conference on systems engineering research
Paper #123

Optimal integration and test plans for software releases of lithographic systems[1]

R. Boumen[†], I.S.M. de Jong[†‡], J.M. van de Mortel-Fronckzak[†] and J.E. Rooda[†]
[†]Eindhoven University of Technology, 5600 MB Eindhoven, the Netherlands
[‡]ASML, 5500 AH Veldhoven, the Netherlands
{R.Boumen, I.S.M.d.Jong, J.M.v.d.Mortel, J.E.Rooda}@tue.nl
Corresponding author: R.Boumen, PhD student

Abstract

This paper describes a method to determine the optimal integration and test plan for embedded systems software releases. The method consists of four steps: 1) describe the integration and test problem in an integration and test model which is introduced in this paper, 2) determine possible test positioning strategies, 3) calculate integration and test plans for each test positioning strategy and 4) analyze the resulting plans on time-to-market and total test time and choose the optimal one. A test positioning strategy determines when a test phase starts and stops. We introduce several possible test phase strategies that can be used within the method. Using this method, for two specific ASML lithographic machine software releases (ASML 2006), we determined the optimal integration and test plans and, hence, the best test positioning strategy for ASML software testing.

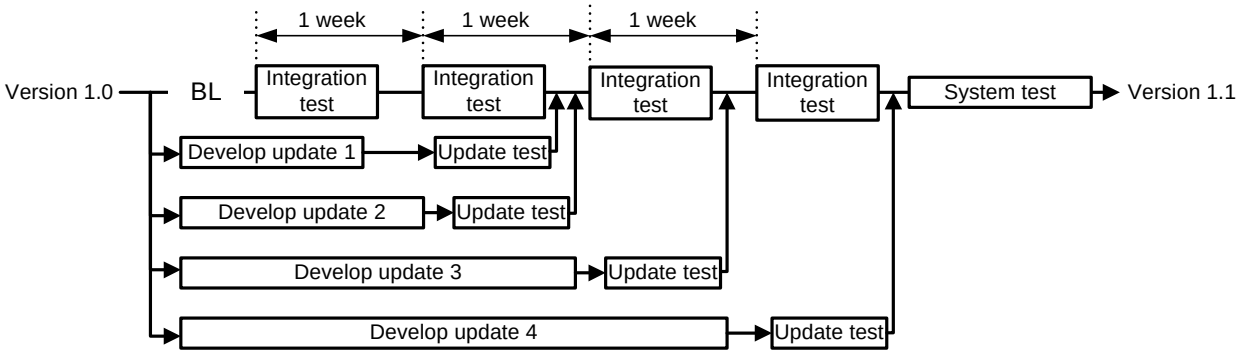
Introduction

Integration and testing of embedded systems software, like the software of an ASML lithographic system, is costly and time consuming. Therefore, testing is performed as early as possible and parallel to the development process. This is done by performing several test phases during the development process. At the end of the development process, a system test phase is performed. After this test phase, the software is released and can be installed on the embedded systems. To reduce time-to-market of a software release, it is important to choose the best integration and test plan. Such a plan describes the integration sequence of the different in parallel developed modules, the tests that should be performed in the test phases, and when test phases should be performed. The integration and test plan depends on the test positioning strategy that determines when a test phase starts and stops. For example, during the development of an ASML software release an integration test phase is performed every week and takes

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

SER 2007, Stevens Institute of Technology, ISBN - 0-9787122-1-8
PROCEEDINGS CSER 2007, March 14-16, Hoboken, NJ , USA

Figure 1. ASML software development process.



approximately one day. The question is whether this test positioning strategy is the best strategy aimed at releasing software as soon as possible. Many other strategies are possible.

In this paper, we introduce a method that can be used to calculate the optimal integration and test plan for a software integration and test problem based on a test positioning strategy. With this method it is possible to investigate which of the introduced test phase strategies results in an optimal integration and test plan. Optimal integration and test plans minimize time-to-market for a specific ASML software release while not increasing the total test effort compared to the current way-of-working. The method uses the integration and test sequencing method described in (Boumen, 2006b, 2006c) which has been successfully applied to optimize integration and test plans of lithographic systems prototypes (Boumen, 2006a, 2007).

