.0	70.0	70.0
s_{27}	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0.05	100.0	70.0	70.0
s_{28}	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0.25	100.0	70.0	70.0
s_{29}	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0.02	100.0	70.0	70.0
C_T	30.0	15.0	5.0	10.0	10.0	10.0	20.0	10.0	10.0	10.0	10.0	5.0	40.0				
C_D	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0				
φ_D	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0	70.0				

Table 39. System test model of case study 3 in Section 4.3

S / T	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	P	I	C_F	φ_F
s_1	0	0	0	0.01	0	0	0	0.01	0	0.01	0	0.01	0.01	0.01	0.01	0.01	0	0	0	0	0	0	0.3	1.0	1.0	1.0
s_2	0	0	0.7	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0.3	0	0	0	0.3	0.8	1.0	1.0	1.0
s_3	0	0	0.6	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0.2	0	0	0	0.3	0.8	1.0	1.0	1.0
s_4	0.1	0	0.1	0	0.1	0.1	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0.3	1.0	1.0	1.0
s_5	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0.4	0	0	0	0.2	0.8	1.0	1.0	1.0
s_6	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0.1	0.3	1.0	1.0	1.0
s_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0.3	1.0	1.0	1.0
s_8	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	1.0	1.0	1.0
s_9	0.1	0	0.2	0.1	0	0.1	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0.3	1.0	1.0	1.0
s_{10}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{11}	0	0	0	0	0	0	0.01	0.01	0.01	0.01	0.01	0	0	0.01	0	0	0	0	0	0.01	0	0	0.01	1.0	1.0	1.0
s_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0.01	1.0	1.0	1.0
s_{13}	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{14}	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{15}	0	0	0	0	0	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{16}	0	0	0	0	0	0	0	0	0.9	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{17}	0	0	0	0	0	0	0	0.9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{18}	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{19}	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{20}	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{21}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.9	0	0.01	1.0	1.0	1.0
s_{22}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{23}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{24}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{25}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{26}	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{27}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{28}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0.1	0.7	0.01	1.0	1.0	1.0
s_{29}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8	1.0	1.0	1.0
s_{30}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{31}	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
C_T	0.5	0.6	3.0	0.4	0.1	1.0	0.1	0.4	0.2	0.4	0.1	1.2	1.5	0.3	0.9	1.2	0.4	0.9	1.2	0.2	1.8	5.5				
C_D	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0				
φ_D	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0				

Table 40. System test model as used in case study 1 in Section 4.1. Part 1 (test case 1 - 22)

S / T	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}	t_{28}	t_{29}	t_{30}	t_{31}	t_{32}	t_{33}	t_{34}	t_{35}	t_{36}	t_{37}	t_{38}	t_{39}	t_{40}	t_{41}	t_{42}	t_{43}	t_{44}	P	I	C_F	φ_F
s_1	0	0	0	0	0	0.01	0	0.01	0.01	0	0	0	0	0	0	0	0.01	0.01	0.01	0.01	0	0.01	0.3	1.0	1.0	1.0
s_2	0	0	0	0.2	0.01	0	0	0	0.1	0	0	0	0	0	0	0.1	0	0.1	0	0	0	0.01	0.8	1.0	1.0	1.0
s_3	0.6	0	0	0.2	0	0	0.1	0.2	0	0	0.1	0	0.1	0	0	0.1	0.1	0.2	0	0	0.1	0.01	0.8	1.0	1.0	1.0
s_4	0.4	0.1	0.2	0	0	0	0	0.1	0	0	0	0	0	0	0	0.1	0	0	0	0	0.1	0	0.3	1.0	1.0	1.0
s_5	0.4	0	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0.8	1.0	1.0	1.0
s_6	0.1	0	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0.3	1.0	1.0	1.0
s_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	1.0	1.0	1.0
s_8	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0.01	0	0	0	0	0.3	1.0	1.0	1.0
s_9	0.6	0	0.1	0	0	0	0	0.1	0	0	0.1	0	0.1	0	0	0	0	0	0	0	0.1	0	0.3	1.0	1.0	1.0
s_{10}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.01	1.0	1.0	1.0
s_{11}	0	0	0.01	0	0	0	0	0.01	0	0	0	0.1	0	0	0.01	0	0.01	0.01	0	0	0	0	0.01	1.0	1.0	1.0
s_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0.01	1.0	1.0	1.0
s_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{17}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{18}	0	0	0.1	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{19}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.01	1.0	1.0	1.0
s_{20}	0.1	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{21}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{22}	0	0.9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{23}	0	0	0	0	0	0	0	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{24}	0	0	0	0	0	0	0	0	0	0	0.9	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{25}	0	0	0	0	0	0	0	0	0	0	0	0	0.9	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{26}	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{27}	0	0	0	0	0	0	0	0	0	0	0	0.9	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{28}	0.1	0	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{29}	0	0	0	0	0	0	0	0	0	0.9	0	0	0	0	0	0	0	0	0	0	0	0	0.8	1.0	1.0	1.0
s_{30}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0.01	1.0	1.0	1.0
s_{31}	0	0	0.7	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	1.0	1.0	1.0
C_T	8.0	0.5	1.1	0.5	1.2	0.4	2.0	1.8	0.9	1.2	0.4	4.8	0.2	0.5	0.1	0.3	1.0	1.1	0.1	13.4	0.9	20.0				
C_D	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0				
φ_D	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0				