The paper is structured as follows. The second section describes the general software development approach that is used within ASML. The third section introduces the integration and test planning method and the possible test phase strategies. The fourth section describes the optimization of ASML software release integration and test plans using the integration and test planning method. The final section gives conclusions.

Software integration and test

Before explaining the integration and test planning method, we first discuss the development approach of software within ASML. ASML uses an incremental concurrent software development approach. Incremental means that the development starts with an old software release that is updated to a new software release. Up to a few hundred software engineers make copies of a (part of) the software and concurrently update each part according to the new requirements. If the development of such an update is finished, it is integrated back in the release. Since this is an ongoing process, the release will slowly migrate to a complete new release. During this migration process the current release (with all updates) is called the baseline (BL). When all functionality is updated, the new release is ready and can be released to the customer. This development approach is illustrated in Figure 1, where each action is shown as a rectangle and the precedence relations are denoted by directed edges.

Software integration. The integration of software is the assembly of a software update with the BL. Within this update, one ore more components may be changed. During the development of this update, the software may be changed such that certain system requirements are not met anymore. To check

whether all requirements are met, tests need to be performed on the BL.

Software test. Within the ASML software development approach three test phases can be distinguished: 1) an update test phase, 2) an integration test phase, and 3) a system test phase. The update test phase is always performed on the software update by the software developers. This test phase is considered part of the development of the update and not discussed further in this paper. The integration test phase is performed on the BL. In the current way of working, test engineers perform this test phase every week. The duration of this test phase is one day. The system test is performed when all functionality is available and all updates are integrated. The duration of this test phase depends on the desired final quality of the software release. The choice of when test phases should start and stop is called the test positioning strategy. The current test positioning strategy of one 8 hour integration test per week and one system test after every update is integrated is a choice that might not result in the shortest time-to-market. If for example, many updates are integrated, longer integration test phases may be needed, while if nothing is integrated, no integration test phase is needed. Using the method described in the following section, the test positioning strategy can be determined that results in the optimal integration and test plan.

Illustration. To illustrate the software integration and test problem we introduce a small example that is also used in the following sections. Suppose we have a software baseline version 1.0 and by developing 4 updates in parallel, this baseline is updated to version 1.1. The software system has 8 requirements that can be tested by performing 8 tests that cover a certain set of these requirements. Each of the 4 updates that are made might cause that these requirements are not met. The goal is to determine the optimal integration and test plan. An integration and test plan consists of the integration sequence, the sequence of the integration test phases and the system test phase that should to be performed. Additionally, it contains the tests that need to be performed within each of these test phases.

Integration and test planning method

The integration and test plan optimization method consists of four steps: 1) describe the integration and test problem in an integration and test model which is introduced in this paper, 2) determine possible test positioning strategies, 3) calculate integration and test plans for each test positioning strategy and 4) analyze the resulting plans on time-to-market and total test time and choose the optimal one. All four steps are discussed in the following subsection.

Integration and test model. The integration and test problem must be defined as an integration and test model as introduced in (Boumen, 2006b, 2006c). This model contains the following basic elements with their properties:

- a set M of modules and the corresponding development times D_m,
- a set I of interfaces and the corresponding creation times D_i,
- a set T of tests and the corresponding execution times D_t,
- a set S of fault states and the corresponding impact S_i.

The modules are the updates of the software baseline that are performed in parallel. The development times are the planned delivery times of these updates. Also, the old software baseline is considered as a software module. The interfaces denote how the updates and old software baseline are connected and therefore how the system can be integrated. For software, an interface is constructed between each software update and the old software version meaning that each update can be integrated with the old software baseline. The test set is the set of available tests. The fault state set denotes the possible faults that may occur. For the software

integration and test problem, the fault states set is the set of requirements that may not be met by the system. The impact of a fault state denotes the importance of this fault state.

Besides the basic elements, the model also contains relations between these basic elements:

- relation R_{im} between the interfaces and modules denotes which interface is created between which modules,
- relation R_{st} between tests and fault states denotes which tests cover which fault states with a coverage between 0 and 1,
- relation R_{sm} between the fault states and the modules denotes which modules introduce which fault states with a certain fault probability,
- relation R_{tm} between tests and modules denotes which modules must be integrated before a test can be executed.