Table 41. System test model as used in case study 1 in Section 4.1. Part 2 (test case 23 - 44)

S / T	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}	P	I	C_F	φ_F
s_1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.04582	1.0	180.0	180.0
s_2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.04582	1.0	180.0	180.0
s_3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.04582	1.0	180.0	180.0
s_4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.04582	1.0	180.0	180.0
s_5	1.0	1.0	1.0	0	1.0	1.0	1.0	1.0	1.0	1.0	0	1.0	1.0	1.0	1.0	1.0	1.0	0	0	1.0	1.0	1.0	1.0	1.0	1.0	0.04582	1.0	180.0	180.0
s_6	1.0	1.0	1.0	0	1.0	1.0	1.0	1.0	1.0	1.0	0	1.0	1.0	1.0	1.0	1.0	1.0	0	0	1.0	1.0	1.0	1.0	1.0	1.0	0.04582	1.0	180.0	180.0
s_7	1.0	1.0	1.0	1.0	0	1.0	1.0	1.0	1.0	0	0	0	1.0	1.0	0	1.0	0	1.0	0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
s_8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.04582	1.0	180.0	180.0
s_9	0	0	1.0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180.0	180.0
s_{10}	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.08317	1.0	180.0	180.0
s_{11}	1.0	1.0	1.0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{12}	1.0	1.0	1.0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{13}	1.0	1.0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{14}	1.0	1.0	1.0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{15}	0	0	0	1.0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
s_{16}	0	0	0	1.0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
s_{17}	1.0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0.16635	1.0	180.0	180.0
s_{18}	1.0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16635	1.0	180.0	180.0
s_{19}	1.0	1.0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{20}	1.0	1.0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.23296	1.0	180.0	180.0
s_{21}	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
s_{22}	0	0	0	1.0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.28841	1.0	180.0	180.0
s_{23}	0	0	0	1.0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.28841	1.0	180.0	180.0
s_{24}	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0.28841	1.0	180.0	180.0
s_{25}	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.23296	1.0	180.0	180.0
s_{26}	1.0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.12899	1.0	180.0	180.0
s_{30}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{31}	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0.31613	1.0	180.0	180.0
s_{32}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.31613	1.0	180.0	180.0
s_{34}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
s_{36}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
s_{37}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
s_{38}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180.0	180.0
s_{39}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180.0	180.0
s_{40}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180.0	180.0
s_{41}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0.28841	1.0	180.0	180.0
s_{42}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{43}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{44}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180.0	180.0
s_{45}	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{46}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{47}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0.09202	1.0	180.0	180.0
s_{48}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0.09202	1.0	180.0	180.0
s_{49}	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{50}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{51}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0.09202	1.0	180.0	180.0
s_{52}	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0.23296	1.0	180.0	180.0
s_{53}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0.09202	1.0	180.0	180.0
s_{54}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{55}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16635	1.0	180.0	180.0
s_{56}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180.0	180.0
s_{57}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.46168	1.0	180.0	180.0
s_{58}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180.0	180.0
s_{59}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
s_{60}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
s_{63}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
s_{66}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.23296	1.0	180.0	180.0
s_{67}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16635	1.0	180.0	180.0
s_{68}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16635	1.0	180.0	180.0
s_{69}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180.0	180.0
C_T	40.0	480.0	120.0	120.0	30.0	240.0	60.0	90.0	30.0	260.0	480.0	30.0	30.0	20.0	60.0	180.0	30.0	60.0	60.0	10.0	60.0	90.0	40.0	60.0	30.0	45.0			
C_D	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0			
φ_D	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0	180.0			