It must be noted that the model introduced in (Boumen, 2006b, 2006c) contains more relations, however for the software integration and test problems these relations are not needed and therefore not explained here.

Using this model, we can calculate the risk in the system at every point during integration and testing. The risk in the system is a measure of the quality of the system and can be used to decide when to start or stop testing. The risk in the system is defined as:

$$R = \sum_{s \in S} P(s) \cdot S_i(s)$$

where *P(s)* is de probability that a fault state *s* is present in the system and $S_i(s)$ denotes the impact of that fault state. The probability depends on the integrated modules and the performed tests. The probability that fault state *s* is present when a set of modules *M'* is integrated, is:

$$P(s) = 1 - \prod_{m \in M'} (1 - R_{sm}(s,m))$$

If a test is performed and passes, the probability that a fault state is present is reduced. The new probability *P'(s)* of a fault state *s* after test *t* is performed becomes:

$$P'(s) = P(s) \cdot (1 - R_{st}(s,t))$$

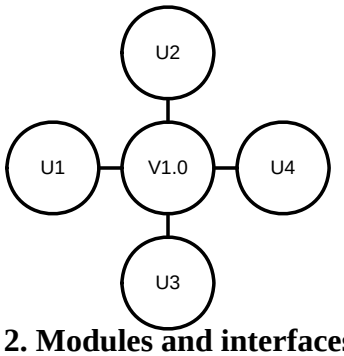


Figure 2. Modules and interfaces of the example model.

Table 1: Tests and fault states of the example model.

R_{st}	t1	t2	t3	t4	t5	t6	t7	t8	S_i
s1	1	0	0	0	1	0	1	0	10
s2	1	0	0	0	1	0	0	1	10
s3	0	1	0	0	1	0	1	0	10
s4	0	1	0	0	1	0	0	1	10
s5	0	0	1	0	0	1	1	0	10
s6	0	0	1	0	0	1	0	1	10
s7	0	0	0	1	0	1	1	0	10
s8	0	0	0	1	0	1	0	1	10
D_t	3	3	3	3	5	5	5	5	

Table 2: Fault states and modules of the example model.

R_{sm}	V1.0	U1	U2	U3	U4
s1	0.2	0	0.1	0	0.1
s2	0.1	0.3	0.1	0	0.1
s3	0	0.2	0	0	0
s4	0.2	0.1	0.3	0.3	0
s5	0.2	0	0.3	0.3	0
s6	0	0	0.1	0.2	0.1
s7	0.3	0	0.1	0.2	0.1
s8	0	0.2	0.1	0.2	0.1
D_m	0	2	4	10	12

In Figure 2 and Tables 1 and 2, the integration and test model of the example introduced in the previous section is shown. In Figure 2, the modules are denoted as nodes and the interfaces as edges. As explained before, each update (U1 through U4) and the old software version (V1.0) are modeled as modules. In Table 1, the tests and fault states and their relations are shown. A value of R_{st}= 1 denotes that test *t* covers fault state *s* with 100%, which means that if test *t* passes, fault state *s* is definitely not present. Also, the fault state impact (S_i) and the test execution durations (D_t) are shown.

In Table 2, the probability that a module introduces a fault state is shown. Also, the development times of each module are shown (D_m). The relations between the tests and the modules are simple: all tests can be performed when the module V1.0 (the old software baseline) is present and integrated. No tests can be done on the individual software updates.

Test positioning strategies. It is possible to start a test phase after each integration action of a subsystem with another second subsystem and test all possible fault states. However, this way of working may not be optimal because too much testing may be done. The test positioning strategy determines when test phases are started and when they stop. An example of such a test positioning strategy is the one currently used by ASML. This strategy determines that an integration test phase starts every week and takes approximately one day. Furthermore, the system test starts after the last integration action and stops when a certain risk threshold is reached. There are many test phase strategies possible. In this paper, we introduce a few alternatives. The following criteria can be used to decide when to start testing, see Figure 3:

- Exclude fault states as soon as possible: this criterion tries to exclude inserted fault states as soon as they can be tested by a certain test. This criterion is denoted by FA.