Table 42. System test model of case study 2 in Section 4.2 (part 1/2)

S / T	t_{28}	t_{29}	t_{30}	t_{31}	t_{32}	t_{33}	t_{34}	t_{35}	t_{36}	t_{37}	t_{38}	t_{39}	t_{40}	t_{41}	t_{42}	t_{43}	t_{44}	t_{45}	t_{46}	t_{47}	t_{48}	t_{49}	t_{50}	t_{51}	t_{52}	t_{53}	t_{54}	P	I	C_F	φ_F
s_1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0	0	1.0	0	1.0	1.0	0	0	0	1.0	0.04582	1.0	180	180
s_2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0	0	0	1.0	1.0	1.0	1.0	1.0	0	0	1.0	0.04582	1.0	180	180
s_3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0	0	0	1.0	0	1.0	1.0	0	0	0	1.0	0.04582	1.0	180	180
s_4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0	0	1.0	0	0	0	1.0	0	1.0	1.0	0	0	0	1.0	0.04582	1.0	180	180
s_5	1.0	1.0	1.0	1.0	1.0	1.0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	1.0	0	1.0	1.0	0	1.0	0	1.0	0.04582	1.0	180	180
s_6	1.0	1.0	1.0	1.0	1.0	1.0	0	0	0	0	0	0	0	0	0	0	1.0	1.0	0	1.0	0	1.0	1.0	0	0	0	1.0	0.04582	1.0	180	180
s_7	0	0	0	0	0	0	0	0	1.0	1.0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0.04582	1.0	180	180
s_8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0	1.0	1.0	1.0	0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0	0	1.0	0.04582	1.0	180	180
s_9	0	0	0	0	0	0	0	1.0	0	0	0	0	1.0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180	180
s_{10}	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0	1.0	1.0	1.0	1.0	0	1.0	0	0	0	0	0	0	1.0	0	0.08317	1.0	180	180
s_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180	180
s_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180	180
s_{17}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16635	1.0	180	180
s_{18}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0.16635	1.0	180	180
s_{19}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{20}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.23296	1.0	180	180
s_{21}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180	180
s_{22}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.28841	1.0	180	180
s_{23}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.28841	1.0	180	180
s_{24}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.28841	1.0	180	180
s_{25}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0.23296	1.0	180	180
s_{26}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0.12899	1.0	180	180
s_{30}	0	0	0	0	1.0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{31}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.31613	1.0	180	180
s_{32}	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.31613	1.0	180	180
s_{34}	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180	180
s_{36}	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180	180
s_{37}	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180	180
s_{38}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180	180
s_{39}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180	180
s_{40}	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180	180
s_{41}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.28841	1.0	180	180
s_{42}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{43}	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{44}	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08317	1.0	180	180
s_{45}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{46}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{47}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{48}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{49}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{50}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0.09202	1.0	180	180
s_{51}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{52}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.23296	1.0	180	180
s_{53}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{54}	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09202	1.0	180	180
s_{55}	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16635	1.0	180	180
s_{56}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0.09202	1.0	180	180
s_{57}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0.46168	1.0	180	180
s_{58}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0.08317	1.0	180	180
s_{59}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180	180
s_{60}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180	180
s_{63}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180	180
s_{66}	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.23296	1.0	180	180
s_{67}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0.16635	1.0	180	180
s_{68}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0.16635	1.0	180	180
s_{69}	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04582	1.0	180	180
C_T	30.0	45.0	0.01	120.0	45.0	45.0	120.0	150.0	20.0	30.0	120.0	120.0	30.0	15.0	30.0	360.0	30.0	60.0	30.0	15.0	30.0	120.0	30.0	240.0	20.0	1920.0	30.0				
C_D	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180				
φ_D	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180				

Table 43. System test model of case study 2 in Section 4.2 (part 2/2)

S / T	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	U
s_1 WP + MI	0.04	0.12	0.0008	0.08	0.002	0.0005	0.0005	0.08	0.005	0.0025	0.04	0.02	0.08	0.0005	0.02	1.0
s_2 IA/QM	0.04	0.04	0.04	0.048	0.2	0.04	0.04	0.008	0.005	0.0025	0.04	0.02	0.08	0	0.02	1.0
s_3 WA	0.04	0.04	0	0.04	0.004	0.08	0.08	0.32	0.005	0.0025	0.04	0.02	0	0	0.02	1.0
s_4 RH	0.04	0.04	0.04	0.04	0.08	1	1	0.04	0.04	0.04	0.04	0.02	0.04	0	0.04	1.0
s_5 PR	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.02	0.04	0	0.04	1.0
s_6 ME: WPlane	0.04	0.04	0	0.04	0.004	0.001	0.001	0.04	0.005	0.0025	0.04	0.02	0	0.4	0.08	1.0
s_7 MC: LO	0.04	0.04	0.04	0.04	0.08	0.08	0.08	0.08	0.04	0.04	0.04	0.02	0	0	0.06	1.0
s_8 ME: Grid + AI	0.04	0.04	0.0133	0.04	0.08	0.08	0.08	0.08	0.04	0.04	0.04	0.02	0	0.4	0.06	1.0
s_9 IL	0.04	0.04	0.0004	0.04	0.04	0.04	0.04	0.04	1	1	0.04	0.02	0	0	0.02	1.0
s_{10} RM	0.04	0.12	0.0008	0.03	0.002	0.0005	0.0005	0.08	0.005	0.0025	1	0.02	0	0	0.02	1.0
s_{11} DC	0.04	0.04	0.0004	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.02	0	0	0.02	1.0
s_{12} WH	0.04	0.03	0.08	0.048	0.004	0.0005	0.0005	0.008	0.005	0.0025	0.04	0.015	0.001	0.0005	0.02	1.0
φ_T	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	
C_T	1	1	1	1	1	1	1	1	1	1	1	1	1	1		
C_D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
φ_D	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Table 44. System model of case study 1 in Section 4.2

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	U
s_1: System	0.007	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	1.0
s_2: SF	0.007	0.007	0.003	0.003	0.000	0.010	0.010	0.010	0.002	0.002	0.010	0.002	1.0
s_3: ME	0.007	0.019	0.133	0.033	0.067	0.013	0.013	0.013	0.000	0.000	0.033	0.000	1.0
s_4: IS	0.007	0.003	0.020	0.000	0.020	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.0
s_5: AM	0.007	0.008	0.007	0.000	0.000	0.020	0.020	0.020	0.000	0.000	0.000	0.100	1.0
s_6: MO	0.007	0.020	0.033	0.000	0.000	0.027	0.027	0.027	0.000	0.000	0.013	0.000	1.0
s_7: WH	0.007	0.021	0.001	0.000	0.000	0.000	0.000	0.000	0.167	0.167	0.000	0.000	1.0
s_8: WS	0.007	0.024	0.067	0.003	0.033	0.000	0.000	0.000	0.020	0.020	0.003	0.000	1.0
s_9: RS	0.007	0.017	0.003	0.003	0.027	0.013	0.013	0.013	0.000	0.000	0.003	0.003	1.0
φ_T	1	1	1	1	1	1	1	1	1	1	1	1	
C_T	1	1	1	1	1	1	1	1	1	1	1	1	
C_D	0	0	0	0	0	0	0	0	0	0	0	0	
T_D	0	0	0	0	0	0	0	0	0	0	0	0	

	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	U
s_1 System	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	1.0
s_2 SF	0.002	0.002	0.002	0.013	0.020	0.027	0.013	0.007	0.007	0.000	1.0
s_3 ME	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.067	1.0
s_4 IS	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.027	0.000	0.000	1.0
s_5 AM	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.0
s_6 MO	0.013	0.067	0.027	0.027	0.027	0.033	0.027	0.027	0.020	0.013	1.0
s_7 WH	0.000	0.000	0.013	0.013	0.013	0.013	0.013	0.013	0.000	0.000	1.0
s_8 WS	0.000	0.000	0.047	0.047	0.047	0.053	0.053	0.040	0.027	0.013	1.0
s_9 RS	0.001	0.053	0.003	0.003	0.003	0.053	0.053	0.040	0.027	0.013	1.0
φ_T	1	1	1	1	1	1	1	1	1	1	
C_T	1	1	1	1	1	1	1	1	1	1	
C_D	0	0	0	0	0	0	0	0	0	0	
T_D	0	0	0	0	0	0	0	0	0	0	

Table 45. Split system model of case study 2 in Section 4.2

BIBLIOGRAPHY

S. Amland, "Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study," *Journal of Systems and Software*, vol. 53, no. 3, pp. 287–295, 2000. (Cited on pages 11, 36, 67, and 93.)

M. Arenthoft, J. Fuchs, Y. Parrod, A. Gasquet, J. Stader, and I. Stokes, "OPTIMUM-AIV: A planning and scheduling system for spacecraft AIV," *Future generation Computer Systems*, vol. 7, pp. 403–412, 1991. (Cited on page 20.)