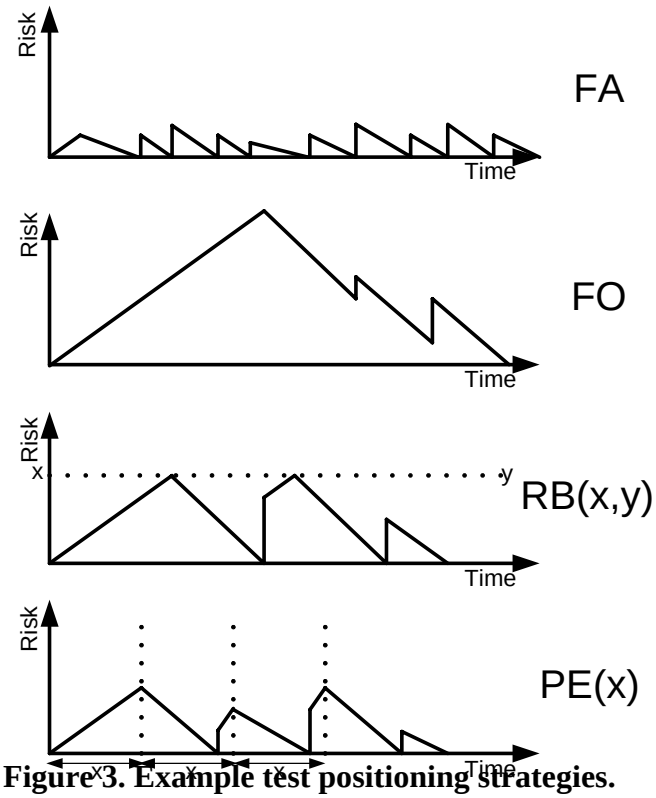


Figure 3. Example test positioning strategies.

- Exclude fault states once: this criterion excludes the fault states when they cannot be introduced anymore by modules that are not yet integrated. This criterion is denoted by FO. This strategy intends to test a certain fault state only once, such that test effort is reduced.
- Risk-based threshold reached: this criterion starts a test phase when the risk reaches a certain risk threshold. The risk threshold is defined as a linear function with a start risk threshold and an end risk threshold (see Figure 3). Such a criterion is denoted by RB(x,y), where x denotes the start risk value and y the end risk value of the threshold.
- Test periodic: this criterion starts a test phase after a certain period.

This criterion is denoted by PE(x) where x denotes the period.

Figure 3 shows the risk in time during the integration and test phase for each of the discussed start criteria.

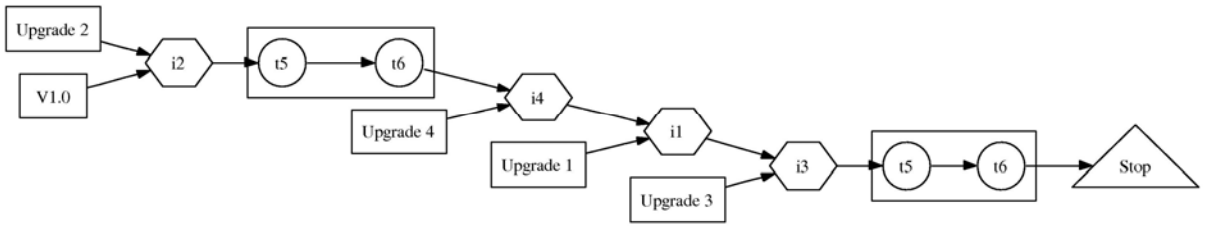
The following criteria can be used to decide when to stop testing:

- Stop testing when a certain duration is reached (e.g. 8 hours). This criterion is denoted by DU(x) where x denotes the duration.
- Stop testing when a certain risk level is reached. Such a criterion is denoted by RL(x,y), where x denotes the start risk value and y the end risk value only now for the stop criterion.

For all examples in Figure 3, the stop criterion is a risk based threshold of 0 (RL(0)). By choosing a start criterion and a stop criterion, a test positioning strategy is constructed that can be used to calculate an optimal integration and test plan.