ASML, "Website ASML NV, Veldhoven, The Netherlands, www.asml.com," Internet, 2007.
URL: *www.asml.com* (Cited on page 85.)

BCS/SIGIST, *Standard for Software Component Testing*, British Computer Society, 2001, british Computer Society Special Interest Group in Software Testing. (Cited on pages 64 and 65.)

B. Beizer, *Software Testing Techniques*, New York, NY, USA: John Wiley & Sons, Inc., 1990. (Cited on page 46.)

O. Benkahla, F. Chevassu, B. Remy, and C. Robach, "Performance evaluation of testing strategies in parallel systems," *EUROMICRO*, p. 0371. (Cited on page 65.)

B. Boehm, *Software engineering economics*, Prentice Hall, 1981. (Cited on page 1.)

R. Boumen, "Integration and test plans for complex manufacturing systems," Ph.D. thesis, Eindhoven University of Technology, 2007. (Cited on pages 35, 52, 53, and 145.)

R. Boumen, I. de Jong, J. Mestrom, J. van de Mortel-Fronczak, and J. Rooda, "Integration sequencing in complex manufacturing systems," SE Report 2006-02, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2006a.
URL: *http://se.wtb.tue.nl/sereports* (Cited on pages 48, 52, and 149.)

R. Boumen, I. de Jong, J. Mestrom, J. van de Mortel-Fronczak, and J. Rooda, "Integration and test sequencing for complex systems," SE Report 2007-07, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2007.
URL: *http://se.wtb.tue.nl/sereports* (Cited on page 149.)

R. Boumen, I. de Jong, J. Vermunt, J. van de Mortel-Fronczak, and J. Rooda, "A risk-based stopping criterion for test sequencing," Internal Report SE 420460, Eindhoven University of Technology, 2006b, Revised version submitted to IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans. (Cited on pages 93 and 94.)

R. Boumen, I. de Jong, J. Vermunt, J. van de Mortel-Fronczak, and J. Rooda, "Test sequencing in complex manufacturing systems," *IEEE Transactions on Systems,Man,and Cybernetics-Part A: Systems and Humans*, vol. 38, no. 1, pp. 25–37, Jan. 2008. (Cited on pages 34, 38, 74, 93, 94, 102, 125, and 128.)

J. Bowles, "The new SAE FMECA standard," *Reliability and Maintainability Symposium, 1998. Proceedings., Annual*, pp. 48–53. (Cited on pages 35 and 98.)

N. Braspenning, R. Boumen, J. van de Mortel-Fronczak, and J. Rooda, "A quantitative method to decide where and when it is profitable to use models for integration and testing," Systems Engineering report 2007-14, Eindhoven University of Technology, 2007, ISSN: 1872-1567. (Cited on pages 47 and 145.)

T. Brugman, "Testing high technology developments at ASML," in *Proceedings of the 9th Nederlandse Testdag*, J. Tretmans and P. Koopman, Eds., Software Technologie Groep, NIII, 2003. (Cited on page 1.)

T. Brugman and F. Beenker, "Summary project plan for the TANGRAM project on model-based testing," , 2003.
URL: *http://www.esi.nl/tangram* (Cited on page 8.)

H. Buus, R. McLees, M. Orgun, E. Pasztor, and L. Schultz, "777 flight controls validation process," *IEEE Transactions on Aerospace and electronic systems*, vol. 33, no. 2, pp. 656–666, 1997. (Cited on page 19.)

K.-Y. Cai, T. Jing, and C.-G. Bai, "Partition testing with dynamic partitioning," in *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, IEEE Computer Society, 2005. (Cited on page 65.)

Ü. Çatalyürek and C. Aykanat, "Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10(7), pp. 673–693, 1999. (Cited on pages 105 and 106.)

Ü. Çatalyürek and C. Aykanat, "Patoh: Partitioning tools for hypergraphs," Tech. rep., Bilkent University, 2002. (Cited on pages 103 and 106.)

H. Chuma, "Increasing complexity and limits of organization in the microlithography industry: implications for science-based industries," *Research Policy*, vol. 35, no. 3, pp. 394–411, 2006.
URL: *http://ideas.repec.org/a/eee/respol/v35y2006i3p394-411.html* (Cited on page 16.)

E. Coffman Jr., M. Garey, and D. Johnson, *Approximation Algorithms for NP-Hard Problems*, chap. Approximation Algorithms for Bin Packing: A Survey, pp. 46–93, PWS Publishing, Boston, 1997.
URL: *http://www.research.att.com/ dsj/papers.html* (Cited on page 102.)

M. Cusumano and R. Selby, "How microsoft builds software," *Communications of the ACM*, vol. 40, no. 6, pp. 53–61, 1997. (Cited on page 22.)

I. Duff, R. Grimes, and J. Lewis, "The Rutherford-Boeing sparse matrix collection," Tech. rep., Rutherford Appleton Laboratory, 1997. (Cited on page 104.)

E. Engel, I. Bogomolni, S. Shachar, and A. Grinman, "Gathering historical life-cycle quality costs to support optimizing the vvt process," in *Proceedings of the 14th International Symposium of INCOSE, Toulouse, France. CD-ROM.*, 2004. (Cited on page 1.)

ETSI, "Testing standards for GSM/GPRS/3G telecommunication devices and infrastructure," www.etsi.org, 1999-2007. (Cited on page 18.)

D. Farren and A. Ambler, "Cost-effective system-level test strategies," in *Proceedings of the IEEE International Test Conference on Driving Down the Cost of Test*, Washington, DC, USA: IEEE Computer Society, 1995, pp. 807–813. (Cited on page 65.)

C. Fiduccia and R. Mattheyses, "A linear-time heuristic for improving network partitions," *Design Automation, 1982. 19th Conference on*, pp. 175–181. (Cited on page 106.)

A. Flint, "Three different MCMs, three different test strategies," *Proceedings of the International Test Conference*, pp. 828–833. (Cited on page 65.)

M. Garey and D. Johnson, "Some simplified NP-complete graph problems," *Theoretical Computer Science*, vol. 1, no. 3, pp. 237–267, 1976. (Cited on page 103.)

P. Giordano and P. Messidoro, "European and international verification and testing standards," in *Proceedings 4th International Symposium on Environmental Testing for Space Programmes, Liege, Belgium*, ESA, ESA SP-467, 2001. (Cited on page 20.)

M. Harrold, D. Rosenblum, G. Rothermel, and E. Weyuker, "Empirical studies of a prediction model for regression test selection," *IEEE Transactions on software engineering*, vol. 27, no. 3, pp. 248–263, 2001. (Cited on pages 67 and 93.)

J. Helton, "Uncertainty and sensitivity analysis in the presence of stochastic and subjective uncertainty," *Journal of Statistical Computation and Simulation*, vol. 57, no. 1 - 4, pp. 3–76, 1997. (Cited on page 36.)

B. Hendrickson and T. Kolda, "Graph partitioning models for parallel computing," *Parallel Computing*, vol. 26, no. 12, pp. 1519–1534, 2000. (Cited on page 103.)

International Electrotechnical Commission, "IEC61508: Functional safety of electronical/electronic/programmable electronic safety-related systems part 7 overview of techniques and measures," IEC Standard, 2005.
URL: *http://www.iec.ch/functionalsafety* (Cited on page 18.)

ISO-9126-1, "Information technology - software product quality - part 1: Quality model," International Organization for Standardization, 2000-03-20. (Cited on page 145.)

I. de Jong, R. Boumen, J. van de Mortel-Fronczak, and J. Rooda, "Software reliability qualification for semi-conductor manufacturing systems," *Advanced Semiconductor Manufacturing Conference*, pp. 326–332. (Cited on page 88.)

I. de Jong, R. Boumen, J. van de Mortel-Fronczak, and J. Rooda, "Integration and test sequencing for manufacturing systems," in *Proceedings of the 2006 INCOSE International Symposium*, INCOSE, 2006. (Cited on page 56.)

I. de Jong, R. Boumen, J. van de Mortel-Fronczak, and J. Rooda, "An overview of integration and test plans in organizations with different business drivers," in *Proceedings of the 5th Annual Conference on Systems Engineering Research (CSER)*, B. Sauser and G. Muller, Eds., vol. 1, Stevens Institute of Technology, Stevens Institute of Technology, 2007a. (Cited on page 14.)

I. de Jong, R. Boumen, J. van de Mortel-Fronczak, and J. Rooda, "Parallelizing test phases using graph partitioning algorithms," SE Report 2007-11, Eindhoven university of technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2007b.
URL: *http://se.wtb.tue.nl/sereports* (Cited on page 102.)