Integration and test plan optimization. Now that we have defined a model and we can choose from several test phase strategies, we can calculate an optimal integration and test plan for each test positioning strategy. To do so, we use a combination of two algorithms. The first algorithm is the integration sequencing algorithm as described in (Boumen 2006c) and calculates an optimal integration sequence based on the modules and interfaces. This algorithm originates from assembly sequencing methods as introduced by (de Mello et. al 1991) and test dependency graphs described by (Hahn 2001). The algorithm performs an AND/OR graph search. The AND/OR graph represents all possible ways to disassemble a system into its modules. The optimal sequence of disassembly actions also represents the optimal sequence of integration actions. Depending on the test positioning strategy, a test phase is performed after an integration action of one subsystem, that may consists of one or more modules, with another subsystem. If such a test phase should be performed, the second algorithm calculates an optimal test sequence for that test phase given the set of tests that may be performed given the current assembly of modules and the fault states with their corresponding probabilities of being present. This algorithm is described in (Boumen, 2006b) and also performs an AND/OR graph search. This algorithm originates from diagnosis methods as described by (Pattipati et. al 1990). The test AND/OR graph denotes all possible sequences of tests. The algorithm takes the test sequence with the least duration and returns this duration to the integration sequencing algorithm that uses this duration as the duration of the test phase.

For the introduced example, an optimal integration and test plan is calculated for several test positioning strategies. The resulting integration sequence for the test positioning strategy with start criterion RB(15,15) is shown in Figure 5. This figure shows the integration and test plan as a directed tree, where rectangle nodes denote the development of modules, hexagonal nodes denote the integration of two subsystems, round nodes denote test actions and the directed edges denote the precedence relations. In this integration sequence one integration test phase and one system test phase (both pictured as rectangles around tests) are shown. The test sequence for both test phases is to perform first test 5 and then test 6.



Analysis. Integration and test plans are compared by looking at two parameters:

Table 3: Results illustration

Test positioning strategy		Time-to-market	System test time	Total test time
Start criterion	Stop criterion			
FA	RL(0,0)	44	8	72
FO	RL(0,0)	22	10	41
RB(15,15)	RL(0,0)	24	10	48
PE(10)	RL(0,0)	28	10	56

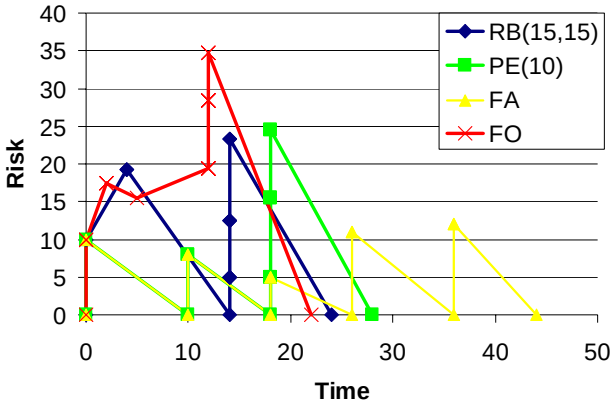
- the time-to-market of the software baseline,
- the total test time during the integration and test phase of a software baseline.

The time-to-market of a software baseline is the duration between the start of the development until the end of the system test phase. The maximal development time of all modules dominates the time-to-market since this is the minimal time-to-market. The total time-to-market is therefore the sum of the maximal development time and the duration of the last system test phase. Therefore, we often look at the last system test duration to compare different integration and test plans. The total test time is the sum of al integration test phase durations and the duration of the system test phase. This parameter is interesting because it shows the costs that have to be spent on testing.

In Figure 4, the risk profile is shown for the four different strategies that were investigated. In Table 3, the resulting durations for the remaining strategies are shown. As can be seen, the optimal integration and test plan is created with the test positioning strategy with start criterion FO. This solution has the minimal time-to-market and the least total test time.

Case studies

In the previous section, we introduced a method to determine the optimal integration and test plan for an integration and test problem. We also showed with a simple illustration how this method can be used for a software releasing problem. In this section, we



determine an optimal integration and test plan for two ASML software releases. This is done in two case studies described in the following subsections. The assumptions that were made during these case studies are:

- the modules are independent of each other,
- the quality of the BL during development is not important,
- only the test times are taken into account.