I. de Jong, R. Boumen, J. van de Mortel-Fronczak, and J. Rooda, "Selecting a suitable system architecture for integration and testing," in *Proceedings of the 2007 Bits&Chips Testdag*, 2007c. (Cited on page 145.)

I. de Jong, R. Boumen, J. van de Mortel-Fronczak, and J. Rooda, "Test set improvement using a next-best-test-case algorithm," SE Report 2007-12, Eindhoven university of technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2007d.
URL: *http://se.wtb.tue.nl/sereports* (Cited on page 125.)

I. de Jong, R. Boumen, J. van de Mortel-Fronczak, and J. Rooda, "Test strategy analysis for manufacturing systems," SE Report 2007-10, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands, 2007e.
URL: *http://se.wtb.tue.nl/sereports* (Cited on pages 64, 102, and 111.)

C. Kaner, J. Falk, and H. Nguyen, *Testing Computer Software*, John Wiley & Sons, 1999. (Cited on page 67.)

S. Kaplan, "The words of risk analysis," *Risk Analysis*, vol. 17, no. 4, pp. 407–417, August 1997.
URL: *http://www.blackwell-synergy.com.janus.libr.tue.nl/doi/abs/10.1111/j.1539-6924.1997.tb00881.x* (Cited on pages 11 and 36.)

G. Karypis, "Multilevel hypergraph partitioning," Technical report 02-25, University of Minnesota, Minneapolis, 2002. (Cited on page 103.)

G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, Washington, DC, USA: IEEE Computer Society, 1998, pp. 1–13. (Cited on page 103.)

B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970. (Cited on page 106.)

T. Koomen and M. Pol, *Test process improvement: a practical step-by-step guide to structured testing*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. (Cited on page 7.)

D. Kung, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen, "A test strategy for object-oriented programs," 1995. (Cited on page 65.)

B. Laugen, N. Acur, H. Boer, and J. Frick, "Best manufacturing practices - what do the best-performing companies do?" *International Journal of Operations and Production Management*, vol. 25, no. 2, pp. 131–150, 2005. (Cited on page 15.)

I. Levendel, "Testing large real-time systems: Harnessing software variance to reduce cost and increase quality," in *Workshop on High performance, Fault adaptive, Large Scale Real-Time systems*, T. Bapty, Ed., Vanderbilt University, Vanderbilt University, 2002. (Cited on page 67.)

S. Lim and J. Cong, "Multiway partitioning with pairwise movement," *iccad*, pp. 512–516. (Cited on page 124.)

R. Mahoney, "Integrating manufacturing test strategy with manufacturing production strategy," *AUTOTESTCON '97. 1997 IEEE Autotestcon Proceedings*, pp. 394–397. (Cited on page 65.)

D. McQueen, M. Kamal-Saadi, G. Byrne, M. Burke, and D. Lee, *Future Mobile Handsets*, Informa telecoms and media, 2006. (Cited on page 19.)

G. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, pp. 1–4, 1965.
URL: *http://download.intel.com/research/silicon/moorespaper.pdf* (Cited on page 1.)

G. Muller, *System architecting*, Embedded Systems Institute, 2007.
URL: *http://www.gaudisite.nl/SystemArchitectureBook.pdf* (Cited on page 145.)

G. Muller, "CAFCR: A multi-view method for embedded systems architecting; balancing genericity and specificity," Ph.D. thesis, Technische Universiteit Delft, Jan. 2004. (Cited on page 146.)

NIST/SEMATECH, *e-Handbook of Statistical Methods*, NIST/SEMATECH, 2003-2006.
URL: *http://www.itl.nist.gov/div898/handbook/* (Cited on pages 88, 89, 90, and 163.)

K. Pattipati, S. Deb, M. Dontamsetty, and A. Maitra, "START: System testability analysis and research tool," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 6, no. 1, pp. 13–20, Jan 1991. (Cited on pages 34 and 157.)

K. Pawar, U. Menon, and J. Riedel, "Time to market," *Integrated manufacturing systems*, vol. 5, no. 1, pp. 14–22, 1994. (Cited on page 15.)

S. L. Pfleeger, "Risky business: what we have yet to learn about risk management," *Journal of Systems and Software*, vol. 53, no. 3, pp. 265–273, 2000. (Cited on pages 11 and 36.)

M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Englewood Cliffs, NJ: Prentice Hall, 2001. (Cited on page 8.)

V. Raghavan, M. Shakeri, and K. Pattipati, "Optimal and near-optimal test sequencing algorithms with realistic test models," *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, vol. 29, no. 1, pp. 11–26, Jan 1999. (Cited on page 69.)