Case study 1. This case study deals with a large software release. In Table 4, the properties of the model are shown. The software updates and their belonging delivery times are known in advance. For each of these software updates, a test engineer estimates the probability that the update introduces a fault state (requirement failure). Also, test engineers have estimated the coverage of each test on each requirement for the complete

available test set. The test durations are also estimated. After the model is created, several test positioning strategies are considered, including the standard ASML test positioning strategy that starts an integration test phase every week for one day. For each of these strategies, an optimal integration and test plan is calculated. The optimal strategy should reduce the time-to-market as much as

possible, while not increasing the total test time.

Table 4: Case study model properties.

Property	Case study 1	Case study 2
# Modules	260	122
# Interfaces	259	121
# Fault states	55	55
# Tests	169	169

In Table 5, the results of different test phase strategies are shown, that is the time-to-market and total test time of the resulting solutions. In Figure 5, the risk profiles of the most interesting solutions are shown.

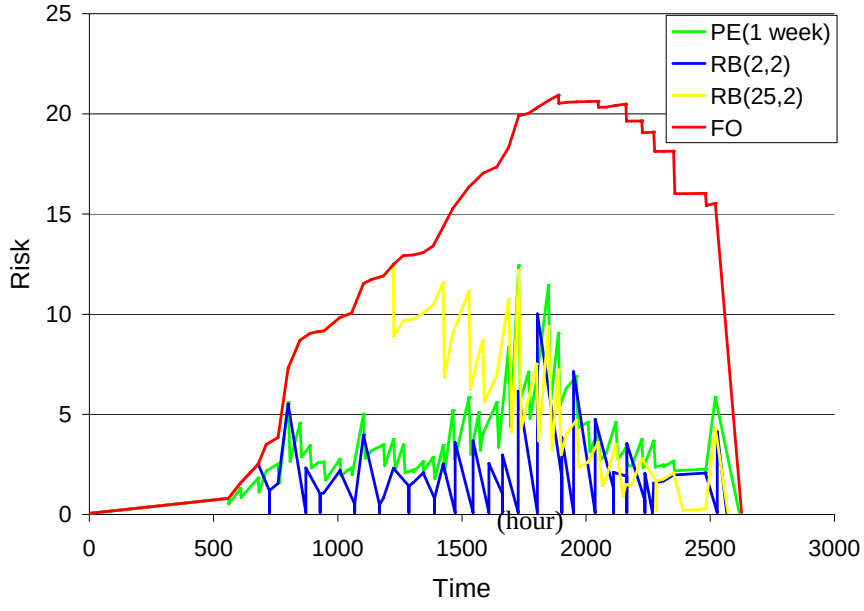
We can conclude from this case study that the test positioning strategy with start criterion RB(2,2) results in the least time-to-market (almost 45% reduction in the system test duration). However, this strategy also results in a large total test time. Strategy RB(25,2) (a risk based line that starts at 25 and ends at 2) also reduces time-to-market and reduces the total test time a little compared to the current ASML situation.

Case study 2. The second case study is the optimization of the integration and test plan of a smaller software release. The properties of this model are also shown in Table 4. For different test phase strategies, the optimal integration and test plan is calculated. In Table 6, the time-to-market and total test time of the different solutions are shown. The test positioning strategy with start criterion RB(1,1) has the best time-to-market, but this test positioning strategy also increases the total test time drastically. The strategy with start criterion RB(20,2) both reduces the time-

Table 5: Results case study 1

Test positioning strategy		Time to market (hour)	System test duration (hour)	Difference system test time	Total test time (hour)	Difference total test time
Start criterion	Stop criterion					
PE(1week)	DU(8 hours)	2617	97	-	2769	-
FA	DU(8 hours)	2618	98	1%	2792	1%
FO	RL(0.1,0.1)	2626	106	9%	2593	-6%
RB(2,2)	RL(0.1,0.1)	2566	46	-53%	3800	37%
RB(25,2)	RL(20,0.2)	2571	51	-47%	2748	-1%

Figure 5. Case study 1 risk results.



to-market and total test time, but the RB(25,2) strategy reduces the time-to-market even more for a little increase in the total test time. Depending on the specific importance of total test time, either of them can be chosen.

Conclusions

In this paper, we introduced a method to determine an optimal test positioning strategy for a software release integration and test problem. The method consists of creating a model of the problem, determining a set of possible test phase strategies, and calculating for each strategy an optimal integration and test plan. A test positioning strategy determines when a test phase should start and stop. The different integration and test plans can be compared on total test time and time-to-market which is the duration of the critical path in the integration and test plan.

We used this method to determine an optimal integration and test plan for ASML software releases that are used to control lithographic systems. This is done for two different software releases. Both case studies showed that the current test positioning strategy used by ASML is theoretically not optimal. This test positioning strategy determines that every week a test phase is started with a fixed duration of one day. The optimal integration and test plan was obtained by using a risk-based test positioning strategy that starts testing when the risk reaches a certain risk level. This risk threshold may decrease in time to reduce the total test time.

However, this theoretical result is obtained by making the assumption that the quality of the BL during development is not important. This assumption is in reality not true, since the updates are copied from the BL, which therefore needs to be of reasonable quality. Therefore, we propose to use a test positioning strategy that keeps the risk at a convenient level during development, while decreasing the risk towards the desired end risk level at the end of development.

As future research we are investigating whether the test costs may be dependant on time or the integrated modules. Also, the diagnosis and fix costs may be larger when the complete system is integrated instead of one single module.

References

ASML, http://www.asml.com, 2006.

Boumen, R., De Jong, I.S.M., van de Mortel-Fronczak, J.M. and Rooda, J.E., "Test time reduction by optimal test sequencing", *Proceedings of the 16th International Symposium of INCOSE*, 2006.

Boumen, R., and De Jong, I.S.M. and Vermunt, J.W.H. and van de Mortel-Fronczak, J.M. and Rooda, J.E., "Test sequencing in complex manufacturing systems," Accepted for *IEEE Transactions on Syst., Man, and Cybern. - Part A, 2006.*

Boumen, R., De Jong, I.S.M., Mestrom, J.M.G., van de Mortel-Fronczak, J.M. and Rooda, J.E., "Integration sequencing for complex systems," Eindhoven University of Technology, ISSN:1872-1567, Report number 2006-2, 2006.

Table 6: Results case study 2

Test positioning strategy		Time to market (hour)	System test duration (hour)	Difference system test time	Total test time (hour)	Difference total test time
Start criterion	Stop criterion					
PE(1 week)	DU(8 hours)	704	80	-	148	-
FA	DU(8 hours)	700	76	-5%	173	17%
FO	RL(0.2, 0.2)	697	73	-9%	106	-28%
RB(1, 1)	RL(0.2, 0.2)	669	45	-44%	550	272%
RB(20, 2)	RL(15, 0.2)	684	60	-25%	135	-9%
RB(25, 2)	RL(22 - 0.2)	674	50	-38%	153	3%

Boumen, R., De Jong, I.S.M., van de Mortel-Fronczak, J.M. and Rooda, J.E., "Optimal integration and test planning applied to lithographic systems", *Submitted to the 17th Intern. Symposium of INCOSE*, 2007.

De Mello, L. S. H., Sanderson, A. C., "A correct and complete algorithm for the generation of mechanical assembly sequences," *IEEE transactions on Robotics and Automation, vol. 7, no. 2, pp. 228–240,* April 1991.

Hanh, V.L., Akif, K., Traon Y.L. and Jézéquel J.M., "Selecting an efficient oo integration testing strategy: An experimental comparison of actual strategies", *Proceedings of ECOOP 2001,* 2001.

Pattipati, K.R. and Alexandridis, M.G., "Application of heuristic search and information theory to sequential diagnosis". *IEEE Trans. Syst. Man, Cybern.*, 20:872–887, July/August 1990.

Biography

R. Boumen received his M.Sc. degree in Mechanical Engineering from the Eindhoven University of Technology, the Netherlands, in 2004. As a master student, he worked in the field of supervisory machine control of lithographic machines. Since 2004 he has been a Ph.D. student at the Eindhoven University of Technology. His research concerns test strategy within the Tangram project.

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

J.M. van de Mortel-Fronczak graduated in computer science at the AGH University of Science and Technology of Cracow, Poland, in 1982. In 1993, she received the Ph.D. degree in computer science from the Eindhoven University of Technology, the Netherlands. Since 1997 she works as an assistant professor at the Department of Mechanical Engineering, Eindhoven University of Technology. Her research interests include specification, design, analysis and verification of supervisory machine control systems.

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