A. Raven, *Consider it Pure Joy... An introduction to clinical trials*, Cambridge Healthcare Research Ltd., 1997. (Cited on page 21.)

A. Raven, *Beyond What is Written. A researchers guide to good clinical practice*, Cambridge Healthcare Research Ltd., 1998. (Cited on page 21.)

S. Riyavong, "Experiments on sparse matrix partitioning," Tech. Rep. WN/-PA/03/32, CERFACS, 2003.
URL: *http://www.cerfacs.fr/algor/reports/2003/WN_PA_03_32.pdf* (Cited on page 104.)

A. de Ron and J. Rooda, "Equipment effectiveness: OEE revisited," *IEEE Transactions on Semiconductor Manufacturing*, vol. 18, no. 1, pp. 190–196, Feb. 2005. (Cited on page 159.)

G. Rothermel and M. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on software engineering*, vol. 22, no. 8, pp. 529–551, 1996. (Cited on pages 67 and 93.)

A. Rozinat, I. de Jong, C. Günther, and W. van der Aalst, "Process Mining of Test Processes: A Case Study," BETA Working Paper Series, WP 220, Eindhoven University of Technology, Eindhoven 2007. (Cited on pages 159 and 162.)

E. Sasidhar, L. Alkalai, and A. Chatterjee, "Test strategies for a 3-d stack multichip module space flight computer," in *Proceedings of the 1997 International Conference on Multichip Modules*, 1997. (Cited on page 65.)

SEI, "Introduction to the architecture of the CMMI framework," 2007. **URL:** *ftp://ftp.sei.cmu.edu/pub/documents/07.reports/07tn009.pdf* (Cited on page 6.)

SEMI, *SEMI E10-0600 Specification for definition and measurement of equipment reliability, availability, and maintainability (RAM)*, Semiconductor Equipment and Materials International, 1986-2000. (Cited on page 88.)

V. Stavridou, "Integration standards for critical software intensive systems," in *ISESS '97: Proceedings of the 3rd International Software Engineering Standards Symposium (ISESS '97)*, Washington, DC, USA: IEEE Computer Society, 1997, p. 99. (Cited on page 47.)

K. Tai, "Condition-based software testing strategies," in *Computer Software and Applications Conference, 1990. COMPSAC 90. Proceedings., Fourteenth Annual International*, 1990. (Cited on page 65.)

J. Tretmans, Ed., *Tangram: Model-based integration and testing of complex high-tech systems*, Embedded Systems Institute, 2007. (Cited on pages 10 and 14.)

B. Vastenhouw and R. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM Review*, vol. 47(1), pp. 67–95, 2005. (Cited on page 106.)

E. van Veenendaal, "Guidelines for testing maturity - the test maturity model," Tech. rep., TMMi-Foundation, 2006. **URL:** *http://www.tmmifoundation.org/downloads/resources/-TestMaturityModel.TMMi.pdf* (Cited on page 6.)

M. Villacourt and M. Mahaney, "Designing a reliability demonstration test on a lithography expose tool using Bayesian techniques," *IEEE Transactions on components, packaging, and manufacturing technology - Part A*, vol. 17, no. 3, pp. 458–462, 1994. (Cited on pages 88 and 90.)

ABOUT THE AUTHOR

Ivo de Jong was born on May 3th, 1971 in Rotterdam. He holds a B.Sc. in Laboratory informatics and automation from Breda Polytechnic. He worked as a software engineer at various companies in the USA and The Netherlands, before he started at ASML in 1996. At ASML, he worked as a test engineer, integrator, software integration team leader, software release project leader and project leader of a major reliability improvement project. Integration, testing and problem tracking have been the red threat in his career at ASML. In 2003, he joined the TANGRAM research project as a Ph.D. student at the Systems Engineering section of the Mechanical Engineering Department at the Eindhoven University of Technology. His research concerns integration and test planning, integration and test strategies, integration and test sequencing and the integration and test process. The resulting thesis, written under supervision of prof.dr.ir. J.E. Rooda and dr.ir. J.M. van de Mortel-Fronczak, is titled: Integration and test strategies for complex manufacturing machines.

TITLES IN THE IPA DISSERTATION SERIES SINCE 2002

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

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

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

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

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

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. 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

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

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

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

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

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

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

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

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

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

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

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

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

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

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

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

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

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

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

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

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

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

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

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

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

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

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

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J.Mathijssen. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W.Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multidisciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08